

CODEGYM

Raising the bar

LẬP TRÌNH CĂN BẢN?

Xây dựng nền tảng lập trình trong 1 tháng



Mục lục

Lời mở đầu	1
Hướng dẫn sử dụng sách	2
Chương 1 - Nhập môn lập trình	5
1. Mục tiêu	5
2. Giới thiệu.....	5
3. Máy tính hoạt động như thế nào?.....	5
4. Phần mềm được tạo ra như thế nào?	9
5. Quy trình tạo ra một phần mềm	10
6. Các vai trò trong lập trình.....	11
7. Ngôn ngữ lập trình	13
8. Thuật toán	14
9. Mô tả thuật toán bằng mã giả.....	15
10. Mô tả thuật toán bằng lưu đồ	16
11. Một số cấu trúc thường gặp trong thuật toán	17
12. Ngôn ngữ lập trình JavaScript	22
13. Chương trình đầu tiên	22
14. Cài đặt công cụ lập trình.....	23
15. Bài thực hành	24
16. Bài tập.....	25
17. Bài kiểm tra	26
18. Tổng kết.....	28
Chương 2 - Biến, kiểu dữ liệu và toán tử	30
1. Mục tiêu	30
2. Giới thiệu.....	30
3. Biến.....	30
4. Kiểu dữ liệu	34
5. Phép toán toán học	36
6. Phép toán logic.....	40
7. Phép toán so sánh.....	43
8. Đọc dữ liệu từ bên ngoài	44

9. Hiển thị dữ liệu	46
10. Mã sạch	47
11. Các lỗi thường gặp	48
13. Bài thực hành	49
14. Bài tập.....	50
15. Bài kiểm tra	50
16. Tổng kết.....	52
Chương 3 - Câu lệnh điều kiện	53
1. Mục tiêu	53
2. Giới thiệu.....	53
3. Cấu trúc điều kiện	54
4. Cấu trúc điều kiện if-else	54
5. Cấu trúc điều kiện if-else lồng nhau.....	56
6. Cấu trúc điều kiện if-else bậc thang	57
7. Cấu trúc điều kiện switch-case	58
8. Mã sạch	61
9. Các lỗi thường gặp	62
10. Bài thực hành	66
11. Bài tập.....	77
12. Bài kiểm tra	77
13. Tổng kết.....	79
Chương 4 - Câu lệnh lặp	80
1. Mục tiêu	80
2. Giới thiệu.....	80
3. Câu lệnh lặp	80
4. Câu lệnh lặp while	81
5. Câu lệnh lặp do-while.....	82
6. Câu lệnh lặp for	82
7. Câu lệnh lặp lồng nhau	84
8. Câu lệnh break.....	85
9. Câu lệnh continue	86
10. Mã sạch	86

11. Các lỗi thường gặp	87
12. Bài thực hành	88
13. Bài tập.....	93
14. Bài kiểm tra	94
15. Tổng kết.....	96
Chương 5 - Mảng	97
1. Mục tiêu	98
2. Giới thiệu.....	98
3. Mảng	98
4. Duyệt qua các phần tử của mảng	102
5. Mảng nhiều chiều	103
6. Các thao tác thông dụng với mảng	106
7. Mã sạch	108
8. Các lỗi thường gặp	109
9. Bài thực hành	110
10. Bài tập.....	115
11. Bài kiểm tra	116
12. Tổng kết.....	118
Chương 6 - Hàm	119
1. Mục tiêu	119
2. Giới thiệu.....	119
3. Hàm.....	120
4. Giá trị trả về của hàm	124
5. Tham số của hàm	125
6. Phạm vi của biến	125
7. Hàm đệ quy	126
8. Mã sạch	127
9. Các lỗi thường gặp	129
10. Bài thực hành	130
11. Bài tập.....	133
12. Bài kiểm tra	134
13. Tổng kết.....	136

Chương 7 - Thuật toán tìm kiếm.....	137
1. Mục tiêu	137
2. Giới thiệu.....	137
3. Tìm kiếm tuyến tính.....	137
4. Tìm kiếm nhị phân	139
5. Độ phức tạp của thuật toán.....	142
6. Các lỗi thường gặp	144
7. Bài thực hành	146
8. Bài tập.....	150
9. Bài kiểm tra	150
10. Tổng kết.....	151
Chương 8 - Thuật toán sắp xếp.....	152
1. Mục tiêu	152
2. Giới thiệu.....	152
3. Thuật toán sắp xếp nổi bọt.....	152
4. Thuật toán sắp xếp chèn	155
5. Thuật toán sắp xếp chọn	157
6. Các lỗi thường gặp	159
8. Bài thực hành	159
9. Bài tập.....	161
10. Bài kiểm tra	162
11. Tổng kết.....	162
Tổng kết.....	164
Đội ngũ tác giả	167

Lời mở đầu

Với sự phát triển nhanh chóng của ngành Công nghệ Thông tin trong những năm gần đây, nhu cầu nhân sự trong ngành này tăng mạnh ở cả quy mô thế giới lẫn Việt Nam. Học lập trình không còn là một lựa chọn xa lạ với nhiều thanh niên Việt Nam nữa. Ngày càng có nhiều bạn trẻ muốn tìm đến với công việc lập trình, bắt đầu bằng việc tham gia vào các trường đào tạo chính quy, các trung tâm dạy lập trình, hoặc tự học.

Với hàng chục năm làm việc trong ngành Công nghệ Thông tin và trực tiếp tham gia vào công tác đào tạo lập trình viên, chúng tôi nhận ra rằng các bạn trẻ gặp rất nhiều khó khăn trong những thời điểm đầu tiên tiếp xúc với công việc này. Có rất nhiều rào cản khiến cho việc học lập trình trở nên khó khăn hơn, có thể kể đến như do chương trình đào tạo được thiết kế không tốt, giáo trình khó hiểu, chương trình không sát với thực tế, giáo trình không giúp người học xây dựng được kiến thức một cách bài bản và vững vàng. Bên cạnh đó, mặc dù có rất nhiều sách dạy lập trình bằng tiếng Anh rất tốt, nhưng nhiều người lại không sử dụng được tiếng Anh. Đó là những thiệt thòi và khó khăn mà những bạn trẻ đang muốn tìm đến ngành nghề lập trình đang gặp phải hiện nay.

Với mục tiêu giúp những người mới bắt đầu học lập trình được thuận lợi hơn, đội ngũ Giảng viên của Hệ thống Đào tạo Lập trình Hiện đại CodeGym đã biên soạn nên cuốn sách này và gửi đến các bạn. Cuốn sách này dành cho những người bắt đầu từ con số 0 và mong muốn phát triển năng lực lập trình trong thời gian nhanh nhất. Cuốn cẩm nang được thiết kế một cách khoa học để giúp người học tự xây dựng năng lực của mình thông qua từng hoạt động một. Không có quá nhiều thuật toán phức tạp, nhanh chóng làm ra được các sản phẩm để chứng minh sự tiến bộ của người học, sử dụng ngôn ngữ lập trình Javascript với độ tương tác cao là những đặc điểm nổi bật của cuốn cẩm nang này.

Mặc dù đội ngũ tác giả chúng tôi đã nỗ lực trong việc hoàn thiện cuốn sách này với tiêu chí dễ hiểu, khoa học và hiệu quả dành cho người mới bắt đầu, tuy nhiên khó để tránh khỏi các sai sót trong quá trình biên soạn. Vì vậy, chúng tôi rất mong nhận được các ý kiến phản hồi và đóng góp của mọi người thông qua email info@codegym.vn. Cảm ơn và chúc các bạn gặt hái được nhiều giá trị từ cuốn sách này.

*Trân trọng!
Đội ngũ tác giả.*

Hướng dẫn sử dụng sách

Cuốn sách này hướng đến mục tiêu giúp người học phát triển cả *kiến thức lẫn kỹ năng*. Cho nên, để khai thác cuốn sách một cách hiệu quả nhất, người học nên thực hiện các công việc sau đây:

- **Ghi chép:** Cần có một cuốn sổ để ghi chép, hoặc sử dụng các công cụ số như các trình soạn thảo hoặc Mind Map. Việc ghi chép sẽ giúp người học đảm bảo được mức độ hiểu của mình đối với các khái niệm quan trọng. Ghi chép cũng giúp cho người đọc ghi nhớ tốt hơn các khái niệm.
- **Tìm kiếm thông tin ngoài cuốn sách:** Có những khái niệm mà người đọc không hiểu được, ngay cả khi đã đọc đi đọc lại nhiều lần. Người đọc nên tìm kiếm thông tin về các khái niệm đó ở những nguồn khác ngoài cuốn sách, để tìm cách hiểu chúng. Lí do là bởi vì có nhiều cách để trình bày một khái niệm nào đó, các tác giả mặc dù đã cố gắng để trình bày chúng một cách dễ hiểu và phù hợp với phần đông các độc giả, nhưng có thể ở một vài chỗ và đối với một vài người thì vẫn bị khó hiểu. Đừng đầu hàng, mà nên tìm cách để hiểu chúng ở phạm vi ngoài cuốn sách.
- **Luyện tập:** Thực hành là thao tác bắt buộc để có thể phát triển kỹ năng. Người học không thể có được kỹ năng tốt nếu chỉ đọc cuốn sách từ đầu tới cuối mà không thực hành trong thực tế. Việc thực hành nên tăng theo cấp độ từ dễ đến khó. Đối với các bài thực hành có hướng dẫn, người học có thể làm theo các hướng dẫn đó. Nhưng sau khi làm theo thì cần phải tiếp tục thực hành để mình có thể tự làm được mà không cần làm theo hướng dẫn.
- **Tổng kết sau mỗi chương:** Sau khi hoàn thành mỗi chương, người học cần xem lại phần ghi chép của mình, đối chiếu với các mục tiêu để xem thử mình đã phát triển được những năng lực gì. Nếu có phần nào mình chưa hiểu hoặc chưa làm được thì hãy cố gắng tìm thêm sự trợ giúp để có thể giải quyết đầy đủ.

Cuốn sách này bao gồm 8 chương, lần lượt đề cập đến các khái niệm nền tảng trong lập trình. Tuy nhiên, nội dung của các chương không hoàn toàn được trình bày một cách tuyến tính, mà được tổ chức theo lối tích hợp. Nghĩa là một khái niệm có thể được trình bày trải đều qua nhiều chương khác nhau, giúp cho người học có cơ hội cọ xát nhiều lần, lặp đi lặp lại các thao tác thực hành, giúp xây dựng được một nền tảng hiểu biết và kỹ năng bền vững. Sau đây là cấu trúc của mỗi chương, người đọc nên hiểu cấu trúc này để có cách khai thác nội dung một cách hiệu quả nhất:

- **Mục tiêu:** Là nơi định nghĩa rõ ràng và ngắn gọn về những kiến thức và kỹ năng mà người đọc sẽ xây dựng được khi hoàn thành mỗi chương. Người học cần ghi chép lại mục tiêu của từng chương, để sau này tự đánh giá lại xem mình đã đạt được các mục tiêu đó hay chưa. Nếu không nắm được các mục

tiêu thì người học sẽ rơi vào trạng thái hoang mang, bởi vì không biết mình đang hướng đến điều gì.

- **Giới thiệu:** Là nơi trình bày ngắn gọn về mục đích và nội dung của chương. Phần này giúp người học trả lời được câu hỏi WHAT (tôi đang học cái gì?) và WHY (thấy được sự liên quan giữa những khái niệm mình sẽ học với các ứng dụng thực tế).
- **Các mục nội dung:** Phần chính của mỗi chương sẽ trình bày về các *khái niệm* và *ứng dụng* của chúng. Mỗi khái niệm sẽ được trình bày đầy đủ *định nghĩa*, *diễn giải*, *các tình huống ứng dụng*, *các ví dụ cụ thể* và những thông tin khác xung quanh khái niệm đó.
- **Mã sạch:** Mã sạch là một phần đặc biệt của cuốn sách để định hướng người đọc trở thành một lập trình viên tốt ngay từ đầu. Trong lập trình, có nhiều cách để giải quyết một vấn đề, nhưng chỉ có một số cách thực sự gọn gàng và hiệu quả. Mã sạch là cách để chúng ta tổ chức mã nguồn thực sự khoa học và hướng đến sự hiệu quả nói chung.
- **Các lỗi thường gặp:** Phần này liệt kê các lỗi mà một lập trình viên non kinh nghiệm thường mất nhiều thời gian để nhận ra và khắc phục. Chúng ta không thể liệt kê hết tất cả các lỗi có thể xảy ra khi làm việc với một cấu trúc lập trình nào đó, tuy nhiên việc nhận biết được các lỗi thường gặp sẽ giúp cho những người mới học lập trình tiết kiệm được khá nhiều thời gian.
- **Bài thực hành:** Đây là nội dung được thiết kế theo hướng “*cầm tay chỉ việc*”, có tác dụng giúp người đọc bắt đầu rèn luyện kỹ năng. Các bài thực hành được thiết kế theo hướng tăng dần về độ khó và quy mô, tương ứng với sự kỳ vọng tăng trưởng về kỹ năng của người đọc. Kết thúc phần *Bài thực hành*, người đọc có thể bắt tay vào tự thực hiện các bài tập để nâng cao hơn nữa kỹ năng của mình.
- **Bài tập:** Là cách thức để người học tự rèn luyện kỹ năng. Nếu nhìn vào lộ trình tổng quan, chúng ta sẽ đi từ việc học lí thuyết (thông qua phần *Mục tiêu*, *Giới thiệu*, *Các khái niệm*, *Các ví dụ*,...) rồi sau đó đến rèn luyện kỹ năng (thông qua phần *Thực hành* và *phần Bài tập*). Khi hoàn thành phần *Bài tập* thì kỹ năng của người đọc sẽ thành thạo hơn so với khi chỉ mới hoàn thành phần *Bài thực hành*.
- **Bài kiểm tra:** Đây là mục giúp người đọc tự đánh giá lại được kiến thức và kỹ năng của mình sau khi hoàn thành được một chương. Đáp án của các câu hỏi được đặt ở phần cuối của bài kiểm tra, giúp người học tự chấm điểm được. Nếu có những khái niệm nào mà mình chưa trả lời được thì người học nên quay trở lại phần trước đó để đọc kỹ hơn.

- **Tổng kết:** Là nơi giúp người học rà soát lại nhanh những kiến thức và kỹ năng mình đã học được. Phần tổng kết này được viết sẵn với nội dung khá ngắn gọn, nhưng chúng tôi vẫn khuyến khích người học nên tự viết lại phần tổng kết của riêng mình với mức độ chi tiết hơn theo cách hiểu của mình. Người học cũng tự đối chiếu lại với các mục tiêu được liệt kê ở đầu chương, để xem mình đã đạt được các mục tiêu đó hay chưa.

codegym.vn

Chương 1 - Nhập môn lập trình

Các khái niệm căn bản về máy tính và lập trình

1. Mục tiêu

- Giải thích được tổng quan cách hoạt động của máy tính
- Phân biệt được phần cứng và phần mềm
- Liệt kê được một số phần cứng thông dụng
- Liệt kê được một số phần mềm thông dụng
- Biết được vai trò của phần cứng, phần mềm và người dùng
- Trình bày được tổng quan quá trình tạo ra một phần mềm
- Giải thích được vai trò và ý nghĩa của ngôn ngữ lập trình
- Biết được tổng quan quy trình sản xuất một phần mềm
- Phân biệt được các vai trò thông dụng trong một quy trình sản xuất phần mềm
- Giải thích được ý nghĩa của thuật toán
- Mô tả được thuật toán bằng mã giả
- Mô tả được thuật toán bằng lưu đồ
- Cài đặt được phần mềm để bắt đầu viết mã
- Tạo được ứng dụng phần mềm đầu tiên

2. Giới thiệu

Trong chương đầu tiên của cuốn sách, chúng ta sẽ tìm hiểu về những khái niệm căn bản nhất về thế giới máy tính và lập trình. Đây là những khái niệm cơ bản nhất, cần thiết nhất trước khi bắt đầu viết những dòng mã đầu tiên. Chúng ta sẽ tìm hiểu về những khái niệm liên quan đến máy tính, phần mềm, người dùng, công việc sản xuất phần mềm, các bước để làm ra phần mềm. Cuối cùng, chúng ta sẽ cài đặt môi trường và chuẩn bị cho việc tạo ra một phần mềm đầu tiên.

Kết thúc chương này, chúng ta có thể bắt tay vào viết được những dòng mã đầu tiên để tạo ra các phần mềm.

3. Máy tính hoạt động như thế nào?

3.1. Định nghĩa máy tính

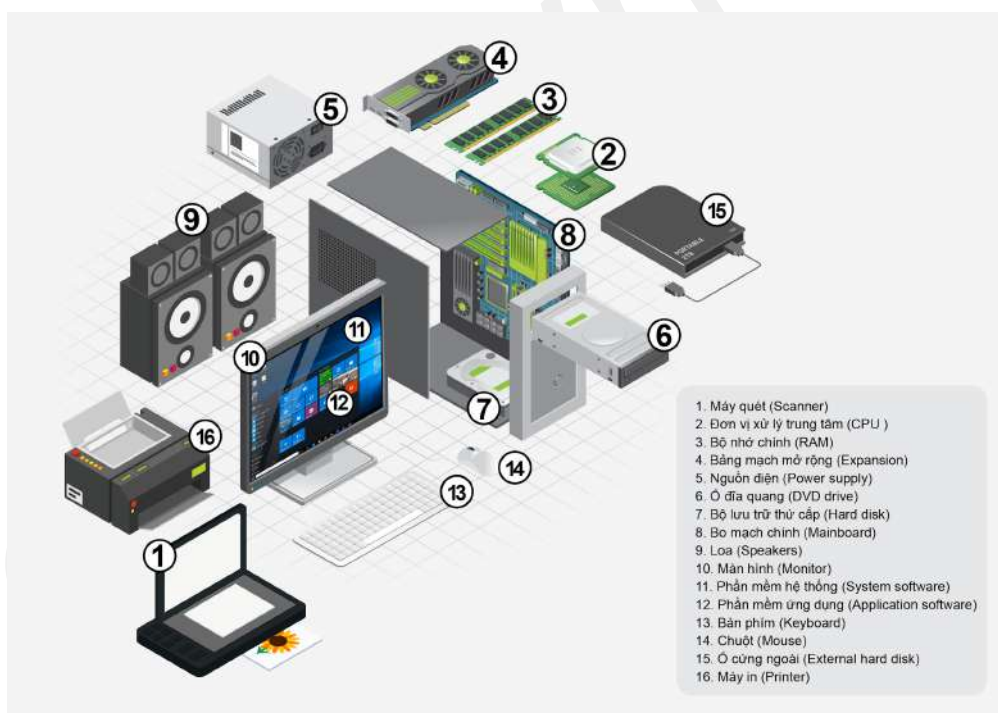
Thuật ngữ máy tính thường được sử dụng để chỉ đến các thiết bị điện tử có khả năng xử lý dữ liệu và chuyển đổi dữ liệu thành những thông tin hữu ích. Máy tính có được các khả năng này là nhờ sự kết hợp giữa các vi mạch điện tử và các tập lệnh do các lập trình viên tạo nên.

Có thể phân loại máy tính theo nhiều tiêu chí khác nhau, chẳng hạn như về mục đích sử dụng, kích thước, cấu tạo hoặc công nghệ. Một số loại máy tính thường được nhắc đến là: Máy tính tính lớn (mainframe), máy tính nhỏ (minicomputer), siêu máy tính (supercomputer), máy vi tính (microcomputer) hoặc máy tính cá nhân (personal computer), máy tính xách tay (laptop). Các thiết bị như điện thoại thông minh (smart phone) hoặc đồng hồ thông minh (smart watch) cũng là một dạng máy tính.

Thông thường, khi người ta nói đến một hệ thống máy tính thì bao gồm 3 thành phần là: Phần cứng, Phần mềm và Người dùng.

3.2. Phần cứng

Các thành phần vật lý kiến tạo lên máy tính được gọi là *phần cứng*. Phần cứng là bất cứ bộ phận nào của máy tính mà bạn có thể chạm tay vào được. Phần cứng máy tính bao gồm các *thiết bị điện tử tích hợp với nhau* được sử dụng để điều khiển các hoạt động nhập xuất và xử lý của máy tính. Một số phần cứng cơ bản bao gồm bộ nguồn điện, CPU (Central Processing Unit - Bộ xử lý trung tâm), RAM (Random Access Memory – Bộ nhớ Truy xuất Ngẫu nhiên), Motherboard (Bảng mạch chính, còn gọi là Mainboard), một số card mở rộng, thiết bị ngoại vi và các thành phần khác.



Hình 1.1: Các thành phần cấu tạo của máy tính

- Bảng mạch chính (Motherboard): Là bảng mạch quan trọng nhất, có nhiệm vụ kết nối các thành phần khác lại với nhau để hoạt động trong một thể thống nhất
- CPU – Bộ xử lý trung tâm: Có thể coi như bộ não của máy tính; thiết bị này tổ chức và thực hiện các chỉ thị của người dùng hoặc phần mềm

- Bộ nhớ: Là một bảng mạch điện tử nhỏ bên trong máy tính. Chương trình chạy trên máy tính sẽ được nạp vào bộ nhớ và chạy từ đó. Bộ nhớ được phân thành hai loại là bộ nhớ sơ cấp và bộ nhớ thứ cấp. Bộ nhớ sơ cấp còn được gọi là bộ nhớ chính. Chúng bao gồm RAM (Random Access Memory – Bộ nhớ Truy xuất Ngẫu nhiên) hoặc ROM (Read-Only Memory – Bộ nhớ Chỉ đọc). Bộ nhớ thứ cấp đề cập tới các bộ lưu trữ trong hoặc ngoài được sử dụng cho các dữ liệu bền vững như đĩa mềm, ổ băng từ, đĩa quang (CD) hoặc ổ USB, v.v.
- Thiết bị đầu vào: Cho phép nhận dữ liệu và các chỉ thị từ người dùng hoặc từ hệ thống máy tính khác. Chẳng hạn như: bàn phím, chuột, đầu đọc đĩa CD, camera, màn hình cảm ứng, v.v.
- Thiết bị đầu ra: Cho phép hiển thị kết quả thực thi các mệnh lệnh. Chẳng hạn như: màn hình, máy in, loa, máy chiếu, v.v.

3.3. Phần mềm

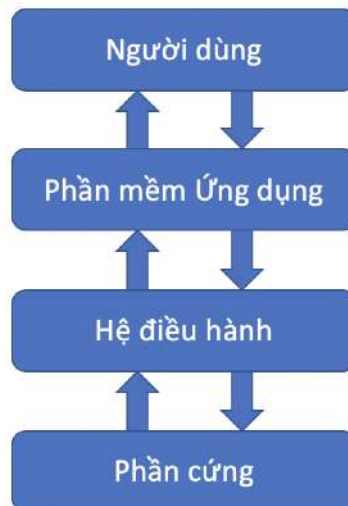
Phần mềm máy tính là tập hợp các chương trình máy tính và các dữ liệu có liên quan để cung cấp cho máy tính các chỉ dẫn cần thiết về những gì mà nó phải thực hiện. Trái ngược hẳn với phần cứng, phần mềm là một thứ vô hình, chúng ta không thể chạm được vào nó. Bạn có thể liên tưởng rằng máy tính như một thực thể sống với *phần xác* (đó là phần cứng) và *phần hồn* (đó là phần mềm). Với phần cứng, bạn có thể lắp ráp thành một máy tính. Tuy nhiên, máy tính cần phải có phần mềm để thực hiện các nhiệm vụ của mình.

Phần mềm máy tính có thể phân ra thành một số loại chính như phần mềm hệ thống, phần mềm ứng dụng và phần mềm lập trình.

- Phần mềm hệ thống: là những phần mềm máy tính được thiết kế để vận hành các thiết bị phần cứng và cung cấp, duy trì một nền tảng để chạy các phần mềm ứng dụng. Chẳng hạn như: Hệ điều hành, phần mềm diệt vi-rút, phần mềm quản lý mạng cho máy tính, v.v.
- Phần mềm ứng dụng (hoặc còn gọi tắt là ứng dụng): được thiết kế để giúp người dùng thực hiện một hay nhiều các công việc cụ thể nào đó. Ví dụ như các phần mềm doanh nghiệp, phần mềm kế toán, bộ phần mềm văn phòng, phần mềm xử lý ảnh và các phần mềm nghe nhạc, xem video, v.v.
- Phần mềm lập trình, những phần mềm này giúp các lập trình viên máy tính tạo ra các phần mềm khác. Ví dụ như phần mềm để viết mã nguồn, phần mềm để biên dịch, phần mềm để cài đặt, v.v.

3.4. Hệ điều hành

Hệ điều hành là một dạng phần mềm đặc biệt, trực tiếp được cài đặt lên các phần cứng để điều khiển chúng, cung cấp môi trường để các phần mềm khác có thể hoạt động, và đồng thời cũng cung cấp môi trường để người dùng tương tác với máy tính.



Hình 1.2: Mối liên quan giữa Người dùng - Ứng dụng – Hệ điều hành – Phần cứng

Có rất nhiều các hệ điều hành khác nhau được sử dụng cho các hệ thống máy tính, có thể kể đến như: Microsoft Windows, MacOS, các dòng Linux khác nhau (như Ubuntu, Debian, CentOS, Fedora...). Ngoài ra, còn có các hệ điều hành dành riêng cho các thiết bị di động, chẳng hạn như: Android, iOS, KaiOS,... Thậm chí, ngày nay, một số đơn vị đã nghĩ đến việc hợp nhất các hệ điều hành cho máy tính cá nhân với các hệ điều hành dành cho các thiết bị di động để gia tăng tính tương thích.

3.5. Người dùng

Những người sử dụng máy tính để làm các công việc cụ thể nào đó được gọi là Người dùng (User). Do máy tính chỉ là các công cụ, cho nên chúng cần phải có người điều khiển thì mới đem lại các lợi ích cụ thể. Người dùng có thể được phân loại là power user (Người dùng quyền lực), đó là những người dùng hiểu biết về hệ thống máy tính, hoặc end user (Người dùng cuối), đó là những người dùng không cần có quá nhiều hiểu biết về máy tính, mà chỉ cần học cách sử dụng các phần mềm đã được tạo sẵn để xử lý các nghiệp vụ hằng ngày.

3.6. Dữ liệu

Dữ liệu bao gồm các sự việc độc lập hoặc các mẫu thông tin, bản thân chúng thường không mang lại ý nghĩa cho con người. Máy tính đọc và lưu trữ dữ liệu ở dạng như văn bản, số liệu, hình ảnh hoặc âm thanh dưới cùng một dạng đó là các con số. Do đó, dữ liệu máy tính là dữ liệu số, nghĩa là chúng được tối giản xuống thành số. Nhìn chung, nghiệp vụ tính toán quan trọng nhất của máy tính là tập hợp dữ liệu (được gọi là đầu vào - input), xử lý chúng thành các dữ liệu đầu ra (output) hữu ích cho con người.

Hầu hết các máy tính ngày nay đều sử dụng hệ thống số nhị phân, bao gồm 2 giá trị là 0 và 1. Ngoại trừ một số máy tính rất đặc biệt khác, có thể có các dạng dữ liệu khác,

chẳng hạn như máy tính lượng tử. Ở trong máy tính, có thể sử dụng một số cơ chế khác nhau để biểu diễn giá trị 0 và 1, chẳng hạn cơ chế quang học (phản xạ thì là 1, không phản xạ là 0), cơ chế từ trường (nam thì 0, bắc thì 1), cơ chế hiệu điện thế (không có điện thế là 0, có điện thế là 1),...

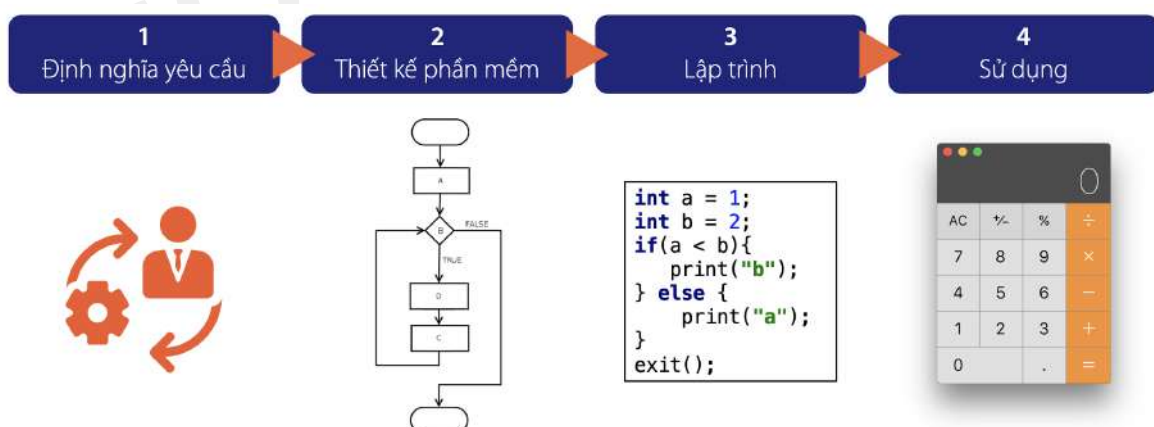
Mỗi một giá trị 0 hoặc 1 khi lưu trữ trong bộ nhớ thì được gọi là 1 *bit*. Một nhóm 8 *bit* thì được gọi là 1 *byte*. Byte là đơn vị lưu trữ nhỏ nhất mà máy tính sử dụng. Sau byte còn có các đơn vị khác lớn hơn để biểu diễn độ lớn của dữ liệu, chẳng hạn như: kilobyte (**kb**), megabyte (**MB**), gigabyte (**GB**), terabyte (**TB**)... Bảng chuyển đổi giữa các đơn vị đo lường bộ nhớ được mô tả dưới đây:

Đơn vị	Ký hiệu	Giá trị xấp xỉ (byte)	Giá trị chính xác (byte)
Kilobyte	KB	1 000	1024
Megabyte	MB	1 000 000	1 048 576
Gigabyte	GB	1 000 000 000	1073 741 824
Terabyte	TB	1 000 000 000 000	1099 511 627 776

Bảng 1.1: Giá trị của các đơn vị đo độ lớn của dữ liệu

4. Phần mềm được tạo ra như thế nào?

Phát triển phần mềm là một công việc bao gồm rất nhiều hoạt động khác nhau, đòi hỏi sự cộng tác, hợp tác giữa nhiều người với nhiều kỹ năng khác nhau. Có thể hình dung một cách đơn giản thì công việc này bao gồm các hoạt động như Định nghĩa yêu cầu, Thiết kế phần mềm, Lập trình, Sử dụng.



Hình 1.3: Các công đoạn chính của việc phát triển phần mềm

- Định nghĩa yêu cầu: Tìm hiểu, phân tích để nắm rõ nhu cầu của người dùng về phần mềm mà mình sắp phát triển.
- Thiết kế phần mềm: Dựa trên các thông tin đã thu thập được, chúng ta phân tích và đưa ra các cách để xử lý vấn đề của người dùng thông qua các tính năng của phần mềm.
- Lập trình: Dựa trên bản thiết kế về các tính năng đã có, người lập trình viên sẽ viết ra các dòng lệnh để biến bản thiết kế đó trở thành một phần mềm có thể chạy thực sự.
- Sử dụng: Sau khi phần mềm đã hoàn tất thì nó được bàn giao cho người dùng để xử lý các tác vụ mà trước đó đã định nghĩa.

Trên đây, chúng ta chỉ mới liệt kê các công đoạn chính để làm ra một phần mềm. Trong thực tế, có thể có thêm rất nhiều các công đoạn khác, hoặc là cách sắp xếp các công việc khác, tùy thuộc vào quy trình phát triển phần mềm của nhóm hoặc tổ chức.

5. Quy trình tạo ra một phần mềm

Để tạo ra một phần mềm, cần có sự cộng tác giữa rất nhiều người, sử dụng và chia sẻ các tài nguyên trong một khoảng thời gian, do đó việc đưa ra các quy trình làm việc là cần thiết.

Thời kỳ đầu, khi mới xuất hiện máy tính và phần mềm, các phần mềm thường nhỏ và đơn giản. Nhưng ngày nay, các hệ thống phần mềm thường rất lớn, đòi hỏi rất nhiều công sức để phát triển. Do đó, rất khó để một lập trình viên có thể hoàn thành được hết các công việc trong một khoảng thời gian cần thiết. Các phần mềm thường được sản xuất bởi các nhóm, hoặc nhiều nhóm cộng tác với nhau. Để cho việc cộng tác giữa các cá nhân và các nhóm được diễn ra thuận lợi thì chúng ta thiết lập các quy trình.

Quy trình là gì? Quy trình được hiểu đơn giản là các quy định về trình tự các bước để làm việc. Ai làm việc gì? Vào lúc nào? Sử dụng công cụ gì? Tiêu chuẩn gì?...

Có nhiều dạng quy trình khác nhau được sử dụng trong các nhóm phần mềm, có thể liệt kê như: Thác nước (Waterfall), Xoắn ốc (Spiral), Scrum... Trong những năm gần đây, triết lý Agile và các phương pháp của nó đã được truyền bá và trở nên rất thông dụng.



Hình 1.4: Minh họa về quy trình thác nước (waterfall)

6. Các vai trò trong lập trình

Có nhiều người tham gia vào trong quá trình làm ra các phần mềm, họ sẽ đóng các vai trò khác nhau và thực hiện các công việc khác nhau. Sau đây là danh sách một số các vai trò thường thấy:

Kỹ sư phần mềm (Software Engineer)

Kỹ sư phần mềm là người thiết kế và lập trình phần mềm ở mức hệ thống. Kỹ sư phần mềm là người hiểu các chức năng của hệ thống, trao đổi với khách hàng để xác định các chức năng của hệ thống đang xây dựng. Kỹ sư phần mềm là người giao tiếp nhiều và đồng thời cũng có nền tảng kỹ thuật và các kỹ năng lập trình tốt. Kỹ sư phần mềm còn được biết đến với các tên gọi tương tự như: Kiến trúc sư phần mềm, Kỹ sư hệ thống.

Chuyên gia phân tích hệ thống (Systems Analyst)

Chuyên gia Phân tích Hệ thống là người nghiên cứu và phân tích các vấn đề nghiệp vụ để sau đó đưa ra các thiết kế hệ thống thông tin nhằm cung cấp giải pháp, việc này thường xuất phát từ yêu cầu từ các bộ phận kinh doanh hoặc từ khách hàng. Chuyên gia Phân tích Hệ thống thu thập các yêu cầu và xác định chi phí cũng như thời gian cần thiết để triển khai dự án. Công việc này đòi hỏi việc kết hợp giữa các kỹ năng nghiệp vụ và kiến thức kỹ thuật, đồng thời phải giao tiếp tốt với các bên. Chuyên

gia phân tích hệ thống còn được biết đến với các tên gọi như: Chuyên gia sản phẩm, Kỹ sư hệ thống, Chuyên gia giải pháp, Nhà thiết kế kỹ thuật.

Chuyên gia phân tích nghiệp vụ (Business Analyst)

Chuyên gia Phân tích Nghiệp vụ là người đóng vai trò trung gian quan trọng, làm việc với cả đội ngũ kỹ thuật, các cấp quản lý và với người dùng cuối. Chuyên gia Phân tích Nghiệp vụ là người đưa ra các cải tiến về quy trình và hoạt động nghiệp vụ thông qua việc sử dụng các công nghệ kỹ thuật. Vai trò này được xác định theo từng dự án, bắt đầu bằng việc phân tích các nhu cầu của khách hàng, thu thập và tài liệu hoá các yêu cầu, lập kế hoạch để xây dựng thiết kế cho giải pháp công nghệ. Chuyên gia Phân tích Nghiệp vụ cần phải có hiểu biết về công nghệ, tuy nhiên không nhất thiết phải là một chuyên gia công nghệ. Chuyên gia phân tích nghiệp vụ còn được biết đến với các tên gọi: Kiến trúc sư nghiệp vụ, Chuyên gia thông tin.

Chuyên viên Hỗ trợ Kỹ thuật (Technical Support)

Chuyên viên Hỗ trợ Kỹ thuật là người giải quyết các vấn đề trong quá trình hoạt động của các hệ thống. Nhiều chuyên gia hỗ trợ kỹ thuật làm việc trong các công ty sản xuất và cung cấp phần cứng, và cũng có nhiều chuyên gia hỗ trợ kỹ thuật ở các doanh nghiệp nhằm hỗ trợ, theo dõi và bảo trì các hệ thống được sử dụng hằng ngày. Nhiều công việc đòi hỏi các chuyên gia với nền tảng và kinh nghiệm kỹ thuật tốt.

Kỹ sư mạng (Network Engineer)

Kỹ sư mạng là một trong các công việc rất cần thiết trong ngành IT, là người thực hiện các thao tác cài đặt, quản trị, duy trì và nâng cấp các hệ thống giao tiếp, xử lý các vấn đề liên quan đến mạng lưới trong các công ty. Kỹ sư mạng cũng là người chịu trách nhiệm về bảo mật, lưu trữ dữ liệu và các chiến lược khôi phục nếu có sự cố xảy ra. Kỹ sư mạng còn biết đến với tên gọi: Kỹ sư phần cứng, Chuyên gia mạng.

Quản lý Dự án (Project Manager)

Quản lý Dự án là người tổ chức các nhóm phát triển, phân bổ thời gian và tài nguyên để đảm bảo các dự án đạt được các yêu cầu về chức năng, đúng thời gian và nằm trong ngân sách cho phép. Quản lý Dự án điều phối tất cả các hoạt động từ khi mới bắt đầu dự án cho đến khi kết thúc. Vai trò này đòi hỏi kinh nghiệm và nền tảng vững chắc về kỹ thuật cũng như các kỹ năng mềm để làm việc tốt với các nhóm phát triển và các nhà quản lý cấp cao. Quản lý Dự án còn biết đến với tên gọi: Trưởng dự án.

Nhà phát triển (Developer)

Nhà phát triển là người trực tiếp tạo ra phần mềm thông qua việc sử dụng các ngôn ngữ lập trình và các công cụ hỗ trợ. Nhà phát triển thường làm việc cộng tác trong các nhóm để đảm bảo các tính năng của sản phẩm được xây dựng và đáp ứng được các yêu cầu như thiết kế ban đầu. Vai trò này đòi hỏi kiến thức và kỹ năng tốt về công nghệ, công cụ và các ngôn ngữ lập trình. Nhà phát triển còn được biết đến với các tên gọi: Lập trình viên (programmer), Nhân viên viết mã (coder).

Kiểm thử viên Phần mềm (Software Tester)

Kiểm thử viên phần mềm là người tham gia vào công tác đảm bảo chất lượng phần mềm thông qua việc phát hiện các lỗi tiềm tàng và hỗ trợ nhóm phát triển trong việc xử lý các lỗi. Kiểm thử viên phần mềm thực hiện các thao tác phân tích nghiệp vụ, lập kế hoạch kiểm thử, viết các kịch bản kiểm thử, thực thi các ca kiểm thử và viết các báo cáo kiểm thử. Kiểm thử viên Phần mềm còn được biết đến với tên gọi: Kiểm thử viên, Nhân viên kiểm thử.

Chủ Sản phẩm (Product Owner)

Chủ sản phẩm là người chịu trách nhiệm xác định và đảm bảo các chức năng của hệ thống. Chủ sản phẩm tìm hiểu và nghiên cứu các yêu cầu của người dùng cuối, đưa ra các giải pháp để đáp ứng được các yêu cầu, quản lý tiến độ và chất lượng của các chức năng trong suốt quá trình phát triển. Chủ sản phẩm là người có hiểu biết về thị trường, về các hệ thống phần mềm và sử dụng các công cụ quản lý sản phẩm và quản lý dự án. Chủ Sản phẩm còn được biết đến với các tên gọi: Quản lý sản phẩm.

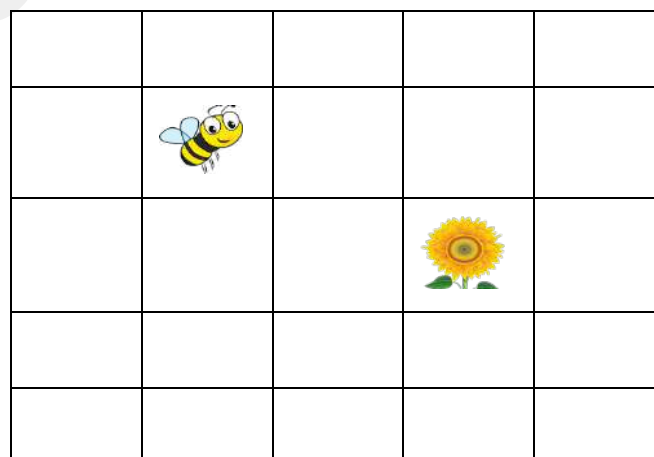
ScrumMaster (ScrumMaster)

ScrumMaster là người chịu trách nhiệm trong việc đảm bảo các nhóm Scrum hoạt động tốt và chuyển giao được sản phẩm chất lượng cao. ScrumMaster nắm rõ khung làm việc Scrum, các kỹ thuật phát triển và là người làm việc thường xuyên với nhóm phát triển. ScrumMaster có thể là người am hiểu về công nghệ hoặc không.

7. Ngôn ngữ lập trình

Ngôn ngữ lập trình là công cụ để lập trình viên viết ra các chỉ dẫn cho máy tính thực thi. Có thể hiểu ngôn ngữ lập trình như là một cách để giao tiếp giữa lập trình viên và máy tính. Ngôn ngữ lập trình được sử dụng để tạo ra các phần mềm.

Các ngôn ngữ lập trình quy định các chỉ dẫn, dựa vào đó lập trình viên sẽ sắp xếp chúng để tạo ra các chỉ dẫn có ý nghĩa. Một ví dụ để minh họa cho ý nghĩa của ngôn ngữ lập trình: Nếu chúng ta có các chỉ dẫn **ĐI THẲNG**, **RẼ TRÁI**, **RẼ PHẢI** thì chúng ta sẽ sắp xếp các chỉ dẫn đó để đưa được chú ong ở bản đồ sau đi đến được vị trí của bông hoa.



Hình 1.5: Chỉ dẫn để chú ong đi đến bông hoa

Chẳng hạn, các chỉ dẫn có thể là:

- RẼ PHẢI – ĐI THẲNG – ĐI THẲNG
- ĐI THẲNG – ĐI THẲNG – RẼ PHẢI
- ĐI THẲNG – RẼ PHẢI – ĐI THẲNG

Ngày nay, có hàng trăm, thậm chí hàng nghìn ngôn ngữ lập trình khác nhau. Có những ngôn ngữ lập trình thông dụng như: Java, JavaScript, PHP, C#, Python, Ruby,... và cũng có nhiều ngôn ngữ lập trình rất chuyên biệt, chỉ sử dụng để xử lý những tình huống đặc thù nào đó. Mỗi ngôn ngữ lập trình đều có những đặc điểm riêng, phù hợp với những tình huống khác nhau, do đó việc lựa chọn ngôn ngữ lập trình phù hợp với một tình huống cụ thể cũng là một nhiệm vụ của lập trình viên.

Tuy có nhiều ngôn ngữ lập trình với các đặc trưng riêng, nhưng có một số chỉ dẫn cơ bản luôn hiện diện trong bất kỳ ngôn ngữ lập trình nào:

- Nhập dữ liệu: những chỉ dẫn giúp chương trình thu thập được dữ liệu từ bàn phím, thiết bị ngoại vi, tập tin, mạng, v.v.
- Xuất dữ liệu: những chỉ dẫn giúp chương trình có thể hiển thị dữ liệu lên màn hình, lưu dữ liệu vào tập tin, thay đổi trạng thái của thiết bị ngoại vi, gửi đi tín hiệu mạng, v.v.
- Tính toán: thực hiện những phép toán cơ bản như cộng hay nhân
- Thực thi có điều kiện: kiểm tra một số điều kiện và theo đó lựa chọn khối chỉ dẫn phù hợp để thực thi.
- Lặp lại: thực hiện một số chỉ dẫn một cách lặp đi lặp lại

Trong cuốn sách này, chúng ta sẽ sử dụng ngôn ngữ lập trình JavaScript để viết các phần mềm. Nên nhớ rằng, khi học lập trình, ngôn ngữ lập trình chỉ là một công cụ chứ không phải là toàn bộ những gì chúng ta cần học. Điều quan trọng nhất là chúng ta cần học tư duy lập trình, tư duy giải quyết vấn đề bằng các chỉ dẫn. Khi nắm được tư duy lập trình rồi thì chúng ta hoàn toàn có thể sử dụng một số ngôn ngữ khác nhau để tạo ra phần mềm. Điều này có nghĩa là, sau một thời gian lập trình thì việc học thêm các ngôn ngữ khác là một việc khá dễ dàng, hầu hết các lập trình viên ngày nay đều thuần thục một vài ngôn ngữ lập trình chứ không chỉ sử dụng một ngôn ngữ duy nhất.

8. Thuật toán

Khi chúng ta học lập trình thì không chỉ cần học một ngôn ngữ lập trình nào đó mà cần phải học tư duy giải quyết vấn đề. Khi đã có tư duy giải quyết vấn đề thì chúng ta có thể sử dụng các ngôn ngữ lập trình khác nhau để xây dựng các ứng dụng. Hay nói

cách khác, ngôn ngữ lập trình chính là công cụ để hiện thực hoá tư duy giải quyết vấn đề cho một bài toán cụ thể.

Thuật toán, còn gọi là giải thuật, là một tập hợp hữu hạn các chỉ thị hay cách thức được định nghĩa rõ ràng cho việc hoàn tất một số sự việc từ một trạng thái ban đầu cho trước. Chúng ta có thể sử dụng các cách khác nhau để mô tả thuật toán, chẳng hạn như bằng lời nói, bằng các hình vẽ hoặc bằng các ký hiệu khác.

Ví dụ:

Có hai bình A và B đựng hai loại chất lỏng khác nhau, chẳng hạn bình A đựng cồn, bình B đựng dầu. Yêu cầu trao đổi (swap) chất lỏng đựng trong hai bình.

Thuật toán:

- Yêu cầu phải có thêm một bình thứ ba gọi là bình C
- Bước 1: Đổ cồn từ bình A vào bình C
- Bước 2: Đổ dầu từ bình B vào bình A
- Bước 3: Đổ cồn từ bình C vào bình B

Trong lập trình, có 2 cách phổ biến để mô tả các thuật toán đó là Mã giả và Lưu đồ.

9. Mô tả thuật toán bằng mã giả

Mã giả (pseudo-code) là cách mô tả thuật toán bằng ngôn ngữ tự nhiên. Thông thường chúng ta sử dụng các từ tiếng Anh để mô tả thuật toán. Mã giả là mã không thực thi được và thông thường cũng không có các quy định chặt chẽ về cú pháp của mã giả. Chúng ta có thể tùy biến cách sử dụng từ, miễn sao đạt được mục đích là các bên liên quan có thể hiểu được thuật toán của chúng ta.

Ví dụ: Sử dụng mã giả để mô tả thuật toán giải phương trình bậc nhất với các cơ số a và b:

```

1. BEGIN
2. INPUT a, b
3. IF a = 0 THEN
4.   IF b = 0 THEN
5.     PRINT "Phương trình vô số nghiệm"
6.   ELSE
7.     PRINT "Phương trình vô nghiệm"
8.   END IF
9. ELSE
10.  PRINT "Phương trình có nghiệm x = -b / a"
11. END IF
12. END
  
```

Giải thích:

- Dòng 1: BEGIN đánh dấu nơi bắt đầu thuật toán
- Dòng 2: INPUT dấu việc nhập dữ liệu cần thiết cho thuật toán
- Dòng 3: IF a = 0 đánh dấu việc thực thi theo điều kiện về tham số a = 0.
- Dòng 4: IF b = 0 đánh dấu việc thực thi theo điều kiện về tham số b = 0.
- Dòng 5: PRINT là chỉ thị để in ra nghiệm tương ứng với điều kiện a = 0 và b = 0.

- Dòng 6: ELSE đánh dấu việc thực thi phương án thay thế khi điều kiện về tham số $b = 0$ không thỏa mãn.
- Dòng 7: PRINT là chỉ thị để in ra nghiệm tương ứng với điều kiện $a = 0$ và $b \neq 0$.
- Dòng 8: END IF đánh dấu kết thúc quá trình thực thi theo điều kiện về tham số $b = 0$.
- Dòng 9: ELSE chỉ thị thực thi phương án thay thế khi điều kiện về tham số $a = 0$ không thỏa mãn.
- Dòng 10: PRINT là chỉ thị để in ra nghiệm tương ứng với điều kiện $a \neq 0$ và $b \neq 0$.
- Dòng 11: END IF đánh dấu kết thúc quá trình thực thi theo điều kiện về tham số $a = 0$.
- Dòng 12: END đánh dấu nơi kết thúc thuật toán.

Ưu điểm của việc sử dụng mã giả đó là gần với tự nhiên, ai cũng có thể sử dụng được. Nhược điểm của mã giả đó là nó không có các quy định chặt chẽ nên có thể dẫn đến tình huống là các bên không hiểu được nhau.

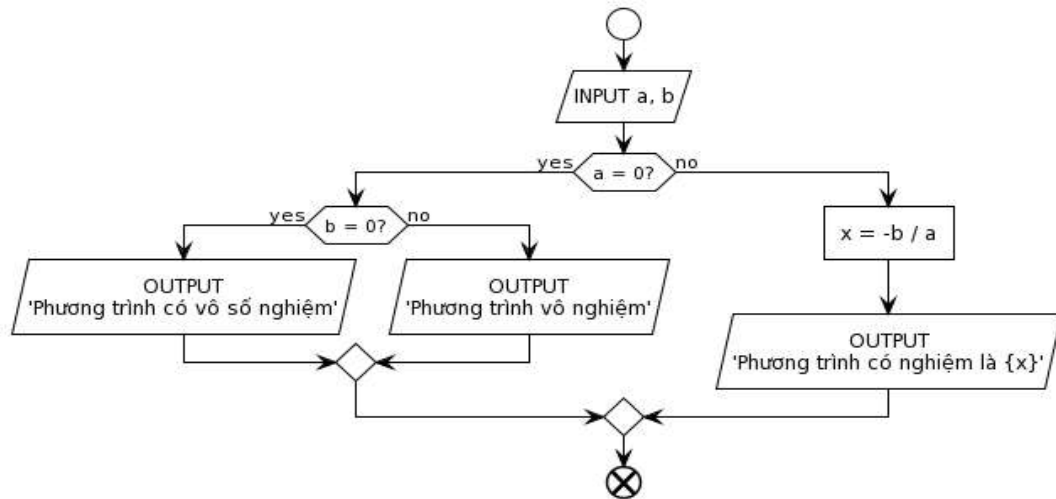
10. Mô tả thuật toán bằng lưu đồ

Lưu đồ là cách sử dụng các ký hiệu được quy định trước để mô tả thuật toán. Ưu điểm của lưu đồ là có các quy định chặt chẽ về từng ký hiệu, việc này giúp cho các bên thống nhất về cách sử dụng và dễ hiểu nhau hơn.

Biểu tượng	Mô tả
	Thể hiện trình tự thực hiện các thao tác
	Điểm bắt đầu hoặc kết thúc của một tiến trình
	Thực hiện các phép tính toán
	Xét điều kiện và lựa chọn hướng đi tiếp dựa vào điều kiện đó
	Nhập dữ liệu vào hoặc xuất thông tin ra

Bảng 1.2: Các biểu tượng thông dụng trong lưu đồ

Ví dụ: Lưu đồ sau đây mô tả thuật toán để tìm nghiệm của phương trình bậc 2:



Giải thích:

- Nút tròn rỗng mô tả nơi bắt đầu thuật toán
- Hình bình hành mô tả nơi nhập vào giá trị đầu vào
- Hình chữ nhật mô tả nơi thực hiện phép tính
- Hình bình hành mô tả nơi hiển thị kết quả
- Nút tròn với dấu X mô tả nơi kết thúc thuật toán.

11. Một số cấu trúc thường gặp trong thuật toán

Thông thường, trình tự các bước thực hiện của thuật toán là tuyến tính từ trên xuống dưới. Nhưng trong nhiều tình huống, chúng ta cần thay đổi luồng thực thi đó thay đổi, chẳng hạn như ra các quyết định dựa trên một điều kiện nào đó, hoặc lặp đi lặp lại các hành động giống nhau. Trong những tình huống như vậy, chúng ta sẽ sử dụng các cấu trúc đặc trưng như cấu trúc điều kiện hoặc cấu trúc lặp.

11.1. Cấu trúc điều kiện

Cấu trúc điều kiện, còn được biết đến với tên gọi *cấu trúc lựa chọn*, là dạng cấu trúc được sử dụng trong các tình huống chúng ta cần ra các quyết định dựa trên một điều kiện cho trước.

Có một số dạng cấu trúc điều kiện cơ bản như sau:

- Cấu trúc 1: **Nếu** <điều kiện> (đúng) **thì** thực hiện <công việc>
- Cấu trúc 2: **Nếu** <điều kiện> (đúng) **thì** thực hiện <công việc 1>, **ngược lại** (điều kiện sai) **thì** thực hiện <công việc 2>
- Cấu trúc 3: Trường hợp <i> thì thực hiện <công việc i>

Ví dụ 1:

Bài toán kiểm tra xem một số có phải là số chẵn hay không, nếu là số chẵn thì hiển thị thông báo. Để xác định việc này, chúng ta căn cứ vào một đặc điểm là nếu một số là chẵn thì nó sẽ chia hết cho 2 (tức là dư 0).

Trong bài toán này chúng ta sử dụng dạng cấu trúc điều kiện **Nếu** <điều kiện> (đúng) **thì** thực hiện <công việc>. Trong đó, các bước thực hiện là:

1. Nhập vào một số *num*
2. Tính *r* là phần dư của phép chia *num* cho 2
3. Kiểm tra xem *r* có bằng 0 hay không
4. Nếu *r* bằng 0 thì hiển thị thông báo “Number is even”

Mã giả:

```

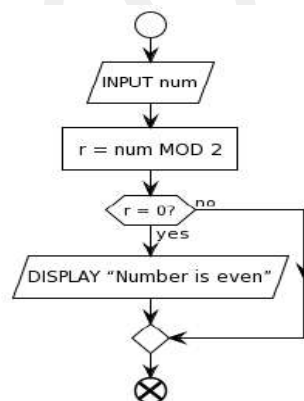
1. BEGIN
2. INPUT num
3. r = num MOD 2
4. IF r=0
5.     DISPLAY "Đây là số chẵn"
6. END IF
7. END

```

Giải thích:

- Dòng 1: BEGIN đánh dấu bắt đầu chương trình
- Dòng 2: Nhập giá trị *num*
- Dòng 3: Tính giá trị *r* theo công thức *r* bằng số dư của *num* chia cho 2
- Dòng 4: Kiểm tra xem thử *r* có bằng 0 hay không. Nếu *r* bằng 0 thì thực hiện dòng 5. Nếu *r* khác 0 thì bỏ qua dòng 5 và thực hiện dòng 6.
- Dòng 5: Hiển thị dòng chữ “Đây là số chẵn”
- Dòng 6: END IF đánh dấu kết thúc khối lệnh IF
- Dòng 7: END đánh dấu kết thúc chương trình

Lưu đồ:



Giải thích:

Ở trong lưu đồ trên, hình quả trám được sử dụng để kiểm tra trường hợp *r* bằng 0, có hai trường hợp xảy ra được mô tả bằng hai hướng Yes và No.

Ví dụ 2:

Bài toán kiểm tra xem một số là số chẵn hay là số lẻ, hiển thị thông báo tương ứng cho cả hai trường hợp.

Trong bài toán này chúng ta sử dụng dạng cấu trúc điều kiện **Nếu** <điều kiện> (đúng) **thì** thực hiện <công việc 1>, **ngược lại** (điều kiện sai) **thì** thực hiện <công việc 2>.

Trong đó, các bước thực hiện là:

- Nhập vào một số *num*

- Tính r là phần dư của phép chia num cho 2
- Kiểm tra xem r có bằng 0 hay không
- Nếu r bằng 0 thì hiển thị thông báo “Number is Even”
- Nếu ngược lại (tức là r khác 0) thì hiển thị thông báo “Number is Odd”.

Mã giả:

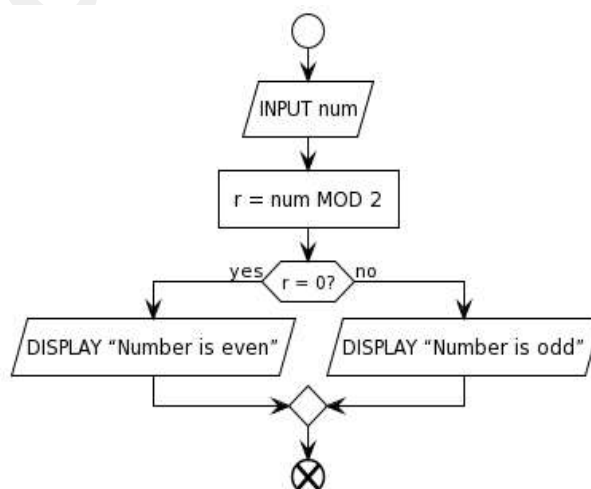
```

1. BEGIN
2. INPUT num
3.  $r = num \text{ MOD } 2$ 
4. IF  $r = 0$ 
5.   DISPLAY "Đây là số chẵn"
6. ELSE
7.   DISPLAY "Đây là số lẻ"
8. END IF
9. END
  
```

Giải thích:

- Dòng 1: BEGIN đánh dấu bắt đầu chương trình
- Dòng 2: Nhập giá trị num
- Dòng 3: Tính giá trị r theo công thức r bằng số dư của num chia cho 2
- Dòng 4: Kiểm tra xem thử r có bằng 0 hay không. Nếu r bằng 0 thì thực hiện dòng 5. Nếu r khác 0 thì thực hiện dòng 7.
- Dòng 5: Hiển thị dòng chữ “Đây là số chẵn”
- Dòng 6: ELSE đánh dấu đoạn mã sẽ thực thi trong trường hợp mà r khác 0
- Dòng 7: Hiển thị dòng chữ “Đây là số lẻ”
- Dòng 8: END IF đánh dấu kết thúc khối lệnh IF
- Dòng 9: END đánh dấu kết thúc chương trình

Lưu đồ:



11.2. Cấu trúc lặp

Cấu trúc lặp cho phép thực hiện lặp đi lặp lại các công việc nào đó dựa vào một điều kiện cho trước. Chúng ta thường sử dụng cấu trúc lặp để tự động hoá những công việc có tính chất giống nhau, giúp cho mã nguồn trở nên ngắn gọn hơn.

Có hai dạng cấu trúc lặp cơ bản như sau:

- Lặp xác định trước số lần lặp: Là dạng lặp mà khi viết chương trình, người lập trình đã xác định được công việc sẽ lặp bao nhiêu lần. Chẳng hạn: hiển thị danh sách 100 khách hàng, tính tổng giá tiền của 10 sản phẩm, in bảng cửu chương (bảng tính nhân từ 1 đến 9) v.v.
- Lặp không xác định số lần lặp: là loại lặp mà khi viết chương trình người lập trình chưa xác định được công việc sẽ lặp bao nhiêu lần. Số lần lặp sẽ được xác định tùy thuộc vào một số yếu tố cụ thể khi chương trình thực thi. Chẳng hạn: sao chép một file từ nơi này sang nơi khác (chúng ta không biết trước dung lượng của file), cho một nhân vật trong trò chơi chuyển động (chúng ta không biết trước khi nào thì nhân vật dừng lại), kim đồng hồ chuyển động v.v.

Ví dụ 1:

Hiển thị 1000 lần dòng chữ "Scooby".

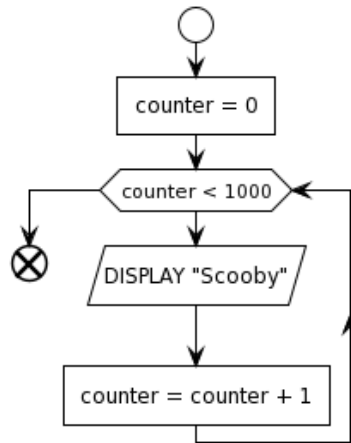
Trong bài toán này, chúng ta đã biết trước số lần lặp là 1000, do đó chúng ta sử dụng dạng lặp thứ nhất. Các bước thực hiện là:

1. Khai báo biến *counter* với giá trị ban đầu là 0
2. Kiểm tra điều kiện xem liệu *counter* có nhỏ hơn 1000 hay không
3. Nếu *counter* nhỏ hơn 1000 thì:
4. Hiển thị dòng chữ "Scooby"
5. Tăng giá trị của biến *counter* thêm 1 giá trị
6. Quay lại Bước 2
7. Nếu *counter* không nhỏ hơn 1000 (tức là bằng hoặc lớn hơn) thì kết thúc chương trình

Mã giả:

```
1. BEGIN
2. counter = 0
3. WHILE (counter < 1000)
4. DO
5.   DISPLAY "Scooby"
6.   counter = counter + 1
7. END DO
8. END
```

Lưu đồ:



Ví dụ 2:

Cho phép người dùng lần lượt nhập vào các số tự nhiên, tính tổng tất cả các số mà người dùng đã nhập. Không hạn chế số lượng lần nhập. Khi người dùng nhập vào số 0 thì hiển thị kết quả và kết thúc chương trình.

Trong bài toán này, chúng ta không biết trước số lần lặp. Các bước thực hiện là:

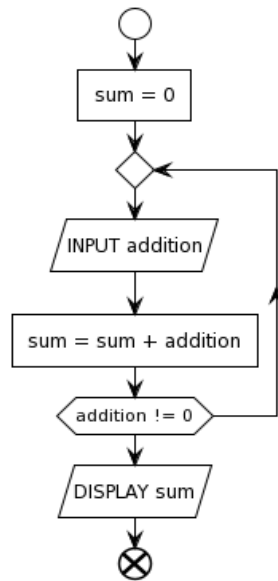
1. Khai báo một biến *sum* với giá trị ban đầu là 0
2. Cho phép người dùng nhập vào một giá trị cho biến *addition*
3. Cộng dồn giá trị của biến *addition* vào biến *sum*
4. Kiểm tra xem biến *addition* có bằng 0 hay không
5. Nếu biến *addition* khác 0 thì lặp lại Bước 2
6. Nếu biến *addition* bằng 0 thì hiển thị giá trị của biến *sum* và kết thúc

Mã giả:

```

1. BEGIN
2. sum = 0
3. DO
4.  INPUT addition
5.  sum = sum + addition
6. WHILE (addition != 0)
7.  DISPLAY sum
8. END
  
```

Lưu đồ:



12. Ngôn ngữ lập trình JavaScript

JavaScript có khởi nguồn là một ngôn ngữ lập trình để chạy trên các trình duyệt, chủ yếu là để gia tăng tính tương tác và trải nghiệm đối với người dùng web. Về sau, JavaScript và hệ sinh thái của nó đã được cải tiến và phát triển để có thể chạy được trên các môi trường khác, chúng ta có thể sử dụng JavaScript để phát triển các ứng dụng phía back-end, mobile hay thậm chí là các ứng dụng desktop.

Trong phạm vi của cuốn sách này, chúng ta sẽ sử dụng JavaScript trên môi trường của trình duyệt. Để thực thi mã JavaScript, chúng ta cần tạo ra một trang web và nhúng mã JavaScript vào trang web đó.

Lưu ý: Nếu trình duyệt bị vô hiệu hoá tính năng thực thi JavaScript thì chương trình JavaScript trên trang web sẽ không được thực thi.

13. Chương trình đầu tiên

Ứng dụng JavaScript đầu tiên của chúng ta sẽ có một chức năng là hiển thị một thông báo nhỏ lên trên trang web với nội dung "Hello. JavaScript is running."

Đoạn mã của file HTML:

```

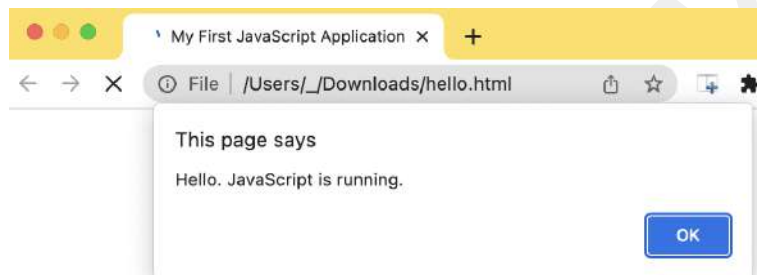
1. <html>
2. <head>
3.   <title>My First JavaScript Application</title>
4. </head>
5. <body>
6.   <script>
7.     alert("Hello. JavaScript is running.");
8.   </script>
9. </body>
10. </html>
  
```

Bạn có thể sao chép đoạn mã trên vào một text editor bất kỳ (chẳng hạn notepad) và lưu lại thành một file có phần mở rộng là *.html*, chẳng hạn *hello.html*.



```
<html>
  <head>
    <title>My First JavaScript Application</title>
  </head>
  <body>
    <script>
      alert("Hello. JavaScript is running.");
    </script>
  </body>
</html>
```

Mở file này bằng một trình duyệt web sẽ cho thấy một dialog được hiển thị như dưới:



Điều gì đã diễn ra?

- Trong file HTML, chúng ta sử dụng thẻ `<script>` để nhúng các đoạn mã JavaScript vào, từ dòng 6 đến dòng 8
- Khi mở trang web bằng trình duyệt, trình duyệt sẽ phiên dịch đoạn mã JavaScript ở giữa dòng 6 và dòng 8 và thực thi
- Hàm `alert()` ở dòng 7 là một hàm được sử dụng rất phổ biến trong JavaScript để hiển thị các thông báo.

Lưu ý: Cấu trúc của file HTML luôn luôn bao gồm các thẻ `<html>`, `<head>`, `<title>`, `<body>` như ví dụ trên. Chúng ta sẽ chưa đi tìm hiểu sâu về HTML ở giai đoạn này.

14. Cài đặt công cụ lập trình

Để bắt đầu tạo ra các phần mềm, chúng ta cần cài đặt các công cụ cần thiết. Trong đó, một trình soạn thảo mã nguồn (code editor) là điều cần thiết.

Có rất nhiều các trình soạn thảo mã nguồn trên thị trường hiện nay, hầu hết chúng đều hỗ trợ các tính năng cần thiết để giúp cho việc viết mã trở nên dễ dàng hơn. Các tính năng chính của chúng thường bao gồm:

- Hiển thị mã nguồn ở các màu sắc khác nhau, giúp dễ phân biệt
- Gợi ý khi viết, giúp viết mã nhanh hơn
- Hỗ trợ biên dịch hoặc phiên dịch mã nguồn
- Hỗ trợ thực thi mã nguồn để quan sát kết quả

- Hỗ trợ debug (dò lỗi) để quan sát tiến trình thực thi của mã
- Và còn nhiều tính năng khác: Tích hợp trình quản lý phiên bản (SVN, Git...), tự động định dạng mã nguồn, tái cấu trúc mã nguồn...

IDE – Integrated Development Environment

IDE là thuật ngữ để nói đến các trình soạn thảo mã nguồn trong đó có tích hợp nhiều chức năng và môi trường để hỗ trợ cho việc lập trình. IDE là viết tắt của Integrated Development Environment (nghĩa là Môi trường Phát triển Tích hợp). Một IDE có thể chỉ hỗ trợ một công nghệ hoặc hỗ trợ nhiều công nghệ khác nhau. Một số IDE nổi tiếng như: WebStorm, PhpStorm, IntelliJ, Netbeans, Eclipse, Visual Studio, Visual Studio Code, Atom, Sublime Text, v.v.

Đối với các tính năng cơ bản thì hầu hết các IDE đều đáp ứng được, chúng chỉ khác nhau ở một số tiện ích riêng. Trong khuôn khổ của cuốn sách này, chúng ta có thể dùng WebStorm hoặc Visual Studio Code để viết mã.

15. Bài thực hành

Các bài thực hành sau đây sẽ hướng dẫn cách giải các bài toán đơn giản và ghi giải pháp ra dưới dạng mã giả hoặc lưu đồ. Chúng ta hãy lấy giấy và bút để thực hành các bài này.

Bài 1: Thuật toán chuyển đổi nhiệt độ

Mô tả thuật toán nhập một giá trị là độ $^{\circ}\text{C}$ (Celsius) và chuyển nó sang độ $^{\circ}\text{F}$ (Fahrenheit). Công thức chuyển đổi là:

$$F = \frac{9 \times C}{5} + 32$$

Hướng dẫn:

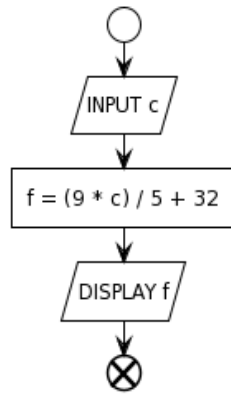
Trong bài toán này, chúng ta sẽ lần lượt thực hiện các bước sau:

- Nhập vào giá trị độ C
- Tính giá trị độ F dựa theo công thức đã cho
- Hiển thị giá trị độ F

Mã giả:

```
1. BEGIN
2. INPUT c
3. f = (9 * c) / 5 + 32
4. DISPLAY f
5. END
```

Lưu đồ:



Bài 2: Thuật toán tính điểm trung bình

Mô tả thuật toán tính điểm trung bình của học sinh. Nhập vào điểm số 3 môn Toán, Lý và Hoá, sau đó hiển thị điểm số trung bình.

Hướng dẫn:

Trong bài này, chúng ta sẽ thực hiện các bước sau:

- Nhập vào điểm số của 3 môn Toán, Lý và Hoá
- Tính điểm trung bình
- Hiển thị điểm trung bình

Mã giả:

```

1. BEGIN
2. INPUT math, physics, chemical
3. medium = (math + physics + chemical) / 3
4. DISPLAY medium
5. END
  
```

Bạn hãy thực hành vẽ lưu đồ cho thuật toán này.

16. Bài tập

Bài 1: Thuật toán chuyển đổi tiền tệ

Hãy viết mã giả và vẽ lưu đồ để mô tả thuật toán chuyển đổi từ Đô la Mỹ sang Việt Nam Đồng. Nhập vào giá trị Đô la Mỹ, sau đó hiển thị giá trị Việt Nam Đồng tương ứng. Tỷ giá chuyển đổi là 1/23000.

Công thức chuyển đổi:

$$1 \text{ Đô la Mỹ} = 23000 \text{ Việt Nam Đồng}$$

Bài 2: Thuật toán tìm giá trị lớn nhất trong 3 số

Hãy viết mã giả và vẽ lưu đồ để mô tả thuật toán tìm giá trị lớn nhất trong 3 số. Nhập vào 3 số, sau đó thực hiện các phép so sánh lần lượt từng cặp số để tìm ra giá trị lớn nhất trong 3 số đó.

Gợi ý: Chúng ta sẽ cần sử dụng cấu trúc điều kiện.

Bài 3: Thuật toán tìm giá trị lớn nhất trong dãy số

Hãy viết mã giả và vẽ lưu đồ để mô tả thuật toán tìm giá trị lớn nhất trong một loạt các số được nhập vào. Nhập vào số n là số lượng các số, sau đó nhập lần lượt n số và tìm ra giá trị lớn nhất trong đó.

Gợi ý: Chúng ta sẽ cần sử dụng kết hợp cấu trúc lặp và cấu trúc điều kiện.

Bài 4: Thuật toán xếp hạng sinh viên

Hãy viết mã giả và vẽ lưu đồ để mô tả thuật toán xếp hạng sinh viên. Nhập vào điểm thi của sinh viên, hiển thị phân loại sinh viên theo các khoảng:

Điểm thi	Xếp hạng
Điểm ≥ 75	Loại A
$60 \leq \text{Điểm} < 75$	Loại B
$45 \leq \text{Điểm} < 60$	Loại C
$35 \leq \text{Điểm} < 45$	Loại D
Điểm < 35	Loại E

Gợi ý: Chúng ta sẽ cần sử dụng nhiều cấu trúc điều kiện nối tiếp nhau.

17. Bài kiểm tra

Câu 1: Điền từ còn thiếu vào định nghĩa sau:

Lập trình là quá trình tạo ra tập các ____ để ____ cho máy tính hoàn thành một ____ nào đó.

- a) chỉ dẫn (instruction), ra lệnh, công việc (task)
- b) hướng dẫn, yêu cầu, chương trình
- c) chỉ dẫn (instruction), hướng dẫn, công việc (task)
- d) chỉ dẫn (instruction), ra lệnh, sự kiện

Câu 2: Đây là các hoạt động trong lập trình?

- a) Viết code
- b) Phân tích
- c) Tìm hiểu yêu cầu
- d) Thiết kế

Câu 3: Phát biểu nào sau đây là SAI về ưu nhược điểm của việc sử dụng mã giả hoặc lưu đồ để mô tả thuật toán?

- a) Sử dụng lưu đồ giúp cho các bên thống nhất về cách sử dụng các ký hiệu và dễ hiểu nhau hơn

- b) Nhược điểm của lưu đồ là có các quy định quá chặt chẽ về từng ký hiệu không phải ai cũng nhớ để sử dụng được
- c) Nhược điểm của mã giả là không có các quy định chặt chẽ dẫn đến tình huống các bên không hiểu được nhau
- d) Mã giả gần với tự nhiên, ai cũng có thể sử dụng được.

Câu 4: Kết luận nào sau đây ĐÚNG khi nói về việc học lập trình

- a) Ngôn ngữ lập trình chính là công cụ để hiện thực hoá tư duy giải quyết vấn đề cho một bài toán cụ thể
- b) Học lập trình thì không chỉ cần học một ngôn ngữ lập trình nào đó mà cần phải học tư duy giải quyết vấn đề
- c) Học lập trình là học một loại ngôn ngữ lập trình để ra lệnh cho máy tính thực hiện 1 yêu cầu nào đó
- d) Học một ngôn ngữ lập trình chúng ta chỉ cần quan tâm đến cú pháp của ngôn ngữ đó

Câu 5: Điền các từ còn thiếu vào phát biểu sau:

Mã giả là cách để mô tả thuật toán bằng ____ Thông thường chúng ta sử dụng ____ để mô tả thuật toán. Mã giả là mã ____ và thông thường cũng không có các quy định chặt chẽ về cú pháp của mã giả.

- a) ngôn ngữ tự nhiên, các từ tiếng Anh, không thực thi được
- b) ngôn ngữ tự nhiên, code, không thực thi được
- c) ngôn ngữ tự nhiên, code, có thể thực thi được
- d) ngôn ngữ máy, các từ của nước bản địa, không thể thực thi được

Câu 6: Lưu đồ giúp chúng ta xem xét lại và gỡ rối chương trình một cách dễ dàng?

- a) Đúng
- b) Sai

Câu 7: Phát biểu nào sau đây đúng về khái niệm "Thuật toán"?

- a) Thuật toán là tập hợp các bước để đưa ra đáp án cho một bài toán
- b) Thuật toán bao gồm các chỉ thị để giải quyết một vấn đề
- c) Thuật toán bao gồm một số bước để giải quyết một vấn đề
- d) Thuật toán là cách giải một bài toán

Câu 8: Phát biểu nào sai về việc sử dụng các ký hiệu trong flow chart?

- a) Ký hiệu hình chữ nhật đánh dấu các thao tác nhập xuất dữ liệu
- b) Ký hiệu hình eclipse đánh dấu điểm bắt đầu hoặc kết thúc của thuật toán
- c) Ký hiệu hình thoi đánh dấu các bước rẽ nhánh
- d) Ký hiệu hình bình hành đánh dấu các bước tính toán

Câu 9: Điền các từ còn thiếu vào phát biểu sau:

Một ____ cơ bản là việc thực thi tuần tự những câu lệnh đến khi một điều kiện cụ thể nào đó là đúng hay sai

- a) Rẽ nhánh

- b) Khai báo biến
- c) Vòng lặp
- d) Thực thi hàm

Câu 10: Điền các từ còn thiếu vào phát biểu sau:

___lựa chọn một công việc để thực hiện căn cứ vào một điều kiện nào đó.

- a) Biến
- b) Vòng lặp
- c) Hàm
- d) Cấu trúc điều kiện

Đáp án: Câu 1: a; Câu 2: a,b,c,d; Câu 3: c; Câu 4: b; Câu 5: a; Câu 6: a; Câu 7: b; Câu 8: a, d; Câu 9: c; Câu 10: d

18. Tổng kết

- Một hệ thống máy tính bao gồm phần cứng, phần mềm và người dùng
- Phần cứng là những thành phần có thể sờ nắm được tạo nên máy tính
- Phần mềm được ví như linh hồn của máy tính, không thể sờ nắm được
- Các công việc cơ bản để tạo ra một phần mềm bao gồm: Thu thập yêu cầu, phân tích, thiết kế, viết mã, kiểm thử, phát hành, bảo trì
- Ngôn ngữ lập trình là công cụ để giao tiếp với máy tính bằng cách viết ra các chỉ dẫn
- Có nhiều vai trò khác nhau tham gia vào trong quá trình sản xuất phần mềm, chẳng hạn như: lập trình viên, kiểm thử viên, quản lý dự án, chuyên gia phân tích nghiệp vụ, ScrumMaster, Product Owner...
- Giải thuật (hay còn gọi là thuật toán) là các bước để xử lý một vấn đề
- Có nhiều cách để mô tả thuật toán, 2 cách thông dụng đó là dùng mã giả và lưu đồ
- Mã giả là cách sử dụng các từ ngữ tự nhiên quen thuộc với con người
- Máy tính không hiểu mã giả, hay nói cách khác, mã giả không thể thực thi được
- Lưu đồ là cách sử dụng các ký hiệu được quy định trước để mô tả thuật toán
- Luồng thực thi của mã nguồn là tuyến tính từ trên xuống dưới
- Có thể thay đổi luồng thực thi của chương trình thông qua các cấu trúc như cấu trúc điều kiện và vòng lặp
- Cấu trúc điều kiện cho phép lựa chọn thực thi một đoạn mã dựa vào một điều kiện
- Cấu trúc lặp cho phép thực hiện nhiều lần một đoạn mã nào đó
- JavaScript là một ngôn ngữ được sử dụng phổ biến trên các giao diện web
- Ngày nay JavaScript còn có thể sử dụng để tạo các ứng dụng back-end, mobile...

- IDE (Integrated Development Environment) là thuật ngữ được sử dụng để chỉ đến các công cụ soạn thảo mã nguồn được tích hợp thêm các môi trường và tính năng khác để hỗ trợ cho việc lập trình

codegym.vn

Chương 2 - Biến, kiểu dữ liệu và toán tử

Thực hiện các phép tính toán

1. Mục tiêu

- Mô tả được khái niệm biến
- Khai báo và sử dụng được biến
- Mô tả được các kiểu dữ liệu thông dụng
- Sử dụng được đúng kiểu dữ liệu phù hợp
- Sử dụng được các toán tử số học
- Sử dụng được các toán tử logic
- Sử dụng được các toán tử so sánh

2. Giới thiệu

Ta đã biết từ chương trước, mọi chương trình máy tính đều thực hiện 4 công việc: nhận dữ liệu, lưu trữ, xử lý, xuất thông tin. Chẳng hạn, trong một ứng dụng bán hàng trực tuyến, người dùng sẽ nhập vào số lượng sản phẩm muốn mua, hệ thống sẽ dựa vào đó để tính được tổng số tiền mà người mua phải trả và hiển thị thông tin đó cho người dùng.

Chương này sẽ giúp chúng ta lựa chọn đúng kiểu dữ liệu phù hợp với từng tình huống. Chẳng hạn, để lưu trữ tên của người dùng thì chọn kiểu dữ liệu gì? Để lưu tuổi của người dùng thì chọn kiểu dữ liệu gì? Để lưu giới tính của người dùng thì chọn kiểu dữ liệu gì?...

Đối với các giá trị của từng kiểu dữ liệu, chúng ta có thể thực hiện được các phép toán khác nhau. Chẳng hạn, chúng ta có thể thực hiện các phép tính Cộng, Trừ, Nhân, Chia giữa các giá trị số. Chúng ta có thể thực hiện phép so sánh giữa các giá trị có kiểu ký tự, v.v.

Hoàn thành chương này, chúng ta có thể tạo được các ứng dụng trong đó thực hiện được các phép tính toán căn cứ vào dữ liệu mà mình nhập vào.

3. Biến

Khái niệm biến

Trong địa lý, nếu chúng ta được yêu cầu chỉ vị trí của vùng nằm ở tọa độ 16°B và 112°Đ ở trên quả địa cầu, phần lớn chúng ta sẽ mất khá nhiều thời gian để suy nghĩ, tìm kiếm và sau đó mới xác định được đúng tọa độ của vùng đó.

Hình 2.1



Tuy nhiên, nếu chúng ta được yêu cầu chỉ vị trí của Quần đảo Hoàng Sa, phần lớn chúng ta đều dễ dàng xác định được dễ dàng và nhanh chóng.

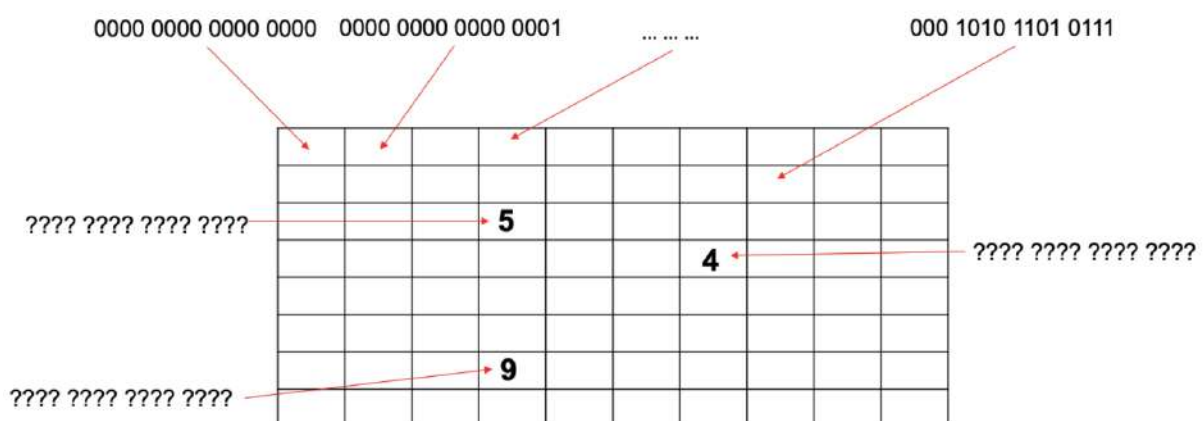
Tương tự như vậy, thật khó để có thể nhớ được toạ độ của các thành phố thủ đô của các nước, nhưng chúng ta lại khá dễ dàng để xác định được vị trí của thành phố thông qua tên của chúng, chẳng hạn như Paris, Berlin, Bangkok, v.v.

Điều gì làm nên sự khác biệt trong việc sử dụng các toạ độ và sử dụng các tên gọi? Rõ ràng, ai cũng biết là việc sử dụng các tên gọi sẽ giúp dễ nhớ hơn, còn toạ độ thường được dùng trong các tính toán khoa học. Việc sử

dụng tên gọi đại diện cho một vùng nào đó trên bản đồ sẽ giúp cho chúng ta dễ dàng hơn khi làm việc với nó.

Trong thế giới máy tính cũng vậy. Dữ liệu được lưu trong bộ nhớ. Bộ nhớ bao gồm nhiều ô nhớ. Mỗi ô nhớ có một địa chỉ riêng của nó để xác định ô nhớ đó. Cũng giống như toạ độ để xác định một vùng đất vậy. Tệ hơn, địa chỉ của các ô nhớ còn nhiều, dài dòng và khó nhớ hơn gấp nhiều lần so với kinh độ và vĩ độ trên trái đất. Có hàng triệu, thậm chí là hàng tỷ ô nhớ trong mỗi máy tính.

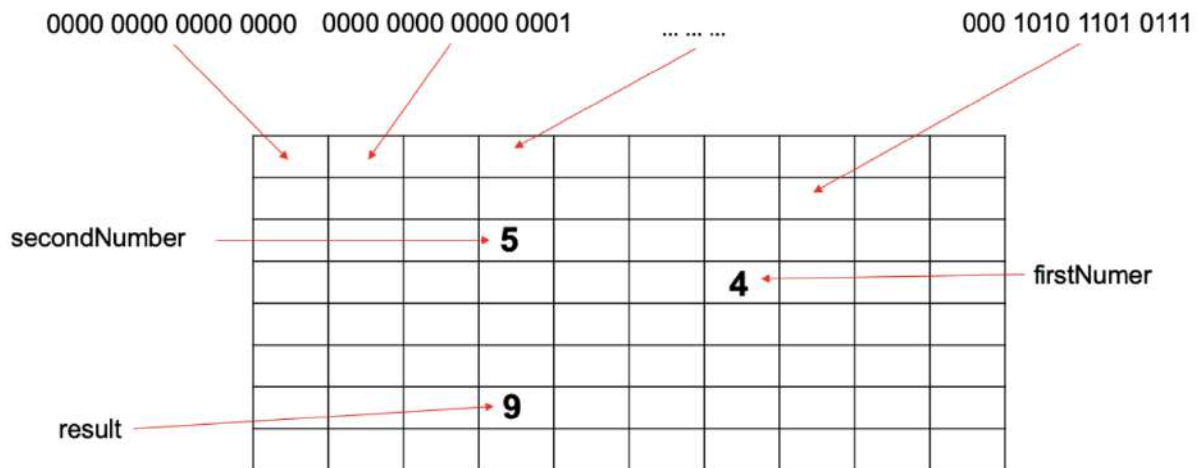
Giả sử, chúng ta cần thực hiện phép trừ 3 số. Như vậy chúng ta bắt đầu nhập vào số thứ nhất, lưu nó vào trong một ô nhớ ngẫu nhiên trong bộ nhớ, chúng ta phải ghi nhớ địa chỉ của ô này. Sau đó, chúng ta nhập số thứ 2 và lại lưu vào trong bộ nhớ, lại phải ghi nhớ địa chỉ ô nhớ này. Khi thực hiện phép cộng, chúng ta phải sử dụng các địa chỉ ô nhớ này để lấy các giá trị ra, rồi lại phải ghi kết quả của phép tính vào một ô nhớ khác, lại phải nhớ địa chỉ của ô nhớ mới.



Hình 2.2: Các ô nhớ

Ở hình trên, các giá trị 5, 4 và 9 đều được lưu trữ ở trong bộ nhớ máy tính ở các vùng khác nhau được gọi là các ô nhớ. Mỗi ô nhớ đều được xác định bởi một địa chỉ khá dài và khó nhớ.

Cứ như vậy, điều gì sẽ xảy ra nếu chúng ta thực hiện một phép tính với sự tham gia của hàng chục giá trị khác nhau? Khả năng của lập trình viên khó có thể nhớ được địa chỉ của từng ấy ô nhớ. Đây là lúc chúng ta cần đặt tên cho các ô nhớ.



Hình 2.3: Sử dụng tên để đại diện cho các ô nhớ

Bây giờ, thay vì phải ghi nhớ địa chỉ của từng ô nhớ, chúng ta sẽ sử dụng các tên để đại diện cho ô nhớ mà mình mong muốn.

Ví dụ, khi nhập vào số thứ nhất, chúng ta đặt tên cho ô nhớ đó là *firstNumber*. Khi nhập vào số thứ hai, chúng ta đặt tên cho ô nhớ đó là *secondNumber*. Khi lưu trữ giá trị của phép tính, chúng ta đặt tên cho ô nhớ mới là *result*.

Chúng ta dễ dàng nhận thấy, cách làm này tốt hơn rất nhiều so với trước đây. Với cách làm này, chúng ta vừa tìm hiểu một khái niệm rất quan trọng trong lập trình, được gọi là *biến*.

Định nghĩa biến: *Biến là một tên gọi đại diện cho một vùng nhớ để lưu trữ dữ liệu trong máy tính.*

Khai báo biến

Khai báo biến là thao tác đặt tên cho một ô nhớ trong bộ nhớ. Sau khi khai báo biến thì chúng ta có thể sử dụng biến đó để thao tác với ô nhớ mà nó đại diện.

Trong JavaScript, chúng ta khai báo một biến bằng cách sử dụng từ khóa *let*. Ví dụ sau đây khai báo lần lượt 3 biến là *radius*, *area* và *diameter*:

```
1. let radius;
2. let area;
3. let diameter;
```

Lưu ý: *Tại các phiên bản JavaScript cũ hơn so với phiên bản phổ biến hiện nay, chúng ta khai báo biến bằng từ khóa *var*.*

Gán giá trị cho biến

Để gán giá trị cho một biến, chúng ta sử dụng toán tử gán (=).

Ví dụ:

```
1. radius = 2.5;
2. area = 19.6;
3. diameter = 15.7;
```

Nói rằng “=” là một toán tử là bởi vì ngoài việc làm cho biến thay đổi giá trị, bản thân phép gán cũng trả về giá trị được gán giống như là kết quả của một phép tính. Vậy nên mặc dù câu lệnh sau trông rồi và ít được dùng trong thực tế thì vẫn là câu lệnh đúng:

```
1. radius1 = radius2 = 2.5;
```

Phép gán *radius2* làm cho *radius2* có giá trị là 2.5, đồng thời “phép tính” này cho kết quả là 2.5, kết quả này được gán vào cho biến *radius1*. Như vậy, kết quả là cả biến *radius1* và biến *radius2* đều có giá trị là 2.5.

Khai báo và khởi tạo giá trị cho biến

Hành động tạo giá trị cho biến lần đầu tiên (nhớ rằng sau đó giá trị của biến có thể được thay đổi bằng cách thực hiện lại phép gán) được gọi là khởi tạo giá trị cho biến. Chúng ta có thể khai báo một biến đồng thời với việc khởi tạo:

```
1. let radius = 2.5;
2. let area = 19.6;
3. let diameter = 15.7;
```

Lưu ý: một biến có thể được cấp lại giá trị, không giới hạn bao nhiêu lần. Trái lại, tại cùng một phạm vi mã (scope), bạn không thể khai báo một biến nhiều lần, chúng ta sẽ học về phạm vi mã sau, nhưng tại thời điểm này bạn có thể hiểu rằng chỉ dẫn sau sẽ gây lỗi bởi vì biến *radius* được khai báo hai lần:

```
1. let radius = 2.5;
2. let radius = 3;
```

```
// Uncaught SyntaxError: Identifier 'radius' has already been declared
```

Quy tắc đặt tên cho biến

Vì một số lý do, trong hầu hết các ngôn ngữ lập trình, chúng ta chỉ có thể sử dụng các chữ cái tiếng Anh và ký tự số để đặt tên cho biến. Tên biến sẽ không có các khoảng trắng (bao gồm cả dấu tab lẫn dấu xuống dòng), ký tự đặc biệt, hay chữ cái có dấu. Chữ cái đầu tiên của biến không được phép là ký tự số.

Riêng với ngôn ngữ JavaScript, ngoài hai quy tắc trên, chúng ta được phép sử dụng hai ký tự là \$ và _ (dấu gạch ngang dưới) như là chữ cái.

Chúng ta có thể kể ra một số tên biến hợp lệ và không hợp lệ như sau:

```
1. let money; // Hợp lệ
2. let 1000dolar; // Không hợp lệ
3. let my money; // Không hợp lệ
4. let _radius; // Hợp lệ
5. let $username; // Hợp lệ
```

Lưu ý: Các ràng buộc được nhắc tới ở đây là Quy tắc (rules), có nghĩa là nếu bạn không tuân thủ, trình biên dịch và thực thi JavaScript sẽ báo lỗi. Bạn phân biệt Quy tắc với các Quy ước (convention) - là những ràng buộc mà nếu tuân thủ thì sẽ rất tốt chứ không phải là bắt buộc. Chúng ta có thể vi phạm các Quy ước mà không ảnh hưởng trực tiếp tới kết quả thực thi của chương trình.

4. Kiểu dữ liệu

Trong máy tính, một giá trị sẽ được thể hiện bằng *dữ liệu* đi kèm với *kiểu giữ liệu*. Cùng một vùng nhớ nhưng nếu được nhìn nhận dưới kiểu dữ liệu khác nhau thì sẽ thể hiện cho những giá trị khác nhau, chẳng hạn, trong một số trường hợp, ký tự 'A' nếu được nhìn nhận dưới kiểu dữ liệu số thì sẽ thể hiện cho số 65. Kiểu dữ liệu sẽ ảnh hưởng tới các toán tử có thể sử dụng lên dữ liệu (chẳng hạn, kiểu dữ liệu ký tự thì không thể áp dụng phép chia được) cũng như cách hành xử của mỗi toán tử. Chẳng hạn hai giá trị số là 1 và 2 khi được áp dụng phép tính cộng sẽ cho kết quả là 3, nhưng hai ký tự "1" và "2" cộng lại sẽ cho kết quả "12".

Định kiểu động

JavaScript là một ngôn ngữ *định kiểu động*. Chúng ta không cần phải khai báo kiểu của các biến, thay vào đó, trình thực thi sẽ suy đoán kiểu của biến thông qua giá trị của nó. Nhờ đó mà đoạn mã sau cho chúng ta kết quả là 3 chứ không phải là "12":

```
1. let foo = 1;  
2. let bar = 2;  
3. console.log(foo + bar); // 3
```

Hai nhóm Kiểu dữ liệu

Trình thực thi JavaScript nhìn dữ liệu thành hai dạng: dữ liệu dạng nguyên thủy và dữ liệu dạng đối tượng. Trong khuôn khổ cuốn sách này chúng ta sẽ chỉ tập trung thao tác với kiểu dữ liệu nguyên thủy.

Kiểu dữ liệu nguyên thủy là kiểu dữ liệu được xây dựng sẵn trong ngôn ngữ JavaScript. Theo đó, hầu hết các toán tử được xây dựng để hỗ trợ xoay quanh các kiểu dữ liệu này.

Các kiểu dữ liệu nguyên thủy

Có các loại dữ liệu nguyên thủy khác nhau, trong đó kiểu Symbol mới được bổ sung gần đây. Kiểu Symbol có mục đích sử dụng khá đặc thù và chúng ta sẽ không bàn tới nó trong cuốn sách này. Các kiểu dữ liệu nguyên thủy còn lại là:

- boolean
- undefined
- number
- string

Kiểu number

Kiểu *number* thể hiện các giá trị số, bao gồm cả số nguyên và số thực. JavaScript chỉ có một kiểu số duy nhất, có giá trị từ $-(2^{53} - 1)$ đến $2^{53} - 1$. Ngoài việc có thể chứa giá trị dấu phẩy động, kiểu số có thêm ba giá trị đại diện: *Infinity* (dương vô cùng), *-Infinity* (âm vô cùng), và *NaN* (Not-a-Number – không phải là một số). Kiểu dữ liệu số được dùng trong các trường hợp như để lưu tuổi của học sinh, giá của sản phẩm, nhiệt độ trong ngày, v.v.

Ví dụ:

```
1. let age = 15;
2. let price = 20.05;
```

Trong ví dụ này, biến *age* và biến *price* có kiểu dữ liệu là số được dùng lần lượt để lưu tuổi của một người và giá của một sản phẩm.

Có một số nguyên duy nhất có hai đại diện: *0* được đại diện bởi *-0* và *+0*. (*0* là một cách viết ngắn gọn của *+0*). Mặc dù điều này hầu như không có tác động nào tới chương trình (biểu thức *(+0 === -0)* cho chúng ta kết quả là *True*), tuy nhiên chúng ta có thể nhận thấy ảnh hưởng này khi chia một số cho *0*:

```
1. let x = 42 / 0; // Infinity;
2. let y = 42 / -0; // -Infinity;
```

Kiểu string

Kiểu *string* được dùng để biểu diễn dữ liệu dạng chuỗi ký tự. Mỗi ký tự có một *chỉ mục* (số thứ tự) của mình ở trong chuỗi. Trong ngôn ngữ JavaScript, các chỉ mục luôn được đánh số từ 0. Ký tự đầu tiên trong chuỗi sẽ chỉ số là 0, tiếp theo là 1, 2, 3... *Độ dài* của chuỗi là số ký tự của chuỗi đó. Kiểu dữ liệu *string* được sử dụng trong những trường hợp như tên của học sinh, địa chỉ nhà ở, tiêu đề của một bài viết, v.v.

Ví dụ: Chuỗi "HELLO" có độ dài là 5, bởi vì nó có chứa 5 ký tự. Chỉ mục của chữ H là 0, chỉ mục của chữ O là 4.

Để khai báo chuỗi thì chúng ta có thể sử dụng dấu nháy đơn hoặc dấu nháy kép. Chẳng hạn:

```
1. let brand = "CodeGym Việt Nam"; //Sử dụng dấu nháy kép
2. let address = 'Hà Nội'; //Sử dụng dấu nháy đơn
```

Lưu ý: Trong một chương trình, nên thống nhất việc sử dụng dấu nháy đơn hoặc nháy kép. Tránh trường hợp sử dụng lộn xộn cả hai cách khiến cho mã nguồn trở nên xấu và khó đọc.

Chuỗi trong JavaScript là bất biến (immutable). Chúng ta không thể thay đổi chuỗi sau khi đã khởi tạo giá trị. Tuy nhiên chúng ta có thể tính toán một chuỗi mới dựa trên giá trị cũ và gán ngược lại vào biến chuỗi.

Kiểu boolean

Kiểu boolean đại diện có hai giá trị logic là *true* và *false*. Kiểu dữ liệu boolean được sử dụng trong các trường hợp cần phân biệt 2 trạng thái là *true* hoặc *false*. Chẳng hạn như là trạng thái bật và tắt của bóng đèn, kết quả của một biểu thức điều kiện trong đó có sử dụng các phép so sánh, v.v.

Trong JavaScript, tất cả các giá trị sau đều được coi là *false*:

- Giá trị *false* của kiểu *boolean*
- Giá trị *+0* và *-0* của kiểu *number*
- Chuỗi rỗng (giá trị *""*)
- Giá trị *undefined* của kiểu *undefined*
- Giá trị *null* của kiểu *object*

Tất cả các giá trị khác sẽ được coi là *true* khi thực hiện phép chuyển đổi kiểu.

```
1. let f1 = Boolean(0) // false
2. let f2 = Boolean("") // false
3. let f3 = Boolean(undefined) // false
4. let f4 = Boolean(-1) // true
5. let f5 = Boolean('hello') // true
```

Kiểu undefined

Undefined mang ý nghĩa là "*chưa xác định*". Một biến chưa được gán giá trị thì có giá trị là *undefined*, đồng thời kiểu dữ liệu của nó cũng là *undefined*.

```
1. let x
2. console.log(x)
3. console.log(typeof x) // undefined
4. x = 3
5. console.log(x) // 3
6. console.log(typeof x) // number
```

5. Phép toán toán học

Toán tử (Operator)

Trong toán học, chúng ta đã quen thuộc với các toán tử (hay còn gọi là ký tự phép toán) như *+*, *-*, ***, *:* ... Chúng được sử dụng trong các biểu thức toán học, thực hiện các tính toán dựa trên giá trị của các toán hạng và trả về một kết quả cuối cùng. Trong các ngôn ngữ lập trình cũng vậy, có nhiều toán tử khác nhau được sử dụng cho nhiều mục đích khác nhau. Có những loại phổ biến như:

- Toán tử toán học được sử dụng trong các phép tính toán học
- Toán tử gán được sử dụng để gán giá trị cho các biến
- Toán tử cộng chuỗi được sử dụng để nối hai chuỗi
- Toán tử so sánh được sử dụng để so sánh các giá trị với nhau
- Toán tử logic được sử dụng để thay đổi giá trị thuộc kiểu dữ liệu boolean

- Toán tử typeof được sử dụng để xác định kiểu dữ liệu của một giá trị

Toán tử toán học (arithmetic)

Toán tử toán học được sử dụng trong các biểu thức toán học. Toán tử toán học làm việc với các giá trị số. Thông thường, trong một biểu thức thì toán tử toán học có 2 toán hạng, nhưng cũng có các trường hợp là một toán hạng. Chẳng hạn, trong biểu thức dưới đây thì toán tử là +, hai toán hạng là 100 và 50:



Danh sách các toán tử toán học được tóm tắt trong bảng sau:

Toán tử	Mô tả
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Chia lấy phần dư (modulus)
++	Tăng 1 giá trị
--	Giảm 1 giá trị

Ví dụ 1: Sử dụng với các giá trị số

```
1. let x = 100 + 50;
```

Ví dụ 2: Sử dụng với các biến

```
1. let a = 5;
2. let b = 10;
3. let c = a + b;
```

Ví dụ 3: Sử dụng với các biểu thức

```
1. let a = 2;
2. let x = (100 + 50) * a;
```

Toán tử Cộng

Toán tử cộng (+) được sử dụng để tính tổng của hai số:

```
1. let x = 5;
2. let y = 2;
3. let z = x + y; // z = 7
```

Toán tử Trừ

Toán tử trừ (-) được sử dụng để tính hiệu của hai số:

```
1. let x = 5;
2. let y = 2;
3. let z = x - y; // z = 3
```

Toán tử Nhân

Toán tử nhân (*) được sử dụng để tính tích của hai số:

```
1. let x = 5;
2. let y = 2;
3. let z = x * y; // z = 10
```

Toán tử Chia

Toán tử chia (/) được sử dụng để tính hiệu của hai số:

```
1. let x = 5;
2. let y = 2;
3. let z = x / y; // z = 2.5
```

Toán tử Chia lấy số dư

Toán tử chia lấy số dư (%) được sử dụng để tính số dư của một phép chia:

```
1. let x = 5;
2. let y = 2;
3. let z = x % y; // z = 1
```

Toán tử Tăng giá trị

Phép tăng giá trị là một phép toán cần duy nhất một toán hạng. Tác dụng của nó là tăng giá trị của một biến *number* được dùng làm toán hạng tăng lên một đơn vị.

```
1. let x = 5;
2. x++; // x = 6
```

Phép tăng giá trị cũng tạo ra kết quả của bản thân phép tính. Giá trị đó phụ thuộc vào vị trí đặt toán tử. Nếu toán tử được đặt trước thì kết quả của phép toán sẽ là giá trị của toán hạng sau khi tăng, còn nếu toán tử đặt sau thì kết quả sẽ là giá trị của toán hạng trước khi tăng. Ví dụ:

```
1. let x = 5;
2. let y = x++; // x = 6, y = 5
3. x = 5
4. let z = ++x // x = 6, y = 6
```

Trong cả hai trường hợp trên thì giá trị của biến x đều được tăng lên là 6. Nhưng kết quả của bản thân phép toán thì khác nhau cho nên giá trị của biến y là khác nhau.

Toán tử giảm giá trị:

Toán tử giảm giá trị (`--`) được sử dụng để giảm giá trị của một biến *number* được dùng làm toán hạng giảm đi một đơn vị.

```
1. let x = 5;
2. x--; // x = 4
```

Lưu ý: Tương tự như toán tử Tăng giá trị, toán tử Giảm giá trị cũng có kết quả trả về và kết quả này cũng bị ảnh hưởng bởi vị trí đặt toán tử.

Toán tử gán (assignment)

Toán tử gán được sử dụng để gán giá trị cho một biến, toán tử gán có thể sử dụng với tất cả các kiểu dữ liệu khác nhau.

```
1. let x = 10;
```

Trong đoạn mã trên, chúng ta gán giá trị `10` cho biến `x`. Sau khi biểu thức này được thực thi thì giá trị của biến `x` sẽ là `10`.

Toán tử gán cơ bản nhất là toán tử bằng, trong đó giá trị phía bên phải dấu bằng được gán cho biến ở phía bên trái của nó.

Ngoài ra, còn có thêm các toán tử gán phái sinh, kết hợp toán tử bằng với các toán tử khác. Chẳng hạn, toán tử cộng gộp là kết hợp giữa toán tử cộng và toán tử bằng.

Biểu thức `x += y` tương đương với biểu thức `x = x + y`.

Bảng sau đây liệt kê các toán tử phái sinh từ toán tử gán.

Toán tử	Ví dụ	Tương đương với
<code>=</code>	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>

Ví dụ 1:

```
1. let x = 5;
```

Ví dụ 2:

```
1. let x = 10;
2. x += 5; // x = 15 (Tương đương với thực hiện gán x = x + 5)
```

Ví dụ 3:

```
1. let x = 10;
```

```
2. x -= 5; // x = 5 (Tương đương với thực hiện gán x = x - 5)
```

Ví dụ 4:

```
1. let x = 10;
2. x *= 5; // x = 50 (Tương đương với thực hiện gán x = x * 5)
```

Ví dụ 5:

```
1. let x = 10;
2. x /= 5; // x = 2 (Tương đương với thực hiện gán x = x / 5)
```

Ví dụ 6:

```
1. let x = 11;
2. x %= 5; // x = 1 (Tương đương với thực hiện gán x = x % 5)
```

Toán tử cộng chuỗi (string concatenate)

Toán tử cộng chuỗi được sử dụng để nối các chuỗi. Cũng có thể sử dụng để nối chuỗi và các số.

Ví dụ 1:

```
1. let txt1 = "John";
2. let txt2 = "Doe";
3. let txt3 = txt1 + " " + txt2; // txt3 = "John Doe"
```

Trong ví dụ trên, giá trị của biến txt3 là một chuỗi "John Doe".

Ví dụ 2:

```
1. let x = 5 + 5; // x = 10
2. let y = "5" + 5; // y = "55"
3. let z = "Hello" + 5; // z = "Hello5"
```

Trong ví dụ trên, giá trị của biến x là số 10 (cộng hai số). Giá trị của biến y là một chuỗi "55" (cộng một chuỗi với một số). Giá trị của biến z là một chuỗi "Hello5" (cộng một chuỗi với một số).

6. Phép toán logic

Toán tử logic được dùng trong các biểu thức logic (*true/false*). Toán tử "VÀ" được ký hiệu bởi hai dấu và (&&). Toán tử "HOẶC" được ký hiệu bởi hai dấu gạch đứng (||). Toán tử "PHỦ ĐỊNH" được ký hiệu bởi một dấu chấm than (!).

Toán tử	Mô tả
&&	Và (AND)
	Hoặc (OR)
!	Phủ định (NOT)

Toán tử &&

Toán tử && trả về giá trị *true* nếu **cả hai** toán hạng đều có giá trị *true*. Chỉ cần một toán hạng có giá trị *false* thì kết quả sẽ là *false*.

Chúng ta có thể tổng hợp các trường hợp giá trị của *a*, *b* và kết quả như trong bảng.

Giá trị biến a	Giá trị biến b	Kết quả (a && b)
true	true	true
true	false	false
false	false	false
false	true	false

Toán tử ||

Toán tử || trả về giá trị *true* nếu **một trong hai** toán hạng có giá trị *true*. Chỉ duy nhất trường hợp nếu cả hai toán hạng có giá trị *false* thì kết quả sẽ là *false*.

Chúng ta có thể tổng hợp các trường hợp giá trị của *a*, *b* và kết quả như trong bảng.

Giá trị biến a	Giá trị biến b	Kết quả (a b)
true	true	true
true	false	true
false	false	false
false	true	true

Toán tử !

Toán tử phủ định đảo ngược giá trị boolean, chẳng hạn giá trị *true* thì chuyển thành *false* và giá trị *false* thì chuyển thành *true*.

Giá trị biến a	Kết quả !a
true	false
false	true

Độ ưu tiên của các toán tử

Trong toán học các phép tính trong một biểu thức có độ ưu tiên nhất định để xác định phép tính nào được tính toán trước. Trong lập trình cũng vậy, nếu trong một biểu thức có sự tham gia của nhiều toán tử thì thứ tự thực hiện được dựa trên độ ưu tiên được quy định sẵn.

Bảng bên dưới liệt kê các thứ tự ưu tiên ứng với từng toán tử:

Toán tử	Độ ưu tiên
Hậu tố (postfix)	expr++ expr--
Một ngôi	++expr --expr +expr -expr ~ !
Nhân	* / %
Cộng	+ -
Dịch	<< >> >>>
Quan hệ	< > <= >= instanceof
Bằng	== !=
Bitwise AND	&
Bitwise exclusive OR	^
Bitwise inclusive OR	
Và	&&
Hoặc	
Ba ngôi	? :
Gán	= += -= *= /= %= &= ^= = <<= >>= >>>=

Nếu muốn thay đổi độ ưu tiên của các toán tử, chúng ta có thể sử dụng phép đóng mở ngoặc tương tự như trong toán học. Đối với các phép toán có cùng độ ưu tiên, chẳng hạn như phép cộng và phép trừ thì thứ tự thực hiện là từ trái sang phải.

Ví dụ:

Khởi tạo hai biến x và y có giá trị lần lượt là 5 và 6:

```
1. let x = 5;
2. let y = 10;
```

Khởi tạo biến z có giá trị là kết quả của một biểu thức, các bước tính toán của biểu thức được thực hiện minh họa ở các dòng phía dưới.

```
let z = (++x * y) < 5 * 10 && 6 > 3;
(6 * y) < 5 * 10 && 6 > 3;
60 < 50 && 6 > 3;
false && true;
false
```

Trong biểu thức trên, trước tiên toán tử ++ được ưu tiên thực thi ($++x = 6$). Tiếp theo các toán tử nhân được thực thi ($6 * y = 60$ và $5 * 10 = 50$). Tiếp theo sẽ thực thi các toán tử so sánh ($60 < 50 = false$ và $6 > 3 = true$). Cuối cùng, toán tử logic && được thực thi. Kết quả biến z sẽ có giá trị là false.

7. Phép toán so sánh

Các toán tử so sánh được dùng để đánh giá mức độ tương quan giữa các giá trị. Kết quả của phép toán so sánh sẽ là *true* hoặc *false*. Chẳng hạn, chúng ta có các toán tử rất quen thuộc là toán tử *Bằng*, toán tử *Lớn hơn*, toán tử *Nhỏ hơn*, toán tử *Lớn hơn hoặc Bằng*, toán tử *Nhỏ hơn hoặc Bằng*.

Lưu ý: Có hai loại toán tử so sánh *Bằng*, một loại chỉ có 2 dấu bằng (==) dùng để so sánh giá trị mà không phân biệt kiểu dữ liệu của chúng, một loại có 3 dấu bằng (===), được sử dụng để so sánh trong trường hợp các giá trị có cùng kiểu. Trong ngôn ngữ Javascript, chúng ta thường sử dụng toán tử có 3 dấu bằng hơn.

Cũng có hai toán tử so sánh *Khác*, một loại bao gồm một dấu chấm than và một dấu bằng (!=), dùng để so sánh khác nhau giữa hai giá trị mà không phân biệt kiểu dữ liệu. Một loại bao gồm một dấu chấm than và hai dấu bằng (!==), được sử dụng để so sánh khác nhau giữa hai giá trị cùng kiểu dữ liệu.

Danh sách các toán tử so sánh được liệt kê trong bảng sau:

Toán tử	Mô tả
==	Bằng về giá trị (không phân biệt kiểu dữ liệu)
===	Bằng về giá trị, đồng thời cùng kiểu dữ liệu
!=	Khác về giá trị (không phân biệt kiểu dữ liệu)
!==	Không bằng về giá trị, hoặc không cùng kiểu dữ liệu
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng

Ví dụ 1:

```
1. let x = 5;
2. let y = x == 5; // y = true
3. let z = x == "5"; // z = true
```

Trong ví dụ trên, chúng ta sử dụng toán tử so sánh *Bằng* với 2 dấu bằng, biến *y* có giá trị là *true*. Biến *z* cũng có giá trị là *true*, bởi vì mặc dù có 2 kiểu dữ liệu khác nhau nhưng giá trị của chúng đều là 5.

Ví dụ 2:

```
1. let x = 5;
2. let y = x === 5; // y = true
3. let z = x === "5"; // z = false
4.
```

Trong ví dụ trên, chúng ta sử dụng toán tử so sánh `===`, giá trị của biến `y` là `true`. Giá trị của biến `z` là `false`, bởi vì các giá trị bằng nhau nhưng lại có kiểu dữ liệu khác nhau (`x` có kiểu dữ liệu là `number` còn `"5"` có kiểu dữ liệu là `string`).

Ví dụ 3:

```
1. let x = 5;
2. let y = x !== 8; // y = true
3. let z = x !== "5"; // z = false
```

Trong ví dụ trên, chúng ta sử dụng toán tử so sánh *Khác* với một dấu chấm than và một dấu bằng, giá trị của biến `y` là `true` và giá trị của biến `z` là `false`.

Ví dụ 4:

```
1. let x = 5;
2. let y = x === 5; // y = false
3. let z = x === "5"; // z = true
```

Trong ví dụ trên, chúng ta sử dụng toán tử so sánh *Khác* bao gồm một dấu chấm than và hai dấu bằng, giá trị của biến `y` là `false`. Giá trị của biến `z` là `true`, bởi vì mặc dù hai giá trị giống nhau nhưng lại khác kiểu dữ liệu.

8. Đọc dữ liệu từ bên ngoài

Trong phạm vi cuốn sách này, chúng ta thực thi chương trình thông qua trình biên dịch của trình duyệt. Chương trình của chúng ta, theo đó, sẽ thu thập dữ liệu từ môi trường bên ngoài chúng thông qua trình duyệt. Có hai phương án mà chúng ta có thể sử dụng ngay được như sau:

- Sử dụng câu hàm `prompt()` để tạo dialog cho người dùng nhập liệu.
- Đọc giá trị của một phần tử `input` trong tài liệu HTML.

Nhận dữ liệu thông qua hộp thoại `prompt`.

Với phương án này, chúng ta sử dụng lời gọi hàm `prompt()` có sẵn trong môi trường thực thi của trình duyệt. Bạn sẽ nghiên cứu kỹ hơn về hàm nói chung ở các chương sau. Tại thời điểm này, hãy chỉ tập trung học cách sử dụng hàm `prompt`:

```
1. let radius = prompt("Vui lòng nhập bán kính hình tròn:");
```

Hàm `prompt()` ở trên sẽ yêu cầu trình duyệt mở một hộp thoại, với lời dẫn được mô tả trong tham số của nó, cùng với một ô nhập liệu và nút chấp nhận. Sau khi người dùng nhấn nút chấp nhận, giá trị họ nhập vào ô nhập liệu sẽ được gán cho biến `radius`:

```
1. console.log(radius); // 2.5
```

Lưu ý: Giá trị nhận được từ prompt luôn có kiểu dữ liệu là string, cho dù người dùng có nhập số hợp lệ đi chăng nữa. Trong trường hợp đó, để chuyển đổi giá trị này thành giá trị số, chúng ta có thể sử dụng hàm `Number()` như dưới đây:

```
1. typeof radius; // "string"
2. radius = Number(radius);
3. console.log(radius); // 2.5
4. typeof radius; // "number"
```

Như vậy, chúng ta có thể viết ngắn gọn quá trình nhập một số vào như sau:

```
1. let radius = Number(prompt("Vui lòng nhập bán kính hình tròn:"));
```

Đọc giá trị của ô input từ trong tài liệu HTML

Trong tài liệu HTML có thể có những thẻ `input`. Các thẻ này được trình duyệt hiển thị dưới dạng những ô nhập liệu mà người dùng có thể nhập giá trị vào. Trình duyệt cũng cung cấp sẵn một số hàm JavaScript để lập trình viên có thể đọc được giá trị của các ô input đó.

Chẳng hạn, chúng ta có một tài liệu HTML với thẻ input như sau:

```
1. <h3>Circles Calculator</h3>
2. <label>
3.   Bán kính: <input type="text" id="radius">
4.   <button onclick="showArea()">Area</button>
5. </label>
6. <script type="text/JavaScript">
7.   function showArea() { }
8. </script>
```

Chúng ta có thể sử dụng hàm JavaScript như sau để đọc giá trị nhập vào của phần tử input có id là `radius` ở trên:

```
1. let radius = document.getElementById("radius").value;
```

Giải thích

Ở dòng lệnh trên, chúng ta đã sử dụng hàm `getElementById()` có sẵn của đối tượng `document` để lấy về giá trị của thẻ input có tên là `radius`.

Hãy thử hoàn thành hàm `showArea()` với câu lệnh trên để giúp tính ra cho người dùng bán kính của hình tròn. Sau đó thực thi chương trình để xem kết quả.

```
1. function showArea() {
2.   let radius = document.getElementById("radius").value;
3.   console.log("Độc được giá trị bán kính là " + radius);
4.   let area = radius * radius * 3.14;
5.   console.log("Tính được giá trị diện tích là " + area + " với PI = " + 3.14);
6.   alert("Diện tích hình tròn có bán kính " + radius + " là " + area);
7.   console.log("Người dùng đã nhận được kết quả!");
8. }
```

Lưu ý: cũng như `prompt()`, kết quả nhận được từ thuộc tính `value` ở đây cũng luôn có kiểu dữ liệu string.

9. Hiển thị dữ liệu

JavaScript có thể "hiển thị" dữ liệu theo nhiều cách khác nhau:

- Viết thành một phần tử HTML, sử dụng thuộc tính *innerHTML*.
- Viết vào đầu ra HTML sử dụng hàm *document.write()*.
- Viết vào một hộp cảnh báo sử dụng hàm *alert()*.
- Viết vào bảng điều khiển trình duyệt sử dụng hàm *console.log()*.

Sử dụng innerHTML

Để truy cập một phần tử HTML, JavaScript có thể sử dụng phương thức *document.getElementById(id)*. Trong đó, *id* là thuộc tính giúp xác định thành phần HTML mà chúng ta muốn truy cập. Thuộc tính *innerHTML* chính là nội dung bên trong của thành phần HTML đó:

Ví dụ:

```
1. <p id="demo"></p>
2. <script>
3.   document.getElementById("demo").innerHTML = 7;
4. </script>
```

Kết quả: Nội dung của thẻ `<p>` với id là *demo* đã được thay đổi thành 7.

Sử dụng document.write()

Hàm *document.write()* thường được dùng để hiển thị dữ liệu bằng cách viết trực tiếp ra tài liệu HTML.

Ví dụ:

```
1. <h2>My First Web Page</h2>
2. <p>My first paragraph.</p>
3. <button type="button" onclick="document.write(5 + 6)">Try it</button>
```

Sử dụng hàm alert()

Hàm *alert()* có nhiệm vụ in một thông báo popup, nó có một tham số truyền vào là nội dung ta muốn thông báo với người dùng.

Ví dụ:

```
1. <h2>My First Web Page</h2>
2. <p>My first paragraph.</p>
3. <button type="button" onclick="alert(5 + 6)">Try it</button>
```

Sử dụng console.log()

Hàm *console.log()* được sử dụng thường xuyên trong việc tìm lỗi (debug). Hàm này có nhiệm vụ hiển thị ra giá trị của tất cả các loại dữ liệu như *number*, *integer*, *array*, *object*... ra cửa sổ console của trình duyệt.

Tùy thuộc vào các trình duyệt mà cách thức mở panel *console* của trình duyệt sẽ khác nhau, nhưng thường thì chúng ta sử dụng phím F12.

Ví dụ:

```
1. <script>
2.   console.log("xin chao")
3. </script>
```

Kết quả hiển thị sẽ nằm ở panel console như dưới đây.



10. Mã sạch

Quy tắc #1: Quy ước

Cú pháp, với mỗi cộng đồng hay mỗi nhóm phát triển sẽ có các quy định cách đặt tên biến khác nhau, nhưng đều sẽ tuân thủ theo những cách đặt bên dưới đây:

- Cú pháp lạc đà (camelCase): Ký tự đầu tiên của từ đầu tiên viết thường, những ký tự đầu tiên của những từ tiếp theo được viết hoa.
- Cú pháp con rắn (snake_case): Với cú pháp con rắn, tất cả các chữ cái đều viết thường, và các từ cách nhau bởi dấu gạch dưới.

Ví dụ:

Các tên biến theo kiểu *camelCase*:

```
1. let userName;
2. let thisIsTheNameThatFollowTheCamelCase;
```

Các tên biến theo kiểu *snake_case*:

```
1. let user_name;
2. let this_is_the_name_that_follow_the_snake_case;
```

Quy tắc #2: Sử dụng đúng loại từ

Tên biến nên là danh từ, cụm danh từ, hoặc tính từ.

Ví dụ:

```
1. let name; // Nên
2. let color; // Nên
3. let visible; //Nên
4. let productPrice; // Nên
5. let move; // Không nên
6. let show; // Không nên
```

Quy tắc #3: Tên có ý nghĩa

Tên biến cần có ý nghĩa. Khi nhìn vào tên biến nó sẽ trả lời cho câu hỏi vì sao nó tồn tại, nó sẽ làm gì.

```
1. let abc; // Không nên
2. let uName; // Không nên
```

Quy tắc #4: Mỗi biến khai báo trên một dòng

Khi khai báo biến nên tách mỗi dòng khai báo một biến.

Ví dụ:

```
1. let i = 3, j = 5, k = 10; // Không nên
2. let i = 3; // Nên
3. let j = 5; // Nên
4. let k = 10; // Nên
```

11. Các lỗi thường gặp

Các lỗi thường gặp #1: Toán tử “===”

Toán tử “===” so sánh đồng thời giá trị và kiểu dữ liệu. Ví dụ:

```
1. let x = 5;
2. let y = x === 5; // y = true
3. let z = x === "5"; // z = false
```

Trong ví dụ trên, biến *y* có giá trị là *true* vì biến *x* và 5 có cùng giá trị và cùng kiểu dữ liệu. Biến *z* có giá trị là *false* vì biến *x* và “5” có cùng giá trị nhưng khác nhau về kiểu dữ liệu.

Các lỗi thường gặp #2: So sánh số thực

Trong lập trình, không phải bao giờ các phép tính toán trên số thực cũng cho kết quả chính xác tuyệt đối. Việc so sánh bằng (==) giữa hai biểu thức số thực là không đáng tin cậy.

Ví dụ, đoạn mã sau sẽ hiển thị kết quả là *false* thay vì *true* như chúng ta thường nghĩ

```
1. let x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
2. console.log(x == 0.5); // false
3. console.log(x); // 0.5000000000000001
```

Trong ví dụ trên, giá trị của biến *x* không phải là 0.5 mà là 0.5000000000000001

Giải pháp cho tình huống so sánh số thực:

Để xử lý tình huống so sánh giữa các số thực, chúng ta sử dụng cách so sánh gần đúng, dựa trên nguyên tắc: hiệu số của hai giá trị là một số rất nhỏ (epsilon), chẳng hạn như 10^{-7} hoặc 10^{-14} .

Ví dụ:

```
1. const EPSILON = 1E-14;
2. let x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
3. console.log(Math.abs(x - 0.5) < EPSILON); // true
```

Hàm *Math.abs()* trên đây cho chúng ta giá trị tuyệt đối của biểu thức *x - 0.5*. Kết quả nhỏ hơn sai số cho phép và kết quả so sánh là *true*.

13. Bài thực hành

Bài 1: Khai báo các biến thuộc các kiểu khác nhau, gán giá trị cho chúng và in ra tài liệu HTML.

- Biến *i* kiểu *number*, có giá trị là 10
- Biến *f* kiểu *number*, có giá trị là 20.5
- Biến *b* kiểu *boolean*, có giá trị là *true*
- Biến *s* kiểu *string*, có giá trị là "Hà Nội".

Sử dụng hàm *document.write()* để viết kết quả ra màn hình.

Mã tham khảo:

```
1. let i = 10;  
2. let f = 20.5;  
3. let b = true;  
4. let s = 'Hà Nội';  
5.  
6. document.write('i = ' + i);  
7. document.write('<br/>');  
8. document.write('f = ' + f);  
9. document.write('<br/>');  
10. document.write('b = ' + b);  
11. document.write('<br/>');  
12. document.write('s = ' + s);
```

Bài 2: Viết một đoạn mã JavaScript, khai báo biến *width* chứa giá trị độ rộng của hình chữ nhật, biến *height* chứa giá trị chiều cao của hình chữ nhật. In ra màn hình diện tích của hình chữ nhật đó.

Mã tham khảo:

```
1. let width = 20;  
2. let height = 10;  
3. let area = width * height;  
4. document.write('Area = ' + area);
```

Bài 3: Dùng hàm *prompt()* để nhập 2 số từ bàn phím, lưu vào 2 biến là *a* và *b*. Kiểm tra xem *a* có phải là bội số của *b* hay không. Hiển thị kết quả ra màn hình bằng cách sử dụng hàm *alert()*.

Mã tham khảo:

```
1. let a = Number(prompt("Enter a:"));  
2. let b = Number(prompt("Enter b:"));  
3. let result = a % b;  
4. if(result === 0){  
5.   alert("a is the multiple of b");  
6. } else {  
7.   alert("a is not a multiple of b");  
8. }
```

14. Bài tập

Bài 1: Ứng dụng chuyển đổi tiền tệ

Hãy viết một ứng dụng cho phép chuyển đổi giữa VND và các loại tiền tệ khác. Giao diện của ứng dụng tương tự như sau, chương trình sẽ hiển thị hộp thoại để thông báo kết quả.

Money Calculator

From VND to
 From to VND

Bài 2: Ứng dụng máy tính đơn giản

Hãy viết một ứng dụng máy tính đơn giản cho phép thực hiện các phép tính Cộng, Trừ, Nhân, Chia, Bình phương, Lập phương...

Giao diện của ứng dụng có thể như sau:

Simple Calculator

Result: ()

- Chương trình có ba ô nhập liệu và các nút chức năng. Hai ô nhập đầu tiên để người dùng nhập các toán hạng, ô thứ 3 để chương trình hiển thị kết quả tính.
- Các nút chức năng chỉ cần dùng đến một giá trị (ví dụ tính bình phương, lập phương, phân số) thì chỉ dùng đến giá trị của ô nhập liệu thứ nhất.
- Nhấn vào các nút chức năng, chương trình sẽ hiển thị kết quả phép tính tương ứng vào ô nhập liệu thứ 3.
- Khi nhấn vào nút chức năng “Use now”, giá trị vừa tính ra sẽ được đưa vào ô nhập liệu đầu tiên, giá trị ở ô nhập liệu thứ 2 và thứ 3 sẽ được xóa trắng.

15. Bài kiểm tra

Câu 1: Trong JavaScript, nếu $x = 83$ và $y = 9$ thì sau khi thực hiện $x \% = y$, x sẽ có giá trị là bao nhiêu?

- 0
- 1
- 2
- 9

Câu 2: JavaScript có phân biệt hoa thường hay không?

- a) Có
- b) Không

Câu 3: Các định danh nào được đặt đúng?

- a) 1_Name
- b) _Name
- c) name_1

Câu 4: Thẻ HTML nào cho phép đưa các mã JavaScript vào trang web?

- a) <jscript>
- b) <script>
- c) <JavaScript>
- d) <scripting>

Câu 5: Trong JavaScript, đâu là mã để viết chuỗi "Xin chào" vào trang web?

- a) document.write("Xin chào");
- b) response.write("Xin chào");
- c) windows.write("Xin chào");
- d) writeln("Xin chào");

Câu 6: Làm thế nào để viết thông điệp "Xin chào" lên hộp thoại của trình duyệt?

- a) msgBox("Xin chào")
- b) alert("Xin chào")
- c) alertBox="Xin chào"
- d) alertBox("Xin chào")

Câu 7: Đây là các cú pháp đúng để viết chú thích trong JavaScript?

- a) Đây là chú thích
- b) <!-- Đây là chú thích -->
- c) /* Đây là chú thích */
- d) // Đây là chú thích

Câu 8: Đây là dòng lệnh đúng dùng để khai báo một biến trong JavaScript?

- a) char[] name;
- b) let name;
- c) String name;
- d) let \$name;

Câu 9: Biểu thức "1"+2+4 cho kết quả là gì?

- a) "16"
- b) Undefined
- c) "124"
- d) 7

Câu 10: Biểu thức 2+5+"8" cho kết quả là gì?

- a) "258"
- b) 15

- c) "78"
- d) undefined

Đáp án: Câu 1: c; Câu 2: b; Câu 3: b, c; Câu 4: b; Câu 5: a; Câu 6: b; Câu 7: c, d; Câu 8: b, d; Câu 9: c; Câu 10: c.

16. Tổng kết

- Biến là một khái niệm được dùng để đại diện cho một giá trị được lưu trữ trong bộ nhớ của máy tính
- Khi khai báo biến thì cần xác định tên của biến
- Tên của biến cần phải tuân thủ các quy định của từng ngôn ngữ lập trình
- Sau khi khai báo biến thì có thể gán giá trị cho biến
- Dữ liệu trong máy tính được chia thành nhiều kiểu khác nhau
- Một số kiểu dữ liệu thông dụng bao gồm: number, string, boolean, object...
- Có thể thực hiện các phép toán khác nhau trên các giá trị hoặc các biến
- Có các loại toán tử thông dụng như: Toán học, Logic, So sánh
- Các toán tử có độ ưu tiên nhất định, có thể thay đổi độ ưu tiên bằng cách sử dụng dấu ngoặc ()
- Có một vài cách khác nhau để đọc dữ liệu từ bên ngoài, chẳng hạn sử dụng hàm *prompt()* hoặc đọc dữ liệu từ thẻ *input*
- Có một vài cách khác nhau để hiển thị dữ liệu ra bên ngoài, chẳng hạn như *alert()*, *document.write()* hoặc *console.log()*

Chương 3 - Câu lệnh điều kiện

Điều khiển luồng thực thi của chương trình dựa vào một điều kiện nhất định

1. Mục tiêu

- Mô tả được các tình huống cần sử dụng câu lệnh điều kiện
- Trình bày được cú pháp của câu lệnh if-else
- Sử dụng được câu lệnh if để lựa chọn thực thi một khối lệnh
- Sử dụng được câu lệnh if-else để lựa chọn thực thi một trong hai khối lệnh
- Sử dụng được câu lệnh if lồng nhau để lựa chọn dựa trên nhiều điều kiện bao trùm lên nhau
- Sử dụng được câu lệnh if bậc thang để lựa chọn dựa trên nhiều điều kiện liên tiếp nhau
- Trình bày được cú pháp của câu lệnh switch-case
- Sử dụng được câu lệnh switch-case để lựa chọn dựa trên so sánh Bằng trong trường hợp có nhiều nhánh
- Lựa chọn được cấu trúc điều kiện phù hợp để sử dụng trong từng tình huống

2. Giới thiệu

Trong cuộc sống, chúng ta luôn phải đưa ra các quyết định khác nhau dựa trên việc đánh giá các tình huống hiện tại. Chẳng hạn, nếu đi đường gặp đèn đỏ thì chúng ta phải dừng chờ, gặp đèn xanh thì chúng ta đi tiếp. Nếu trời nắng thì chúng ta sẽ tìm các món ăn mát và bù nước, nếu trời lạnh thì chúng ta tìm các món ăn nóng ấm và giữ nhiệt, v.v.

Trong lập trình cũng vậy, chúng ta thường phải thay đổi luồng thực thi của một chương trình dựa trên các điều kiện hiện tại, chẳng hạn khi chúng ta đăng nhập vào một trang web, nếu mật khẩu nhập vào là đúng thì chúng ta sẽ được chuyển đến trang quản trị, còn nếu sai thì chúng ta sẽ phải đăng nhập lại.

Để làm được điều này, chúng ta sử dụng các cấu trúc trong lập trình được gọi là cấu trúc điều kiện, hay còn có một tên gọi khác là cấu trúc lựa chọn.

Hoàn thành chương này, chúng ta sẽ có thể viết được các ứng dụng phần mềm trong đó sẽ thực hiện một số hành động khác nhau dựa trên việc đánh giá các điều kiện.

3. Cấu trúc điều kiện

Các câu lệnh điều khiển

Một chương trình phần mềm thực thi các câu lệnh theo trật tự từ trên xuống dưới. Khi đó chúng ta có thể thay đổi luồng thực thi của một chương trình bằng cách sử dụng các câu lệnh điều khiển luồng (control flow statement).

Các câu lệnh điều khiển của JavaScript:

- Câu lệnh điều kiện (conditional statement)
- Câu lệnh lặp (Loop statement)
- Câu lệnh nhảy (jump statement)

Câu lệnh điều kiện

Trong khi viết mã lệnh, chúng ta muốn thực hiện các hành động khác nhau đối với các quyết định (điều kiện) khác nhau. Khi đó chúng ta có thể sử dụng các lệnh điều kiện trong mã của mình để làm điều này.

Câu lệnh điều kiện còn được gọi là câu lệnh ra quyết định (decision making). Câu lệnh điều kiện cho phép thay đổi luồng thực thi của chương trình. Việc lựa chọn thực thi một khối lệnh dựa trên việc đánh giá một điều kiện cho trước.

JavaScript hỗ trợ các câu lệnh điều kiện:

- Câu lệnh *if-else*: Được sử dụng trong trường hợp muốn đánh giá một biểu thức và rẽ sang 1 hoặc 2 nhánh khác nhau tùy thuộc vào giá trị của biểu thức đó.
- Câu lệnh *switch-case*: Được sử dụng trong trường hợp muốn đánh giá một biểu thức và rẽ sang 1 hoặc nhiều nhánh khác nhau tùy thuộc vào giá trị của biểu thức đó.

4. Cấu trúc điều kiện if-else

Câu lệnh if

Câu lệnh if được sử dụng để xem xét việc thực thi một thao tác nào đó dựa trên điều kiện hiện tại.

Cú pháp:

```
1. if (biểu_thức_điều_kiện) {
2.     khối_mã_được_thực_thi_nếu_điều_kiện_là_đúng
3. }
```

Câu lệnh *if* bao gồm một biểu thức điều kiện và một khối lệnh ở trong phần thân của nó. Tại thời điểm thực thi, nếu biểu thức điều kiện trả về kết quả *true* thì khối lệnh trong phần thân sẽ được thực thi, còn nếu biểu thức điều kiện trả về *false* thì khối lệnh trong phần thân sẽ được bỏ qua.

Ví dụ:

```
1. if (score > 9) {
2.     classification = "Xuất sắc";
```

```
3. }
```

Trong đoạn mã này biểu thức điều kiện là `score > 9`, còn phần thân là `classification = "Xuất sắc"`. Khi thực thi chương trình nếu biến `score` có giá trị lớn hơn 9 thì biến `classification` sẽ được gán giá trị là "Xuất sắc", còn nếu biến `score` có giá trị bằng hoặc nhỏ hơn 9 thì biến `classification` sẽ không được gán giá trị mới.

Lưu ý: Nếu phần thân chỉ có một câu lệnh duy nhất thì chúng ta có thể không cần viết dấu mở và đóng ngoặc, còn nếu trong trường hợp phần thân có nhiều hơn 1 câu lệnh thì chúng ta bắt buộc phải mở và đóng ngoặc.

Câu lệnh if-else

Câu lệnh *if-else* là dạng đầy đủ của câu lệnh *if*.

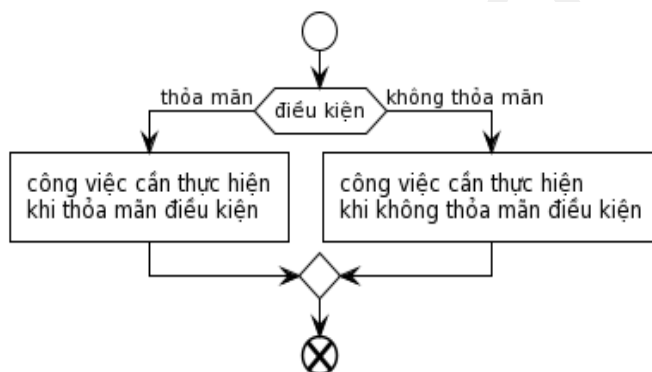
Cú pháp:

```
1. if (biểu_thức_điều_kiện) {
2.   khối_mã_được_thực_thi_nếu_điều_kiện_là_đúng
3. } else {
4.   khối_mã_được_thực_thi_nếu_điều_kiện_trên_là_sai
5. }
```

Câu lệnh *if-else* cũng bao gồm một biểu thức điều kiện, nhưng lại có hai phần thân khác nhau. Một phần thân gắn với phần *if* và một phần thân gắn với phần *else*.

Nếu biểu thức điều kiện trả về *true* thì phần thân gắn với *if* được thực thi, còn nếu biểu thức điều kiện trả về *false* thì phần thân gắn với phần *else* được thực thi.

Luồng thực thi của câu lệnh *if-else* có thể được mô tả như lưu đồ dưới đây:



Hình 3.1: Luồng thực thi của câu lệnh *if-else*

Ví dụ 1:

```
1. if (hour < 18){
2.   greeting = "Good day";
3. } else {
4.   greeting = "Good evening";
5. }
```

Trong đoạn mã trên, biểu thức điều kiện là `hour < 18`, còn phần thân của *if* là `greeting = "Good day"` và phần thân của *else* là `greeting = "Good evening"`.

Khi thực thi, nếu biến *hour* có giá trị nhỏ hơn 18 thì biến *greeting* sẽ được gán giá trị là “Good day”, còn nếu biến *hour* có giá trị lớn hơn hoặc bằng 18 thì biến *greeting* sẽ được gán giá trị là “Good evening”.

Ví dụ 2:

```
1. if (number % 2 == 0){
2.   alert(number + " is even.");
3. } else {
4.   alert(number + " is odd.");
5. }
```

Trong ví dụ trên, chúng ta đánh giá một số nguyên và phân loại số đó là số chẵn hay là số lẻ.

Trong trường hợp này, biểu thức điều kiện là để đánh giá xem số hiện tại có chia hết cho 2 hay không, còn hai phần thân là hiển thị thông báo tương ứng.

Việc đánh giá chia hết cho 2 hay không được thực hiện thông qua phép chia lấy phần dư và sau đó so sánh với giá trị 0. Nếu phần dư bằng không thì chứng tỏ đây là số chẵn, còn nếu phần dư khác 0 thì số đó là số lẻ.

Chẳng hạn, nếu *number* có giá trị là 4 thì kết quả phép chia cho 2 lấy phần dư là 0, do vậy biểu thức điều kiện sẽ trả về đúng, và phần thân của *if* được thực thi.

Nếu *number* có giá trị là 5 thì kết quả phép chia cho 2 lấy phần dư là 1, do vậy biểu thức điều kiện trả về sai, phần thân của *else* được thực thi.

if-else với một dòng lệnh bên trong

Trong trường hợp chỉ có một dòng lệnh bên trong *if* hoặc *else* thì có thể bỏ dấu ngoặc.

Ví dụ:

```
1. if(number % 2 == 0)
2.   alert(number + " is even.");
3. else
4.   alert(number + " is odd.");
```

5. Cấu trúc điều kiện if-else lồng nhau

Ngoài việc sử dụng câu lệnh *if-else* riêng lẻ, chúng ta cũng có thể đặt câu lệnh *if-else* này trong câu lệnh *if-else* khác. Điều này xảy ra khi chúng ta muốn đánh giá nhiều biểu thức điều kiện trước khi thực thi một thao tác nào đó.

Cú pháp:

```
1. if (điều kiện 1) {
2.   //khối lệnh thực thi nếu điều kiện 1 đúng
3.   if (điều kiện 2) {
4.     //khối lệnh thực thi nếu điều kiện 2 đúng
5.   } else {
6.     //khối lệnh thực thi nếu điều kiện 2 sai
7.   }
8. } else {
9.   //khối lệnh thực thi nếu điều kiện 1 sai
10. }
```

Ví dụ:

```

1. if (a > b) {
2.   if (a > c) {
3.     console.log("Số lớn nhất là a = " + a);
4.   } else {
5.     console.log("Số lớn nhất là c = " + c);
6.   }
7. } else {
8.   if (b > c) {
9.     console.log("Số lớn nhất là b = " + b);
10.  } else {
11.    console.log("Số lớn nhất là c = " + c);
12.  }
13. }

```

Trong ví dụ này, chúng ta so sánh các giá trị của 3 biến là a , b và c để tìm ra giá trị lớn nhất trong 3 số đó. Chúng ta cần đặt các câu lệnh điều kiện bên trong các câu lệnh điều kiện khác.

Chẳng hạn, với trường hợp $a = 5$, $b = 4$, $c = 3$ thì biểu thức $a > b$ sẽ trả về kết quả *true*, do đó phần thân của lệnh *if* này được thực thi, phần thân của lệnh *else* tương ứng sẽ được bỏ qua. Sau đó, biểu thức $a > c$ cũng trả về kết quả *true*, do đó phần thân của câu lệnh *if* này cũng sẽ được thực thi, và phần thân của câu lệnh *else* tương ứng sẽ được bỏ qua. Như vậy, chuỗi in ra sẽ là “Số lớn nhất là $a = 5$ ”.

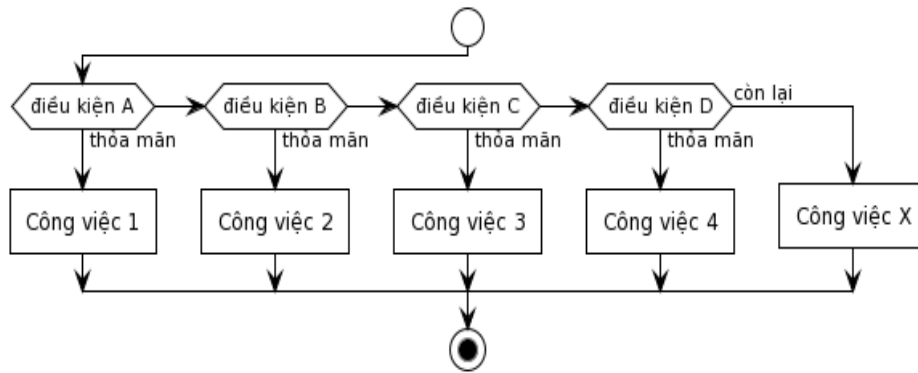
Hỏi nhanh: Hãy thử suy luận để xem luồng thực thi của đoạn mã trên sẽ là như thế nào trong trường hợp $a = 3$, $b = 4$, $c = 5$.

6. Cấu trúc điều kiện if-else bậc thang

Khi chúng ta kết hợp nhiều câu lệnh *if-else* liên tiếp nhau theo dạng *if-else-if* thì chúng ta được một dạng mới thường được gọi là câu lệnh điều kiện *if-else bậc thang*. Gọi là bậc thang bởi vì khi viết chúng được trình bày dạng như các bậc thang hướng từ trên xuống dưới.

Các khối lệnh điều kiện được đặt trong các câu lệnh *if* nối tiếp nhau, nếu một điều kiện nào đó thỏa mãn thì khối lệnh trong thân câu lệnh *if* sẽ được thực thi, còn nếu điều kiện sai thì sẽ chuyển sang đánh giá điều kiện *if* tiếp theo cho đến khi kết thúc.

Luồng thực thi của khối lệnh *if-else* bậc thang có thể mô tả như lưu đồ sau:



Hình 3.2: Luồng thực thi của câu lệnh if-else bậc thang

Cú pháp:

```

1. if (điều kiện 1) {
2.     //khối mã được thực thi nếu điều kiện 1 là đúng
3. } else if (điều kiện 2) {
4.     //khối mã được thực thi nếu điều kiện 1 là sai và điều kiện 2 là đúng
5. } else {
6.     //khối mã được thực thi nếu điều kiện 1 và điều kiện 2 đều sai
7. }
  
```

Ví dụ:

```

1. if (time < 10){
2.     greeting = "Good morning";
3. } else if (time < 20){
4.     greeting = "Good day";
5. } else {
6.     greeting = "Good evening";
7. }
  
```

Trong ví dụ này, nếu giá trị của *time* là 9 thì điều kiện đầu tiên đúng, do đó biến *greeting* sẽ có giá trị là “Good morning”.

Nếu giá trị của *time* là 11 thì điều kiện đầu tiên không đúng, do đó chuyển sang đánh giá điều kiện của câu lệnh *if* tiếp theo, ở đây điều kiện *time < 20* là đúng, do đó biến *greeting* sẽ được gán giá trị là “Good day”.

Nếu giá trị của biến *time* là 21 thì điều kiện đầu tiên không đúng, chuyển sang đánh giá điều kiện thứ hai cũng không đúng, do đó chuyển đến thực thi câu lệnh *else* cuối cùng, và kết quả biến *greeting* sẽ được gán giá trị là “Good evening”.

7. Cấu trúc điều kiện switch-case

Câu lệnh *switch-case* là một bộ lựa chọn đa hướng, nó so sánh giá trị của một biểu thức với một danh sách các hằng số nguyên hoặc hằng ký tự. Chúng ta sử dụng câu lệnh *switch-case* trong những trường hợp muốn phân loại luồng thực thi của chương trình dựa vào các điều kiện so sánh bằng. Khi biểu thức so sánh bằng trả về đúng thì khối lệnh tương ứng được thực thi.

Cú pháp:

```

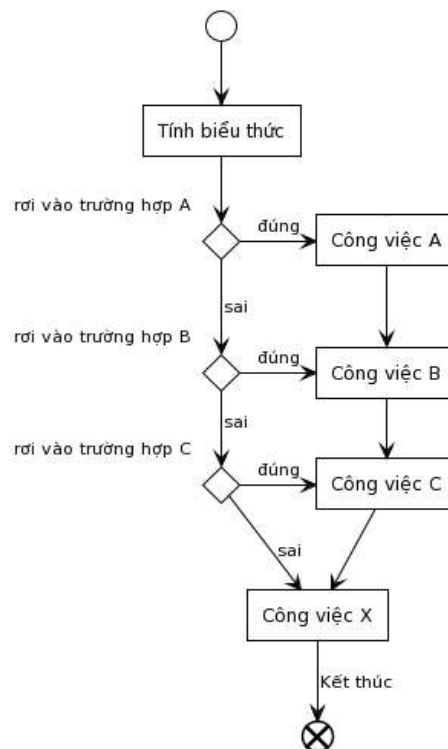
1. switch (param) {
2.   case value1:
3.     //khối lệnh 1
4.   case value2:
5.     //khối lệnh 2
6.   case value3:
7.     //khối lệnh 3
8.   default:
9.     //Khối lệnh mặc định sẽ được thực thi nếu không chọn được khối lệnh nào khác
10. }

```

Trong đó *param* là một biểu thức. Trình thực thi sẽ so sánh giá trị của *param* lần lượt với giá trị đi theo sau các *case*. Nếu gặp một *case* mang theo giá trị bằng với giá trị của *param* thì các khối lệnh đi sau đó sẽ được thực thi.

Trong trường hợp không tìm thấy *case* nào mang giá trị khớp với *param* thì khối lệnh đi theo sau *default* sẽ được thực thi. *default* là một khối lệnh không bắt buộc.

Lưu ý: Cho dù là thực thi mã đi theo *case* hay *default*, lệnh *switch* sẽ còn tiếp tục thực thi mã của tất cả các *case* (hay cả mã của *default*) đứng sau đó, mà không cần xem xét bất cứ điều kiện gì. Quá trình đó có thể được biểu diễn bằng lưu đồ như sau:



Hình 3.3: Luồng thực thi của câu lệnh switch-case

Chúng ta có thể thấy luồng thực thi nối thẳng từ khối lệnh này tới khối lệnh khác mà không thông qua bất kỳ khối hình thoi (khối xét điều kiện) nào. Vì hình ảnh này mà cách thực thi này được gọi là *fall-through* (rơi xuyên qua) trong tiếng Anh.

Ví dụ:

```

1. let switcher = 1;
2. switch (switcher) {
3. case 0:
4.   alert("Đèn sáng");
5. case 1:
6.   alert("Đèn tắt");
7. default:
8.   alert("Không tìm thấy công tắc");
9. }

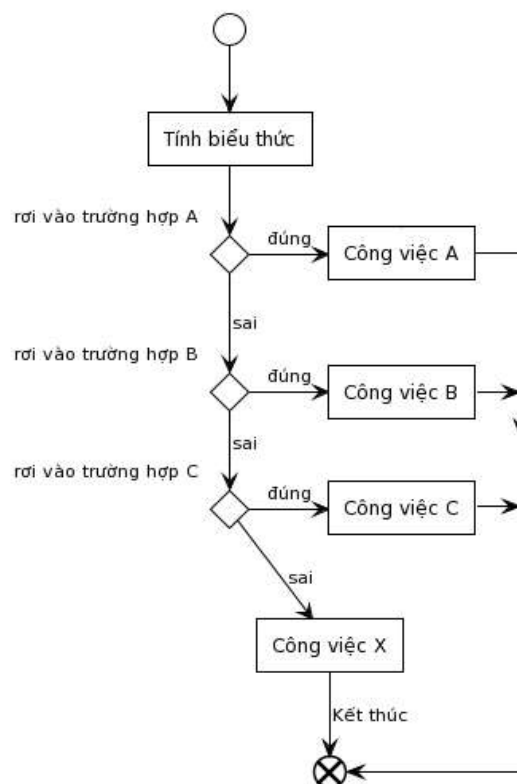
```

Mặc dù chỉ có một case mang giá trị khớp với *switcher*, trình duyệt sẽ thực thi mã của cả case 1 lẫn *default*, và kết quả là trình duyệt sẽ hiển thị 2 thông báo alert thay vì chỉ 1.

Từ khóa break

Chúng ta có thể ứng dụng sự đặc biệt trong cách thực thi của lệnh *switch* để dùng chung mã giữa các case, nhờ đó tạo ra được những lệnh *switch* ngắn gọn. Tuy nhiên để khắc phục hiện tượng luồng thực thi rơi xuyên qua tất cả các case phía sau, thay vì chỉ thực thi những mã cần thiết, chúng ta cần vận dụng từ khóa *break*.

Từ khóa *break* sẽ làm luồng thực thi ngay lập tức thoát ra khỏi câu lệnh *switch* hiện tại.



Hình 3.3: Luồng thực thi của câu lệnh *switch-case* khi có *break* sau mỗi case

Từ khóa *break* thường được đặt làm câu lệnh cuối cùng của mỗi case, nhờ thế, hiện tượng fall-through được khắc phục.

Ví dụ:

```

1. let switcher = 0;
2. switch (switcher) {
3.   case 0:
4.     alert("Đèn sáng");
5.     break;
6.   case null:
7.   case undefined:
8.     alert("Không tìm thấy công tắc");
9.   case 1:
10.  default:
11.    alert("Đèn tắt");
12. }

```

Nhờ có *break*, giờ đây nếu *switcher* có giá trị là 0 thì chỉ có alert “Đèn sáng” hiện lên. Nếu *switcher* mang giá trị là 1 thì alert “Đèn tắt” vẫn hiện lên. Nếu chúng ta thực thi đoạn mã trên với giá trị *switcher* là *null* hay *undefined*, sẽ có 2 alert hiện lên.

Lưu ý:

Lệnh *switch* khớp giá trị của *param* với các *case* bằng toán tử `===` chứ không phải `==`.

Ví dụ:

```

1. let switcher = "0";
2. switch (switcher) {
3.   case 0:
4.     alert("Đèn sáng");
5.     break;
6.   case null:
7.   case undefined:
8.     alert("Không tìm thấy công tắc");
9.   case 1:
10.  default:
11.    alert("Đèn tắt");
12. }

```

Đoạn mã trên sẽ khớp giá trị của *switcher* với *case default* thay vì *case 0* như chúng ta thường nghĩ.

8. Mã sạch

Quy tắc #1: Quy ước

- Trong các câu lệnh điều kiện, phía sau các từ khóa *if*, *else*, *switch* sẽ là khoảng trắng và tiếp theo là biểu thức điều kiện, kết thúc dòng đầu tiên sẽ là ký tự `{` để đánh dấu bắt đầu khối mã lệnh nằm bên trong câu lệnh điều kiện đó, trước dấu `{` sẽ là một khoảng trắng.
- Khối mã lệnh thực thi bên trong câu lệnh điều kiện phải lùi đầu dòng vào bốn khoảng trắng.
- Ký tự `}` đánh dấu kết thúc khối mã lệnh nằm trong câu điều kiện phải được viết riêng biệt tại dòng cuối cùng

Ví dụ:

Không nên:

```
1. if(number > 0)
2. { number += 1;
3.   alert('The number is' + number);}
```

Nên:

```
1. if (number > 0) {
2.   number += 1;
3.   alert('The number is' + number);
4. }
```

Quy tắc #2: Viết có ý nghĩa

Viết mã nguồn cũng như viết một bài văn, chúng ta cần viết thế nào để cho người đọc mã nguồn dễ dàng hiểu được ý nghĩa của từng dòng mã. Một cách đơn giản để đạt được tính dễ hiểu đối với các biểu thức như trên đó là "tách biến": Đưa biểu thức của chúng ta ra bên ngoài khối lệnh *if* và gán giá trị của nó cho một biến (với tên gọi có ý nghĩa).

Ví dụ:

Trước khi tách biến:

```
1. if (number % 2 == 0){
2.   alert(number + " is even.");
3. } else {
4.   alert(number + " is odd.");
5. }
6.
```

Sau khi tách biến:

```
1. let isEven = number % 2 == 0;
2. if (isEven) {
3.   alert(number + " is even.");
4. } else {
5.   alert(number + " is odd.");
6. }
```

9. Các lỗi thường gặp**Lỗi thường gặp #1: Quên dấu ngoặc**

Ví dụ:

```
1. if (radius >= 0)
2.   area = radius * radius * PI;
3.   alert("The area is " + area); // Luôn được thực thi, không phụ thuộc vào điều kiện
```

Trong đoạn mã bên trên, chúng ta muốn thực thi 2 câu lệnh ở trong điều kiện *if*, tuy nhiên lại quên mất việc đặt chúng vào trong một khối đóng mở ngoặc. Như vậy, khi

điều kiện đúng thì chỉ có một câu lệnh ở trên được thực thi, dòng lệnh `alert("The area is" + alert)` sẽ luôn được thực thi mà không bị phụ thuộc vào điều kiện (`radius >= 0`). Trong trường hợp này chúng ta cần phải sửa lại, bổ sung cặp dấu `{ }` vào vị trí phù hợp.

Sau khi bổ sung cặp dấu ngoặc `{}`:

```
1. if (radius >= 0) {
2.   area = radius * radius * PI;
3.   alert("The area is " + area);
4. }
```

Lỗi thường gặp #2: Đặt dấu ; ở cuối câu lệnh điều kiện

Lỗi thứ hai đó là đặt thừa dấu chấm phẩy sau điều kiện của `if`. Trong trường hợp này câu lệnh `if` không có tác dụng bởi vì nó kết thúc ngay sau khi đánh giá điều kiện. Có nghĩa là khối lệnh đằng sau nó sẽ luôn được thực thi cho dù điều kiện ở trước có đúng hay không.

Ví dụ:

```
1. if (radius >= 0); {
2.   area = radius * radius * PI;
3.   alert("The area is " + area);
4. }
```

Để khắc phục lỗi này, chúng ta chỉ cần xóa dấu chấm phẩy ở cuối câu lệnh `if` đi. Xóa dấu `(;)` ở cuối câu lệnh `if`:

```
1. if (radius >= 0) {
2.   area = radius * radius * PI;
3.   alert("The area is " + area);
4. }
```

Lỗi thường gặp #3: So sánh giá trị boolean

Lỗi thứ ba xảy ra khi chúng ta so sánh các giá trị thuộc kiểu dữ liệu boolean. Bản chất thì đây không phải là một lỗi, bởi vì các dòng lệnh vẫn thực thi đúng như mong đợi. Nhưng việc so sánh giá trị kiểu boolean như trong trường hợp này là không cần thiết.

Không nên:

```
1. if (even == true) {
2.   alert("It is even.");
3. }
```

Chúng ta sẽ dễ dàng thấy rằng kết quả của biểu thức so sánh `even = true` là hoàn toàn giống với giá trị của biến `even`. Nếu `even` có giá trị `true` thì biểu thức sẽ có giá trị `true`. Nếu `even` có giá trị là `false` thì biểu thức sẽ có giá trị là `false`. Như vậy thì việc sử dụng biểu thức ở đây là không cần thiết.

Giá trị của biến even	Kết quả của biểu thức (even == true)
true	true
false	false

Nên:

```
1. if (even) {
2.   alert("It is even.");
3. }
```

Lỗi thứ ba xảy ra khi chúng ta so sánh các giá trị thuộc kiểu dữ liệu boolean. Bản chất thì đây không phải là một lỗi, bởi vì các dòng lệnh vẫn thực thi đúng như mong đợi. Nhưng việc so sánh giá trị kiểu boolean như trong trường hợp này là không cần thiết. Chẳng hạn, chúng ta sẽ dễ dàng thấy rằng kết quả của biểu thức so sánh `even == true` là hoàn toàn giống với giá trị của biến `even`. Nếu `even` có giá trị `true` thì biểu thức sẽ có giá trị `true`. Nếu `even` có giá trị là `false` thì biểu thức sẽ có giá trị là `false`. Như vậy thì việc sử dụng biểu thức ở đây là không cần thiết.

Giá trị của biến even	Kết quả của biểu thức (even == true)
true	true
false	false

Lỗi thường gặp #4: Nhầm lẫn khối if-else

Lỗi thứ 4 đó là nhầm lẫn khi kết hợp các khối lệnh if-else, lỗi này thường xảy ra khi chúng ta quên các dấu đóng mở ngoặc hoặc nhầm lẫn do khoảng cách đầu dòng của các câu lệnh không được căn chỉnh cẩn thận.

Không nên:

```
1. if (first_number > second_number)
2.   if (first_number > third_number)
3.     alert("A");
4. else
5.   alert("B");
6.
```

Ở ví dụ trên, khi nhìn vào đoạn mã thứ nhất, chúng ta thường nghĩ rằng khối lệnh `else` được áp dụng cho câu lệnh điều kiện đầu tiên. Có nghĩa là nếu `i` nhỏ hơn hoặc bằng `j` thì sẽ hiển thị thông báo “B”.

Tuy nhiên thực tế thì không phải như thế. Câu lệnh `else` được áp dụng cho điều kiện if thứ hai ở bên trong câu lệnh if đầu tiên. Do đó, điều kiện `else` này chỉ được thực thi nếu điều kiện đầu tiên đúng và điều kiện thứ hai sai, tức là `i` lớn hơn `j` và `i` nhỏ hơn `k`.

Nên:

```

1. if (first_number > second_number) {
2.   if (first_number > third_number) {
3.     alert("A");
4.   } else {
5.     alert("B");
6.   }
7. }

```

Lỗi thường gặp #5: Gán giá trị boolean

Lỗi thường gặp thứ 5 là thực hiện việc so sánh để gán các giá trị tương ứng cho một biến. Đây thực chất cũng không phải là một lỗi, bởi vì chương trình vẫn thực thi đúng như mong đợi, nhưng việc sử dụng câu lệnh điều kiện trong trường hợp này là không cần thiết.

Không nên:

```

1. if (number % 2 === 0) {
2.   even = true;
3. } else {
4.   even = false;
5. }
6.

```

Lỗi thường gặp thứ 5 là thực hiện việc so sánh để gán các giá trị tương ứng cho một biến. Đây thực chất cũng không phải là một lỗi, bởi vì chương trình vẫn thực thi đúng như mong đợi, nhưng việc sử dụng câu lệnh điều kiện trong trường hợp này là không cần thiết.

Chẳng hạn, trong đoạn mã này, giá trị của biến `even` hoàn toàn tương ứng với giá trị của biểu thức so sánh kết quả phép chia lấy số dư và giá trị 0. Do đó, thay vì sử dụng `if-else` để gán giá trị thì chúng ta hoàn toàn có thể gán trực tiếp kết quả của biểu thức cho biến `even`.

Giá trị của biến <code>even</code>	Kết quả của biểu thức (<code>number % 2 === 0</code>)
true	true
false	false

Trong đoạn mã trên, giá trị của biến `even` hoàn toàn tương ứng với giá trị của biểu thức so sánh kết quả phép chia lấy số dư và giá trị 0. Do đó, thay vì sử dụng `if-else` để gán giá trị thì chúng ta hoàn toàn có thể gán trực tiếp kết quả của biểu thức cho biến `even`.

Giá trị của biến even	Kết quả của biểu thức (number % 2 === 0)
true	true
false	false

Nên:

```
1. let even = number % 2 === 0;
```

Lỗi thường gặp #6: Lặp mã

Lỗi thường gặp cuối cùng đó là việc lặp mã. Đây thực ra cũng không phải là một lỗi, bởi vì chương trình vẫn thực thi đúng như mong đợi nhưng việc mã bị lặp lại là một điều tối kỵ trong lập trình.

Ví dụ:

```
1. //Không nên
2. if (inState) {
3.   tuition = 5000;
4.   alert("The tuition is " + tuition);
5. } else {
6.   tuition = 15000;
7.   alert("The tuition is " + tuition);
8. }
```

Trong trường hợp này, hai câu lệnh `alert()` y hệt nhau cùng xuất hiện trong một đoạn mã ngắn. Chúng ta có thể dễ dàng xử lý bằng cách đơn giản là đưa câu lệnh đó ra cuối khối `if-else`. Như vậy thì câu lệnh `alert()` này được thực thi cả trong hai trường hợp là biến `inState` có giá trị là `true` hoặc `false`. Nghĩa là hành vi của chương trình không bị thay đổi.

Nên:

```
1. if (inState) {
2.   tuition = 5000;
3. } else {
4.   tuition = 15000;
5. }
6. alert("The tuition is " + tuition);
```

10. Bài thực hành

Bài 1: Kiểm tra năm nhuận

Mục tiêu:

Luyện tập sử dụng cấu trúc điều kiện `if`.

Mô tả:

Trong phần này, chúng ta sẽ phát triển một ứng dụng nhằm kiểm tra xem một năm có phải là năm nhuận hay không. Ứng dụng cho phép người dùng nhập vào một năm, sau đó sẽ đưa ra thông báo là năm đó là năm nhuận hay không phải là năm nhuận. Năm nhuận là một năm đặc biệt, được cộng thêm một ngày để giữ cho lịch được đồng bộ với lịch thiên văn.

Cách xác định năm nhuận: Những năm chia hết cho 4 là năm nhuận, ngoại trừ những năm chia hết cho 100 mà không chia hết cho 400. Từ đó, có thể rút gọn thành các quy tắc xác định năm nhuận: “Những năm chia hết cho 4 mà không chia hết cho 100 là năm nhuận. Những năm chia hết cho 100 mà không chia hết cho 400 thì KHÔNG PHẢI là năm nhuận. Những năm chia hết đồng thời cho 100 và 400 là năm nhuận”.

Hướng dẫn:

Bước 1: Tạo dự án mới với file index.html và khối lệnh JavaScript

Bước 2: Nhập dữ liệu từ bàn phím

```
1. let year = parseInt(prompt("Enter a year"));
```

Bước 3: Phân loại và hiển thị kết quả

```
1. if (year % 4 == 0) {
2.   if (year % 100 == 0) {
3.     if (year % 400 == 0) {
4.       alert(year + " is a leap year");
5.     } else {
6.       alert(year + " is NOT a leap year");
7.     }
8.   } else {
9.     alert(year + " is a leap year");
10.  }
11. } else {
12.   alert(year + " is NOT a leap year");
13. }
```

Bước 4: Chạy và quan sát kết quả.

Sử dụng các năm sau để kiểm tra tính đúng đắn của chương trình:

Năm	Kết quả
12	Năm nhuận
13	Năm không nhuận
1000	Năm không nhuận
2000	Năm nhuận

Lưu ý:

Với các điều kiện như trên, chúng ta có thể chỉnh sửa để mã nguồn tốt hơn như sau:

```

1. let year = parseInt(prompt("Enter a year"));
2. let isLeapYear = false;
3. if (year % 4 == 0) {
4.   if (year % 100 == 0) {
5.     if (year % 400 == 0) {
6.       isLeapYear = true;
7.     }
8.   } else {
9.     isLeapYear = true;
10.  }
11. }
12. if (isLeapYear) {
13.   alert(year + " is a leap year");
14. } else {
15.   alert(year + " is NOT a leap year");
16. }

```

Đảm bảo clean code:

Trong đoạn mã trên, các điều kiện bên trong các câu lệnh if sẽ là khó hiểu, bởi vì bản thân các biểu thức không trực tiếp nói rõ ý nghĩa của nó. Chúng ta có thể sử dụng kỹ thuật tách biến để các câu lệnh này dễ hiểu hơn. Ví dụ:

```

1. let isLeapYear = false;
2.
3. let isDivisibleBy4 = year % 4 == 0;
4. if (isDivisibleBy4) {
5.   let isDivisibleBy100 = year % 100 == 0;
6.   if (isDivisibleBy100) {
7.     let isDivisibleBy400 = year % 400 == 0;
8.     if (isDivisibleBy400) {
9.       isLeapYear = true;
10.    }
11.  } else {
12.    isLeapYear = true;
13.  }
14. }
15.
16. if (isLeapYear) {
17.   alert(year + " is a leap year");
18. } else {
19.   alert(year + " is NOT a leap year");
20. }

```

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/01-kiem-tra-nam-nhuan>

Bài 2: Luyện tập với cấu trúc if...else

Mục tiêu:

Luyện tập sử dụng cấu trúc if/else.

Mô tả:

Sử dụng cấu trúc if...else, viết chương trình nhập vào một chuỗi câu trả lời cho câu hỏi "What is the "official" name of JavaScript?". Nếu câu trả lời nhập vào là

“ECMAScript”, thì hiển thị thông báo: “Right!”, ngược lại hiển thị thông báo: “Didn’t know? ECMAScript!”

Hướng dẫn:

Bước 1: Tạo file index.html

Bước 2: Thêm thẻ <script> thực hiện viết các mã JavaScript

Bước 3: Khai báo biến value nhận giá trị được nhập vào từ hộp thoại

```
1. let value = prompt("What is the 'official' name of JavaScript?", "");
```

Bước 4: Sử dụng cấu trúc if...else kiểm tra giá trị của value

```
1. if (value == 'ECMAScript') {
2.   alert("Right!");
3. } else {
4.   alert("You don't know? ECMAScript!");
5. }
```

Bước 5: Thực thi chương trình. Quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/02-luyen-tap-voi-cau-truc-if-else>

Bài 3: Luyện tập với cấu trúc if...else...if

Mục tiêu

Luyện tập sử dụng cấu trúc if...else...if

Mô tả

Viết chương trình kiểm tra đăng nhập hệ thống của người dùng theo yêu cầu sau:

- Yêu cầu người dùng nhập tên từ bàn phím.
- Nếu tên nhập vào là “Admin” thì yêu cầu nhập mật khẩu.
- Nếu mật khẩu là “TheMaster” thì in ra chuỗi “Welcome”.
- Nếu mật khẩu nhập vào là null in ra chuỗi “Canceled”.
- Còn lại in ra chuỗi “Wrong password”.
- Nếu tên nhập vào là null in ra chuỗi “Canceled”.
- Còn lại in ra chuỗi “I don’t know you”.

Hướng dẫn

Bước 1: Tạo file login.html

Bước 2: Thêm thẻ <script> thực hiện viết các mã JavaScript

Bước 3: Tạo biến userName nhận giá trị nhập vào từ hộp thoại

```
1. let userName = prompt("Who's there?", "");
```

Bước 4: Kiểm tra giá trị của userName

```
1. if (userName == 'Admin') {
2.   //code vào đây
3. } else if (userName == null) {
4.   alert("Canceled");
5. } else {
6.   alert("I don't know you");
7. }
```

Bước 5: Nếu userName là "Admin". Khai báo biến pass nhận giá trị nhập vào từ hộp thoại.

```
1. let pass = prompt('Password?', "");
```

Bước 6: Kiểm thử

```
1. if (pass == 'TheMaster') {
2.   alert("Welcome!");
3. } else if (pass == null) {
4.   alert('Canceled. ');
5. } else {
6.   alert("Wrong password");
7. }
```

Bước 7: Thực thi chương trình. Quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/03-luyen-tap-voi-cau-truc-if-else-if>

Bài 4: Luyện tập với cấu trúc switch...case

Mục tiêu:

Luyện tập với cấu trúc switch...case

Mô tả:

Viết lại cấu trúc if sau thành cấu trúc switch...case

```
1. if (browser == 'Edge') {
2.   alert("You've got the Edge!");
3. } else if (browser == 'Chrome' || browser == 'Firefox' || browser == 'Safari' || browser == 'Opera') {
4.   alert('Okay we support these browsers too');
5. } else {
6.   alert("We hope that this page looks ok!");
7. }
```

Hướng dẫn

Cấu trúc if được viết lại với cấu trúc switch...case như sau:

```
1. switch (browser) {
2.   case 'Edge':
3.     alert("You've got the Edge!");
4.     break;
5.   case 'Chrome':
6.   case 'Firefox':
7.   case 'Safari':
8.   case 'Opera':
9.     alert('Okay we support these browsers too');
10.    break;
11.  default:
12.    alert("We hope that this page looks ok!");
13. }
```

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/04-luyen-tap-voi-cau-truc-switch-case>

Bài 5: Luyện tập với cấu trúc switch...case

Mục tiêu:

Luyện tập với cấu trúc switch...case.

Mô tả:

Viết lại cấu trúc if sau thành cấu trúc switch...case.

```
1. let a = parseInt(prompt('a?', " "));
2. if (a == 0) {
3.   alert(0);
4. }
5. if (a == 1) {
6.   alert(1);
7. }
8. if (a == 2 || a == 3) {
9.   alert('2,3');
10. }
```

Hướng dẫn:

Cấu trúc if được viết lại với cấu trúc switch...case như sau:

```
1. let a = parseInt(prompt('a?', " "));
2. switch (a) {
3.   case 0:
4.     alert(0);
5.     break;
6.   case 1:
7.     alert(1);
8.     break;
9.   case 2:
10.    case 3:
11.      alert('2,3');
12.      break;
13. }
```

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/05-luyen-tap-voi-cau-truc-switch-case>

Bài 6: Luyện tập với cấu trúc if...else

Mục tiêu:

Luyện tập với cấu trúc if...else.

Mô tả:

Sử dụng cấu trúc if...else, nhập vào một số, kiểm tra giá trị của số nhập vào.

- Nếu số nhập vào lớn hơn 0 hiển thị 1
- Nếu số nhập vào nhỏ hơn 0 thì hiển thị -1
- Còn lại hiển thị 0

Hướng dẫn:

Bước 1: Tạo file sign.html

Bước 2: Khai báo biến value nhận giá trị đầu vào từ hộp thoại, mặc định không nhập gì value = 0

```
1. let value = prompt('Type a number', 0);
```

Bước 3: Viết khối if...else..if

```
1. if (value > 0) {
2.   alert(1);
3. } else if (value < 0) {
4.   alert(-1);
5. } else {
6.   alert(0);
7. }
```

Bước 4: Thực thi chương trình. Nhập giá trị 0, -5, 5. Quan sát kết quả nhận được

Bước 5: Thay thế cấu trúc if bằng cấu trúc switch...case. Làm lại các bước như trên.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/06-luyen-tap-voi-cau-truc-if-else>

Bài 7: Tính tổng 2 số

Mục tiêu:

Luyện tập sử dụng cấu trúc điều kiện với toán tử "?", ":"

Mô tả:

Hãy viết chương trình nhập vào giá trị cho a và b. Tính tổng a và b, nếu tổng nhỏ hơn 4, hiển thị chuỗi Below, ngược lại hiển thị Over. Lưu ý sử dụng toán tử ? :

Hướng dẫn:

Bước 1: Tạo file add.html

Bước 2: Thêm thẻ <script> thực hiện viết các mã JavaScript

```
1. <!DOCTYPE html>
2. <html>
3. <body>
4.   <script>
5.     //code vào đây
6.   </script>
7. </body>
8. </html>
```

Bước 3: Khai báo biến a, b nhập vào giá trị cho a, b từ hộp thoại

```
1. let a = parseInt(prompt("a: "));
2. let b = parseInt(prompt("b: "));
```

Bước 4: Khai báo biến result lưu kết quả

```
1. let result = (a + b < 4) ? 'Below' : 'Over';
2. alert(result);
```

Bước 5: Thực thi chương trình. Quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/07-tinh-tong-2-so>

Bài 8: Luyện tập với chuỗi

Mục tiêu:

Luyện tập với chuỗi.

Mô tả:

Viết chương trình nhập vào giá trị cho chuỗi message.

- Nếu giá trị nhập vào là Employee thì hiển thị chuỗi Hello.
- Còn lại nếu giá trị nhập vào là Director thì hiển thị chuỗi Greetings
- Còn lại nếu giá trị nhập vào chuỗi rỗng thì hiển thị No login
- Còn lại hiển thị chuỗi rỗng

Sử dụng toán tử ? : để viết chương trình với cấu trúc điều kiện như sau:

```
1. let message;
2. if (login === 'Employee') {
3.   message = 'Hello';
4. } else if (login === 'Director') {
5.   message = 'Greetings';
6. } else if (login === "") {
7.   message = 'No login';
8. } else {
9.   message = "";
10. }
```

Hướng dẫn:

Bước 1: Tạo file *employee.html*

Bước 2: Thêm thẻ <script> thực hiện viết các mã JavaScript

```
1. <!DOCTYPE html>
2. <html>
3. <body>
4.   <script>
5.     //code vào đây
6.   </script>
7. </body>
8. </html>
9.
```

Bước 3: Khai báo biến message

```
1. let login = prompt("Login?", "");
2. let message = (login === "Employee") ?
3.   "Hello" :
4.   (login === "Director") ?
5.     "Greetings" :
6.     (login === "") ?
7.       "No login" :
8.       "";
```

Bước 4: Hiển thị thông báo

```
1. alert(message);
```

Bước 5: Thực thi chương trình. Quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/08-luyen-tap-voi-chuoi>

Bài 9: Sự kiện bàn phím

Mục tiêu:

Luyện tập xử lý sự kiện bàn phím.

Mô tả:

Di chuyển hình ảnh nhân vật Nobita lên, xuống, sang trái, sang phải sử dụng các phím tương ứng. Việc này được thực hiện bằng cách thay đổi thuộc tính tọa độ của thẻ khi xử lý event.



Hình 3.5: Minh họa sự kiện bàn phím

Hướng dẫn:

Bước 1: Chèn ảnh nobita.png vào tài liệu html

```
1. 
```

Bước 2: Xây dựng các hàm xử lý khi nhấn phím lên

```
1. function upArrowPressed() {  
2.   let element = document.getElementById("nobita");  
3.   element.style.top = parseInt(element.style.top) - 5 + 'px';  
4. }
```

Bước 3: Xây dựng các hàm xử lý khi nhấn phím xuống

```
1. function downArrowPressed() {  
2.   let element = document.getElementById("nobita");  
3.   element.style.top = parseInt(element.style.top) + 5 + 'px';  
4. }
```

Bước 4: Xây dựng các hàm xử lý khi nhấn phím sang trái

```
1. function leftArrowPressed() {  
2.   let element = document.getElementById("nobita");  
3.   element.style.left = parseInt(element.style.left) - 5 + 'px';  
4. }
```

Bước 5: Xây dựng các hàm xử lý khi nhấn phím sang phải

```

1. function rightArrowPressed() {
2.   let element = document.getElementById("nobita");
3.   element.style.left = parseInt(element.style.left) + 5 + 'px';
4. }

```

Bước 6: Xây dựng các hàm nhận các phím

```

1. function moveSelection(evt) {
2.   switch (evt.keyCode) {
3.     case 37:
4.       leftArrowPressed();
5.       break;
6.     case 39:
7.       rightArrowPressed();
8.       break;
9.     case 38:
10.      upArrowPressed();
11.      break;
12.     case 40:
13.      downArrowPressed();
14.      break;
15.   }
16. }
17.

```

Bước 7: Xây dựng hàm docReady()

```

1. function docReady() {
2.   window.addEventListener('keydown', moveSelection);
3. }

```

Bước 8: Gọi hàm docReady()

```

1. <body onload="docReady()">

```

Bước 9: Chạy chương trình. Sử dụng các phím lên-xuống-sang trái-sang phải để di chuyển ảnh và quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/09-su-kien-ban-phim>

Bài 10: Sự kiện chuột

Mục tiêu:

Luyện tập thao tác sự kiện chuột.

Mô tả:

Làm sử dụng sự kiện click chuột để di chuyển hình ảnh một quả bóng. Mỗi lần click quả bóng sẽ được di chuyển sang trái/phải một khoảng cách.

Hướng dẫn:

Bước 1: Tạo giao diện như hình



Click button below to move the image to right

Hình 3.6: Minh họa sự kiện chuột

```
1. <form>
2.   
3.   <p>Click button below to move the image to right</p>
4.   <input type="button" value="Click Me" />
5. </form>
```

Bước 2: Tạo biến imgObj để lưu ảnh

```
1. let imgObj = null;
```

Bước 3: Sử dụng hàm getElementById () để nhận được một đối tượng DOM và sau đó gán nó cho biến imgObj.

```
1. imgObj = document.getElementById('myImage');
```

Bước 4: Tạo hàm init () để khởi tạo imgObj mà chúng ta đã đặt vị trí và các thuộc tính còn lại.

```
1. function init() {
2.   imgObj = document.getElementById('myImage');
3.   imgObj.style.position = 'relative';
4.   imgObj.style.left = '0px';
5. }
```

Bước 5: Tạo hàm moveRight () để tăng khoảng cách trái 10 pixel. Bạn cũng có thể đặt nó là một giá trị âm để di chuyển nó sang bên trái.

```
1. function moveRight() {
2.   imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
3. }
```

Bước 6: Gọi hàm init() khi trình duyệt được tải lần đầu

```
1. window.onload = init;
```

Bước 7: Xác định sự kiện onclick cho button

```
1. <input type="button" value="Click Me" onclick="moveRight();" />
```

Bước 8: Chạy chương trình. Click chuột vào button Click Me và quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-03/10-su-kien-chuot>

11. Bài tập

Bài 1: Ứng dụng tính chỉ số cân nặng của cơ thể

Chỉ số khối cơ thể (Body mass index-BMI) là một thước đo sức khỏe dựa trên cân nặng và chiều cao. Nó được tính bằng cách lấy cân nặng đơn vị tính kilogam chia cho bình phương của chiều cao đơn vị tính mét. Công thức:

$bmi = weight / (height * 2)$

Chỉ số BMI đối với người trên 20 tuổi được phân loại và diễn giải theo bảng sau:

BMI	Diễn giải
$BMI < 18.5$	Thiếu cân
$18.5 \leq BMI < 25.0$	Bình thường
$25.0 \leq BMI < 30.0$	Thừa cân
$30.0 \leq BMI$	Mập

Ví dụ: Một người có cân nặng là 65Kg và chiều cao là 1.75m thì BMI là $65 / 1.75^2 = 22.22$. Chỉ số này nằm trong khoảng 18.5 đến 25.0 cho nên được phân loại là Bình thường.

Hãy viết một chương trình cho phép nhập vào chiều cao và cân nặng, sau đó đưa ra thông báo về chỉ số BMI và diễn giải nó.

Bài 2: Ứng dụng tính số ngày trong tháng

Hãy viết một ứng dụng cho phép tính được số ngày trong một tháng. Giao diện của ứng dụng tương tự như sau. Chương trình sẽ hiển thị số ngày của tháng tương ứng.

Tháng 1 có 31 ngày

Hình 3.7: Minh họa ứng dụng tính số ngày trong tháng

- Chương trình có ô nhập vào một số là số thứ tự của tháng trong năm
- Nhấn nút "Tính số ngày" thì sẽ hiển thị số ngày của tháng đó
- Lưu ý: Trong trường hợp tháng 2, kết quả sẽ là "Tháng 2 có 28 hoặc 29 ngày".

12. Bài kiểm tra

Câu 1: Xác định kết quả của biến number trong đoạn mã sau:

```
1. let number = 5;
2. if (number >= 5) {
3.   number += 1;
```

```
4. } else {
5.   number -= 1;
6. }
```

- a) 4
- b) 7
- c) 6
- d) 5

Câu 2: Trong JavaScript, cách nào sau đây sử dụng để kiểm tra điều kiện "number" bằng 5 trước khi thực thi các mã lệnh khác?

- a) if number = 5 then
- b) if number == 5 then
- c) if (number == 5)
- d) if (number = 5)

Câu 3: Trong JavaScript, cách nào sau đây sử dụng để kiểm tra điều kiện "number" khác 5 trước khi thực thi các mã lệnh khác?

- a) if number != 5 then
- b) if (number != 5)
- c) if (number > 5)
- d) if (number =! 5)

Câu 4: Xác định kết quả của biến number trong đoạn mã sau:

```
1. let number = 5;
2. if (number++ > 5) {
3.   number += 1;
4. } else {
5.   number -= 1;
6. }
```

- a. 6
- b. 7
- c. 4
- d. 5

Câu 5: JavaScript hỗ trợ những cấu trúc nào khi xử lý về điều kiện rẽ nhánh?

- a. if then else
- b. switch case
- c. when case
- d. if else

Câu 6: Xác định giá trị của biến message sau khi chạy đoạn mã sau:

```
1. let day = "Mon";
2. let message = "";
3. switch (day) {
4.   case "Mon":
5.     message = "Ngày đầu tuần";
6.   case "Wed":
7.     message = "Ngày giữa tuần";
```

```
8. case "Sat":  
9. case "Sun":  
10. message = "Ngày nghỉ";
```

- a) Xảy ra lỗi
- b) "Ngày nghỉ"
- c) "Ngày giữa tuần"
- d) "Ngày đầu tuần"

Đáp án: Câu 1: c, Câu 2: c, Câu 3: b, Câu 4: d, Câu 5: b, d, Câu 6: b

13. Tổng kết

- Câu lệnh điều kiện được sử dụng để lựa chọn việc thực thi một khối lệnh dựa vào một điều kiện cho trước
- Câu lệnh điều kiện còn được gọi là câu lệnh rẽ nhánh hoặc câu lệnh lựa chọn
- Có các câu lệnh điều kiện thông dụng là if-else và switch-case
- Có thể kết hợp các câu lệnh điều kiện if-else để hình thành dạng lồng nhau hoặc dạng bậc thang
- Câu lệnh switch case được sử dụng trong trường hợp muốn so sánh bằng và rẽ nhiều nhánh khác nhau
- Từ khoá break được dùng để ngắt luồng thực thi của một khối lệnh trong switch-case.

Chương 4 - Câu lệnh lặp

Thực hiện các thao tác lặp đi lặp lại nhiều lần dựa vào các điều kiện nhất định.

1. Mục tiêu

- Trình bày được ý nghĩa của vòng lặp
- Trình bày được cú pháp của vòng lặp for
- Sử dụng được vòng lặp for
- Trình bày được cú pháp của vòng lặp while
- Sử dụng được vòng lặp while
- Trình bày được cú pháp của vòng lặp do-while
- Sử dụng được vòng lặp do-while
- Sử dụng được vòng lặp lồng nhau
- Lựa chọn được vòng lặp phù hợp trong từng tình huống cụ thể

2. Giới thiệu

Vòng lặp là dạng cấu trúc cho phép tự động thực hiện một khối lệnh lặp đi lặp lại nhiều lần dựa vào một điều kiện cho trước. Vòng lặp giúp cho lập trình viên viết được các mã nguồn ngắn gọn hơn so với việc phải viết lặp lại những dòng mã tương tự nhau. Chẳng hạn, trong thực tế chúng ta có thể sử dụng vòng lặp để hiển thị danh sách khách hàng trong một ứng dụng bán hàng trực tuyến; hoặc hiển thị danh sách bạn bè trong một ứng dụng mạng xã hội; hoặc để tạo hiệu ứng chuyển động của nhân vật trong một trò chơi; hoặc để sao chép một file lớn từ nơi này sang nơi khác, v.v.

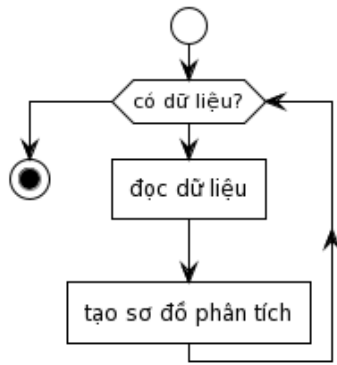
Chương này sẽ trình bày các dạng vòng lặp được sử dụng phổ biến và cách để nhận diện và lựa chọn được vòng lặp phù hợp với tình huống thực tế.

Hoàn thành chương này, chúng ta có thể tạo được các ứng dụng mà trong đó có nhiều thao tác được tự động thực thi lặp lại nhiều lần.

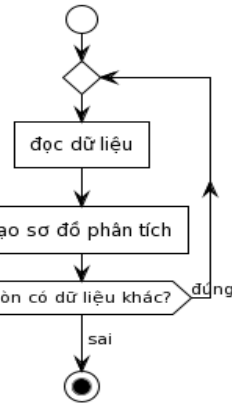
3. Câu lệnh lặp

Tại chương trước chúng ta đã tìm hiểu về các câu lệnh điều kiện. Đặc trưng của các câu lệnh này là dựa trên một điều kiện cho trước, luồng thực thi của chương trình sẽ rẽ nhánh sang các khối lệnh khác nhau.

Hãy xem xét hai lưu đồ dưới đây, khi gặp một điều kiện nào đó, luồng thực thi của chương trình được chuyển tới một vị trí nào đó trong quá khứ.



Hình 4.1



Hình 4.2

Trong cả 2 lưu đồ trên, chừng nào vẫn còn dữ liệu thì chương trình sẽ tiếp tục tạo thêm sơ đồ phân tích, chúng ta gọi đây là phép lặp.

Nhờ có phép lặp, chúng ta có khả năng cho thực thi một khối mã nhiều lần mà không cần phải viết lại, giúp chương trình ngắn hơn, dễ đọc hơn, và quan trọng nhất là “rõ ý” hơn.

4. Câu lệnh lặp while

Câu lệnh lặp while chính là câu lệnh được mô tả bởi hình 4.1 ở trên, cú pháp của câu lệnh này như sau:

```

1. while (loop-continuation-condition) {
2.     // statement(s)
3. }
  
```

Trong cú pháp trên, *loop-continuation-condition* là một biểu thức có giá trị *boolean*. Quá trình thực thi sẽ bắt đầu bằng việc tính toán và đánh giá biểu thức *loop-continuation-condition* (điều kiện tiếp diễn vòng lặp). Nếu biểu thức điều kiện có giá trị là *true*, các câu lệnh trong thân vòng lặp sẽ được thực thi. Sau khi các câu lệnh trong thân vòng lặp thực thi xong, quá trình đánh giá sẽ được thực hiện lại, và thân vòng lặp sẽ được thực thi hết lần này đến lần khác chừng nào *loop-continuation-condition* vẫn còn *true*. Do đó câu lệnh này mới có tên như hiện tại (while trong tiếng Anh nghĩa là “chừng nào”).

Sau đây là một ví dụ:

```

1. while(number > 0) {
2.     console.log(number);
3.     number--;
4. }
  
```

Trong ví dụ trên, chừng nào giá trị của biến *number* còn lớn hơn 0 thì giá trị đó sẽ được hiển thị, sau đó thì biến *number* sẽ giảm giá trị đi 1 đơn vị. Cho đến khi giá trị của *number* bằng 0 thì vòng lặp sẽ kết thúc.

Lặp vô hạn

Lặp vô hạn là trường hợp xảy ra khi biểu thức điều kiện luôn luôn trả về giá trị *true*. Chẳng hạn, các vòng lặp sau đây đều thực thi vô hạn:

```
1. while(true) {  
2. }
```

Hoặc

```
1. let i = 0;  
2. while(i < 10) {  
3.   console.log(i);  
4. }
```

5. Câu lệnh lặp do-while

Ứng với hình 4.2 là câu lệnh lặp do-while, nó có cấu trúc như sau:

```
1. do {  
2.   // statement(s)  
3. } while (loop-continuation-condition);
```

Cấu trúc và cách thực thi của câu lệnh *do-while* rất giống với câu lệnh *while*, ngoại trừ cách bắt đầu vòng lặp đầu tiên. Trong khi lệnh *while* tính toán và đánh giá biểu thức điều kiện trước rồi mới bắt đầu khối lệnh trong thân, thì lệnh *do-while* thực thi khối lệnh trong thân trước, sau đó mới tiến hành tính toán và đánh giá. Hệ quả của việc này là thân câu lệnh chắc chắn được thực thi ít nhất một lần, dù cho biểu thức điều kiện là đúng hay sai.

Về mặt lý thuyết, mọi trường hợp cần dùng tới câu lệnh *while* thì chúng ta đều có thể chuyển đổi sang sử dụng câu lệnh *do-while* và ngược lại. Tuy nhiên thực tế cho thấy với mỗi trường hợp, thường có một vòng lặp giúp chúng ta viết mã đẹp hơn trường hợp còn lại.

Ví dụ:

```
1. do {  
2.   console.log(number);  
3.   number--;  
4. } while (number > 0);
```

Trong ví dụ này, các câu lệnh trong khối *do* được thực hiện ít nhất là một lần. Kể cả trường hợp là giá trị của biến *number* nhỏ hơn hoặc bằng 0.

6. Câu lệnh lặp for

Ngoài *while* và *do-while*, câu lệnh *for* cũng là một câu lệnh lặp hay được sử dụng.

Cú pháp:

```

1. for (initial-action; loop-continuation-condition; action-after-each-iteration) {
2.     // statement(s);
3. }

```

Câu lệnh *for* được cấu tạo từ một thân vòng lặp - nơi chứa những câu lệnh cần lặp, đi kèm với đó là 3 *cấu hình* được mô tả giữa *cặp dấu ngoặc tròn*, 3 cấu hình này được phân cách bởi hai dấu chấm phẩy. Cả ba cấu hình đều có thể để trống. Chúng mang những ý nghĩa khác nhau.

- *Initial-action* (hành động khởi tạo): Câu lệnh này sẽ được thực thi một lần duy nhất. Nếu để trống *initial-action* thì tương đương với không thực thi bất kỳ khởi tạo nào.
- *loop-continuation-condition*: Là một biểu thức điều kiện. Biểu thức này sẽ được tính toán để quyết định xem thân vòng lặp có được thực thi hay không. Nếu biểu thức trả về giá trị *true* thì thân vòng lặp sẽ được thực thi, còn nếu trả về *false* thì thân vòng lặp sẽ được bỏ qua (hay nói cách khác là vòng lặp được kết thúc). Nếu bỏ trống, trình thực thi sẽ sử dụng giá trị *true* để thay thế.
- *action-after-each-iteration* (câu lệnh thực thi sau mỗi lần lặp): Câu lệnh này sẽ được thực thi sau mỗi lần thực thi vòng lặp. Bỏ trống *action-after-each-iteration* tương đương với không thực thi bất kỳ lệnh kết thúc nào.

Luồng thực thi của vòng lặp for

Mô tả sau đây sẽ liệt kê các bước thực thi của một vòng lặp *for*.



Hình 4.3: Các bước thực thi của một vòng lặp *for*

Các bước:

1. Các câu lệnh *initial-action* được thực thi
2. Biểu thức điều kiện được đánh giá. Nếu điều kiện đúng thì khối lệnh bên trong thân vòng lặp được thực thi. Nếu điều kiện sai thì vòng lặp kết thúc
3. Khối lệnh bên trong thân vòng lặp được thực thi
4. Các câu lệnh *action-after-each-iteration* được thực thi
5. Bắt đầu vòng lặp mới từ bước 2 – 3 – 4.

Ví dụ 1:

Chúng ta viết đoạn mã in 10 ký tự bằng câu lệnh *for* như sau:

```

1. let dot = prompt("Mời bạn nhập ký tự:");
2. let line = "";
3. for (let count = 1; count <= 10 ; count++) {
4.     line += dot;
5. }
6. console.log(line);

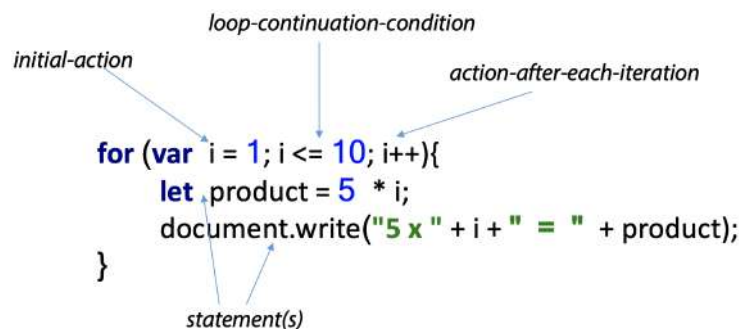
```

Giải thích:

- Biến *dot* được sử dụng để lưu ký tự do người dùng nhập vào.
- Biến *line* được sử dụng để lưu chuỗi các ký tự.
- Dòng số 3 là nơi khai báo vòng lặp *for*, dòng số 4 là thân vòng lặp.
- Biểu thức *let count = 1* là hành động khởi tạo.
- Biểu thức *count <= 10* là biểu thức điều kiện.
- Biểu thức *count++* là biểu thức được thực thi sau mỗi vòng lặp.

Ví dụ 2:

Trong ví dụ sau, chúng ta sẽ sử dụng một vòng lặp *for* để hiển thị bảng tính nhân của số 5.



Hình 4.4: Vòng lặp *for* hiển thị bảng tính nhân

Kết quả:

```

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
  
```

7. Câu lệnh lặp lồng nhau

Trong thực tế, có những trường hợp mà chúng ta cần có một thao tác lặp ở bên trong một thao tác lặp khác. Chẳng hạn như để vẽ một bàn cờ có các dòng và các cột, ta có thể sử dụng một vòng lặp để vẽ các dòng, và bên trong từng dòng thì ta có thể sử dụng một vòng lặp khác để vẽ các ô. Hoặc một ví dụ khác là để hiển thị bảng cửu chương, ở ví dụ phía trên chúng ta chỉ mới sử dụng một vòng lặp để hiển thị bảng tính nhân của 5, như vậy chúng ta có thể sử dụng một vòng lặp khác nữa để tính bảng tính nhân của các số khác từ 1 đến 9.

Trong những trường hợp như vậy, chúng ta đặt một vòng lặp vào bên trong một vòng lặp khác, chúng ta gọi là các vòng lặp lồng nhau.

Ví dụ:

```
1. for (let i = 0; i < 5; i++){
2.   for (let j = 0; j < 10; j++){
3.     document.write("*");
4.   }
5.   document.write('<br/>');
6. }
```

Kết quả:

```
*****
*****
*****
*****
*****
```

Trong ví dụ này, vòng lặp ở ngoài tương ứng với từng dòng, vòng lặp ở trong tương ứng với từng cột. Vòng lặp ở ngoài sẽ in ra 5 dòng, và trong mỗi dòng đó vòng lặp ở trong sẽ in ra 10 ký tự.

Lưu ý: Chúng ta có thể lồng nhiều hơn 2 vòng lặp vào với nhau, không có giới hạn về số lượng vòng lặp lồng nhau. Tuy nhiên, càng nhiều vòng lặp lồng nhau thì độ phức tạp của thuật toán càng tăng. Do đó, thông thường chúng ta sẽ cố gắng làm phẳng các thuật toán để hạn chế số lượng vòng lặp lồng nhau.

8. Câu lệnh break

Trên thực tế, đôi khi thực thi một vòng lặp chúng ta không cần thiết phải thực thi đến điểm cuối cùng. Khi thỏa mãn một điều kiện nào đó chúng ta muốn thoát khỏi vòng lặp đó chúng ta sẽ sử dụng lệnh *break*.

Trong khi vòng lặp đang thực thi mà gặp câu lệnh *break* thì chương trình sẽ thoát khỏi vòng lặp đó và thực thi lệnh ngay sau vòng lặp. Nếu có nhiều vòng lặp lồng nhau thì lệnh *break* sẽ được áp dụng cho vòng lặp gần nhất.

Ví dụ:

```
1. let message = "";
2. let index;
3. for (index = 0; index < 5; index++) {
4.   if (index === 3) {
5.     break;
6.   }
7.   message += "The number is " + index + "<br>";
8. }
```

Kết quả hiển thị:

The number is 0

The number is 1

The number is 2

Trong đoạn mã trên, câu lệnh *break* được thực thi khi điều kiện *index === 3* được thỏa mãn. Do đó, mặc dù vòng lặp của chúng ta được chạy đến 5 lần nhưng khi index bằng 3 thì nó đã thoát và không thực thi nữa. Do đó, kết quả là chỉ có 3 dòng được in ra.

9. Câu lệnh continue

Lệnh *continue*, thay vì kết thúc cả câu lệnh lặp như lệnh *break*, chỉ bỏ qua việc thực thi vòng lặp hiện tại. Lưu ý rằng nếu được sử dụng trong vòng lặp *for*, sau khi *continue*, lệnh được mô tả trong *action-after-each-iteration* vẫn sẽ được thực thi.

Ví dụ:

```
1. let text = "";
2. for (let i = 0; i < 10; i++) {
3.   if (i === 3) {
4.     continue;
5.   }
6.   text = text + i;
7. }
8. console.log(text);
```

Kết quả: "012456789".

Đề ý rằng giá trị 3 không xuất hiện trong kết quả xuất của đoạn mã bởi vì lần lặp thứ 4 (với giá trị *i* bằng 3) đã được bỏ qua.

Hỏi nhanh: Nếu chúng ta thay câu lệnh *continue* ở trên thành câu lệnh *break* thì kết quả sẽ là gì?

10. Mã sạch

Quy tắc #1: Quy ước

- Trong các lệnh của vòng lặp, phía sau các từ khóa *for*, *while*, do sẽ là khoảng trắng và tiếp theo là biểu thức điều kiện với từ khóa *for* và *while*, kết thúc dòng đầu tiên sẽ là ký tự *{* để đánh dấu bắt đầu khối mã lệnh cần được thực thi nằm bên trong vòng lặp, trước dấu *{* sẽ là một khoảng trắng.
- Khối mã lệnh thực thi bên trong vòng lặp phải lùi đầu dòng vào bốn khoảng trắng.
- Ký tự *}* đánh dấu kết thúc khối mã lệnh nằm trong vòng lặp phải được viết riêng biệt tại dòng cuối cùng.

Ví dụ:

Không nên:

```
1. for (let index = 1; index <= 10; index++)
2. { product = 5 * index;
3.   document.write("5 x " + index + " = " + product);}
```

Nên:

```
1. for (let index = 1; index <= 10; index++) {
2.   product = 5 * index;
3.   document.write("5 x " + index + " = " + product);
4. }
```

Quy tắc #2: Viết có ý nghĩa

Đối với câu lệnh vòng lặp cũng tương tự như viết bất kỳ mã lệnh nào, đều phải tuân thủ quy tắc viết có ý nghĩa. Việc đọc, hiểu một đoạn mã lệnh bên trong vòng lặp sẽ được chạy liên tục bao nhiêu lần, chạy đến khi nào thì dừng sẽ trở nên khó khăn, nếu như các điều kiện ban đầu không được viết rõ ràng, dễ hiểu.

Ví dụ:

Trước khi tách biến:

```
1. let cars = ["Audi", "BMW", "Honda", "Hyundai"];
2. for (let i = 0; i <= cars.length; i++) {
3.   document.write("The " + i + "th car is " + cars[i]);
4. }
```

Sau khi tách biến:

```
1. let cars = ["Audi", "BMW", "Honda", "Hyundai"];
2. let totalCars = cars.length;
3. for (let index = 0; index <= totalCars; index++) {
4.   document.write("The " + index + "th car is " + cars[index]);
5. }
```

11. Các lỗi thường gặp

Lỗi thường gặp #1: Vòng lặp vô hạn

Khi sử dụng các vòng lặp, những người mới bắt đầu thường gặp một lỗi đó là vô tình để vòng lặp thực thi vô hạn. Việc này dẫn đến tình trạng ứng dụng bị treo hoặc bị tắt đột ngột.

Nguyên nhân chính dẫn đến tình trạng này là do không xem xét cẩn thận điều kiện dừng, dẫn đến tình trạng giá trị của biểu thức điều kiện luôn đúng.

Ví dụ 1:

```
1.   for(var   index   =   0;   index   >=   0;   index++)   {
document.write("index = " + index + "<br/>");
2. }
```

Ví dụ 2:

```
1. var index = 0;
2. while (index >= 0) {
3.   document.write("index = " + index + "<br/>");
4.   i++;
5. }
```

Trong hai trường hợp trên, điều kiện dừng vòng lặp là index nhỏ hơn 0, trong vòng lặp index được khởi tạo bằng 0 và luôn tăng một đơn vị sau mỗi lần lặp, biểu thức $\text{index} \geq 0$ luôn đúng vì thế vòng lặp rơi vào trạng thái lặp vô hạn.

Lỗi thường gặp #2: Số lần lặp không đúng

Lỗi này tuy nhỏ nhưng rất hay bị mắc phải, đặc biệt với những lập trình viên mới.

Ví dụ: In ra 100 lần câu "Welcome to JavaScript!".

Mã triển khai sử dụng vòng while:

```
1. let index = 0;
2. while (index <= 100) {
3.   document("Welcome to JavaScript!<br/>");
4.   count++;
5. }
```

Hoặc mã triển khai sử dụng vòng lặp for:

```
1. for(let index = 0; index <=100; index++) {
2.   document("Welcome to JavaScript!<br/>");
3. }
```

Kết quả sẽ in ra 101 câu "Welcome to JavaScript!".

Bởi vì, khi nhìn vào biểu thức điều kiện ta thấy điều kiện để dừng vòng lặp là biến $\text{index} \leq 0$ mà index ban đầu được khởi tạo bằng 0. Có nghĩa vòng lặp sẽ chạy từ 0 đến 100 (101 lần) và sẽ in ra 101 câu "Welcome to JavaScript!".

Để giải quyết tình huống này, chúng ta chỉ cần sửa biểu thức điều kiện thành $\text{index} < 100$. Tuy là một lỗi nhỏ nhưng nếu không cẩn thận chúng ta rất hay mắc phải lỗi này.

12. Bài thực hành

Bài 1: Sử dụng vòng lặp for

Mục tiêu:

Luyện tập sử dụng vòng lặp for.

Mô tả:

Viết chương trình hiển thị chuỗi "The number is N" 5 lần sử dụng vòng lặp for. Với N sẽ hiển thị từ 0 đến 4.

Hướng dẫn:

Bước 1: Phân tích 3 phần trong vòng lặp for

Khởi tạo: $i = 0$

Điều kiện: $i < 5$

Lệnh lặp lại: $i = i + 1$

Bước 2: Viết mã vòng lặp for

```
1. let i;
2. for (i = 0; i < 5; i++) {
3.   text += "The number is " + i + "<br/>";
4. }
```

Lưu ý: biến *i* có thể được khởi tạo tại thời điểm khai báo, hoặc trong phần thân của vòng lặp.

Bước 3: Toàn bộ đoạn mã lệnh

```
1. let text = "";
2. let i;
3. for (i = 0; i < 5; i++) {
4.   text += "The number is " + i + "<br>";
5. }
6. document.getElementById("demo").innerHTML = text;
```

Bước 4: Khai báo thẻ `<p id="demo">` trong `<body>`

Bước 5: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/dev/chapter-04/01-su-dung-vong-lap-for>

Bài 2: Sử dụng vòng lặp for

Mục tiêu:

Luyện tập sử dụng vòng lặp for.

Mô tả:

Viết chương trình nhập vào một số bất kỳ lớn hơn 0. Tính tổng các phần tử từ 1 đến số vừa nhập vào.

Hướng dẫn:

Bước 1: Khai báo biến để lưu số vừa nhập vào

```
1. let num = prompt("Enter your number: ");
```

Bước 2: Khai báo biến total và khởi tạo cho total giá trị ban đầu là 0, biến total dùng để lưu tổng.

```
1. let total = 0;
```

Bước 3: Xây dựng các phần của vòng lặp for

Khởi tạo: `let i = 1`

Điều kiện: `i <= num`

Lệnh lặp lại: `i += 1`

```
1. for (let i = 1; i <= num; i += 1) {
2.   //phần thân vòng lặp thực hiện tính tổng
3. }
```

Bước 4: Viết mã thực hiện tính tổng

```
1. total = total + i;
```

Bước 5: Hiển thị kết quả

```
1. alert(total);
```

Bước 7: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-04/02-su-dung-vong-lap-for>

Bài 3: Sử dụng vòng lặp while

Mục tiêu:

Luyện tập sử dụng vòng lặp while

Mô tả:

Viết chương trình nhận vào một số từ hộp thoại, việc nhập kết thúc khi người dùng nhập vào giá trị -1. Mỗi giá trị nhập được sẽ được hiển thị ra và thực hiện tính tổng các giá trị đó.

Hướng dẫn:

Bước 1: Khai báo biến num để lưu giá trị nhập vào từ hộp thoại.

```
1. let num = parseInt(prompt("Enter a number: "));
```

Bước 2: Khai báo biến total lưu tổng.

```
1. let total = num;
```

Bước 3: Phân tích vòng lặp

Điều kiện vòng lặp thực hiện: num != -1

Bước 4: Viết mã phần thân vòng lặp

```
1. while( num != -1 ) {
2.   num = parseInt(prompt("Enter a number: "));
3.   if(num!= -1){
4.     alert(number);
5.     //phần code tính tổng
6.     total += num;
7.   }
8. }
```

Bước 5: Hiển thị tổng

```
1. alert("total:" + total);
```

Bước 6: Thực thi chương trình. Quan sát kết quả

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-04/03-su-dung-vong-lap-while>

Bài 4: Sử dụng vòng lặp while

Mục tiêu:

Luyện tập sử dụng vòng lặp while.

Mô tả:

Viết chương trình hiển thị thẻ <hr> theo độ rộng từ 1 đến 100. Kết quả sẽ được hiển thị như hình:



Hướng dẫn:

Bước 1: Phân tích vòng lặp while

Điều kiện vòng lặp thực hiện: Độ rộng thẻ hr ≤ 100

Vòng lặp thực hiện thế nào: Hiển thị thẻ hr với độ rộng bắt đầu từ 1 đến 100.

Bước 2: Viết vòng lặp while

```
1. let i = 1;
2. while (i < 100) {
3.   //phân thân vòng lặp
4.   document.write("<hr width = " + i + "%>");
5.   i++;
6. }
```

Bước 3: Chạy chương trình, quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-04/04-su-dung-vong-lap-while>

Bài 5: Sử dụng vòng lặp do...while

Mục tiêu:

Luyện tập sử dụng vòng lặp do...while.

Mô tả:

Sử dụng vòng lặp do while để viết lại chương trình yêu cầu nhập các số từ 1 -> 10 ở ví dụ trong phần vòng lặp while.

Hướng dẫn:

Bước 1: Tạo file *dowhilesample.html*. Tạo thẻ `<script>` và viết các mã lệnh thực thi trong đó.

Bước 2: Tạo biến value để lưu giá trị người dùng nhập vào

```
1. let value = null;
```

Bước 3: Viết mã thực thi

```
1. do {
2.   value = prompt("Nhập vào số từ 1 -> 10");
3. } while (value < 1 || value > 10);
```

Bước 4: Hiển thị ra màn hình giá trị vừa nhập

```
1. alert("Số bạn vừa nhập là " + value);
```

Bước 5: Thực thi chương trình. Quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-04/05-su-dung-vong-lap-do-while>

Bài 6: Sử dụng vòng lặp lồng nhau**Mục tiêu:**

Luyện tập sử dụng vòng lặp lồng nhau

Mô tả:

Chương trình hiển thị bảng với dữ liệu như sau

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Hướng dẫn:**Bước 1:** Phân tích vòng lặp sử dụng trong bài toán

Vòng lặp for ngoài dùng để in ra số dòng. Gồm 10 dòng. Do đó vòng lặp sẽ là:

```
1. for (i = 1; i <= 10; i++)
```

Vòng lặp trong in ra giá trị từng cột trên mỗi dòng. Gồm 10 cột cho mỗi dòng. Do đó vòng lặp sẽ là:

```
1. for (j = 1; j <= 10; j++)
```

Mỗi cột có giá trị bằng $i * j$.

Bước 2: Viết mã sử dụng vòng lặp for lồng nhau

```
1. let sout;
2. sout = "<table border='1' width='300' cellspacing='0' cellpadding='3'>"
```

```

3. for (let i = 1; i <= 10; i++) {
4.   sout = sout + "<tr>";
5.   for (let j = 1; j <= 10; j++) {
6.     sout = sout + "<td>" + i * j + "</td>";
7.   }
8.   sout = sout + "</tr>";
9. }
10. sout = sout + "</table>";
11. document.write(sout);

```

Bước 3: Thực thi chương trình, quan sát kết quả.

Bước 4: Có thể thử nghiệm thay đổi mã trên với vòng lặp while lồng nhau

```

1. let sout, i, j;
2. sout = "<table border='1' width='300' cellspacing='0' cellpadding='3'>";
3. i = j = 1;
4. while (i <= 10) {
5.   sout = sout + "<tr>";
6.   while (j <= 10) {
7.     sout = sout + "<td>" + i * j + "</td>";
8.     j++;
9.   }
10.  sout = sout + "</tr>";
11.  j = 1;
12.  i++;
13. }
14. sout = sout + "</table>";
15. document.write(sout);

```

Bước 5: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-04/06-su-dung-vong-lap-long-nhau>

13. Bài tập

Bài 1: Sinh bảng cửu chương

Mô tả:

Hãy viết một ứng dụng để in ra trang web một bảng cửu chương với giao diện như sau:

2 x 1 = 2	2 x 2 = 4	2 x 3 = 6	2 x 4 = 8	2 x 5 = 10	2 x 6 = 12	2 x 7 = 14	2 x 8 = 16	2 x 9 = 18
3 x 1 = 3	3 x 2 = 6	3 x 3 = 9	3 x 4 = 12	3 x 5 = 15	3 x 6 = 18	3 x 7 = 21	3 x 8 = 24	3 x 9 = 27
4 x 1 = 4	4 x 2 = 8	4 x 3 = 12	4 x 4 = 16	4 x 5 = 20	4 x 6 = 24	4 x 7 = 28	4 x 8 = 32	4 x 9 = 36
5 x 1 = 5	5 x 2 = 10	5 x 3 = 15	5 x 4 = 20	5 x 5 = 25	5 x 6 = 30	5 x 7 = 35	5 x 8 = 40	5 x 9 = 45
6 x 1 = 6	6 x 2 = 12	6 x 3 = 18	6 x 4 = 24	6 x 5 = 30	6 x 6 = 36	6 x 7 = 42	6 x 8 = 48	6 x 9 = 54
7 x 1 = 7	7 x 2 = 14	7 x 3 = 21	7 x 4 = 28	7 x 5 = 35	7 x 6 = 42	7 x 7 = 49	7 x 8 = 56	7 x 9 = 63
8 x 1 = 8	8 x 2 = 16	8 x 3 = 24	8 x 4 = 32	8 x 5 = 40	8 x 6 = 48	8 x 7 = 56	8 x 8 = 64	8 x 9 = 72
9 x 1 = 9	9 x 2 = 18	9 x 3 = 27	9 x 4 = 36	9 x 5 = 45	9 x 6 = 54	9 x 7 = 63	9 x 8 = 72	9 x 9 = 81

Hướng dẫn: Sử dụng 2 vòng lặp lồng nhau. Một vòng lặp để chèn các dòng và một vòng lặp để chèn các ô.

Bài 2: Hiển thị các số nguyên tố đầu tiên

Mô tả:

Trong phần này, chúng ta sẽ phát triển một ứng dụng cho phép hiển thị 20 số nguyên tố đầu tiên.

Hướng dẫn:

Bước 1: Khai báo biến số nguyên numbers và nhập cho nó một giá trị từ bàn phím để lưu số lượng số nguyên tố cần in ra.

Bước 2: Khai báo biến count và gán cho nó giá trị 0, biến này để đếm xem số lượng số nguyên tố cần in ra đã bằng numbers hay chưa.

Bước 3: Khai báo biến N và gán cho giá trị 2, biến này để kiểm tra xem các giá trị nó nhận được có phải là số nguyên tố không, mỗi lần lặp giá trị của biến sẽ được tăng lên 1.

Bước 4: Trong khi count < numbers thì:

- Kiểm tra xem N có phải là số nguyên tố không. Nếu N là số nguyên tố thì in ra giá trị của N và tăng giá trị của count lên 1
- Giá trị của N tăng lên 1 để kiểm tra số tiếp theo

Bài 3: Các số chia hết cho 7

Tính tổng của 30 số đầu tiên chia hết cho 7 trong các số tự nhiên.

Bài 4: Số fibonacci chia hết cho 5

Tìm số đầu tiên trong dãy fibonacci chia hết cho 5.

14. Bài kiểm tra

Câu 1: JavaScript hỗ trợ những lệnh nào để thực hiện vòng lặp?

- for
- do/while
- for/in
- while

Câu 2: Lệnh 'for' nào thực hiện lặp 5 lần?

- a) for(i=0; i<5; i++)
- b) for(i=0; i<5; i+=2)
- c) for(i=0; i<6; i++)
- d) for(i=0; i<4; i++)

Câu 3: Xác định giá trị của biến count sau khi thực hiện đoạn mã sau:

```

1. let count = 0;
2. while (count++ < 10) {
3.     count++;
4. }

```

- a) 11
- b) Không có giá trị nào đúng
- c) 9
- d) 10

Câu 4: Xác định giá trị của biến count sau khi thực hiện đoạn mã sau:

```

1. let count = 0;
2. do {
3.     count++;
4. } while (count++ < 10);

```

- a) Không có giá trị nào đúng
- b) 10
- c) 12
- d) 11

Câu 5: Đây là đoạn mã thực hiện lặp tương đương với đoạn mã sau:

```

1. let count = 0;
2. while (count < 5) { count++; }

```

- a) var count = 0; for(count < 5; count ++){ }
- b) Không có đoạn mã nào tương đương
- c) for(count = 0; count <= 5; count++){ }
- d) for(count = 0; count < 5){ count++; }

Câu 6: cho phép một tập các chỉ thị được thực thi chừng nào một điều kiện xác định đạt còn đúng.

- a) Cấu trúc
- b) Vòng lặp
- c) Tất cả đều sai
- d) Toán tử

Câu 7: Dự đoán kết quả đoạn mã sau:

```

1. let x = 1;
2. for (let i = 0; i < 3; i++) {
3.     x = x + 1;
4. }
5. console.log(x);

```

- a) 1
- b) 4
- c) 2
- d) 3

Câu 8: Dự đoán kết quả đoạn mã sau:

```

1. let x = 1, i;
2. for (i = 0; i < 3; i = i + 2) {
3.   x = x + 1;
4. }
5. console.log(x);
6. console.log(i);

```

- a) 2,4
- b) 3,2
- c) 3,4
- d) 2,1

Câu 9: Dự đoán kết quả của đoạn mã dưới đây:

```

1. let i = 0;
2. while (i < 3) {
3.   console.log(i);
4.   i++;
5. }

```

- a. 1 2 3
- b. Lỗi
- c. 0 1 2
- d. 0 1 2 3

Câu 10: Dự đoán kết quả của đoạn mã dưới đây:

```

1. let i = 0;
2. while (i <= 3) {
3.   console.log("hi");
4.   i++;
5. }

```

- a) hi bye
- b) hihi
- c) hi bye hi
- d) Hihihhi

Đáp án: Câu 1: a, b, c, d; Câu 2: c; Câu 3: a; Câu 4: c; Câu 5: d; Câu 6: b; Câu 7: b; Câu 8: c; Câu 9: c; Câu 10 d.

15. Tổng kết

- a) Cấu trúc lặp giúp tự động thực hiện lặp đi lặp lại các tác vụ
- b) Các cấu trúc lặp thông dụng: for, while, do-while
- c) Cấu trúc lặp for được sử dụng phù hợp nhất trong những tình huống mà chúng ta biết trước số lần lặp
- d) Cấu trúc lặp while và do-while được sử dụng phù hợp nhất trong những tình huống mà chúng ta không biết trước số lần lặp
- e) Phần thân của cấu trúc lặp do-while được thực hiện ít nhất là 1 lần, kể cả trong trường hợp biểu thức điều kiện trả về false ngay từ đầu

- f) Có thể lồng các vòng lặp vào với nhau, càng nhiều vòng lặp lồng nhau thì độ phức tạp của mã càng cao
- g) Câu lệnh break được sử dụng để ngắt luồng thực thi của vòng lặp
- h) Câu lệnh continue được sử dụng để bỏ qua vòng lặp hiện tại.

codegym.vn

Chương 5 - Mảng

Lưu trữ và xử lý nhiều giá trị cùng dạng một cách dễ dàng.

1. Mục tiêu

- Trình bày được khái niệm mảng
- Giải thích được mục đích và ý nghĩa của mảng
- Khai báo và sử dụng được mảng 1 chiều
- Khai báo và sử dụng được mảng 2 chiều
- Sử dụng được vòng lặp để duyệt mảng 1 chiều
- Sử dụng được vòng lặp để duyệt mảng 2 chiều
- Thực hiện được các thao tác cơ bản với mảng

2. Giới thiệu

Trong các tình huống thực tế, có nhiều trường hợp chúng ta phải lưu trữ nhiều dữ liệu. Chẳng hạn như lưu trữ danh sách tên của hàng trăm sinh viên, điểm thi của từng sinh viên, lưu trữ danh sách các sản phẩm, lưu trữ danh sách các bài hát trong một album. Nếu sử dụng biến để chứa những giá trị này, chúng ta sẽ phải khai báo hàng trăm biến, thậm chí là hàng nghìn biến, điều này gây ra nhiều khó khăn cho lập trình viên và gần như là không thực tế.

Trong những tình huống như vậy, chúng ta có thể sử dụng cấu trúc dữ liệu mảng để lưu trữ dữ liệu dễ dàng hơn.

Hoàn thành chương này, chúng ta có thể xây dựng các ứng dụng trong đó có lưu trữ và xử lý nhiều dữ liệu cùng dạng.

3. Mảng

Mảng là gì?

Mảng là một loại biến đặc biệt, được dùng để lưu trữ nhiều giá trị. Trong các trường hợp mà chúng ta cần lưu trữ một danh sách các giá trị, chẳng hạn như danh sách khách hàng, danh sách sản phẩm, danh sách cấu hình... thì mảng là một lựa chọn tốt. Chẳng hạn, nếu chúng ta sử dụng biến bình thường để lưu 3 tên xe thì đoạn mã sẽ như sau:

```
1. let car1 = "Toyota";  
2. let car2 = "Subaru";  
3. let car3 = "BMW";
```

Thử tưởng tượng, nếu chúng ta có danh sách 100 chiếc xe thì sẽ làm sao? Chẳng lẽ chúng ta lại đi khai báo 100 biến tương ứng. Điều này có vẻ không hợp lý chút nào.

Nếu chúng ta sử dụng mảng, mã nguồn của chúng ta sẽ được đơn giản hoá đi rất nhiều:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
```

Một mảng có thể chứa nhiều giá trị trong một biến duy nhất, và chúng ta có thể truy cập các giá trị thông qua chỉ số của nó.

Các khái niệm khi làm việc với mảng

Khi khai báo và làm việc với mảng, chúng ta cần biết các khái niệm sau:

- Tên mảng (name): Phải tuân thủ theo quy tắc đặt tên của biến.
- Độ dài của mảng (length): Là số lượng giá trị mà mảng có.
- Phần tử (item): Là một giá trị trong mảng.
- Chỉ số (index): Là vị trí của một phần tử trong mảng

Ví dụ:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
```

Trong ví dụ này:

- Tên của mảng là: *cars*
- Độ dài của mảng là: 3
- Các phần tử của mảng là: **"Toyota"**, **"Subaru"** và **"BMW"**

Chỉ số của phần tử (item index) là vị trí của phần tử trong mảng. Chỉ số đầu tiên là 0, tiếp sau là 1, 2, 3... cho đến hết. Chỉ số của phần tử cuối cùng của mảng sẽ là *length - 1*. Trong đó *length* là độ dài của mảng. Chẳng hạn, nếu mảng có 6 phần tử thì chỉ số của phần tử cuối cùng sẽ là 5.

Ví dụ:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
2. console.log(cars[0]); // Toyota
3. console.log(cars[1]); // Subaru
```

Cú pháp khai báo mảng

JavaScript hỗ trợ một số cú pháp khác nhau để khai báo mảng.

Cách 1: Sử dụng dấu ngoặc vuông ([]):

Cú pháp:

```
1. let arr = [element1, element2, element3];
```

Ví dụ:

```
1. let arr = ["Toyota", "Subaru", "BMW"];
```

Lưu ý:

Dấu cách, hoặc dấu xuống dòng không có ảnh hưởng gì tới việc khai báo mảng. Chẳng hạn, chúng ta có thể khai báo trên nhiều dòng như sau:

```
1. let arr = [
2.     "Toyota",
```

```
3.     "Subaru",
4.     "BMW"
5. ];
```

Cách 2: Sử dụng từ khóa *new*:

```
1. let arr = new Array("Toyota", "Subaru", "BMW");
```

Việc sử dụng từ khóa *new* hay dấu ngoặc vuông đều cho kết quả như nhau. Thông thường thì cú pháp sử dụng dấu ngoặc vuông được sử dụng nhiều hơn do ngắn gọn, dễ đọc.

Lưu ý:

Khi khai báo mảng sử dụng từ khóa *new*, nếu chúng ta truyền vào một số thì số đó sẽ là độ dài của mảng. Ví dụ, *new Array(5)* thì sẽ tạo ra một mảng có 5 phần tử, và giá trị của cả 5 phần tử đều là *undefined*. Tuy nhiên, *new Array(5, 10)* thì lại tạo ra một mảng có 2 phần tử, phần tử đầu tiên có giá trị là 5 và phần tử thứ hai có giá trị là 10.

Truy xuất một phần tử của mảng

Để truy xuất đến một phần tử của mảng, chúng ta sử dụng tên mảng và chỉ số của phần tử đó.

Ví dụ:

Truy xuất đến phần tử đầu tiên của mảng *cars*:

```
1. let toy = cars[0];
```

Chúng ta cũng có thể sử dụng cách tương tự để gán giá trị cho các phần tử của mảng.

Ví dụ:

Gán giá trị mới cho phần tử ở vị trí số 3 của mảng *cars*:

```
1. cars[3] = "Hyundai";
```

Độ dài của mảng

Chúng ta sử dụng thuộc tính *length* để biết độ dài của mảng.

Ví dụ:

```
1. let countOfCars = cars.length; // 4
```

Trong ví dụ trên, độ dài của mảng là 4.

Thêm phần tử vào mảng

Hàm *push()* được sử dụng để thêm một phần tử mới vào phần cuối của mảng.

Ví dụ:

```
1. cars.push("Kia");
2. console.log(cars[cars.length - 1]); // Kia
```

Hàm *unshift()* được sử dụng để thêm một phần tử mới và phần đầu của mảng.

Ví dụ:

```
1. cars.unshift("Ferrari");
2. console.log(cars[0]); // Ferrari
```

Xóa phần tử của mảng

Hàm `pop()` được sử dụng để xóa phần tử cuối cùng của mảng và trả về giá trị của phần tử đó.

Ví dụ:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
2. console.log(cars[0]); //Toyota
3. console.log(cars.length); // 3
4. let lastElement = cars.pop();
5. console.log(cars.length); // 2
6. console.log(lastElement); //BMW
```

Hàm `shift()` được sử dụng để xóa đi phần tử đầu tiên của mảng và trả về giá trị của phần tử đó.

Ví dụ:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
2. console.log(cars.length); // 3
3. let firstElement = cars.shift();
4. console.log(cars.length); // 2
5. console.log(firstElement); //Toyota
```

Sắp xếp mảng

Hàm `sort()` được sử dụng để sắp xếp các phần tử của mảng theo một trật tự nhất định.

Ví dụ:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
2. cars.sort();
3. console.log(cars); //[ "BMW", "Subaru", "Toyota" ]
```

Chúng ta cũng có thể đảo ngược trật tự của một mảng bằng cách sử dụng hàm `reverse()`.

Ví dụ:

```
1. let cars = ["Toyota", "Subaru", "BMW"];
2. cars.sort();
3. console.log(cars); //[ "BMW", "Subaru", "Toyota" ]
4. cars.reverse();
5. console.log(cars); //[ "Toyota", "Subaru", "BMW" ]
```

Mặc định, phương thức `sort()` sẽ so sánh các phần tử theo trật tự của các ký tự trong bảng chữ cái. Chẳng hạn, phần tử "Apple" sẽ được đưa lên trước phần tử "Banana". Tuy nhiên, điều này sẽ gây sai sót khi so sánh các chữ số, chẳng hạn, số 100 sẽ được đưa lên trước số 25 (bởi vì số 1 đứng trước số 2 trong bảng chữ cái). Trong trường hợp này, chúng ta cung cấp cho hàm `sort()` một hàm so sánh, để nó thực hiện đúng chức năng của mình.

Ví dụ:

```
1. let points = [40, 100, 1, 5, 25, 10];
2. points.sort(function (a, b) {
```

```

3.   return a - b;
4. } // [ 1, 5, 10, 25, 40, 100 ]
5. points.sort(function (a, b) {
6.   return b - a;
7. } // [ 100, 40, 25, 10, 5, 1 ]

```

Lưu ý:

- Hàm so sánh là một hàm có mục đích giúp so sánh 2 phần tử.
- Hàm so sánh có 2 tham số
- Hàm so sánh trả về giá trị kiểu số: âm (nhỏ hơn 0), dương (lớn hơn 0) và 0.
 - Âm có nghĩa là phần tử đầu nhỏ hơn phần tử sau
 - Dương có nghĩa là phần tử đầu lớn hơn phần tử sau
 - 0 có nghĩa là 2 phần tử bằng nhau

4. Duyệt qua các phần tử của mảng

Khi làm việc với mảng, rất thường xuyên chúng ta cần phải duyệt qua các phần tử của mảng. Có một số cách khác nhau để duyệt mảng, trong đó 2 cách phổ biến là sử dụng vòng lặp for hoặc sử dụng phương thức forEach của mảng.

Duyệt mảng với vòng lặp for

Khi sử dụng vòng lặp for, chúng ta thực hiện việc đếm biến chạy từ vị trí số 0, tức là vị trí của phần tử đầu tiên, đến vị trí số $n - 1$, tức là vị trí của phần tử cuối cùng.

Ví dụ:

```

1. let fruits = ['Banana', 'Orange', 'Apple', 'Mango'];
2. let text = "";
3. for(let index = 0; index < fruits.length; index++) {
4.   text += fruits[index] + '<br/>';
5. }

```

Kết quả:

```

Banana
Orange
Apple
Mango

```

Trong ví dụ này, vòng lặp for chạy biến đếm từ số 0 đến số 3 tức là vị trí của phần tử cuối cùng.

Duyệt mảng với phương thức forEach

Một cách khác để duyệt qua các phần tử của mảng đó là sử dụng phương thức forEach của mảng. Phương thức này nhận vào tham số là một hàm (chúng ta sẽ tìm hiểu về hàm trong chương sau), trong đó có các phần tử của mảng.

Ví dụ:

```

1. var numbers = [4, 9, 16, 25];
2. numbers.forEach(function(item) {
3.     document.write(item + '<br/>');
4. });

```

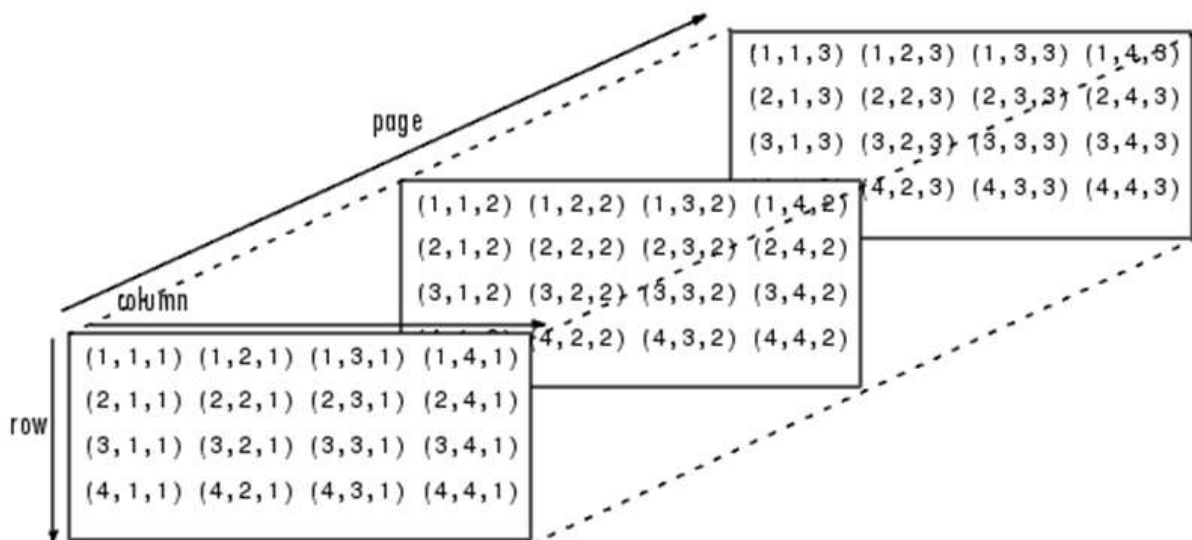
Kết quả:

4
9
16
25

Trong ví dụ này, lần lượt 4 phần tử của mảng sẽ được duyệt qua và hiển thị lên tài liệu. Biến *item* lần lượt đại diện cho từng phần tử của mảng, từ phần tử đầu tiên cho đến phần tử cuối cùng.

5. Mảng nhiều chiều

Phần tử của một mảng có thể là một mảng khác. Khi này chúng ta có mảng nhiều chiều. Có thể có mảng 2 chiều, 3 chiều... hoặc nhiều hơn. Mảng càng nhiều chiều thì độ phức tạp khi xử lý càng cao.



Hình 5.1: Mảng nhiều chiều

Truy cập phần tử của mảng nhiều chiều

Mảng một chiều cần 1 chỉ số để xác định vị trí của phần tử mảng.

Mảng hai chiều cần 2 chỉ số để xác định vị trí của phần tử mảng

	chiều					
chiều	5	8	9	2	10	3
	4	2	6	8	1	0
	5	9	11	2	4	6

Mảng 3 chiều cần 3 chỉ số để xác định vị trí của phần tử mảng

The diagram illustrates a 3D structure composed of blocks, with three red arrows and the word "chiều" (dimension) indicating the three axes:

- chiều** (length): A red arrow pointing diagonally upwards and to the right, indicating the depth of the structure.
- chiều** (width): A red arrow pointing horizontally to the right, indicating the width of the structure.
- chiều** (height): A red arrow pointing vertically downwards, indicating the height of the structure.

The structure itself is a 3D arrangement of blocks, with the top layer being a 6x6 grid of blocks, the middle layer being a 6x6 grid of blocks, and the bottom layer being a 6x6 grid of blocks. The blocks are arranged in a way that shows the three dimensions of the structure.

Mảng hai chiều

Mảng hai chiều là mảng nhiều chiều được sử dụng phổ biến. Mảng hai chiều là một mảng mà có mỗi phần tử là một mảng một chiều. Có thể coi mảng hai chiều là một bảng gồm n dòng và m cột:

	Cột 0	Cột 1	Cột 2	Cột 3
Dòng 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Dòng 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Dòng 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Tên mảng: a
 chỉ số dòng: 0, 1, 2
 chỉ số cột: 0, 1, 2, 3

/codegym.vn/ - 104

Mảng trên có 3 dòng và 4 cột, tổng cộng 12 phần tử, các phần tử đó sẽ được truy cập theo chỉ số từ [0][0], [0][1],... đến [2][3].

Khởi tạo mảng hai chiều

Để khởi tạo mảng hai chiều, chúng ta có thể sử dụng các cặp dấu ngoặc vuông, trong đó mỗi dòng được khai báo như là một mảng một chiều. Trong ví dụ sau, chúng ta khai báo một mảng hai chiều bao gồm 3 dòng và 3 cột. Mỗi dòng được khai báo như một mảng một chiều.



Hình 5.6: Minh họa mảng hai chiều

```
1. let cities = [
2.   ["Hanoi", "Saigon", "DaNang"],
3.   ["New York", "California", "Miami"],
4.   ["Tokyo", "Nagofa", "Osaka"]
5. ];
```

Cách thứ 2 để khai báo mảng hai chiều đó là sử dụng từ khóa *new* như minh họa trong ví dụ sau:

```
1. let cities = new Array(3);
2. for (let i = 0; i < 3; i++) {
3.   cities[i] = new Array(3);
4. }
5. cities[0][0] = "Hanoi";
6. cities[0][1] = "Saigon";
7. cities[0][2] = "DaNang";
8. cities[1][0] = "New York";
9. cities[1][1] = "California";
10. cities[1][2] = "Miami";
11. cities[2][0] = "Tokyo";
12. cities[2][1] = "Nagofa";
13. cities[2][2] = "Osaka";
```

Cả hai cách này đều cho kết quả giống nhau, nhưng cách thứ 2 này rườm rà hơn, do đó chúng ta thường sử dụng cách thứ nhất để khởi tạo mảng.

Duyệt mảng đa chiều

Để duyệt phần tử của mảng đa chiều, chúng ta sử dụng các vòng lặp lồng nhau. Ví dụ dưới đây sử dụng 2 vòng lặp lồng nhau để duyệt mảng hai chiều đã khai báo ở trên.

```
1. for (let i = 0; i < cities.length; i++) {
2.   for (let j = 0; j < cities[i].length; j++) {
3.     document.write(cities[i][j] + "<br>");
4.   }
5. }
```

Vòng for bên ngoài sẽ duyệt qua các dòng, trong ví dụ trên, vòng for bên ngoài sẽ duyệt qua 3 dòng (dòng 1: ["Ha Noi", "Sai Gon", "Da Nang"], dòng 2: ["New Your", "California", "Miami"], dòng 3: ["Tokyo", "Nagoya", "Osaka"]). Trong mỗi dòng (tương ứng với mảng một chiều) vòng for bên trong sẽ duyệt qua các phần tử của mảng một chiều (Ví dụ tại dòng 1, vòng for bên trong sẽ duyệt qua các phần tử "Ha Noi", "Sai Gon", "Da Nang").

6. Các thao tác thông dụng với mảng

Mảng là một cấu trúc được sử dụng nhiều trong các ứng dụng, và cũng có rất nhiều các thao tác khác nhau có thể thực hiện với mảng. Trong phần này, chúng ta sẽ cùng liệt kê một số thao tác cơ bản và thường thấy khi làm việc với mảng.

Khởi tạo giá trị ngẫu nhiên cho các phần tử

Ví dụ:

```
1. let matrix = new Array(10, 10);
2. for (let row = 0; row < matrix.length; row++) {
3.   for (let column = 0; column < matrix[row].length; column++) {
4.     matrix[row][column] = Math.floor((Math.random() * 100) + 1);
5.   }
6. }
```

Trong ví dụ này, chúng ta đã sử dụng hàm *random()* của lớp *Math* để sinh ra các số ngẫu nhiên nằm trong khoảng từ 0 đến 100.

Tính tổng các phần tử số

Ví dụ:

```
1. let total = 0;
2. for (let row = 0; row < matrix.length; row++) {
3.   for (let column = 0; column < matrix[row].length; column++) {
4.     total += matrix[row][column];
5.   }
6. }
```

Trong ví dụ này, biến *total* lưu trữ giá trị của tổng tất cả các phần tử. Ban đầu biến *total* có giá trị là 0, sau đó nó lần lượt "tích lũy" thêm giá trị của từng phần tử. Đến cuối cùng, giá trị của *total* chính là tổng của tất cả các phần tử.

Tính tổng các phần tử số theo từng cột

Ví dụ:

```
1. for (let column = 0; column < matrix[0].length; column++) {
2.   let total = 0;
3.   for (let row = 0; row < matrix.length; row++){
4.     total += matrix[row][column];
5.   }
6.   console.log("Sum for column " + column + " is " + total);
7. }
```

Trong ví dụ này, biến *total* được dùng để lưu trữ giá trị tổng của các phần tử trong từng cột. Khác với ví dụ trước đó, trong ví dụ này biến *total* được khai báo ở trong vòng lặp đầu tiên, và như vậy sau mỗi lần lặp thì biến này lại được khai báo lại và có giá trị là 0.

Tìm hàng có tổng lớn nhất

Ví dụ:

```
1. let maxRow = 0;
2. let indexOfMaxRow = 0;
3. for (let column = 0; column < matrix[0].length; column++) {
4.   maxRow += matrix[0][column];
5. }
6. for (let row = 1; row < matrix.length; row++) {
7.   let totalOfThisRow = 0;
8.   for (let column = 0; column < matrix[row].length; column++) {
9.     totalOfThisRow += matrix[row][column];
10.  }
11.  if (totalOfThisRow > maxRow) {
12.    maxRow = totalOfThisRow;
13.    indexOfMaxRow = row;
14.  }
15. }
16. console.log("Row " + indexOfMaxRow + " has the maximum sum of " + maxRow);
```

Trong ví dụ này, biến *maxRow* được dùng để lưu trữ giá trị tổng lớn nhất. Biến *indexOfMaxRow* được dùng để lưu trữ chỉ số của dòng có tổng giá trị lớn nhất. Biến *totalOfThisRow* được dùng để lưu trữ tổng giá trị của từng dòng.

Trộn ngẫu nhiên các phần tử

Ví dụ:

```
1. for (let row = 0; row < matrix.length; row++) {
2.   for (let column = 0; column < matrix[row].length; column++) {
3.     let randomRow = parseInt(Math.random() * matrix.length);
4.     let randomColumn = parseInt(Math.random() * matrix[row].length);
5.     let temp = matrix[row][column];
6.     matrix[row][column] = matrix[randomRow][randomColumn];
7.     matrix[randomRow][randomColumn] = temp;
8.   }
9. }
```

Trong ví dụ này, chúng ta sử dụng hàm `random()` của lớp `Math` để sinh ra ngẫu nhiên các giá trị dòng và cột. Sau đó thực hiện phép hoán đổi giá trị của các phần tử ngẫu nhiên tại vị trí dòng và cột đó.

Sao chép phần tử của mảng

Ví dụ:

```
1. let sourceArray = [2, 3, 1, 5, 10];
2. let targetArray = new Array(sourceArray.length);
3. for (let i = 0; i < sourceArray.length; i++) {
4.   targetArray[i] = sourceArray[i];
5. }
```

Trong ví dụ này, chúng ta tạo một mảng `targetArray` có độ dài bằng với độ dài của mảng `sourceArray`. Sau đó, chúng ta duyệt qua lần lượt từng phần tử của mảng `sourceArray` và gán giá trị của phần tử đó cho từng phần tử tương ứng của mảng `targetArray`.

7. Mã sạch

Quy tắc #1: Không sử dụng mảng quá nhiều chiều

Mảng càng nhiều chiều thì thuật toán để xử lý càng trở nên phức tạp, khó hiểu, khó kiểm soát. Do đó, cần suy xét rất cẩn thận để hạn chế việc sử dụng mảng quá nhiều chiều. Chẳng hạn, để duyệt qua một mảng có 3 chiều, thông thường chúng ta sẽ cần tới 3 vòng lặp lồng nhau, dẫn đến độ phức tạp của thuật toán là rất cao.

Trong những tình huống như vậy, chúng ta có thể áp dụng kỹ thuật làm phẳng (flatten) để giảm số chiều của mảng.

Quy tắc #2: Không lưu trữ các dữ liệu khác biệt vào trong một mảng

Chúng ta đều biết, mỗi phần tử của mảng được đối xử như một biến độc lập, do đó chúng có thể chứa bất kỳ một giá trị nào. Chẳng hạn, chúng ta có thể sử dụng một mảng để lưu tên, tuổi và địa chỉ của khách hàng, thậm chí là có thể lưu luôn mã hoá đơn, hay danh sách sản phẩm mà khách hàng ấy đã mua.

Như vậy, mảng của chúng ta sẽ có dạng như:

"John"	"Madrider"	20	"A8821"	"iPhone"	2	5/2019	true
--------	------------	----	---------	----------	---	--------	------

Nhìn vào mảng này, thật khó để có thể hình dung được dữ liệu mà chúng ta đang lưu trữ là gì. Và cũng thật khó để có thể duyệt và thực hiện các thao tác trên tập dữ liệu này. Do đó, chúng ta nên tránh sử dụng mảng vào những mục đích tương tự như vậy. Thay vào đó, chúng ta sử dụng đối tượng để biểu diễn dữ liệu thay cho mảng (Chúng ta sẽ tìm hiểu về đối tượng ở trong Chương 9).

Quy tắc #3: Tên mảng dạng số nhiều

Mảng là một biến đặc biệt, cho phép lưu trữ nhiều giá trị, do đó tên của mảng nên được đặt ở dạng số nhiều.

Ví dụ:

Không nên:

```
1. let user = ['John', 'Rose', 'Bob'];
```

Nên:

```
1. let users = ['John', 'Rose', 'Bob'];
```

8. Các lỗi thường gặp

Lỗi thường gặp #1: Chỉ số bắt đầu từ 1

Ví dụ:

Bài toán đặt ra, cho một mảng gồm 5 số nguyên, hiển thị ra màn hình tổng các số trong mảng.

```
1. let total = 0;
2. let numbers = [1, 2, 3, 4, 5];
3. let index = 0;
4. for (index = 1; index < numbers.length; index++) {
5.   total += numbers[index];
6. }
7. alert("Total is: " + total);
```

Kết quả hiển thị:

Total is: 14

Mục đích của đoạn mã trên là tính tổng các số trong mảng numbers. Các số từ 1 đến 5 sẽ có tổng là 15, nhưng kết quả hiển thị là 14. Bởi vì, khi sử dụng vòng lặp for để duyệt mảng thì index được bắt đầu từ 1, nhưng mảng luôn luôn bắt đầu từ vị trí 0, có nghĩa là mảng numbers được duyệt từ phần tử thứ hai `numbers[1]` thay vì phải bắt đầu từ `numbers[0]`, khi đó việc tính tổng các số trong mảng sẽ bị sai lệch.

Lỗi thường gặp #2: Nhầm lẫn chỉ số cuối cùng

Đôi khi trong lúc duyệt mảng chúng ta sẽ bị nhầm lẫn chỉ số cuối cùng của mảng, khi duyệt mảng từ vị trí 0 thay vì sử dụng `array.length - 1` chúng ta lại sử dụng `array.length`, sẽ dẫn đến dữ liệu tính toán bị sai lệch.

Ví dụ:

Bài toán đặt ra, cho một mảng gồm 5 số nguyên, hiển thị ra màn hình số cuối cùng trong mảng.

```
1. let numbers = [1, 2, 3, 4, 5];
2. alert("The last element is: " + numbers[numbers.length]);
```

Kết quả hiển thị:

The last element is: undefined

Trong đoạn mã trên `numbers.length` là 5, nhưng mảng luôn luôn bắt đầu từ `numbers[0]`, vậy phần tử cuối cùng phải là `numbers[4](numbers.length - 1)` khi đó `numbers[numbers.length]` sẽ có không có giá trị (*undefined*).

9. Bài thực hành

Bài 1: Tạo và thao tác với mảng

Mục tiêu:

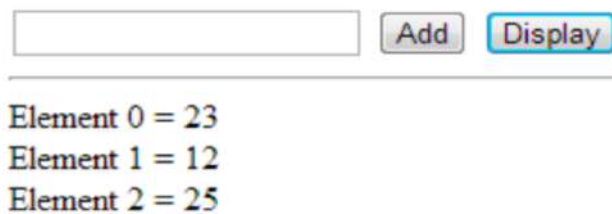
Luyện tập tạo mảng, thao tác với mảng.

Mô tả:

Viết một chương trình JavaScript để thêm vào các phần tử trong một mảng và hiển thị chúng.

Giao diện mẫu như sau:

Sample Screen:



The screenshot shows a web interface with a text input field, an 'Add' button, and a 'Display' button. Below the buttons, the array elements are displayed as follows:

```

Element 0 = 23
Element 1 = 12
Element 2 = 25
  
```

Hình 5.7: Minh họa chương trình thêm phần tử vào mảng

Hướng dẫn:

Bước 1: Tạo file `arrayElement.html`. Thêm mã lệnh html để tạo giao diện:

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <meta charset=utf-8 />
5.   <title>JS Bin</title>
6.   <style>
7.     body {padding-top:50px}
8.   </style>
9. </head>
10. <body>
11.   <input type="text" id="txtValue"></input>
12.   <input type="button"
13.     id="btnAdd"
14.     value="Add"
15.     onclick="add_element_to_array();"></input>
16.   <input type="button"
17.     id="btnDisplay"
18.     value="Display"
19.     onclick="display_array();"></input>
20.   <div id="result"></div>
21. </body>
22. </html>
  
```

Thêm sự kiện onclick cho button *Add* gọi đến hàm *add_element_to_array()* và *Display* gọi đến hàm *display_array()*.

Bước 2: Tạo biến *x* và khởi tạo bằng 0, *x* là chỉ số truy cập tới các phần tử trong mảng.

```
1. let x = 0;
```

Bước 3: Tạo mảng array để lưu các giá trị được nhập vào

```
1. let array = Array();
```

Bước 4: Xây dựng hàm *add_element_to_array()* để thêm mới một phần tử từ form Lấy giá trị từ input-text và gán phần tử ở vị trí *x* trong mảng mỗi lần hàm *add_element_to_array()* được gọi

Tăng giá trị *x* lên 1

Hiển thị phần tử vừa được thêm vào mảng

Gán giá trị rỗng cho input-text

Mã lệnh thực thi có thể như sau:

```
1. function add_element_to_array() {
2.   array[x] = document.getElementById("txtValue").value;
3.   alert("Element: " + array[x] + " Added at index " + x);
4.   x++;
5.   document.getElementById("txtValue").value = "";
6. }
```

Bước 5: Xây dựng hàm *display_array()* để hiển thị các phần tử trong mảng

Tạo biến *e* để lưu thẻ `<hr>` mỗi lần hiển thị một phần tử thẻ `<hr>` sử dụng để phân cách các phần tử trong mảng: `var e = "<hr/>"`

Dùng vòng lặp for duyệt toàn bộ mảng: `for (var i=0; i<array.length; i++)`

Mỗi lần duyệt lấy giá trị của các phần tử cộng dồn vào biến *e*. Thêm thẻ `<br/>` cuối mỗi lần duyệt.

Kết thúc lặp. Hiển thị biến *e*.

Mã lệnh thực thi có thể như sau:

```
1. function display_array() {
2.   let e = "<hr/>";
3.   for (let i = 0; i < array.length; i++) {
4.     e += "Element " + i + " = " + array[i] + "<br/>";
5.   }
6.   document.getElementById("result").innerHTML = e;
7. }
```

Bước 6: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-05/01-tao-va-thao-tac-voi-mang>

Bài 2: Đảo ngược các phần tử trong mảng

Mục tiêu: Luyện tập tạo mảng, thao tác với mảng.

Mô tả:

Viết một chương trình JavaScript thực hiện đảo ngược các giá trị trong một mảng đã cho. Không sử dụng phương thức reverse().

Ví dụ mảng gồm các phần tử:

`[-3,5,1,3,2,10];`

Sau khi gọi hàm reverse() mảng trên sẽ đảo ngược thành:

`[10,2,3,1,5,-3]`

Hướng dẫn:

Bước 1: Khai báo mảng x và khởi tạo các phần tử trong mảng:

```
1. let x = [-3, 5, 1, 3, 2, 10];
```

Bước 2: Khai báo biến first và gán giá trị bằng 0. Biến này để duyệt từ phần tử đầu tiên trong mảng.

```
1. let first = 0;
```

Bước 3: Khai báo biến last và gán giá trị bằng độ dài mảng trừ 1. Biến này để duyệt phần tử cuối cùng trong mảng.

```
1. let last = x.length - 1;
```

Bước 4: Dùng vòng lặp chạy với điều kiện first < last, trong quá trình lặp đổi chỗ phần tử đầu tiên và cuối cùng cho nhau. Sau đó tăng biến first lên 1 để duyệt phần tử tiếp theo, giảm last đi một.

```
1. while (first < last) {
2.   let b = x[first];
3.   x[first] = x[last];
4.   x[last] = b;
5.   first++;
6.   last--;
7. }
```

Bước 5: In ra mảng sau khi đảo ngược

```
1. document.write(x);
```

Bước 6: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-05/02-dao-nguoc-cac-phan-tu-trong-mang>

Bài 3: Tìm giá trị trong mảng

Mục tiêu: Luyện tập tạo mảng, thao tác với mảng.

Mô tả:

Viết một chương trình JavaScript để tìm một giá trị được nhập vào từ hộp thoại có nằm trong mảng hay không. Nếu có in ra vị trí của phần tử đó.

Hướng dẫn:

Bước 1: Nhập giá trị cần tìm từ hộp thoại và lưu vào biến value

```
1. let value = prompt("Enter a number: ");
```

Bước 2: Khai báo mảng x và khởi tạo giá trị ban đầu cho mảng

```
1. let x = [-3, 5, 1, 3, 2, 10];
```

Bước 3: Dùng vòng lặp duyệt toàn bộ mảng x. Kiểm tra nếu một phần tử trong mảng x bằng giá trị được nhập vào thì hiển thị thông báo:

```
1. for (let i = 0; i < x.length; i++) {
2.   if (value == x[i]) {
3.     alert("Value " + x[i] + "found at " + i);
4.   }
5. }
```

Bước 4: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-05/03-tim-gia-tri-trong-mang>

Bài 4: Tìm giá trị lớn nhất trong mảng

Mục tiêu:

Luyện tập tạo mảng, thao tác với mảng.

Mô tả:

Viết một chương trình JavaScript để tìm giá trị lớn nhất trong mảng cho trước.

Hướng dẫn:

Bước 1: Khai báo mảng x và khởi tạo giá trị ban đầu cho mảng

```
1. let x = [-3, 5, 1, 3, 2, 10];
```

Bước 2: Khai báo biến max để lưu giá trị lớn nhất mặc định là phần tử đầu tiên trong mảng

```
1. let max = x[0];
```

Bước 3: Khai báo biến index mặc định gán giá trị 0 để lưu chỉ số của phần tử lớn nhất trong mảng

```
1. let index = 0;
```

Bước 4: Dùng vòng lặp duyệt toàn bộ mảng x từ phần tử tiếp theo đến hết mảng. Kiểm tra nếu một phần tử trong mảng x có giá trị lớn hơn max thì gán max chính là phần tử đó và gán index bằng vị trí của phần tử đó.

```
1. for (let i = 1; i < x.length; i++) {
2.   if (x[i] > max) {
3.     max = x[i];
4.     index = i;
5.   }
6. }
```

Bước 5: In ra phần tử lớn nhất và vị trí của nó trong mảng

```
1. alert("max: " + max + " at position " + index);
```

Bước 6: Thực thi chương trình, quan sát kết quả.

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-05/04-tim-gia-tri-lon-nhat-trong-mang>

Bài 5: Tạo bàn cờ caro đơn giản

Mục tiêu:

Luyện tập mảng hai chiều với vòng lặp for, truy cập và thay đổi các phần tử trong mảng 2 chiều.

Mô tả:

Xây dựng game caro đơn giản. Bàn cờ hiển thị đơn giản gồm N dòng và M cột. Mỗi giá trị trong cột nhận giá trị mặc định là 0.

Caro Game Simple

```
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Change Value

Hình 5.8: Minh hoạ caro game 1

Khi người dùng click vào nút “Change Value” sẽ hiển thị hộp thoại yêu cầu nhập vị trí phần tử cần thay đổi và giá trị phần tử cần thay đổi. Sau khi thay đổi màn hình hiển thị lại bàn cờ như sau:

Caro Game Simple

```
0 0 0 0 0
0 x 0 0 0
0 0 0 x 0
0 0 0 0 0
0 0 0 0 0
```

Change Value

Hình 5.9: Minh hoạ caro game 2

Hướng dẫn:

Bước 1: Tạo file carosimple.html.

Bước 2: Tạo giao diện sử dụng thẻ html

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <body>
4.   <h3>Caro Game Simple</h3>
5.   <p id="carogame" />
6.   <input type="button" value="Change Value" onclick="changeValue()">
7. </body>
8. </html>
```

Bước 3: Hiển thị bàn cờ

```

1. let b = document.getElementById("carogame ");
2. let board = [];
3. let data = "";
4. for (let i = 0; i < 5; i++) {
5.   board[i] = new Array(0, 0, 0, 0, 0);
6. }
7. for (let i = 0; i < 5; i++) {
8.   data += "<br/>";
9.   for (let j = 0; j < 5; j++) {
10.    data += board[i][j] + " ";
11.   }
12. }
13. data += "<br/><br/><input type='button' value='Change Value' onclick='changeValue()'>"
14. b.innerHTML = data;

```

Bước 4: Xử lý sự kiện khi người dùng click chuột vào nút "Change Value"

```

1. function changeValue() {
2.   let positionX = prompt("X: ");
3.   let positionY = prompt("Y: ");
4.   data = "";
5.   board[positionX][positionY] = "x";
6.   for (let i = 0; i < 5; i++) {
7.     data += "<br/>";
8.     for (let j = 0; j < 5; j++) {
9.       data += board[i][j] + " ";
10.    }
11.   }
12.   data += "<br/><br/><input type='button' value='Change Value' onclick='changeValue()'>"
13.   b.innerHTML = "<hr/>" + data;
14. }
15.

```

Bước 5: Chạy chương trình và thay đổi vị trí trên bàn cờ. Quan sát kết quả

Mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/tree/dev/chapter-05/05-tao-ban-co-ca-ro-don-gian>

10. Bài tập

Bài 1: Chèn dấu (-) giữa 2 số chẵn

Hãy viết một chương trình nhận một số nhập vào và chèn dấu (-) giữa 2 số chẵn. Ví dụ nếu nhập vào 025468 thì kết quả của chương trình sẽ là 0-254-6-8.

Bài 2: Chuyển các ký tự thường thành ký tự hoa và ngược lại

Hãy viết một chương trình nhập vào một chuỗi và chuyển các ký tự chữ thường trong chuỗi vừa nhập sang thành dạng chữ hoa và ngược lại. Ví dụ: nếu nhập vào chuỗi 'The Quick Brown Fox' kết quả của chương trình là 'tHE qUICK bROWN fOX'.

Bài 3: Xóa phần tử khỏi mảng một chiều

Viết chương trình khởi tạo/nhập vào một mảng số nguyên gồm 10 phần tử, nhập/tạo phần tử số nguyên V. Chương trình kiểm tra xem V có thuộc mảng đã cho không, nếu V thuộc mảng đã cho xóa V khỏi mảng (bản chất việc xóa ở đây tức là dịch phần tử ở bên phải V sang vị trí của V, và gán 0 cho phần tử cuối cùng của mảng).

Bài 4: Sắp xếp mảng một chiều

Viết chương trình khởi tạo/nhập vào một mảng số nguyên gồm 10 phần tử. Chương trình sắp xếp mảng theo thứ tự giảm dần và hiển thị ra mảng đã được sắp xếp.

Bài 5: Ứng dụng từ điển đơn giản

Mô tả:

Hãy phát triển một ứng dụng từ điển đơn giản. Ứng dụng cho phép tra cứu các từ tiếng Anh sang tiếng Việt. Danh sách các từ được lưu trữ trong các mảng.

Hướng dẫn:

Bước 1: Tạo một trang web với một form đơn giản cho phép người dùng nhập từ cần tìm kiếm.

Bước 2: Viết mã JavaScript

- Tạo 2 mảng có độ dài bằng nhau để lưu trữ danh sách các từ. Mảng 1 lưu trữ các từ tiếng Anh, mảng 2 lưu trữ các từ tiếng Việt tương ứng.
- Khi tìm kiếm, tìm vị trí của từ tiếng Anh trong mảng 1. Nếu tìm thấy thì hiển thị từ tiếng Việt ở cùng vị trí trong mảng 2.
- Nếu không tìm thấy thì hiển thị thông báo không tìm thấy.

11. Bài kiểm tra

Câu 1: Mảng là một biến đặc biệt, chỉ có thể chứa một giá trị?

- Đúng
- Sai

Câu 2: Cú pháp đúng khai báo mảng?

- let arr = [item1; item2; ...];
- let arr = {item1; item2; ...};
- let arr = [item1, item2, ...];
- let arr = {item1, item2, ...};

Câu 3: Đây là khai báo mảng đúng?

- let car = ["Saab", "Volvo"];
- let car = ["Saab"; "Volvo"];
- let = {"Saab", "Volvo"};
- let car = new Array("Saab", "Volvo", "BMW");

Câu 4: Dự đoán kết quả đoạn mã sau:

```
1. let x = [1, 2, 4];
2. console.log(x);
```

- a) [1, 2, 4]
- b) []
- c) Không đáp án nào đúng
- d) [1, 2, 4, null]

Câu 5: Dự đoán kết quả đoạn mã sau:

```
1. let x = [1, 3, 5, 7];
2. let y = [2, 4, 6];
3. x[2] = y[x.length - 3];
4. console.log(x);
```

- a) Không đáp án nào đúng
- b) [1,3,5,7]
- c) [1,3,4,5]
- d) [1,3,4,7]

Câu 6: Dự đoán kết quả đoạn mã sau:

```
1. let x = [1, 3, 5, 7];
2. let y = [2, 4, 6];
3. x[2] = y[x.length - 3] + x[x[0] + y[0]];
4. console.log(x);
```

- a) Không đáp án nào đúng
- b) [2,3,11,7]
- c) [1,3,11,7]
- d) [1,3,11,13]

Câu 7: Đọc đoạn mã sau:

```
1. let iMax = 20;
2. let jMax = 10;
3. let f = new Array();
4. for (i = 0; i < iMax; i++) {
5.   f[i] = new Array();
6.   for (j = 0; j < jMax; j++) {
7.     f[i][j] = 0;
8.   }
9. }
```

Giá trị của *f.length* là gì?

- a) 20
- b) 200
- c) Đoạn mã có lỗi
- d) 10

Câu 8: Đoạn mã sau trả về kết quả gì?

```
1. let x = [2, 4, 6, 3, 7];
2. let y = x.length - 2;
3. console.log(x[y]);
```

- a) 4

- b) 3
- c) 6
- d) Đoạn mã có lỗi

Câu 9:

```
1. let x = [2, 4, 5];
2. let counter = 0;
3. if (x[counter] < 5) {
4.     console.log('code');
5. } else {
6.     console.log('gym');
7. }
```

Đoạn mã trên trả về kết quả gì?

- a) gym
- b) Đoạn mã có lỗi
- c) code
- d) Codegym

Câu 10: Dự đoán giá trị của x sau khi thực thi đoạn mã lệnh sau:

```
1. let a = new Array(12, false, "text");
2. x = 10;
3. if (a[1]) {
4.     x = 20;
5. } else x = 30;
```

- a) 10
- b) Đoạn mã có lỗi
- c) 20
- d) 30

Đáp án: Câu 1: b; Câu 2: c; Câu 3: a, d; Câu 4: a; Câu 5: d; Câu 6: c; Câu 7: a; Câu 8: b; Câu 9: c; Câu 10: d;

12. Tổng kết

- Mảng cho phép lưu trữ nhiều giá trị cùng kiểu
- Các khái niệm của mảng: Tên mảng, kiểu dữ liệu, kích thước, phần tử, chỉ số
- Tên của mảng tuân theo quy tắc của tên biến
- Chỉ số của phần tử đầu tiên là 0, chỉ số của phần tử cuối cùng là length – 1
- Có thể sử dụng vòng lặp for và for-each để duyệt mảng
- Để sao chép mảng thì cần sao chép lần lượt từng phần tử của mảng
- Mảng đa chiều được sử dụng phổ biến là 2 chiều
- Số lượng “chiều” của mảng bằng với số lượng chỉ số để truy xuất đến một phần tử của mảng
- Có thể sử dụng 2 vòng lặp lồng nhau để duyệt qua các phần tử của mảng hai chiều.

Chương 6 - Hàm

Tổ chức được mã nguồn dưới dạng các đơn vị để dễ quản lý và có thể tái sử dụng được

1. Mục tiêu

- Trình bày được khái niệm và mục đích của hàm
- Trình bày được cú pháp khai báo hàm
- Trình bày được cú pháp gọi hàm
- Giải thích được tham số của hàm
- Giải thích cách sử dụng câu lệnh return trong hàm
- Trình bày được phạm vi của biến
- Khai báo và sử dụng được hàm không tham số
- Khai báo và sử dụng được hàm có tham số
- Khai báo và sử dụng được hàm có return
- Khai báo và sử dụng được hàm đệ quy

2. Giới thiệu

Hàm là một khái niệm quen thuộc với hầu hết mọi người, kể cả những người mới bắt đầu học lập trình, bởi vì tên gọi này đã được chúng ta biết đến trước đó trong toán học.

Hãy cùng nhớ lại một vài hàm đơn giản mà chúng ta đã từng biết đến trong toán học, chẳng hạn như:

$$f(x) = x^2$$

$$g(x, y) = \sqrt{x} + \sqrt{y}$$

$$h(x, y) = f(x) + g(x, y)$$

- f, g, h được gọi là tên hàm
- x và y được gọi là các biến số
- $x^2, \sqrt{x} + \sqrt{y}$ và $f(x) + g(x, y)$ được gọi là các biểu thức

Chúng ta có thể đánh giá các hàm f, g và h với các giá trị cụ thể của biến số. Chẳng hạn như:

$$f(5) = 5^2 = 25$$

$$g(9, 25) = \sqrt{9} + \sqrt{25} = 3 + 5 = 8$$

$$h(9, 25) = 81 + 8 = 89$$

Có nghĩa là:

- Với biến $x = 5$ thì hàm f sẽ cho giá trị là 25

- Với biến $x = 9$ và $y = 25$ thì hàm g sẽ cho giá trị là 8
- Với biến $x = 9$ và $y = 25$ thì hàm h sẽ cho giá trị là 89

Chúng ta cũng dễ dàng nhận thấy có hai bước để làm việc với hàm đó là *định nghĩa hàm* và *sử dụng hàm*.

Trong lập trình, chúng ta cũng có một khái niệm tương tự như vậy. Khái niệm này cũng được gọi là hàm, nhưng tất nhiên là chúng ta sẽ không định nghĩa hàm bằng các ký hiệu như trong toán học. Nội dung của chương này sẽ đề cập đến hàm, cách khai báo hàm và sử dụng hàm.

Hoàn thành chương này, chúng ta có thể phát triển các ứng dụng trong đó có sử dụng hàm để giảm thiểu sự trùng lặp mã nguồn và phân tách các bài toán lớn thành các bài toán nhỏ hơn để dễ giải quyết.

3. Hàm

Hàm là gì?

Hàm là một khối lệnh được sử dụng để thực hiện một công việc nhất định. Trước khi sử dụng hàm thì chúng ta cần khai báo hàm.

Có hai loại hàm khác nhau:

1. Hàm được định nghĩa sẵn (built-in function)
2. Hàm do lập trình viên tự định nghĩa (user-defined function)

Ở các phần trước, chúng ta đã sử dụng một số hàm có sẵn trong JavaScript, chẳng hạn như hàm `console.log()`, `alert()`, `Math.pow()`, `Math.random()`... Để sử dụng các hàm được dựng sẵn trong JavaScript chúng ta chỉ cần gọi đến chúng với đầy đủ tham số được yêu cầu.

Lưu ý: Đôi khi giữa các tên gọi hàm (function), phương thức (method), thủ tục (procedure) có thể dùng thay thế cho nhau nhưng về bản chất ba khái niệm này có sự khác nhau.

Ví dụ:

Hàm `alert()` là một hàm có sẵn của JavaScript, mục đích của hàm này là hiển thị một hộp thông báo. Đoạn mã sau hiển thị một thông báo "Hello JavaScript" bằng cách gọi hàm `alert()`:

```
1. alert("Hello JavaScript");
```

Tương tự như vậy, đoạn mã sau gọi hàm `log()` của đối tượng `console` nhằm hiển thị dòng chữ Hello JavaScript trong cửa sổ console:

```
1. console.log("Hello JavaScript");
```

Các thành phần của hàm

Khi làm việc với hàm, chúng ta cần biết rõ các thành phần của hàm: tên hàm, nhiệm vụ của hàm, tham số đầu vào của hàm, giá trị trả về của hàm.

Tên hàm

Tên hàm là một định danh để đại diện cho hàm. Chúng ta sẽ cần định danh này để sử dụng được hàm. Tên của hàm cần tuân thủ quy tắc đặt tên của ngôn ngữ. Tên của hàm cũng nên là một động từ, bởi vì nó thực hiện một thao tác nào đó.

Tên không tốt	Tên tốt	Lý do
myInterest	calculateInterest	Bắt đầu bằng động từ
FindMaxValue	findMaxValue	Viết thường chữ đầu tiên
get_payment	getPayment	Sử dụng Camel Case

Nhiệm vụ của hàm

Mỗi hàm sẽ thực hiện một nhiệm vụ nhất định nào đó, do vậy, trước khi định nghĩa hàm thì chúng ta cần xác định rõ là hàm này nhằm thực hiện nhiệm vụ gì. Chẳng hạn:

- Hàm tính bình phương của một số
- Hàm tính lũy thừa a^n
- Hàm tính thể tích của hình trụ

Tham số đầu vào

Tham số đầu vào là các giá trị cần thiết để có thể thực hiện tính toán bên trong hàm. Chẳng hạn:

- Hàm tính bình phương của một số sẽ cần tham số đầu vào là một số
- Hàm tính lũy thừa a^n sẽ cần tham số đầu vào là cơ số a và số mũ n
- Hàm tính thể tích hình trụ cần tham số đầu vào là bán kính đáy và chiều cao trụ

Giá trị trả về

Sau khi thực hiện tính toán thì hàm có thể trả về một giá trị để đại diện cho kết quả thực thi. Không phải tất cả các hàm đều có giá trị trả về. Chẳng hạn:

- Hàm tính bình phương, hàm tính lũy thừa, hàm tính thể tích đều có giá trị trả về là một số
- Hàm `alert()` không có giá trị trả về
- Hàm `confirm()` có giá trị trả về là một chuỗi

Ví dụ: Khai báo hàm tính tổng của hai số

```
1. function sum(firstNumber, secondNumber) {
2.   return firstNumber + secondNumber;
3. }
```

Các thành phần của hàm được khai báo ở trên:

- Tên hàm là `sum`
- Nhiệm vụ của hàm là tính tổng 2 tham số được truyền vào
- Tham số đầu vào gồm 2 số: `firstNumber` và `secondNumber`
- Hàm trả về kết quả tổng 2 tham số

Khai báo hàm

Cú pháp:

```
1. function functionName(parameter1, parameter2, ..., parameterN) {
2.   //code to be executed
3. }
```

Trong đó:

- *functionName*: Tên hàm
- *parameter1, parameter2, parameter 3*: Danh sách các tham số
- *code to be executed*: phần thân hàm (các lệnh thực thi hàm)

Sử dụng hàm

Một hàm cần được *khai báo (declare)* trước khi được *gọi (call/invoked)*. Khi gọi hàm chúng ta cần chú ý đến yêu cầu tham số đầu vào. Khi hàm có trả về giá trị chúng ta sẽ khởi tạo một biến để lưu giá trị nhận được khi gọi hàm.

Ví dụ:

Khai báo hàm *sum()* để tính tổng hai số:

```
1. function sum(firstNumber, secondNumber) {
2.   return firstNumber + secondNumber;
3. }
```

Sử dụng hàm *sum()* đã được khai báo ở trên:

```
1. let total = sum(3, 4);
2. document.write(total); //Hiển thị 7 ra màn hình
```

Khi gọi *hàm sum()*, chúng ta truyền vào hai số 3 và 4, kết quả $3 + 4 = 7$ được tính toán bên trong hàm *sum()* và trả về được lưu trong biến *total*.

Hàm giúp tái sử dụng mã nguồn

Ví dụ:

Bài toán đặt ra, tính tổng các số từ 1 đến 10, tính tổng các số từ 20 đến 38, tính tổng các số từ 35 đến 55. Nếu không sử dụng hàm, chúng ta cần viết lặp đi lặp lại các đoạn mã tương tự nhau.

Tính tổng các số từ 1 đến 10:

```
1. let sum = 0;
2. for (let index = 1; index <= 10; index++) {
3.   sum += index;
4. }
5. console.log("Sum from 1 to 10 is " + sum);
```

Tính tổng các số từ 20 đến 38:

```
1. let sum = 0;
2. for (let index = 20; index <= 38; index++) {
3.   sum += index;
4. }
5. console.log("Sum from 20 to 38 is " + sum);
```

Tính tổng các số từ 35 đến 55:

```
1. sum = 0;
2. for (let index = 35; index <= 55; index++) {
3.     sum += index;
4. }
5. console.log("Sum from 35 to 55 is " + sum);
```

Đoạn mã này thực thi tốt và đáp ứng được yêu cầu được đưa ra, tuy nhiên lại xuất hiện rất nhiều dòng mã lặp gần giống nhau. Để tiết kiệm công sức và giúp cho mã nguồn trở nên ngắn gọn và khoa học hơn, chúng ta thử sử dụng hàm trong trường hợp này.

Khai báo hàm *sum()* tính tổng các số nằm trong khoảng từ *startNumber* đến *endNumber*:

```
1. function sum(startNumber, endNumber) {
2.     let total = 0;
3.     for (let index = startNumber; index <= endNumber; index++) {
4.         total += index;
5.     }
6.     return total;
7. }
```

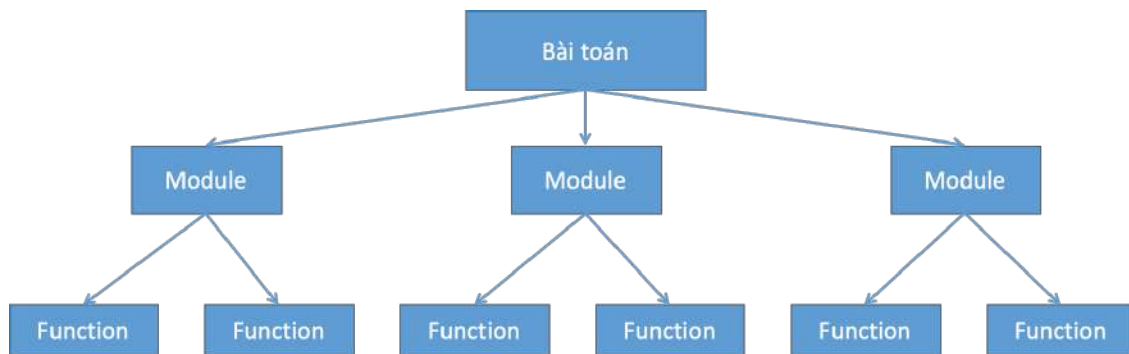
Lần lượt sử dụng hàm *sum* với các tham số khác nhau:

```
1. document.write(sum(1, 10) + "<br>"); //Kết quả: 55
2. document.write(sum(20, 38) + "<br>"); //Kết quả: 552
3. document.write(sum(35, 55) + "<br>"); //Kết quả: 945
```

Như vậy, chúng ta có thể dễ dàng nhận thấy rằng việc sử dụng hàm đã giúp mã nguồn ngắn gọn hơn, tránh trùng lặp và quan trọng hơn hết là chúng ta có thể tái sử dụng một cách rất dễ dàng.

Chiến thuật chia để trị

Hình minh họa dưới đây thể hiện chiến thuật chia để trị nhằm giải quyết các bài toán lớn. Một bài toán hay một chương trình cần giải quyết một nhiệm vụ khá lớn, bao gồm trong đó nhiều công việc phức tạp khác nhau (từng module), từ các module đó chúng ta có thể tách ra từng công việc nhỏ cụ thể được đưa vào hàm. Khi đó chúng ta sẽ tập trung viết mã lệnh để đi giải quyết từng bài toán nhỏ thay vì viết một khối mã lệnh lớn để giải quyết bài toán tổng thể.



Hàm là chiếc hộp đen

Có thể hình dung hàm như là những chiếc hộp đen có công dụng thực hiện các nhiệm vụ nhất định. Đôi khi người sử dụng hàm không cần quan tâm đến bên trong chiếc hộp, chỉ quan tâm đến đầu vào và đầu ra.



Hình: Hàm là chiếc hộp đen

4. Giá trị trả về của hàm

Khi hàm kết thúc quá trình thực thi, nó có thể trả về giá trị cho nơi gọi hàm bằng cách sử dụng từ khóa *return*.

Khi tạo ra một hàm mới, tùy vào mục đích của hàm mà bạn có thể quyết định hàm đó có trả về một giá trị hay không.

Giá trị trả về là kết quả sẽ được trả về tại vị trí hàm được gọi. Mỗi một hàm chỉ có một giá trị trả về duy nhất. Các giá trị này có thể là một biến, một mảng hay một đối tượng, danh sách đối tượng.

Ví dụ:

```

1. function myFunction(a, b) {
2.   return a * b; // hàm trả về giá trị tích của a và b
3. }
4. let x = myFunction(4, 3); // Hàm được gọi, giá trị trả về sẽ được gán cho biến x
  
```

Trong trường hợp hàm không có giá trị trả về (không sử dụng *return*), khi sử dụng hàm, chúng ta sẽ nhận về kết quả có giá trị *undefined*.

Ví dụ:

```

1. function myFunction(a, b) {
2.   const c = a * b; // dòng lệnh này giá trị tích của a và b nhưng không sử dụng return để trả về kết quả
3. }
4. let x = myFunction(4, 3); // x có giá trị là undefined
  
```

Chú ý: Khi gặp câu lệnh *return*, hàm sẽ trả về giá trị ngay tại thời điểm đó. Tất cả câu lệnh ở phía sau dòng lệnh *return* sẽ được bỏ qua.

Ví dụ:

```

1. function printLines() {
2.   console.log('First line');
3.   console.log('Second line');
4.   return;
5.   console.log('Third line');
6. }

```

Khi gọi hàm `printLines()` ở trên, dòng cuối cùng sẽ không bao giờ được thực thi, bởi vì câu lệnh `return` đã chấm dứt việc thực thi ở dòng trước đó. Do đó, chuỗi "Third line" sẽ không bao giờ được in ra.

5. Tham số của hàm

Tham số (parameter)

Tham số (còn được gọi đầy đủ là tham số hình thức – formal parameter) là các biến được khai báo trong phần header khi khai báo hàm.

Đối số (argument)

Khi gọi hàm thì các giá trị của các biến được khai báo trong phần header sẽ được truyền vào. Các giá trị này được gọi là tham số thực (actual parameter) hoặc đối số (argument).

Ví dụ:

```

1. function isEven(number){
2.   return number % 2 == 0;
3. }
4. isEven(5);

```

Ở đoạn mã trên, biến `number` được gọi là tham số, giá trị 5 được gọi là đối số.

6. Phạm vi của biến

Phạm vi (scope) của biến là các vị trí trong chương trình mà một biến có thể được sử dụng. Một biến được khai báo trong một khối lệnh thì được gọi là *biến địa phương* (*local variable*). Phạm vi của biến địa phương bắt đầu từ vị trí nó được khai báo cho đến điểm kết thúc của khối lệnh chứa nó. Biến được khai báo trong một khối lệnh thì không thể được truy cập từ bên ngoài khối.

Ví dụ:

```

1. let country = "Việt Nam";
2. if (country === "Việt Nam") {
3.   let capital = "Hà Nội";
4.   console.log("Trong khối, country: ", country); // Việt Nam
5.   console.log("Trong khối, capital: ", capital); // Hà Nội
6. }
7. console.log("Ngoài khối, country: ", country); // Việt Nam
8. console.log("Ngoài khối, capital: ", capital); // undefined

```

Ở đoạn mã trên, biến *capital* được khai báo bên trong khối lệnh *if*, do đó nó chỉ có phạm vi ở bên trong khối lệnh đó. Việc sử dụng biến *capital* ở ngoài khối lệnh *if* là không đúng, do đó kết quả in ra của câu lệnh đó là *undefined*.

Phạm vi hoạt động của biến trong vòng lặp for

Đối với vòng lặp *for*, biến được khai báo trong phần khởi tạo của vòng lặp thì có phạm vi trong toàn bộ vòng lặp. Biến được khai báo trong phần thân của vòng lặp thì chỉ có phạm vi bên trong thân vòng lặp (tính từ vị trí được khai báo cho đến hết khối lệnh chứa nó).

Ví dụ:

```
1. for(let i = 0; i < 10; i++) { //Điểm bắt đầu phạm vi biến i
2.   //...
3.   //...
4.   let j; //Điểm bắt đầu phạm vi biến j
5.   //...
6.   //...
7.   //...
8. } //Điểm kết thúc phạm vi biến i và j
```

Phạm vi của tham số của hàm

Tham số của hàm cũng là các biến địa phương. Phạm vi của các tham số là trong toàn bộ hàm đó. Mã trong hàm *capitalOf* dưới đây sử dụng được biến địa phương *country*, nhưng mã bên ngoài hàm thì không:

```
1. function capitalOf(country) {
2.   let capital;
3.   if (country === "Việt Nam") {
4.     capital = "Hà Nội";
5.   }
6.   return capital;
7. }
8.
9. console.log(capitalOf("Việt Nam")); // Hà Nội
10. console.log(country); // undefined
```

Ở đoạn mã trên, không thể sử dụng được biến *country* ở bên ngoài hàm *capitalOf()*, do đó kết quả hiển thị sẽ là *undefined*.

7. Hàm đệ quy

Hàm đệ quy (recursive function) là hàm mà có thực hiện lời gọi đến chính nó. Loại hàm này được sử dụng phổ biến trong những tình huống mà chúng ta thực hiện cùng một thao tác nhưng trên các đối tượng khác nhau được phái sinh từ đối tượng trước đó.

Ví dụ:

Sử dụng hàm đệ quy để tính giai thừa của một số.

Công thức tính giai thừa:

$$0! = 1$$

$$n! = n \times (n - 1)!; n > 0$$

Với một số n bất kỳ, việc tính giai thừa của n sẽ được thực hiện như thế nào?

- Nếu $n = 0$ thì trả về kết quả là 1
- Nếu $n = 1$ thì trả về kết quả là 1 (bởi vì $1! = 1 \times 0!$)
- Nếu $n = 2$ thì trả về kết quả là 2 (bởi vì $2! = 2 \times 1!$)
- ...
- Xét một cách tổng quát, bài toán tính $n!$ sẽ được đưa về thành bài toán tính $(n - 1)!$

Hàm tính giai thừa của n có thể được triển khai đơn giản như sau:

```
1. function factorial(n) {
2.   if (n === 0) {
3.     return 1;
4.   }
5.   return n * factorial(n - 1);
6. }
```

Trong đoạn mã trên, chúng ta dễ dàng nhận thấy rằng hàm `factorial()` đã thực hiện lời gọi đến chính nó. Biểu thức `n === 0` được gọi là *điều kiện dừng (stopping condition)* hoặc là *trường hợp cơ sở (base case)*.

Lưu ý: Chúng ta hoàn toàn có thể sử dụng vòng lặp để tính giai thừa của một số. Ví dụ được nêu ra ở đây là để giúp chúng ta dễ hình dung về khái niệm hàm đệ quy.

Lưu ý: Cần rất thận trọng khi đặt điều kiện dừng, bởi vì nếu không có điều kiện dừng hoặc điều kiện dừng không đúng thì có thể dẫn đến tình huống là hàm sẽ được thực thi vô tận. Thử hình dung xem điều gì sẽ diễn ra nếu chúng ta loại bỏ khối lệnh `if` trong hàm `factorial()` ở trên.

Chúng ta sẽ còn quay lại làm việc với hàm đệ quy ở trong phần Thuật toán tìm kiếm nhị phân của Chương 7.

8. Mã sạch

Quy tắc #1: Tên hàm cần có ý nghĩa

Như đã biết, hàm được sinh ra để thực hiện một công việc nhất định. Chính vì thế, tên hàm cần phải là một động từ và mô tả rõ nghĩa nhất về hành động được thực hiện bên trong đó.

Ví dụ:

Không nên:

```
1. function m(firstNumber, secondNumber) {
2.   // Đoạn mã tính tổng hai tham số
3. }
```

Khi nhìn vào tên hàm bên trên chúng ta không thể biết ngay được hàm `m()` sẽ thực hiện công việc, hành động gì nếu như chúng ta không đọc vào phần mã bên trong.

Gây ra việc khó sử dụng khi truyền tham số khi gọi. Để khắc phục lỗi này, chúng ta cần tìm một tên có ý nghĩa để đặt cho hàm này.

Nên:

```
1. function sum(firstNumber, secondNumber) {
2.   // Đoạn mã tính tổng hai tham số
3. }
```

Quy tắc #2: Một hàm chỉ thực hiện một công việc duy nhất

Tránh trường hợp bên trong hàm thực hiện nhiều hơn 2 hành động, như vậy mục đích của hàm được tạo ra sẽ bị sai lệch về ý nghĩa. Điều này sẽ dẫn đến việc khó kiểm soát và khó bảo trì về sau.

Ví dụ:

Hàm *getDaysOfMonth()* sau đây được khai báo để trả về số ngày trong tháng.

```
1. function getDaysOfMonth(month, year) {
2.   switch(month) {
3.     case 1:
4.     case 3:
5.     case 5:
6.     case 7:
7.     case 8:
8.     case 10:
9.     case 12:
10.      return 31;
11.    case 4:
12.    case 6:
13.    case 9:
14.    case 11:
15.      return 30;
16.    case 2:
17.      let isLeapYear = false;
18.      if(year % 4 === 0) {
19.        if(year % 100 === 0) {
20.          if(year % 400 === 0) {
21.            isLeapYear = true;
22.          }
23.        } else {
24.          isLeapYear = true;
25.        }
26.      }
27.      if(isLeapYear) {
28.        return 29;
29.      }
30.      return 28;
31.    }
32. }
```

Thoạt nhìn thì chúng ta thấy hàm này được triển khai khá tốt, và trả về kết quả đúng như mong đợi. Tuy nhiên, xét kỹ hơn thì chúng ta lại thấy hàm này đang làm hai nhiệm vụ: tính số ngày trong tháng và kiểm tra năm nhuận. Điều này là không tốt, do đó chúng ta cần tách hàm này thành các hàm nhỏ hơn.

```

1. function isLeapYear(year) {
2.   if(year % 4 === 0) {
3.     if(year % 100 === 0) {
4.       if(year % 400 === 0) {
5.         return true;
6.       }
7.     } else {
8.       return true;
9.     }
10.  }
11.  return false;
12. }
13.
14. function getDaysOfMonth(month, year) {
15.   switch(month) {
16.     case 1:
17.     case 3:
18.     case 5:
19.     case 7:
20.     case 8:
21.     case 10:
22.     case 12:
23.       return 31;
24.     case 4:
25.     case 6:
26.     case 9:
27.     case 11:
28.       return 30;
29.     case 2:
30.       if(isLeapYear(year)) {
31.         return 29;
32.       }
33.       return 28;
34.     }
35.   }

```

Ở đoạn mã sau khi đã tái cấu trúc, chúng ta có thể thấy rằng nhiệm vụ kiểm tra năm nhuận đã được tách ra thành một hàm mới là *isLeapYear()*, và như vậy thì hàm *getDaysOfMonth()* bây giờ chỉ còn thực hiện một nhiệm vụ là tính số ngày trong tháng mà thôi.

9. Các lỗi thường gặp

Lỗi thường gặp #1: Gọi một hàm không tồn tại

Những người mới học lập trình thường gặp phải lỗi này: gọi một hàm mà hàm đó chưa được định nghĩa. Có nhiều nguyên nhân dẫn đến trường hợp này, chẳng hạn như:

- Hàm thực sự chưa được khai báo
- Chưa nhập (import) file có chứa phần khai báo hàm
- Gọi tên hàm bị sai (đôi khi chỉ là một ký tự viết hoa hoặc viết thường)

Để xử lý những tình huống này, cần phải dò xét lại cẩn thận để phát hiện nguyên nhân (như đã liệt kê ở trên) và khắc phục.

Lỗi thường gặp #2: Điều kiện dừng của hàm đệ quy không đúng

Ví dụ:

Hàm tính giai thừa được triển khai như sau:

```
1. function factorial(n) {
2.   if (n === 2) {
3.     return 2;
4.   }
5.   return n * factorial(n - 1);
6. }
```

Với cách triển khai như thế này, mọi chuyện vẫn diễn ra suôn sẻ nếu chúng ta thực hiện các lời gọi hàm như factorial(5), factorial(10), factorial(2)..

Nhưng sẽ là vấn đề lớn nếu chúng ta gọi hàm factorial() với tham số nhỏ hơn 2, chẳng hạn như là factorial(1) hoặc factorial(0). Trong những trường hợp này, hàm factorial() sẽ được thực thi vô hạn.

Lỗi thường gặp #3: Quên trả về kết quả trong hàm

Với hàm cần trả về giá trị để tái sử dụng, người mới học lập trình thường gặp lỗi quên sử dụng lệnh return để trả về kết quả sau khi tính toán.

Ví dụ:

```
1. function sum(firstNumber, secondNumber) {
2.   const result = firstNumber + secondNumber // Đoạn mã tính tổng hai tham số
3.   // Không return giá trị biến result khi kết thúc hàm
4. }
5.
6. console.log(sum(4,6)); // Kết quả sử dụng hàm sum sai
```

10. Bài thực hành

Bài 1: Chuyển đổi nhiệt độ

Mục tiêu:

Luyện tập xây dựng và sử dụng hàm.

Mô tả:

Xây dựng hàm để chuyển đổi từ độ F sang độ C theo công thức sau:

$$C = (F - 32)/1.8$$

Giao diện mẫu của ứng dụng như sau:

Temperature Converter

Type a value in the Fahrenheit field to convert the value to Celcius:

Fahrenheit

Celcius: -17.22222222222222

Hướng dẫn:**Bước 1:** Thêm mã HTML tạo form

```

1. <p>
2. <label>Fahrenheit</label>
3. <input id="inputFahrenheit" type="number" placeholder="Fahrenheit"
4.   oninput="temperatureConverter(this.value)"
5.   onchange="temperatureConverter(this.value)">
6. </p>
7. <p>Celsius: <span id="outputCelsius"></span></p>

```

Bước 2: Xây dựng hàm temperatureConverter()

Hàm sẽ nhận tham số đầu vào là giá trị cần chuyển đổi. Thực hiện chuyển đổi từ độ F sang C theo công thức $C = (F - 32) / 1.8$

```

1. function temperatureConverter(valNum) {
2.     valNum = parseFloat(valNum);
3.     document.getElementById("outputCelsius").innerHTML = (valNum-32) / 1.8;
4. }

```

Mã nguồn hoàn chỉnh như sau:

```

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   <meta charset="UTF-8">
5.   <title>Title</title>
6. </head>
7. <body>
8.
9. <form>
10.  <p>
11.    <label>Fahrenheit</label>
12.    <input id="inputFahrenheit" type="number" placeholder="Fahrenheit"
13.      oninput="temperatureConverter(this.value)"
14.      onchange="temperatureConverter(this.value)">
15.  </p>
16.  <p>Celsius: <span id="outputCelsius"></span></p>
17. </form>
18.
19. <script>
20.
21.   function temperatureConverter(valNum) {
22.       valNum = parseFloat(valNum);
23.       document.getElementById("outputCelsius").innerHTML = (valNum-32) / 1.8;
24.   }
25.
26. </script>
27. </body>
28. </html>

```

Bước 3: Chạy chương trình và quan sát kết quả

Link mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-06/exercise-01-chuyen-doi-nhiệt-do.html>

Bài 2: Tìm giá trị nhỏ nhất của mảng

Mục tiêu:

Luyện tập tạo và sử dụng hàm, truyền mảng vào hàm.

Mô tả:

Xây dựng hàm nhận vào một tham số là một mảng cho trước. Hàm thực hiện tìm giá trị nhỏ nhất trong mảng và trả về giá trị đó. Sử dụng hàm vừa xây dựng trên với mảng như sau:

arr1: [3, 5, 1, 8, -3, 7, 8]

arr2: [7, 12, 6, 9, 20, 56, 89]

arr3: []

arr4: [0, 0, 0, 0, 0, 0]

Hướng dẫn:

Bước 1: Xây dựng hàm

Các bước thực thi trong hàm:

- Giả sử phần tử đầu tiên trong mảng là giá trị nhỏ nhất. Gán giá trị của phần tử này cho biến min.
- Duyệt mảng từ phần tử tiếp theo. Kiểm tra nếu giá trị của phần tử tiếp theo trong mảng nhỏ hơn min thì gán min = arr[i] (với i ở vị trí của phần tử tiếp theo)
- Kết thúc duyệt mảng, tìm được giá trị min trong mảng. Hàm trả về giá trị min đó.

Code mẫu như sau:

```
1. function minArray(arr) {  
2.   let min = arr[0];  
3.   for(let i = 1; i < arr.length; i++){  
4.     if(arr[i] < min){  
5.       min = arr[i];  
6.     }  
7.   }  
8.   return min;  
9. }
```

Bước 2: Sử dụng hàm

```
1. let arr1 = [3, 5, 1, 8, -3, 7, 8];  
2. let min = minArray(arr1);  
3. alert(min);
```

Bước 3: Chạy chương trình, quan sát kết quả.

Bước 4: Tương tự với mảng được khởi tạo các phần tử mặc định

arr2: [7, 12, 6, 9, 20, 56, 89]

arr3: []

arr4: [0, 0, 0, 0, 0, 0]

Bước 5: Với mảng không có phần tử nào sửa lại hàm minArray để kiểm tra số phần tử trong mảng.

Nếu độ dài mảng bằng 0. Thì trả về -1.

```

1. function minArray(arr) {
2.   if(arr.length == 0)
3.     return -1;
4.   let min = arr[0];
5.
6.   for(let i = 1; i < arr.length; i++){
7.     if(arr[i] < min){
8.       min = arr[i];
9.     }
10.  }
11.  return min;
12. }

```

Vậy khi hàm trả về giá trị -1 ta hiểu rằng mảng này đang rỗng.

Hỏi nhanh:

Tuy nhiên có trường hợp xảy ra là nếu trong mảng có phần tử -1 là phần tử nhỏ nhất thì sẽ xử lý thế nào?

Link mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-06/exercise-02-tim-gia-tri-nho-nhat.html>

11. Bài tập

Bài 1: Kiểm tra số nguyên tố

Hãy xây dựng chương trình để kiểm tra xem một số nguyên bất kỳ có phải là số nguyên tố hay không. Sau đó tìm tất cả các số nguyên tố nhỏ hơn 10000 và hiển thị ra màn hình.

Lưu ý: Số nguyên tố là số tự nhiên khác 0 chỉ có hai ước số dương phân biệt là 1 và chính nó.

Do số 1 chỉ có một ước số dương là chính nó, nên số 1 không phải là số nguyên tố.

Ví dụ, các số sau đây là số nguyên tố:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97...

Bài 2: Ứng dụng chuyển đổi giữa feet và meter

Hãy xây dựng chương trình cho phép chuyển đổi giữa feet sang meter và ngược lại.

Công thức chuyển đổi như sau:

meter = 0.305 * foot

foot = 3.279 * meter

Sử dụng các giá trị trong bảng sau để kiểm tra tính chính xác của chương trình:

Feet	Meters		Meters	Feet
1.0	0.305		20.0	65.574
2.0	0.610		25.0	81.967
...				
9.0	2.745		60.0	196.721
10.0	3.050		65.0	213.115

Bài 3: Tìm số nhỏ nhất trong 3 số

Hãy viết hàm nhận vào 3 số nguyên bất kỳ, trả về số nguyên có giá trị nhỏ nhất.

Bài 4: Tính thể tích hình trụ

Hãy viết một hàm để tính thể tích của hình trụ. Các tham số truyền vào là: bán kính đáy và chiều cao của hình trụ.

12. Bài kiểm tra

Câu 1: Đây là định nghĩa đúng về một hàm trong JavaScript?

- a. `function.FunctionName()`
- b. `function = FunctionName()`
- c. `new FunctionName()`
- d. `function FunctionName()`

Câu 2: Gọi hàm có tên là `myFunction()` như thế nào?

- a. `myFunction();`
- b. `call myFunction();`
- c. `myFunction;`
- d. `exe myFunction();`

Câu 3: Đây là những mô tả đúng về hàm được khai báo dưới đây:

```
function add(a, b) {
    return a + b;
}
```

- a. Hàm thiếu kiểu dữ liệu trả về
- b. Hàm có 2 tham số
- c. Hàm trả về số nguyên là tổng của hai tham số truyền vào
- d. Hàm có tên là add

Câu 4: Cho hàm `func1` như sau:

```
1. function func1(a) {
2.   if (a == 2)
3.     return 2;
4.   return a * func1(a - 1);
5. }
```

Hãy phân tích và xác định giá trị trả về khi gọi hàm func1 với tham số truyền vào là 5.

- a. 2
- b. 120
- c. 24
- d. 5

Câu 5: Một hàm trong JavaScript bắt buộc phải có lệnh return để trả về kết quả?

- a. Đúng
- b. Sai

Câu 6: Hãy phân tích và chỉ ra kết quả thực thi đoạn mã sau:

```
1. function swap(a, b) {  
2.   let temp = a;  
3.   a = b;  
4.   b = temp;  
5. }  
6. let a = 5;  
7. let b = 10;  
8. swap(a, b);  
9. document.write("a = " + a + ", b = " + b);
```

- a. a = 5, b = 5
- b. a = 5, b = 10
- c. a = 10, b = 10
- d. a = 10, b = 5

Câu 7: Xác định kết quả của biến sum sau khi đoạn mã sau được thực thi:

```
1. function add(a, b) {  
2.   return a + b;  
3. }  
4. let sum = add("4", "3");
```

- a. 43
- b. 7
- c. Xảy ra lỗi
- d. not defined

Câu 8: Mô tả nào không đúng về hàm sau:

```
1. function add(a, b) {  
2.   return a + b;  
3. }
```

- a. hàm không có dữ liệu trả về
- b. hàm sai cú pháp
- c. chỉ nhận vào tham số dạng số
- d. hàm có 2 tham số

Câu 9: Gán kết quả của hàm có tên là myFunction, truyền cho hàm tham số là 2 cho biến result trong Javascript ta sử dụng cú pháp nào?

- a. result = exe myFunction(2);
- b. result(2) = myFunction();
- c. result(2) = myFunction;
- d. result = myFunction(2);

Câu 10: Để gọi hàm myFunction có 2 tham số, ta sử dụng cú pháp nào?

- a. myFunction[a, b];
- b. exe myFunction(a, b);
- c. myFunction(a, b);
- d. myFunction(a; b);

Đáp án: Câu 1: d; Câu 2: a; Câu 3: a, b, d; Câu 4: b; Câu 5: b; Câu 6: b; Câu 7: a; Câu 8: a, b, c; Câu 9: d; Câu 10: c;

13. Tổng kết

- Hàm là một khối bao gồm nhiều dòng lệnh nhằm thực thi một tác vụ nhất định
- Hàm giúp tái sử dụng mã nguồn
- Tên, danh sách tham số và giá trị trả về của hàm là các yếu tố cần cân nhắc khi định nghĩa một hàm
- Đặt tên hàm thể hiện rõ ý là một thao tác rất quan trọng để đảm bảo mã sạch
- Tham số là các biến hình thức, đối số là các giá trị được truyền vào khi gọi hàm.
- Câu lệnh return được sử dụng để trả về giá trị của một hàm.
- Phạm vi của biến là các vị trí trong chương trình mà một biến có thể được sử dụng.

Chương 7 - Thuật toán tìm kiếm

Thực hiện được các thao tác tìm kiếm dữ liệu một cách hiệu quả

1. Mục tiêu

- Trình bày được mục đích của các thuật toán tìm kiếm
- Trình bày được ý tưởng và các bước thực hiện của thuật toán tìm kiếm tuyến tính
- Triển khai được thuật toán tìm kiếm tuyến tính trên mảng
- Trình bày được ý tưởng và các bước thực hiện của thuật toán tìm kiếm tuyến tính
- Triển khai được thuật toán tìm kiếm nhị phân trên mảng
- Tính toán được độ phức tạp của thuật toán tuyến tính và tìm kiếm nhị phân
- Phân biệt được các tình huống nên sử dụng thuật toán tìm kiếm tuyến tính và tìm kiếm nhị phân

2. Giới thiệu

Tìm kiếm là một trong những thao tác căn bản nhất khi làm việc với dữ liệu. Tìm kiếm khách hàng, tìm kiếm sản phẩm hoặc tìm kiếm các thông tin khác nói chung đều là những tính năng mà ta bắt gặp hằng ngày ở các phần mềm. Ở mức độ giải thuật cũng vậy, các lập trình viên thường xuyên tiếp xúc với những tình huống mà ở đó cần đến việc triển khai thuật toán tìm kiếm: tìm kiếm xem liệu dữ liệu có tồn tại hay không, tìm kiếm vị trí của dữ liệu, tìm kiếm dữ liệu dựa trên một đặc điểm nhất định, v.v.

Ở trong chương này, chúng ta sẽ thảo luận về một số thuật toán tìm kiếm căn bản nhất. Dựa trên các thuật toán này, về sau chúng ta có thể triển khai thêm các thuật toán tìm kiếm phù hợp để đáp ứng được yêu cầu của tình huống cụ thể. Hai thuật toán tìm kiếm mà chúng ta sẽ đề cập đến đó là tìm kiếm tuyến tính và tìm kiếm nhị phân.

Hoàn thành chương này, chúng ta có thể xây dựng được các tính năng cho các ứng dụng phần mềm mà ở đó cần triển khai thao tác tìm kiếm dựa trên các tiêu chí khác nhau.

3. Tìm kiếm tuyến tính

Tìm kiếm tuyến tính (linear search) là thuật toán mà ở đó chúng ta duyệt dữ liệu lần lượt từ đầu đến cuối để tìm ra phần tử phù hợp với yêu cầu đặt ra. Thuật toán này thuộc nhóm “vét cạn”: chúng ta sẽ cố gắng duyệt và kiểm tra lần lượt tất cả các trường hợp.

Khi thao tác với mảng, việc triển khai thuật toán tìm kiếm tuyến tính khá đơn giản, chỉ cần bắt đầu một vòng lặp ở đầu danh sách và so sánh mỗi phần tử với dữ liệu bạn đang tìm kiếm. Nếu tìm thấy một phần tử phù hợp, quá trình tìm kiếm sẽ kết thúc. Nếu đã duyệt tới cuối danh sách mà không có phần tử nào khớp với giá trị cần tìm thì kết luận dữ liệu tìm kiếm không có trong danh sách.

Giải thuật

- Bắt đầu từ phần tử đầu tiên, so sánh với giá trị muốn tìm, nếu bằng giá trị muốn tìm thì trả về vị trí hiện tại còn nếu không bằng thì chuyển sang phần tử kế tiếp.
- Nếu đến phần tử cuối cùng mà không tìm thấy giá trị nào bằng thì nghĩa là không tìm thấy.

Mã giả

Đầu vào: mảng a có N phần tử và giá trị x là giá trị muốn tìm kiếm

Đầu ra: Trả về vị trí nếu tìm thấy phần tử bằng x , ngược lại trả về -1

Bước 1: $i = 0$ // bắt đầu từ phần tử đầu tiên của dãy

Bước 2: So sánh $a[i]$ với x , có 2 khả năng

- $a[i] = x$: Tìm thấy. Trả về i và dừng lại
- $a[i] \neq x$: Sang bước 3

Bước 3:

- $i = i + 1$ // xét tiếp phần tử kế trong mảng
- Nếu $i \geq N$: Hết mảng, trả về -1 . Ngược lại: Lặp lại Bước 2.

Ví dụ dưới đây minh họa cách thuật toán tìm kiếm tuyến tính làm việc. Giả sử chúng ta đang tìm phần tử có giá trị 12 trong mảng sau:

48	83	42	73	36	12	12	7	93	27	73	54	3	48	69	91
----	----	----	----	----	----	----	---	----	----	----	----	---	----	----	----

So sánh giá trị phần tử ở vị trí $i = 0$ với 12 .

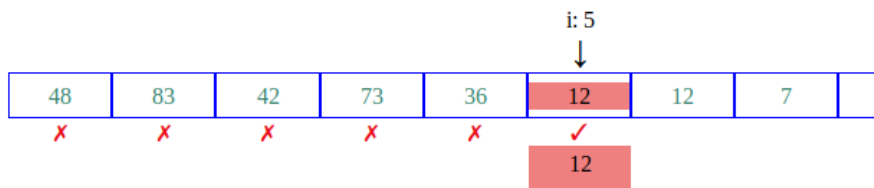
i: 0															
↓															
	48	83	42	73	36	12	12	7							
	X														
	12														

Vì 48 khác 12 nên chuyển sang so sánh với phần tử ở vị trí tiếp theo $i = 1$.

		i: 1													
		↓													
	48	83	42	73	36	12	12	7							
	X	X													
		12													

Vì 83 khác 12 nên chuyển sang so sánh với phần tử ở vị trí tiếp theo $i = 2$.

Cứ lặp lại các bước như vậy đến khi tìm được phần tử có giá trị bằng 12 đầu tiên thì dừng lại.



Trong trường hợp này, đến vị trí $i = 5$ thì tìm thấy giá trị 12 như mong muốn, vậy kết quả trả về là 5.

Cài đặt thuật toán tìm kiếm tuyến tính

Hàm `seqSearch()` sau đây được triển khai để kiểm tra xem dữ liệu data có tồn tại trong mảng arr hay không:

```
1. function seqSearch(arr, data) {
2.   for (let i = 0; i < arr.length; ++i) {
3.     if (arr[i] == data) {
4.       return true;
5.     }
6.   }
7.   return false;
8. }
```

Nếu giá trị của biến `data` được tìm thấy trong mảng, hàm trả về `true` ngay lập tức. Nếu hàm duyệt tới cuối mảng mà không tìm kiếm một phần tử với giá trị phù hợp, hàm trả về `false`.

Thuật toán tìm kiếm tuyến tính còn được sử dụng để giải quyết các bài toán khác như tìm vị trí của một giá trị, tìm phần tử có giá trị nhỏ nhất, tìm phần tử lớn nhất trong mảng, v.v.

4. Tìm kiếm nhị phân

Tìm kiếm nhị phân (binary search) là một giải thuật tìm kiếm nhanh với độ phức tạp thời gian chạy là $O(\log n)$. Giải thuật tìm kiếm nhị phân làm việc dựa trên nguyên tắc chia để trị. Để thuật toán tìm kiếm nhị phân có thể hoạt động trên một tập dữ liệu thì tập dữ liệu đó *phải được sắp xếp*.

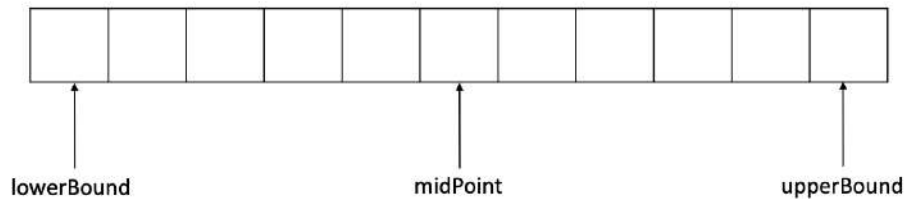
Giải thuật

Binary Search tìm kiếm một giá trị xác định bằng cách so sánh giá trị đó với giá trị của phần tử tại vị trí giữa của tập dữ liệu. Có 3 trường hợp xảy ra:

- Nếu giá trị cần tìm *bằng* với hơn giá trị của phần tử ở giữa của tập dữ liệu thì trả về phần tử đó.
- Nếu giá trị cần tìm *lớn hơn* giá trị của phần tử ở giữa của tập dữ liệu thì giá trị cần tìm sẽ được tìm trong tập dữ liệu con nằm *bên phải* phần tử giữa.
- Nếu giá trị cần tìm *nhỏ hơn* giá trị của phần tử ở giữa của tập dữ liệu thì giá trị cần tìm sẽ được tìm trong tập dữ liệu con nằm *bên trái* phần tử giữa.

Việc tìm kiếm sẽ được tiếp tục cho tới khi tìm hết các phần tử trên tập dữ liệu con.

Giải thích thuật toán bằng hình minh hoạ



Trong hình minh hoạ trên, chúng ta có:

- *lowerBound* là vị trí của phần tử đầu tiên
- *upperBound* là vị trí của phần tử cuối cùng
- *midPoint* là vị trí của phần tử ở giữa (*midPoint* bằng trung bình cộng của *lowerBound* và *upperBound*)

Giả sử *x* là phần tử cần tìm, vậy chúng ta so sánh *x* với giá trị tại *midPoint*. Có 3 khả năng xảy ra:

- Nếu *x* bằng với giá trị tại *midPoint* thì trả về giá trị đó, kết thúc thuật toán
- Nếu *x* nhỏ hơn giá trị tại *midPoint* thì rõ ràng là *x* phải nằm trong khoảng $[lowerBound - midPoint]$, và như vậy thì chúng ta có thể loại bỏ khoảng $[midPoint - upperBound]$.
- Nếu *x* lớn hơn giá trị tại *midPoint* thì rõ ràng là *x* phải nằm trong khoảng $[midPoint - upperBound]$, và như vậy thì chúng ta có thể loại bỏ khoảng $[lowerBound - midPoint]$.

Mã giả

```

1. begin
2.  A // mảng đã được sắp xếp
3.  n // kích cỡ mảng
4.  x // giá trị để tìm kiếm trong mảng
5.  lowerBound = 0 // chỉ số đầu của mảng
6.  upperBound = n-1 // chỉ số cuối của mảng
7.
8.  while x not found
9.    if upperBound < lowerBound
10.      EXIT: x không tồn tại.
11.    midPoint = (lowerBound + upperBound) / 2
12.    if A[midPoint] < x
13.      lowerBound = midPoint + 1
14.
15.    if A[midPoint] > x
16.      upperBound = midPoint - 1
17.    if A[midPoint] = x
18.      EXIT: x được tìm thấy tại midPoint
19.  end while
20. end
21.
```

Ví dụ:

Tìm phần tử có giá trị “35” trong mảng đã được sắp xếp theo trật tự tăng dần sau:

0	3	18	19	35	38	56	65	69	73	76	80	80	88	92	99
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Đầu tiên, xác định chỉ số giữa của mảng theo công thức sau:

chỉ số giữa = (chỉ số đầu + chỉ số cuối) / 2;

Áp dụng cho mảng ở trên chúng ta xác định được chỉ số giữa là: $(0+15)/2 = 7.5$. Chúng ta có thể làm tròn lên hoặc làm tròn xuống. Ở đây chúng ta sẽ làm tròn xuống, vậy chỉ số giữa là 7 và giá trị của phần tử giữa mảng là 65.

Tiếp theo, chúng ta so sánh giá trị của phần tử giữa với giá trị cần tìm. Giá trị của phần tử giữa là 65, giá trị của phần tử cần tìm là 35. Do đó không bằng nhau. Bởi vì giá trị cần tìm nhỏ hơn nên giá trị cần tìm sẽ nằm ở mảng con bên trái phần tử giữa.

0	3	18	19	35	38	56	65	69	73	76	80	80	88	92	99
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Chúng ta thay đổi (*chỉ số cuối* = *chỉ số giữa* - 1) và tiếp tục tìm kiếm chỉ số giữa.

chỉ số cuối = chỉ số giữa - 1

chỉ số giữa = (chỉ số đầu + chỉ số cuối) / 2

Bây giờ, chỉ số giữa được xác định là 3 và giá trị của phần tử giữa là 19. Do giá trị cần tìm là 35, lớn hơn giá trị của phần tử giữa nên giá trị cần tìm sẽ nằm ở mảng con bên phải phần tử giữa.

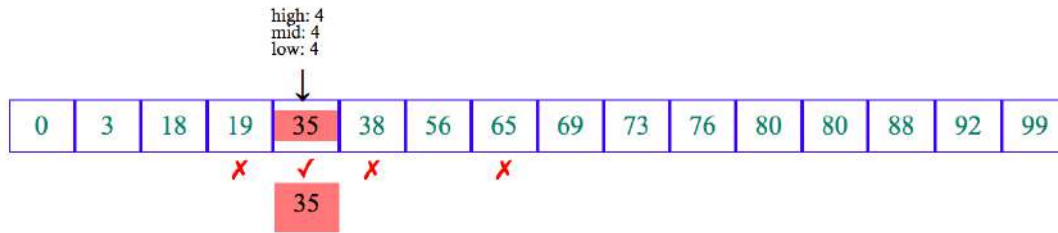
low: 0			mid: 3			high: 6									
↓			↓			↓									
0	3	18	19	35	38	56	65	69	73	76	80	80	88	92	99
			X			X									
				35											

Tiếp tục tìm chỉ số giữa, lúc này chỉ số giữa là 5, và giá trị của phần tử giữa là 38. Do giá trị cần tìm là 35, nhỏ hơn giá trị của phần tử giữa nên giá trị cần tìm sẽ nằm ở mảng con bên trái phần tử giữa.

				low: 4	mid: 5	high: 6									
				↓	↓	↓									
0	3	18	19	35	38	56	65	69	73	76	80	80	88	92	99
			X		X	X									
					35										

Tiếp tục tìm chỉ số giữa, lúc này chỉ số giữa là 4, và giá trị của phần tử giữa là 35, bằng với giá trị cần tìm.

Chúng ta kết luận rằng, giá trị cần tìm là 35, được lưu trữ tại vị trí có chỉ số là 4 trong mảng.



Cài đặt thuật toán tìm kiếm nhị phân

```

1. function binarySearch(data, intArray) {
2.   let lowerBound = 0; // chỉ số đầu
3.   let upperBound = intArray.length - 1; // chỉ số cuối
4.   let midPoint = -1; // chỉ số giữa
5.   let index = -1; // vị trí tìm thấy giá trị cần tìm trong mảng
6.
7.   while (lowerBound <= upperBound) {
8.     midPoint = Math.floor((lowerBound + upperBound) / 2);
9.     if (intArray[midPoint] == data) {
10.      index = midPoint;
11.      break;
12.    } else {
13.      if (intArray[midPoint] < data) {
14.        lowerBound = midPoint + 1;
15.      } else {
16.        upperBound = midPoint - 1;
17.      }
18.    }
19.  }
20.  return index;
21. }
22.

```

5. Độ phức tạp của thuật toán

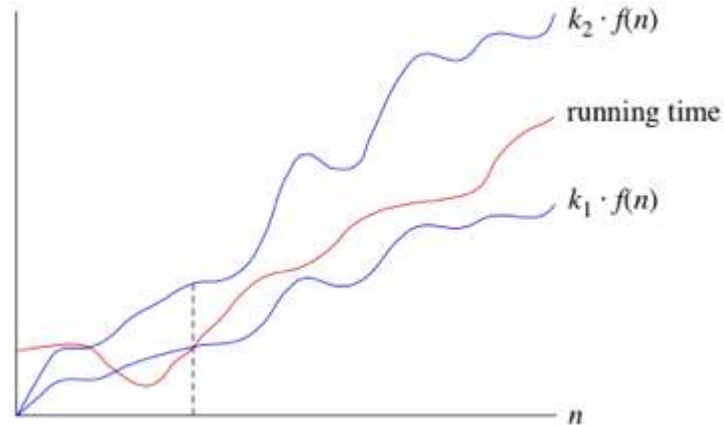
Độ phức tạp thuật toán là một mô hình tính toán để đánh giá mức độ tiêu tốn tài nguyên hệ thống của một thuật toán, bởi vì tài nguyên cần dùng phụ thuộc vào kích cỡ của dữ liệu đầu vào, chúng ta có thể xem độ phức tạp thuật toán là một hàm đặc trưng cho động thái của hệ thống khi kích cỡ đầu vào tăng lên. Thường chúng ta không cần tìm cách xác định chính xác tuyệt đối những hàm này mà chỉ cần tìm ra một ước lượng đủ tốt của chúng.

Để ước lượng độ phức tạp của một thuật toán ta thường dùng khái niệm bậc O -lớn và bậc Θ (đọc là *bậc Theta*). Chúng ta có thể coi chúng là những hàm tiệm cận với hàm tính độ phức tạp thuật toán.

Để diễn giải khái niệm bậc O và bậc Θ , trong mục này chúng ta gọi độ lớn của dữ liệu đầu vào là n .

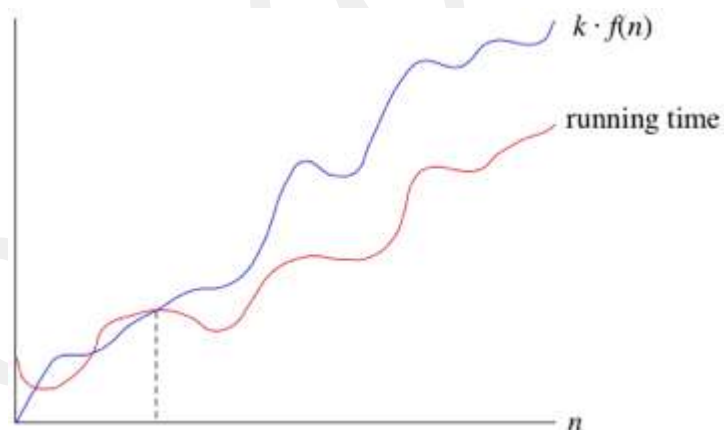
Bậc Theta

Nếu tồn tại một hàm $f(n)$ sao cho tại một giá trị n đủ lớn, thời gian thực thi của hệ thống không lớn hơn giá trị $k_1 \cdot f(n)$ và không nhỏ hơn giá trị $k_2 \cdot f(n)$ trong đó k_1 và k_2 là những hằng số, thì chúng ta nói rằng độ phức tạp thuật toán là *Theta* của $f(n)$.



Bậc O-lớn

Trong một số trường hợp chúng ta không thể ước lượng được một tiệm cận dưới có ý nghĩa thực tế. Chẳng hạn, “trong trường hợp tốt nhất, cả hai thuật toán tìm kiếm đều chỉ mất một lần lặp để tìm được giá trị cần tìm” là một phát biểu đúng, nhưng hoàn toàn vô nghĩa cho mục đích ước lượng độ phức tạp thuật toán. Sẽ tốt hơn nếu chúng ta chỉ phát biểu về tiệm cận trên. Chúng ta sử dụng bậc *O-lớn* vào những lúc như vậy.



Nếu tồn tại một hàm $f(n)$ sao cho tại một giá trị n đủ lớn, thời gian thực thi của hệ thống không lớn hơn giá trị $k \cdot f(n)$ trong đó k là một hằng số, thì chúng ta nói rằng độ phức tạp thuật toán là *O-lớn* của $f(n)$.

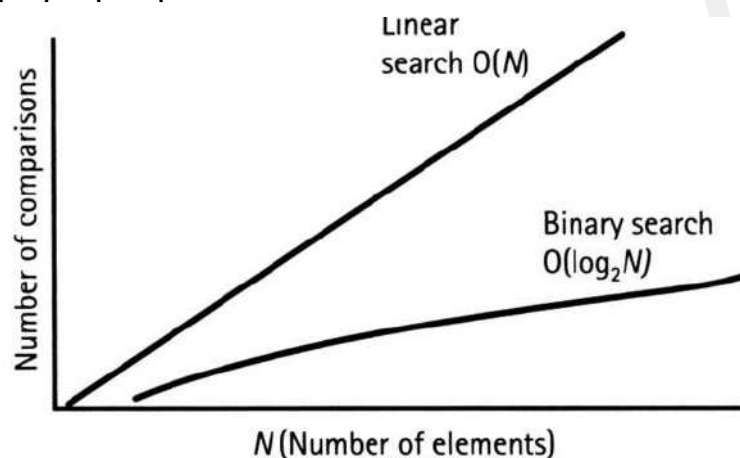
Độ phức tạp của các thuật toán tìm kiếm

Bậc *O-lớn* chính là mô hình tính toán thường được sử dụng để tính độ phức tạp của thuật toán tìm kiếm tuyến tính cũng như tìm kiếm nhị phân.

Đối với thuật toán tìm kiếm tuyến tính, trên mảng có độ dài n , số vòng lặp tối đa phải thực hiện để có thể tìm ra phần tử cần thiết chính bằng n . Tổng thời gian thực thi thuật

toán khi đó sẽ là $k.n + c$, trong đó k là thời gian tính toán trong mỗi vòng lặp và c là thời gian thực thi các bước chuẩn bị trước khi chạy vòng lặp (khởi tạo biến đếm, đặt giá trị ban đầu). Chúng ta không thể biết trước được k do nó phụ thuộc vào độ phức tạp của thuật toán nhận diện phần tử, còn c thì là một hằng số. Do đó mô hình tính toán độ phức tạp ở đây chỉ có một biến số là n và hàm $f(n)$ luôn cho kết quả là n . Ta nói rằng độ phức tạp của thuật toán tìm kiếm tuyến tính là $O(n)$ (đọc là: O lớn của n). Đối với thuật toán tìm kiếm nhị phân, số vòng lặp tối đa phải thực hiện là $\log(n)$. Theo cùng một cách phân tích như trên, độ phức tạp của thuật toán là $O(\log(n))$ (đọc là: O-lớn của lô-ga-rit của n).

Bậc O-lớn của thuật toán tìm kiếm tuyến tính và tìm kiếm nhị phân có thể biểu diễn trên cùng một hệ trục tọa độ như sau:



Nhìn trên sơ đồ, chúng ta có thể thấy rằng ở một n đủ lớn, độ phức tạp của thuật toán tìm kiếm nhị phân tăng rất chậm, trái ngược với độ phức tạp của thuật toán tìm kiếm nhị phân thì tăng nhanh một cách ổn định.

Lưu ý: Như vậy có thể kết luận rằng thuật toán tìm kiếm nhị phân thì có hiệu suất tốt hơn thuật toán tìm kiếm tuyến tính. Tuy nhiên lưu ý rằng, thuật toán nhị phân chỉ có thể thực hiện được trên dữ liệu đã được sắp xếp.

6. Các lỗi thường gặp

Lỗi thường gặp #1: Chỉ số đầu tiên là 1

Ví dụ:

```
1. for (let i = 1; i < array.length; i++) {
2.   if (value === array[i]) {
3.     return i;
4.   }
5. }
```

Đoạn mã trên bắt đầu vòng lặp từ chỉ số 1, nghĩa là từ vị trí thứ 2 trong mảng. Như vậy, có khả năng xảy ra sai sót bởi vì chúng ta đã bỏ qua phần tử đầu tiên của mảng. Cách khắc phục: Luôn luôn bắt đầu với chỉ số đầu tiên của mảng là 0. Đoạn mã đúng là như sau:

```

1. for (let i = 0; i < array.length; i++) {
2.   if (value === array[i]) {
3.     return i;
4.   }
5. }

```

Lỗi thường gặp #2: Duyệt không đúng phần tử cuối cùng

Ví dụ:

```

1. for (let i = 0; i <= array.length; i++) {
2.   if (value === array[i]) {
3.     return i;
4.   }
5. }

```

Đoạn mã trên sử dụng phép so sánh ($\leq$) khiến cho vòng lặp duyệt quá số lượng phần tử của mảng (bởi vì vị trí cuối cùng của mảng là `array.length - 1` chứ không phải là `array.length`). Có 2 cách khắc phục trong trường hợp này:

Cách 1: Sử dụng dấu ($<$) thay vì dấu ($\leq$)

```

1. for (let i = 0; i < array.length; i++) {
2.   if (value === array[i]) {
3.     return i;
4.   }
5. }

```

Cách 2: Chỉ duyệt đến `array.length - 1` thay vì duyệt đến `array.length`

```

1. for (let i = 0; i <= array.length - 1; i++) {
2.   if (value === array[i]) {
3.     return i;
4.   }
5. }

```

Lỗi thường gặp #3: Sử dụng tìm kiếm nhị phân trên tập dữ liệu chưa được sắp xếp

Lưu ý rằng thuật toán tìm kiếm nhị phân chỉ hoạt động đúng trên các tập dữ liệu đã được sắp xếp. Do đó, nếu đầu vào là một tập dữ liệu chưa được sắp xếp thì chúng ta có 2 lựa chọn:

- Sắp xếp dữ liệu trước khi áp dụng thuật toán tìm kiếm nhị phân
- Sử dụng thuật toán tìm kiếm tuyến tính thay cho thuật toán tìm kiếm nhị phân

Lỗi thường gặp #4: Thông báo sớm trong thuật toán tìm kiếm

Đoạn mã dưới đây là một tình huống khi viết chương trình tra từ điển.

```

1. let inpWord = prompt("Nhập vào từ cần tra:");
2. let enWords = ["Hello", "Google", "Code", "Fun"];
3. let viWords = ["Xin chào", "Google", "Viết mã", "Vui"];
4.
5. for (let i = 0; i < viWords.length; i++) {
6.   if (enWords[i] === inpWord) {
7.     alert("Nghĩa của từ này là: " + viWords[i]);
8.   } else {
9.     alert("Không tìm thấy từ này trong từ điển");

```

```

10. }
11. }

```

Với cách viết như trên, vòng lặp nào cũng sẽ không báo về việc tìm thấy hoặc không tìm thấy từ vựng đã nhập. Lỗi này rất thường gặp ở những người mới học lập trình. Để giải quyết tình huống trên, chúng ta chuyển việc thông báo kết quả ra khỏi vòng lặp, và sử dụng từ khoá *break* để ngắt vòng lặp sau khi tìm thấy kết quả. Đoạn mã được viết lại như sau:

```

1. let inpWord = prompt("Nhập vào từ cần tra:");
2. let enWords = ["Hello", "Google", "Code", "Fun"];
3. let viWords = ["Xin chào", "Google", "Viết mã", "Vui"];
4. let foundIndex = -1;
5.
6. for (let i = 0; i < enWords.length; i++) {
7.   if (enWords[i] === inpWord) {
8.     foundIndex = i;
9.     break;
10.  }
11. }
12. if (foundIndex > -1) {
13.   console.log("Nghĩa của từ vựng là: " + viWords[foundIndex]);
14. } else {
15.   console.log("Không tìm thấy từ vựng đã nhập!");
16. }

```

7. Bài thực hành

Bài 1: Tìm một giá trị trong mảng sử dụng thuật toán tìm kiếm nhị phân

Mục tiêu:

Luyện tập sử dụng thuật toán tìm kiếm nhị phân

Mô tả:

Cho trước 1 mảng các số nguyên đã được sắp xếp như sau:

```
array = [1, 2, 3, 4, 6, 7, 9, 11, 12, 14, 15, 16, 17, 19, 33, 34, 43, 45, 55, 66];
```

Hãy tìm số có giá trị bằng 9 được lưu trữ tại vị trí nào trong mảng.

Hướng dẫn:

Bước 1: Xây dựng hàm *binarySearch(data, intArray)* có 2 tham số hình thức là giá trị cần tìm và mảng các số nguyên đã được sắp xếp.

```

1. function binarySearch(data, intArray) {
2.   let lowerBound = 0; // chỉ số đầu
3.   let upperBound = intArray.length - 1; // chỉ số cuối
4.   let midPoint = -1; // chỉ số giữa
5.   let index = -1; // vị trí tìm thấy giá trị cần tìm trong mảng
6.
7.   while (lowerBound <= upperBound) {
8.     midPoint = Math.floor((lowerBound + upperBound) / 2);
9.     if (intArray[midPoint] === data) {
10.      index = midPoint;
11.      break;

```

```

12.     } else {
13.         if (intArray[midPoint] < data) {
14.             lowerBound = midPoint + 1;
15.         } else {
16.             upperBound = midPoint - 1;
17.         }
18.     }
19. }
20. return index;
21. }

```

Bước 2: Khai báo mảng các số nguyên đã được sắp xếp và giá trị cần tìm:

```

1. let array = [1, 2, 3, 4, 6, 7, 9, 11, 12, 14, 15, 16, 17, 19, 33, 34, 43, 45, 55, 66];
2. let data = 9;

```

Bước 3: Truyền mảng các số nguyên và giá trị cần tìm đã được khai báo như đối số của hàm binarySearch:

```

1. let index = binarySearch(data, array);

```

Bước 4: Hiển thị vị trí tìm thấy giá trị cần tìm trong mảng:

```

1. if (index !== -1) {
2.     console.log(data + ' is found at the location: ' + index);
3. } else {
4.     console.log('not found');
5. }

```

Bước 5: Chạy và quan sát kết quả.

Link mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-07/exercise-01-thuat-toan-tim-kiem-nhi-phan.html>

Bài 2: Tìm một giá trị trong mảng sử dụng thuật toán tìm kiếm tuyến tính

Mục tiêu:

Luyện tập sử dụng thuật toán tìm kiếm tuyến tính

Mô

tả:

Cho trước 1 mảng các số nguyên đã được sắp xếp như sau:

array = [1, 2, 3, 4, 6, 7, 9, 11, 12, 14, 15, 16, 17, 19, 33, 34, 43, 45, 55, 66];

Hãy tìm số có giá trị bằng 9 được lưu trữ tại vị trí nào trong mảng.

Hướng

dẫn:

Bước 1: Xây dựng hàm linearSearch(data, intArray) có 2 tham số hình thức là giá trị cần tìm và mảng các số nguyên đã được sắp xếp.

```

1. function linearSearch(data, intArray) {
2.     let index = -1;
3.     for(let i = 0; i < intArray.length; i++){
4.         if(data == intArray[i]){
5.             index = i;
6.         }
7.     }
8.     return index;
9. }

```

Bước 2: Khai báo mảng các số nguyên đã được sắp xếp và giá trị cần tìm:

```
1. let array = [1, 2, 3, 4, 6, 7, 9, 11, 12, 14, 15, 16, 17, 19, 33, 34, 43, 45, 55, 66];
2. let data = 9;
```

Bước 3: Truyền mảng các số nguyên và giá trị cần tìm đã được khai báo như đối số của hàm linearSearch:

```
1. let index = linearSearch(data, array);
```

Bước 4: Hiển thị vị trí tìm thấy giá trị cần tìm trong mảng:

```
1. if (index !== -1) {
2.   console.log(data + ' is found at the location: ' + index);
3. } else {
4.   console.log('not found');
5. }
```

Bước 5: Chạy và quan sát kết quả.

Link mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-07/exercise-02-thuat-toan-tim-kiem-tuyen-tinh.html>

Bài 3: Tìm giá trị lớn nhất và nhỏ nhất.

Mục tiêu:

Luyện tập sử dụng thuật toán tìm kiếm để tìm giá trị nhỏ nhất, lớn nhất.

Mô tả:

Vấn đề lập trình máy tính thường liên quan đến việc tìm kiếm các giá trị tối thiểu và tối đa. Trong một cấu trúc dữ liệu được sắp xếp, việc tìm kiếm các giá trị này là một công việc bình thường. Tuy nhiên, tìm kiếm trên một cấu trúc dữ liệu chưa được phân loại là một nhiệm vụ khó khăn hơn.

Hãy bắt đầu bằng việc xác định chúng ta tìm kiếm giá trị nhỏ nhất trên một mảng.

Thuật toán tìm giá trị nhỏ nhất trên một mảng:

1. Gán phần tử đầu tiên của mảng cho một biến như là giá trị nhỏ nhất.
2. Bắt đầu lặp qua mảng, bắt đầu với phần tử thứ hai, so sánh mỗi phần tử với giá trị nhỏ nhất hiện tại.
3. Nếu phần tử hiện tại có giá trị nhỏ hơn giá trị nhỏ nhất hiện tại, gán phần tử hiện tại làm giá trị nhỏ nhất mới.
4. Chuyển sang phần tử tiếp theo và lặp lại bước 3.
5. Giá trị nhỏ nhất được lưu trữ trong biến đến khi chương trình kết thúc.

Hãy cài đặt thuật toán trên.

Hướng dẫn:

Bước 1: Xây dựng hàm *findMin()* nhận vào một mảng. Hàm làm nhiệm vụ tìm giá trị nhỏ nhất của mảng nhận vào và trả về giá trị nhỏ nhất đó.

```
1. function findMin(arr) {
2.   let min = arr[0];
3.   for (let i = 1; i < arr.length; ++i) {
4.     if (arr[i] < min) {
5.       min = arr[i];
6.     }
7.   }
8.   return min;
9. }
```

Bước 2: Cài đặt thuật toán tìm giá trị lớn nhất

Thuật toán để tìm giá trị lớn nhất hoạt động theo một cách tương tự. Chỉ định phần tử đầu tiên của mảng là giá trị lớn nhất và sau đó lặp phần còn lại của mảng, so sánh từng phần tử với giá trị cực đại hiện tại. Nếu phần tử hiện tại lớn hơn giá trị cực đại hiện tại, giá trị của phần tử đó được lưu trữ trong biến.

Xây dựng hàm *findMax()*:

```
1. function findMax(arr) {
2.   let max = arr[0];
3.   for (let i = 1; i < arr.length; ++i) {
4.     if (arr[i] > max) {
5.       max = arr[i];
6.     }
7.   }
8.   return max;
9. }
```

Bước 3: Xây dựng chương trình tìm cả giá trị nhỏ nhất và giá trị lớn nhất của một mảng

```
1. let nums = [];
2. for(let i = 0; i < 100; i++) {
3.   nums[i] = Math.random()*100;
4. }
5. for(let i = 0; i < nums.length; i++) {
6.   document.write(nums[i] + " ");
7. }
8.
9. let minValue = findMin(nums);
10. document.write("<br/>");
11. document.write("The minimum value is: " + minValue);
12.
13. let maxValue = findMax(nums);
14. document.write("<br/>");
15. document.write("The maximum value is: " + maxValue);
```

Bước 4: Thực thi chương trình quan sát kết quả:

```

34 2 93 21 62 71 81 66 72 56 28 71 74 84 85 48 20 79
88 85 96 2 8 72 14 90 95 70 83 83 25 93 43 7 10 20
16 32 3 98 7 31 7 51 60 80 95 64 94 73 90 70 58 73
68 68 12 28 11 10 24 72 12 48 8 79 40 22 36 24 31 2
98 12 66 87 4 44 91 53 31 15 100 17 72 64 19 35 71
58 90 33 55 12 41 91 27 14 71 72
The minimum value is: 2
The maximum value is: 100

```

Link mã nguồn tham khảo: <https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-07/exercise-03-tim-min-max.html>

8. Bài tập

Bài 1: Tìm số điện thoại

Cho một danh sách các số điện thoại di động của nhiều nhà mạng khác nhau. Hãy viết một hàm để hiển thị các số di động thuộc nhà mạng Viettel.

Bài 2: Game đoán số

Hãy xây dựng một ứng dụng game đoán số được mô tả như sau: Người chơi được phép bí mật chọn trước một số nằm trong khoảng từ 1 đến 100. Nhiệm vụ của ứng dụng sẽ là đoán xem người chơi đã chọn số nào trong khoảng đó.

Ứng dụng thực hiện việc này bằng cách hỏi người chơi và người chơi sẽ phải trả lời trung thực.

Câu hỏi sẽ có dạng: "Có phải số bạn chọn là X hay không? Nếu không phải thì hãy cho tôi biết là nó lớn hơn hay nhỏ hơn X?"

Người chơi sẽ phải trả lời 1 trong 3 trường hợp:

- Số tôi chọn chính là X: trò chơi kết thúc
- Số tôi chọn nhỏ hơn X: ứng dụng tiếp tục hỏi câu tiếp theo
- Số tôi chọn lớn hơn X: ứng dụng tiếp tục hỏi câu tiếp theo

Gợi ý: Bản chất của trò chơi này đó là triển khai thuật toán tìm kiếm nhị phân.

Bài 3: Triển khai thuật toán tìm kiếm nhị phân bằng cách sử dụng hàm đệ quy

Hãy viết một hàm đệ quy để triển khai tìm kiếm nhị phân mà không sử dụng vòng lặp.

Gợi ý: Xem lại nội dung về Hàm đệ quy trong Chương 6.

9. Bài kiểm tra

Câu 1: Độ phức tạp tốt nhất của thuật toán tìm kiếm tuyến tính là?

- $O(n)$
- $O(\log n)$
- $O(n \log n)$
- $O(1)$

Câu 2: Độ phức tạp tệ nhất của thuật toán tìm kiếm nhị phân là gì?

- a) $O(\log n)$
- b) $O(n \log n)$
- c) $O(1)$
- d) $O(n)$

Câu 3: Với một mảng số nguyên đã được sắp xếp theo trật tự giảm dần thì thuật toán tìm kiếm nào có hiệu quả nhất?

- a) Thuật toán tìm kiếm tuyến tính
- b) Thuật toán tìm kiếm nhị phân

Câu 4: Thuật toán tìm kiếm nào là tốt nhất cho một danh sách lớn

- a) Tìm kiếm tuần tự
- b) Sử dụng vòng lặp for-in
- c) Tìm kiếm nhị phân

Câu 5: Trung bình, một thuật toán tìm kiếm tuần tự sẽ mất $N / 2$ số lần so sánh cho một danh sách kích thước N .

- a) Đúng
- b) Sai

Câu 6: Điều sau đây được là đúng đối với thuật toán tìm kiếm nhị phân?

- a) Có thể áp dụng với dữ liệu đã được sắp xếp theo trật tự tăng dần
- b) Có thể áp dụng với dữ liệu đã được sắp xếp theo trật tự giảm dần
- c) Có thể áp dụng với dữ liệu chưa được sắp xếp

Đáp án: Câu 1: d; Câu 2: a; Câu 3: b; Câu 4: c; Câu 5: a; Câu 6: a và b;

10. Tổng kết

- Tìm kiếm dữ liệu là một thao tác được thực hiện thường xuyên trong các ứng dụng
- Thuật toán tìm kiếm tuyến tính hoạt động dựa trên cơ chế duyệt qua lần lượt tất cả các phần tử
- Thuật toán tìm kiếm nhị phân hoạt động dựa trên cơ chế chia nhỏ tập dữ liệu
- Thuật toán tìm kiếm nhị phân chỉ có thể thực hiện được trên tập dữ liệu đã được sắp xếp
- Độ phức tạp của thuật toán là một đại lượng để đo tính hiệu quả của một thuật toán
- Độ phức tạp của thuật toán tìm kiếm nhị phân là nhỏ hơn so với độ phức tạp của thuật toán tìm kiếm tuyến tính

Chương 8 - Thuật toán sắp xếp

Sắp xếp được dữ liệu theo các trật tự và tiêu chí khác nhau

1. Mục tiêu

- Liệt kê được các trường hợp áp dụng của thuật toán sắp xếp
- Trình bày được thuật toán sắp xếp Nổi bọt
- Cài đặt được thuật toán sắp xếp Nổi bọt
- Trình bày được thuật toán sắp xếp Chọn
- Cài đặt được thuật toán sắp xếp Chọn
- Trình bày được thuật toán sắp xếp Chèn
- Cài đặt được thuật toán sắp xếp Chèn
- Phân biệt được độ phức tạp của các thuật toán sắp xếp để có lựa chọn phù hợp

2. Giới thiệu

Sắp xếp là thao tác thay đổi vị trí của các phần tử trong một danh sách để thỏa mãn một tiêu chí nhất định. Chúng ta thường gặp các danh sách có trật tự, chẳng hạn danh sách thí sinh dự thi trong một kỳ thi Đại học được sắp xếp theo tên, danh sách các quốc gia theo độ lớn của diện tích, danh sách khách hàng theo tổng giá trị hợp đồng, v.v. Để thực hiện được thao tác thay đổi vị trí của các phần tử một cách hiệu quả, chúng ta sẽ cần nghiên cứu nhiều thuật toán sắp xếp khác nhau.

Độ phức tạp của một thuật toán sắp xếp phụ thuộc vào số lượng phép so sánh và số lượng phép hoán vị cần phải thực hiện. Tối ưu hóa hai đại lượng này chính là mục tiêu của các thuật toán sắp xếp. Có nhiều thuật toán sắp xếp khác nhau, tùy thuộc vào kích thước và cách phân phối của các phần tử trong danh sách đầu vào mà mỗi loại thuật toán thể hiện một ưu thế riêng.

Kết thúc chương này, chúng ta có thể lựa chọn triển khai một số thuật toán sắp xếp khác nhau để phù hợp với các tình huống cụ thể. Chúng ta sẽ khảo sát 3 thuật toán cơ bản bao gồm: *Sắp xếp Nổi bọt*, *Sắp xếp Chèn*, *Sắp xếp Chọn*.

3. Thuật toán sắp xếp nổi bọt

Sắp xếp nổi bọt (bubble sort) là một thuật toán sắp xếp đơn giản, với thao tác cơ bản là so sánh hai phần tử kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ (swap) cho nhau, việc đổi chỗ này được lặp đi lặp lại cho đến khi không còn phần tử nào đứng không đúng thứ tự.

Có thể tiến hành theo hướng từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang). Sắp xếp nổi bọt còn có tên là sắp xếp bằng so sánh trực tiếp. Nó sử dụng phép so sánh các phần tử nên là một giải thuật sắp xếp kiểu so sánh.

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trong trường hợp xấu nhất và trường hợp trung bình là $O(n^2)$ với n là số phần tử.

Giải thuật sắp xếp nổi bọt là giải thuật chậm nhất trong số các giải thuật sắp xếp cơ bản. Giải thuật này còn chậm hơn giải thuật đổi chỗ trực tiếp mặc dù số lần so sánh bằng nhau, nhưng do đổi chỗ hai phần tử kề nhau nên số lần đổi chỗ nhiều hơn.

Thuật toán này có tên gọi là “nổi bọt” vì hình tượng các giá trị nhỏ hơn (trong trường hợp sắp xếp giảm dần) hoặc lớn hơn (trong trường hợp sắp xếp tăng dần) lần lượt nổi lên phía trên của danh sách.

Giải thuật

Sắp xếp từ trên xuống

Giả sử dãy cần sắp xếp có n phần tử, trật tự sắp xếp là tăng dần. Khi tiến hành từ trên xuống, ta so sánh hai phần tử đầu tiên, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau. Tiếp tục làm như vậy với cặp phần tử thứ hai và thứ ba và tiếp tục cho đến cuối tập hợp dữ liệu, nghĩa là so sánh (và đổi chỗ nếu cần) phần tử thứ $n-1$ với phần tử thứ n . Sau bước này phần tử cuối cùng chính là phần tử lớn nhất của dãy.

Sau đó, quay lại so sánh (và đổi chỗ nếu cần) hai phần tử đầu cho đến khi gặp phần tử thứ $n-2$

Ghi chú: Nếu trong một lần duyệt, không phải đổi chỗ bất cứ cặp phần tử nào thì danh sách đã được sắp xếp xong.

Ví dụ: Để sắp xếp 6 phần tử (2 9 5 4 8 1) theo trật tự tăng dần:

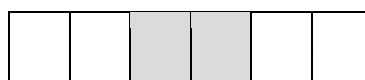
Chúng ta so sánh hai phần tử đầu tiên là 2 và 9, chúng đã đúng trật tự cho nên không cần hoán đổi gì cả.



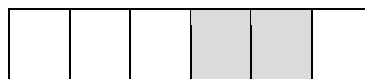
Tiếp theo chúng ta so sánh hai phần tử là 9 và 5, chúng đang không ở đúng trật tự (5 phải đứng trước 9) cho nên chúng ta phải hoán đổi vị trí của 2 phần tử này.



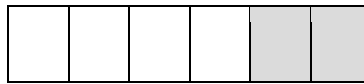
Tiếp theo chúng ta so sánh hai phần tử là 9 và 4, phải hoán đổi vị trí của chúng.



Tiếp theo chúng ta so sánh hai phần tử 9 và 8, phải hoán đổi vị trí của chúng.



Cuối cùng so sánh hai phần tử 9 và 1, phải hoán đổi vị trí của chúng.



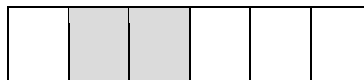
Kết thúc lượt thứ nhất, số 9 đã được “nổi” lên trên cùng. Chúng ta sẽ thực hiện lượt thứ hai với 5 phần tử còn lại là (2 5 4 8 1). Kết thúc lượt thứ hai thì số 8 sẽ được nổi lên. Và cứ như vậy cho đến khi tất cả các phần tử đứng đúng ở vị trí của nó.



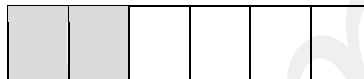
Lượt thứ hai – Số 8 được “nổi” lên



Lượt thứ ba – Số 5 được “nổi” lên



Lượt thứ tư – Số 4 được “nổi” lên



Lượt thứ năm – Số 2 được “nổi” lên

Sắp xếp từ dưới lên

Sắp xếp từ dưới lên so sánh (và đổi chỗ nếu cần) bắt đầu từ việc so sánh cặp phần tử thứ $n-1$ và n . Tiếp theo là so sánh cặp phần tử thứ $n-2$ và $n-1, \dots$ cho đến khi so sánh và đổi chỗ cặp phần tử thứ nhất và thứ hai. Sau bước này phần tử nhỏ nhất đã được “nổi” lên vị trí trên cùng (nó giống như hình ảnh của các “bọt” khí nhẹ hơn được nổi lên trên). Tiếp theo tiến hành với các phần tử từ thứ 2 đến thứ n .

Mã giả

Sắp xếp từ trên xuống

```

1. procedure bubble_sort1(list L, number n) //n=listszie
2.   For number i from n downto 2
3.     for number j from 1 to (i - 1)
4.       if L[j] > L[j + 1] //nếu chúng không đúng thứ tự
5.         swap(L[j], L[j + 1]) //đổi chỗ chúng cho nhau
6.       endif
7.     endfor
8.   endfor
9. endprocedure

```

Sắp xếp từ dưới lên

```

1. procedure bubble_sort2(list L, number n) //n=listszie
2.   For number i from 1 to n-1
3.     for number j from n-1 downto i
4.       if L[j] > L[j + 1] //nếu chúng không đúng thứ tự

```

```

5.         swap(L[j], L[j + 1]) //đổi chỗ chúng cho nhau
6.     endif
7. endfor
8. endfor
9. endprocedure

```

Cài đặt thuật toán sắp xếp nổi bọt

```

1. function bubbleSort(items) {
2.     let length = items.length;
3.     for (let i = 0; i < length; i++) {
4.         for (let j = 0; j < (length - i - 1); j++) {
5.             if (items[j] > items[j + 1]) {
6.                 let tmp = items[j];
7.                 items[j] = items[j + 1];
8.                 items[j + 1] = tmp;
9.             }
10.        }
11.    }
12. }

```

4. Thuật toán sắp xếp chèn

Ý tưởng

Sắp xếp chèn (insertion sort) thực hiện sắp xếp một dãy số bằng cách duyệt từng phần tử và chèn phần tử đó vào vị trí thích hợp trong một danh sách con mới sao cho danh sách con luôn luôn được duy trì dưới dạng đã qua sắp xếp.

Ví dụ: Để sắp xếp một danh sách các số (2 9 5 4 8 1 6) theo trật tự tăng dần. Bước đầu tiên, danh sách con mới chỉ chứa phần tử đầu tiên đó là số 2. Tiếp theo, chúng ta sẽ chèn số 9 vào trong danh sách con đó, vì số 9 lớn hơn 2 cho nên số chỉ phải được “chèn” vào phía sau 2. Sau đó, chúng ta sẽ chèn số 5 vào trong danh sách con, vì số 5 lớn hơn 2 nhưng lại nhỏ hơn 9 cho nên sẽ được chèn vào giữa hai số này. Tương tự như vậy cho các số còn lại cho đến khi tất cả các số đều được chèn vào danh sách con.

	Bước 1: Danh sách con chỉ có một phần tử ban đầu là {2}. Chúng ta chèn số 9 vào sau số 2.
	Bước 2: Danh sách con đã được sắp xếp là {2, 9}. Chèn số 5 vào danh sách con.
	Bước 3: Danh sách con đã được sắp xếp là {2, 5, 9}. Chèn số 4 vào danh sách con.

	Bước 4: Danh sách con đã được sắp xếp là {2, 4, 5, 9}. Chèn số 8 vào danh sách con.
	Bước 5: Danh sách con đã được sắp xếp là {2, 4, 5, 8, 9}. Chèn số 1 vào danh sách.
	Bước 6: Danh sách con đã được sắp xếp là {1, 2, 4, 5, 8, 9}. Chèn số 6 vào danh sách.
	Bước 7: Toàn bộ danh sách đã được sắp xếp.

Quá trình chèn lần lượt các số vào đúng vị trí

Quá trình “chèn” được diễn ra bằng cách dịch chuyển các phần tử ở phía sau vị trí muốn chèn để tạo một chỗ trống cho phần tử mới. Chẳng hạn như ở Bước 3 ở trên, để chèn số 4 vào vị trí trước số 5 thì chúng ta phải dịch chuyển số 9 và số 9 về phía sau để tạo một khoảng trống trước số 5.

[0] [1] [2] [3] [4] [5] [6] list 2 5 9 4	Bước 1: Lưu giá trị 4 vào một biến tạm
[0] [1] [2] [3] [4] [5] [6] list 2 5 9	Bước 2: Dịch chuyển phần tử ở vị trí list[2] sang vị trí list[3].
[0] [1] [2] [3] [4] [5] [6] list 2 5 9	Bước 3: Dịch chuyển phần tử ở vị trí list[1] sang vị trí list[2].
[0] [1] [2] [3] [4] [5] [6] list 2 4 5 9	Bước 4: Gán giá trị của biến tạm vào vị trí list[1].

Quá trình dịch chuyển để chèn một phần tử vào đúng vị trí

Giải thuật

Từ mô tả trên ta đã có một bức tranh khái quát cho giải thuật sắp xếp chèn, chúng ta sẽ có các bước cơ bản cho giải thuật như sau:

```

1. for (let i = 1; i < list.length; i++) {
2.   chèn phần tử list[i] vào danh sách con list[0..i-1] sao cho danh sách list[0..i] luôn được sắp xếp
3. }

```

Cài đặt thuật toán sắp xếp chèn

```

1. function insertionSort(items) {

```

```

2. for (let i = 1; i < items.length; i++) {
3.     let current = items[i];
4.     let j = i-1;
5.
6.     while (j >= 0 && items[j] > current) {
7.         items[j+1] = items[j];
8.         j--;
9.     }
10.    items[j+1] = current;
11. }
12. return items;
13. }

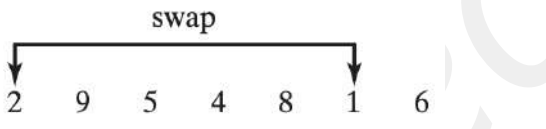
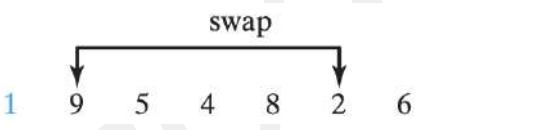
```

5. Thuật toán sắp xếp chọn

Ý tưởng

Chẳng hạn, nếu chúng ta muốn sắp xếp một danh sách theo trật tự tăng dần. Chúng ta sẽ tìm ra phần tử nhỏ nhất trong danh sách, rồi đưa phần tử đó về vị trí đầu tiên. Sau đó, chúng ta tìm phần tử nhỏ nhất trong danh sách còn lại (trừ phần tử đầu tiên trước đó) và đưa nó về vị trí đầu tiên trong danh sách còn lại đó, cứ như thế cho đến phần tử cuối cùng.

Ví dụ: Để sắp xếp danh sách các số {2, 9, 5, 4, 8, 1, 6} sử dụng thuật toán sắp xếp chọn, chúng ta sẽ tiến hành các bước như sau:

	Bước 1: Tìm được phần tử nhỏ nhất trong danh sách đó là 1. Hoán đổi vị trí của số 1 cho số 2 (nằm ở vị trí đầu tiên).
	Bước 2: Danh sách còn lại là {9, 5, 4, 8, 2, 6}, loại trừ số 1 vì nó đã được sắp xếp. Tìm được phần tử nhỏ nhất là 2, hoán đổi vị trí của số 2 với số 9 (nằm ở vị trí đầu tiên của danh sách còn lại).
	Bước 3: Danh sách còn lại là {5, 4, 8, 9, 6}. Tìm được phần tử nhỏ nhất là 4, hoán đổi vị trí của số 4 với số 5.
	Bước 4: Danh sách còn lại là {5, 8, 9, 6}. Tìm được phần tử nhỏ nhất là 5, nhưng không cần hoán đổi vì nó đã nằm ở vị trí đầu tiên.

	Bước 5: Danh sách còn lại là {8, 9, 6}. Tìm được phần tử nhỏ nhất là 6, hoán đổi vị trí số 6 với số 8.
	Bước 6: Danh sách còn lại là {9, 8}. Tìm được phần tử nhỏ nhất là 8, hoán đổi vị trí của số 8 và số 9.
	Bước 7: Danh sách còn lại chỉ một phần tử duy nhất là số 9. Không cần thực hiện thêm thao tác nào nữa. Danh sách đã được sắp xếp.

Quá trình sắp xếp chọn lần lượt các số vào đúng vị trí

Giải thuật

Từ mô tả ở trên, chúng ta có thể có một giải thuật cho thuật toán sắp xếp chọn như sau:

```

1. for (int i = 0; i < list.length - 1; i++) {
2.   tìm phần tử nhỏ nhất trong danh sách list[i..list.length-1];
3.   hoán đổi phần tử nhỏ nhất với phần tử list[i] nếu cần thiết
4.   // list[i] đã nằm đúng vị trí của nó.
5.   // lần lặp tiếp theo được thực hiện trong danh sách còn lại list[i+1..list.length-1]
6. }

```

Cài đặt thuật toán sắp xếp chọn

```

1. function selectionSort(arr) {
2.   let minIndex, temp, len = arr.length;
3.   for (let i = 0; i < len; i++) {
4.     minIndex = i;
5.     // tìm index của giá trị nhỏ nhất
6.     for (let j = i+1; j < len; j++) {
7.       if (arr[j] < arr[minIndex]) {
8.         minIndex = j;
9.       }
10.    }
11.  }
12.  // Đổi chỗ
13.  temp = arr[i];
14.  arr[i] = arr[minIndex];
15.  arr[minIndex] = temp;
16. }
17. return arr;
18. }

```

6. Các lỗi thường gặp

Lỗi thường gặp #1: Duyệt không đúng danh sách

Trong các thuật toán sắp xếp, chúng ta làm việc nhiều với các danh sách/mảng và vòng lặp, do đó rất dễ mắc phải các lỗi liên quan đến chỉ số bắt đầu và kết thúc của danh sách. Hãy rất chú ý đến những nơi mà chúng ta có sử dụng các phần tử đầu (chẳng hạn như `list[0]` hoặc `list[1]`, và những phần tử cuối (chẳng hạn như `list[list.length]` hoặc `list[list.length - 1]`).

Lỗi thường gặp #2: Sử dụng nhầm chỉ số

Trong các thuật toán sắp xếp thì thường có nhiều vòng lặp lồng nhau, do đó sẽ có các biến khác nhau để đại diện cho các chỉ số. Thông thường chúng ta sẽ dùng các biến `i`, `j`, `k`... để đại diện cho các chỉ số lần lượt từ vòng lặp ngoài cùng cho đến bên trong.

Hãy rất chú ý đến những nơi có truy xuất đến các phần tử ở trong vòng lặp lồng nhau, chẳng hạn như `list[i]`, `list[j]`, `list[k]`...

8. Bài thực hành

Bài 1: Triển khai thuật toán sắp xếp nổi bọt

Mục tiêu:

Luyện tập triển khai thuật toán sắp xếp nổi bọt.

Mô tả:

Hãy viết một hàm triển khai thuật toán sắp xếp nổi bọt. Hàm này có mô tả như sau:

```
function bubbleSort(arr, asc);
```

Trong đó:

- `arr` là một mảng bất kỳ
- `asc` là trật tự mà chúng ta muốn sắp xếp, nếu `asc` có giá trị `true` thì sắp xếp theo trật tự tăng dần, còn nếu `asc` có giá trị `false` thì sắp xếp theo trật tự giảm dần.

Hướng dẫn:

- Hãy tham khảo thuật toán và mã nguồn trong Mục 3 – Thuật toán sắp xếp nổi bọt để viết một hàm triển khai thuật toán sắp xếp nổi bọt
- Hãy thử nghiệm với một mảng số nguyên theo trật tự giảm dần
- Hãy thử nghiệm với một mảng chuỗi theo trật tự tăng dần

Link mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-08/exercise-01-sap-xep-noi-bot.html>

Bài 2: Triển khai thuật toán sắp xếp chèn

Mục tiêu:

Luyện tập triển khai thuật toán sắp xếp chèn.

Mô tả:

Hãy viết một hàm triển khai thuật toán sắp xếp chèn. Hàm này có mô tả như sau:

```
function insertionSort(arr, asc);
```

Trong đó:

- *arr* là một mảng bất kỳ
- *asc* là trật tự mà chúng ta muốn sắp xếp, nếu *asc* có giá trị *true* thì sắp xếp theo trật tự tăng dần, còn nếu *asc* có giá trị *false* thì sắp xếp theo trật tự giảm dần.

Hướng dẫn:

- Hãy tham khảo thuật toán và mã nguồn trong Mục 4 – Thuật toán sắp xếp chèn để viết một hàm triển khai thuật toán sắp xếp chèn
- Hãy thử nghiệm với một mảng số nguyên theo trật tự giảm dần
- Hãy thử nghiệm với một mảng chuỗi theo trật tự tăng dần

Link mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-08/exercise-02-sap-xep-chen.html>

Bài 3: Triển khai thuật toán sắp xếp chọn

Mục tiêu:

Luyện tập triển khai thuật toán sắp xếp chọn.

Mô tả:

Hãy viết một hàm triển khai thuật toán sắp xếp chọn. Hàm này có mô tả như sau:

```
function selectionSort(arr, asc);
```

Trong đó:

- *arr* là một mảng bất kỳ
- *asc* là trật tự mà chúng ta muốn sắp xếp, nếu *asc* có giá trị *true* thì sắp xếp theo trật tự tăng dần, còn nếu *asc* có giá trị *false* thì sắp xếp theo trật tự giảm dần.

Hướng dẫn:

- Hãy tham khảo thuật toán và mã nguồn trong Mục 3 – Thuật toán sắp xếp chọn để viết một hàm triển khai thuật toán sắp xếp chọn
- Hãy thử nghiệm với một mảng số nguyên theo trật tự giảm dần
- Hãy thử nghiệm với một mảng chuỗi theo trật tự tăng dần

Link mã nguồn tham khảo:

<https://github.com/codegym-vn/introduction-to-programming-with-javascript/blob/main/chapter-08/exercise-03-sap-xep-chon.html>

9. Bài tập

Bài 1: Minh họa thuật toán sắp xếp nổi bọt

Mục tiêu:

Hiểu rõ về cơ chế hoạt động của thuật toán sắp xếp nổi bọt.

Mô tả:

Hãy hiển thị tất cả các kết quả của từng bước sắp xếp để hiểu hơn về thuật toán sắp xếp nổi bọt.

Bài 2: Minh họa thuật toán sắp xếp chèn

Mục tiêu:

Hiểu rõ về cơ chế hoạt động của thuật toán sắp xếp chèn.

Mô tả:

Hãy hiển thị tất cả các kết quả của từng bước sắp xếp để hiểu hơn về thuật toán sắp xếp chèn.

Bài 3: Minh họa thuật toán sắp xếp chọn

Mục tiêu:

Hiểu rõ về cơ chế hoạt động của thuật toán sắp xếp chọn.

Mô tả:

Hãy hiển thị tất cả các kết quả của từng bước sắp xếp để hiểu hơn về thuật toán sắp xếp chọn.

Bài 4: Nối mảng đã được sắp xếp

Mục tiêu:

Luyện tập triển khai thuật toán sắp xếp chèn.

Mô tả:

Hãy viết một hàm để nối hai mảng `arr1` và `arr2` trong đó mảng `arr1` đã được sắp xếp. Kết quả là mảng `arr1` sẽ bao gồm các phần tử của `arr2` và vẫn giữ được trật tự sắp xếp. Hàm này có mô tả như sau:

```
function orderedMerge(arr1, arr2);
```

Trong đó:

- `arr1` là một mảng đã được sắp xếp
- `arr2` là một mảng bất kỳ

Ví dụ, nối mảng `[1, 3, 5, 8, 9]` với mảng `[2, 1, 7, 4]` thì sẽ được mảng `[1, 1, 2, 3, 4, 5, 7, 8, 9]`.

10. Bài kiểm tra

Câu 1: Độ phức tạp của thuật toán sắp xếp nổi bọt là bao nhiêu?

- a) $O(n)$
- b) $O(\log(n))$
- c) $O(n^2)$
- d) $O(1)$

Câu 2: Độ phức tạp của thuật toán sắp xếp chèn là bao nhiêu?

- a) $O(n)$
- b) $O(\log(n))$
- c) $O(n^2)$
- d) $O(1)$

Câu 3: Độ phức tạp của thuật toán sắp xếp chọn là bao nhiêu?

- a) $O(n)$
- b) $O(\log(n))$
- c) $O(n^2)$
- d) $O(1)$

Câu 4: Nếu một danh sách có n phần tử và đã được sắp xếp thì thuật toán sắp xếp nổi bọt sẽ thực hiện bao nhiêu phép so sánh?

- a) n
- b) $n - 1$
- c) $n!$
- d) $(n - 1)!$

Câu 5: Trong thuật toán sắp xếp chọn, nếu chúng ta muốn sắp xếp theo trật tự giảm dần thì thao tác đầu tiên là sẽ tìm ra phần tử nhỏ nhất hay lớn nhất?

- a) Tìm phần tử nhỏ nhất
- b) Tìm phần tử lớn nhất

Câu 6: Điều nào sau đây là đúng đối với thuật toán tìm kiếm nhị phân?

- a) Có thể áp dụng với dữ liệu đã được sắp xếp theo trật tự tăng dần
- b) Có thể áp dụng với dữ liệu đã được sắp xếp theo trật tự giảm dần
- c) Có thể áp dụng với dữ liệu chưa được sắp xếp

Đáp án: Câu 1: c; Câu 2: c; Câu 3: c; Câu 4: b; Câu 5: b; Câu 6: a và b.

11. Tổng kết

- Thuật toán sắp xếp nổi bọt so sánh hai phần tử liền kề nhau và hoán đổi vị trí cho nhau nếu cần thiết

- Thuật toán sắp xếp chèn duyệt lần lượt qua phần tử và chèn nó vào vị trí phù hợp trong một danh sách con
- Thuật toán sắp xếp chọn tìm ra phần tử nhỏ nhất (hoặc lớn nhất) trong danh sách và đưa nó về vị trí đầu tiên (hoặc cuối cùng)
- Độ phức tạp trong trường hợp tệ nhất của các thuật toán sắp xếp chèn, sắp xếp chọn, sắp xếp nổi bọt là $O(n^2)$

codegym.vn

Tổng kết

Như vậy là bạn đã hoàn thành được cuốn sách này, hiểu được những khái niệm nền tảng nhất trong lập trình, có được những kỹ năng căn bản nhất để có thể tạo ra được những chương trình phần mềm một cách bài bản. Bạn nên biết rằng, trong khuôn khổ của một cuốn sách nhập môn, chúng ta chỉ mới tiếp cận được những phần căn bản nhất dành cho người mới bắt đầu, do vậy để tiếp tục phát triển năng lực đầy đủ thì bạn vẫn sẽ phải tiếp tục việc học. Cuốn sách này hoàn thành nhiệm vụ của nó là giúp cho bạn có một khởi đầu thuận lợi và bài bản nhất.

Các nội dung chính mà bạn đã học trong cuốn sách là:

- Giới thiệu tổng quan về lập trình
- Biến, kiểu dữ liệu và toán tử
- Câu lệnh điều kiện
- Câu lệnh lặp
- Mảng
- Hàm
- Thuật toán tìm kiếm
- Thuật toán sắp xếp

Bạn hãy đặc biệt chú ý đến các phần *Mã sạch* trong từng chương, vì đây là những hướng dẫn quan trọng để sau này bạn hình thành được kỷ luật và thói quen lập trình tử tế.

Đội ngũ tác giả rất cảm ơn bạn đã đánh giá cao giá trị và sử dụng cuốn sách này trọn vẹn. Trong quá trình biên soạn cuốn sách, chắc hẳn sẽ có một số nội dung được trình bày chưa thỏa đáng, hoặc có các lỗi nhỏ. Đội ngũ tác giả rất vui lòng nếu nhận được các phản hồi của các bạn để cải thiện cho các phiên bản sau tại codegym@codegym.vn



Quét QR code để truy cập *Hòm thư góp ý*!

Tài nguyên lập trình

Trong khuôn khổ của CodeGym, chúng tôi có cung cấp thêm rất nhiều những tài nguyên khác phục vụ cho người học, bạn có thể truy cập các trang web sau để tìm hiểu thêm.

<u>Website CodeGym</u>	https://codegym.vn/
<u>Kiến thức lập trình</u>	https://kienthuclaptrinh.vn/
<u>Học JavaScript</u>	https://hocjavascript.net/
<u>Học Java</u>	https://hocjava.com/
<u>Học Spring Boot</u>	https://hocspringboot.net/
<u>Học PHP</u>	https://hocphp.net/
<u>Học Laravel</u>	https://hoclaravel.net/
<u>Học Spring MVC</u>	https://hocspringmvc.net/

Hỗ trợ tư vấn, định hướng

Đội ngũ chuyên gia tại CodeGym luôn sẵn sàng tư vấn, định hướng và giải đáp các thắc mắc về nghề lập trình, bạn có thể kết nối với CodeGym để được hỗ trợ theo thông tin dưới đây:

CodeGym Mỹ Đình (Hà Nội) - Nhà số 23, Lô TT-01, KĐT Mon City, đường Hàm Nghi, - Mỹ Đình 2, Nam Từ Liêm, Hà Nội - info@codegym.vn - facebook.com/codegym.hn - codegym.vn 0246 253 8829	CodeGym Quảng Trị 133 Lý Thường Kiệt, Tp Đông Hà, Quảng Trị - quangtri@codegym.vn - https://www.facebook.com/LaptrinhQT - codegym.vn/trung-tam-codegym-quang-tri 0702 723 723 - 0977 376 416
CodeGym Thanh Xuân (Hà Nội) - Tầng 2, tòa 17T4 Hapulico, số 1 Nguyễn Huy Tưởng, - Thanh Xuân, Hà Nội - info@codegym.vn - facebook.com/codegym.vn - codegym.vn 0246 253 8829	CodeGym Đà Nẵng - Số 280 Trần Hưng Đạo, phường An Hải Tây, quận Sơn Trà, - Tp Đà Nẵng - danang@codegym.vn - facebook.com/codegymdanang - danang.codegym.vn 0236 651 7021
CodeGym Huế 28 Nguyễn Tri Phương, Phường Phú Nhuận, Tp Huế - hue@codegym.vn - facebook.com/codegymhue - hue.codegym.vn 0234 629 1888	CodeGym Sài Gòn 21K Nguyễn Văn Trỗi, Phường 12, Quận Phú Nhuận, Tp HCM - saigon@codegym.vn - facebook.com/codegymsaigon - codegym.vn 083 997 3636 - 0985 481 005
CodeGym Online - Tầng 11, tòa nhà MD Complex, số 68 Nguyễn Cơ Thạch, Phường Cầu Diễn, Quận Nam Từ Liêm, Tp Hà Nội - online.codegym.vn - facebook.com/huanluyenlaptrinhonline 0989 965 376	CodeGym Thành Đô - QL 32 - Lai Xá - Kim Chung - Hoài Đức - Hà Nội - https://ibs.thanhdo.edu.vn/ - facebook.com/hockysucongngghethongtin 093 368 62 66

Đội ngũ tác giả

Nguyễn Khắc Nhật
Nguyễn Khánh Tùng
Nguyễn Bình Sơn
Phan Văn Luân
Dương Trung Dũng
Đặng Huy Hoà
Dư Thanh Hoàng

