**Bài 1: Căn bản lập trình**

1, **Lập trình** là quá trình tạo ra tập các chỉ dẫn (**instruction**) để **ra lệnh** cho máy tính hoàn thành một công việc (**task**) nào đó

Lập trình bao gồm rất nhiều hoạt động: Tìm hiểu yêu cầu, phân tích, thiết kế, viết code, kiểm thử, triển khai, bảo trì, mở rộng…

2, **Ngôn ngữ lập trình** là **phương tiện** để lập trình viên viết ra các chỉ dẫn cho máy tính.

3, **Thuật toán (algorithm)**

* Thuật toán/giải thuật bao gồm các chỉ thị để giải quyết một vấn đề
* Có thể sử dụng các cách khác nhau để mô tả thuật toán, các cách thông dụng là:

• Mã giả (pseudo-code)

• Lưu đồ (flowchart)

**4, Mã giả (pseudo-code)**

* Liệt kê tuần tự các bước bằng ngôn ngữ tự nhiên để biểu diễn thuật toán

• Ưu điểm:

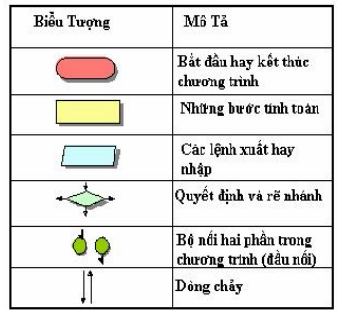
* Đơn giản, không cần kiến thức về cách biểu diễn (lưu đồ, ngôn ngữ lập trình)

• Nhược điểm:

* Dài dòng, không cấu trúc
* Đôi lúc khó hiểu, không diễn đạt được thuật toán

**4, Lưu đồ (flowchart)**

* Lưu đồ mô tả giải thuật bằng các sơ đồ hình khối, mỗi khối quy định một hành động.



**Bài 2: Git and HTML**

**1,** **Source Code Management System**– Hệ thống quản lý mã nguồn là một phần mềm hỗ trợ:

* Phối hợp giữa các thành viên trong một nhóm phát triển phần mềm
* Quản lý tập tin và kiểm soát phiên bản
* Các nhà phát triển khả năng làm việc đồng thời trên các tập tin, hợp nhất với các thay đổi khác của nhà phát triển khác
* Theo dõi và kiểm tra các thay đổi được yêu cầu và thực thi
* Theo dõi tình trạng sửa lỗi và thực thi

**2,Version Control System (VCS)** – Hệ thống quản lý phiên bản mã nguồn là một phần mềm hỗ trợ:

* Khôi phục lại phiên bản cũ của các file
* Khôi phục lại phiên bản cũ của toàn bộ dự án
* Xem lại các thay đổi đã được thực hiện theo thời gian
* Xem ai là người thực hiện thay đổi cuối cùng có thể gây ra sự cố
* Khôi phục lại các file vô tình xoá mất

|  |  |
| --- | --- |
| Tập trung | Phân tán |
| - Toàn bộ mã nguồn và lịch sử thay đổi được lưu trữ trên một máy chủ tập trung (central server)  - Mỗi nhà phát triển tải xuống mã nguồn từ máy chủ trung tâm, chỉnh sửa và tải lên lại khi hoàn thành  - Đồng bộ hóa và hợp nhất phiên bản được thực hiện thông qua máy chủ tập trung  - Đòi hỏi kết nối mạng ổn định để truy cập và tải xuống mã nguồn từ máy chủ tập trung  - Rủi ro dự án cao khi máy chủ tập trung gặp sự cố, có thể ảnh hưởng đến toàn bộ dự án  - Quản lý lịch sử thay đổi tập trung trên máy chủ | - mỗi nhà phát triển có một bản sao đầy đủ của kho mã nguồn và lịch sử thay đổi trên máy tính cá nhân của mình  - Mỗi nhà phát triển quản lý phiên bản trên máy tính cá nhân, có thể làm việc ngoại tuyến và giao tiếp với các phiên bản khác khi cần thiết  - Nhà phát triển có thể đồng bộ hóa và hợp nhất phiên bản trực tiếp với nhau, giúp tiết kiệm thời gian và làm việc độc lập  - Không phụ thuộc vào kết nối mạng, cho phép nhà phát triển làm việc offline và chỉ đồng bộ khi cần thiết  - Rủi ro dự án thấp do mỗi nhà phát triển có bản sao đầy đủ và độc lập của mã nguồn  - Mỗi nhà phát triển quản lý lịch sử thay đổi trên máy tính cá nhân, tạo ra một lịch sử phiên bản riêng biệt |

**3, Git và Github**

* **Git** là một hệ thống điều khiển phiên bản (version control system) theo hình thức phân tán
* **Git** được sử dụng để quản lý mã nguồn (source code) và ghi nhận các thay đổi
* **GitHub** là một dịch vụ cung cấp kho lưu trữ mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm.

**4, Snapshot** là sao lưu tất cả sự thay đổi toàn bộ mã nguồn tại thời điểm hiện tại

* Các snapshot như là các lát cắt được tạo ra trong quá trình phát triển
* Lập trình viên quyết định lúc nào thì tạo một snapshot
* Có thể quay lại một snapshot bất kỳ

**5, Commit** là cách để tạo các **snapshot**

* Các **commit** thường được tạo ra khi có một thay đổi đáng kể đối với mã nguồn:
  + Tạo một tính năng mới
  + Sửa được một lỗi
  + Cải tiến mã nguồn
* Một **commit** bao gồm các thông tin:
  + Thay đổi ở các file so với trước
  + Một tham chiếu đến commit trước nó (gọi là commit cha)
  + Một mã băm đại diện, thường có dạng như

87878747939740429190ca307289c494311e27fe

**6, Repository** là nơi chứa toàn bộ mã nguồn

* Repository bao gồm toàn bộ các file và lịch sử của các file đó
* Repository chứa tất cả các commit
* Có 2 loại repository:
  + Local Repository: Ở trên máy của lập trình viên
  + Remote Repository: Ở trên một máy chủ chia sẻ (chẳng hạn như GitHub)

**7, Clone:** Sao chép một Remote Repository về máy của lập trình viên

**8, Pull**: Cập nhật mã nguồn từ một Remote Repository về Local Repository

**9, Push:** Đẩy mã nguồn từ Local Repository lên Remote Repository

**10, Các câu lệnh cơ bản của Git**

* **git clone**: Sao chép một repository
* **git init**: Khởi tạo một repository
* **git add**: Đưa các file vào trong vùng staged
* **git commit**: Ghi nhận các thay đổi
* **git push**: Đưa các thay đổi từ local repository lên remote repository
* **git pull**: cập nhật từ remote về local. Cố gắng làm thay đổi trạng thái local phù hợp với nội dung. Sử dụng khi có 1 người làm việc trên 1 nhánh. Hạn chế xung đột
* **git fetch**: không làm thay đổi trạng thái local. Dùng trong trường hợp có nhiều người làm việc trên cùng 1 nhánh.

**11, HTML** là ngôn ngữ được sử dụng để tạo ra cácWebpage

* là viết tắt của chữ Hypertext Markup Language (Ngôn ngữ Đánh dấu Siêu Văn bản)
  + Đánh dấu: sử dụng các thẻ
  + Siêu văn bản: có thể đi đến văn bản khác thông qua các liên kết (link)

**12, Các thẻ HTML:**

**<!DOCTYPE>** Xác định cho trình duyệt biết phiên bản HTML mà bạn đang sử dụng

**<html>** Xác định một tài liệu HTML

**<head>** Xác định phần đầu của tài liệu HTML (chứa các thẻ cung cấp thông tin cho trang web)

**<title>** Xác định tiêu đề của trang web

**<body>** Xác định phần thân của tài liệu HTML (chứa những phần tử sẽ được hiển thị lên màn hình trình duyệt)

**<h1> - <h6>** Tiêu đề

**<p> or <span>** Xác định một đoạn văn bản

**<br>** Chèn một ngắt xuống dòng

**<hr>** Tạo một đường kẻ phân cách nằm ngang

**<!-- -->** Xác định một đoạn chú thích

**<ul><li>** Danh sách không có thự tự

**<ol><li>** Danh sách có thứ tự

**<img src = >** Hình ảnh

**<a href= >** Liên kết tới 1 web khác

**<video src = >** Video

<div> để tạo ra một khối hoặc container. Nhóm các khối block

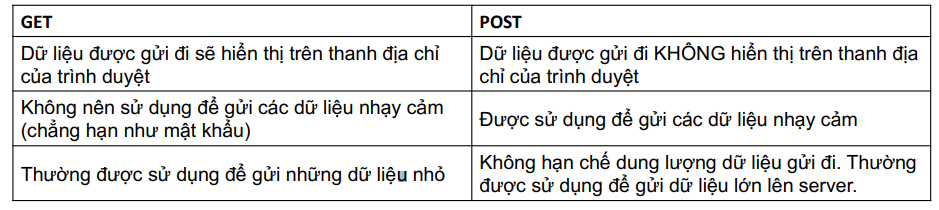
<span> nhóm các khối inline

**Bài 3; HTML Form & Table**

**1, Form**) là cơ chế cho phép người dùng nhập dữ liệu và gửi về server

* Thẻ **<form>** được sử dụng để tạo một form
* Thuộc tính **action** có giá trị là một URL, quy định nơi dữ liệu được gửi đến
* Thuộc tính **name** quy định tên của form, sau đó có thể được dung để truy xuất đến các giá trị trong form
* Thẻ **<fieldset>** và **<legend>** được sử dụng để nhóm các trường dữ liệu trong form
* Thuộc tính **method** được sử dụng để quy định cách thức dữ liệu được gửi lên server

**2, Phương thức POST và GET**

****

**POST:** Bảo mật nên cần được mã hóa dữ liệu, đóng gói, tránh đánh mất dữ liệu nên sẽ chậm hơn phương thức **GET**

**3, Thẻ <input>** được dùng để tạo các trường nhập dữ liệu thông dụng mà chúng ta thường thấy

* Thuộc tính **name**: tên của trường dữ liệu
* Thuộc tính **disabled**: vô hiệu hóa trường dữ liệu, không cho người dùng tương tác
* Thuộc tính **value**: giá trị của trường dữ liệu
* Thuộc tính **type**: loại trường dữ liệu (text, password…)

**Lưu ý**: Dữ liệu được gửi lên server dưới dạng các cặp tên-giá trị

• **Input type: text** được sử dụng để nhập dữ liệu chuỗi ngắn, chẳng hạn như tên, địa chỉ…

**• Input type**: **password** được sử dụng để nhập password, chẳng hạn như trong form đăng ký người dùng, hoặc form đăng nhập…

**• Input type**: **submit** được sử dụng để tạo một nút nhấn, khi người dung nhấn vào nút submit thì dữ liệu sẽ được gửi lên server

**• Input type: reset** được sử dụng để xóa các dữ liệu mà người dùng vừa mới nhập vào trong form, tất cả các trường dữ liệu sẽ quay trở về dữ liệu mặc định trước đó

**• Input type: radio** được sử dụng để cho phép người dùng lựa chọn MỘT trong số các giá trị cho trước

**• Input type: checkbox** được sử dụng để cho phép người dùng lựa chọn KHÔNG, MỘT, hoặc NHIỀU giá trị cho trước

**• Input type: button** được sử dụng để tạo một nút, nút này có thể được gắn với sự kiện của Javascript để xử lý các thao tác của người dùng

**• Các loại input khác:** color, date, datetime-local, email, month, number, range, search, tel, time, url, week

**4, Thẻ select và option:** được sử dụng để cho phép người dung lựachọn từ một danh sách cho trước. Trường nhập dữ liệu này còn đượcgọi là dropdown hoặc combobox

* So sách với input[radio] thì dropdown chiếm ít không gian hơn nhưng lại yêu cầu người dùng phải nhấn chuột thì mới hiển thị đầy đủ các danh sách

**5, Thẻ textarea** được sử dụng để nhập dữ liệu dạng text lớn, chẳng hạn như nội dung của một bài báo, mô tả của một sản phẩm…

* Các thuộc tính quan trọng của thẻ textarea:
  + rows: số lượng dòng (chiều cao)
  + cols: số lượng cột (chiều rộng)

**6, Bảng trong HTML:** Bảng được sử dụng để thể hiện dữ liệu dưới dạng hàng và cột

* Sử dụng bảng khi muốn hiển thị một danh sách các đối tượng với nhiều thuộc tính khác nhau
* Các thành phần của bảng:
  + Cột (**column**)
  + Dòng (**row**)
  + Đường viền (**border**)
  + Tiêu đề (**header**)
  + **Cellspacing**: Khoảng cách giữa các đường viền
  + **Cellpadding**: Khoảng cách từ nội dung đến đường viên của ô **<th> <td>**
* Các thẻ được dùng đẻ tạo bảng: **<table>, <tr>, <td>, <th>**
  + **<table>:** Khai báo một bảng
  + **<tr>:** Khai báo một dòng
  + **<th>:** Khai báo một ô tiêu đề
  + **<td>:** Khai báo một ô dữ liệu
  + **<caption>:** định nghĩa phụ đề bảng
* Ô chiếm nhiều cột: Sử dụng thuộc tính **colspan**
* Ô chiếm nhiều dòng: Sử dụng thuộc tính **rowspan**

**Bài 4: Javascript**

**1, Javascript:** là ngôn ngữ lập trình được sử dụng nhiều trên web. JS có thể sử dụng lập trình game, web-service, xử lý sự kiện người dùng

* JS sẽ có các chức năng sau:
  + Thay đổi giao diện
  + Thay đổi nội dung
  + Xử lý sự kiện người dùng => Tăng tính tương tác giữa người dùng và website
  + Có thể viết cả front end và back end không cần sử dụng thêm một ngôn ngữ nào khác.
* Nhược điểm: Không thể chạy trên trình duyệt không hỗ trợ or bị tắt JS
  + Tính bảo mật không cao

**2, Nhúng Javascript vào trang web**

* Viết mã Javascript bên trong thẻ <script></script> (internal)
* Sử dụng file.js. Khai báo đường dẫn của file.js trong thuộc tính src của thẻ <script> (external)
* Viết trực tiếp mã JavaScript trong thẻ <html>(inline)

\*Đối với các dự án lớn hoặc các file js dài thì thường sử dụng cách nhúng External: Tách biệt code HTML và JS dễ kiểm soát => Dễ dàng sữa chữa, nâng cấp JS mà không gây ảnh hưởng đến HTML

JS nên nhúng ở cuối body:

* Tăng tốc độ tải trang
* Tránh gây các lỗi không đáng có nên như file js có tương tác đến thành phần chưa được render ở HTML

|  |  |  |
| --- | --- | --- |
|  | **Ưu điểm** | **Nhược điểm** |
| **Inline** | * Dễ nhận thấy ngay tại thẻ đang sử dụng mã JS * Không cần upload nhiều file | * Nếu quá nhiều dòng html và quá nhiều các lệnh js thì rối * Không thể tái sử dụng JS (mỗi thẻ chỉ dùng 1 lần) |
| **Internal** | * Dễ nhận thấy sự kiện được bắt ngay tại trang html * Có thể tái sử dụng (trong cùng 1 file html) * Không cần upload nhiều file | * Không thể tái sử dụng cho nhiều file html khác nhau |
| **External** | * Có thể tái sử dụng linh hoạt tại nhiều nơi trong file html và nhiều file html * File html riêng và js riêng làm cho source code rõ rang, sạch đẹp hơn | * Không hiển thị trưc tiếp => khi cần fix, update, delete 1 hay nhiều tính năng thì phải cần tham chiếu chính xác * Nhiều file |

**3, Hàm thông báo trong JS**

* Alert(): Thông báo thông tin đến người dùng
* Prompt(): Thông báo cho người dùng, đồng thời yêu cầu người dùng cung cấp thông tin
* Confirm(): Thông báo xác nhận true or false

**4, Hiển thị dữ liệu đầu ra**

* Dùng document.write(“……….”)
* Dùng document.getElementById(“myID”).innerText
* Dùng document.getElementById(“myID”).innerHTML
* Dùng console.log()

**5, Debug** (dò lỗi) là quá trình tìm kiếm và sửa chữa các lỗi trong một chương trình

* **Các loại lỗi thường gặp:**
  + Lỗi cú pháp (syntax error) dễ phát hiện bởi vì compiler đưa ra thông báo
  + Lỗi thực thi (runtime error) cũng dễ phát hiện bởi vì interpreter cũng hiển thị lỗi trên màn hình
  + Lỗi logic thường khó phát hiện hơn
  + Các lỗi logic còn được gọi là bug
  + Thông thường, cần kết hợp nhiều cách để tìm được bug
* **Một số phương pháp debug:**
  + Đọc mã nguồn (hand-trace)
  + Chèn các lệnh in ra các giá trị trong từng đoạn của chương trình để kiểm tra các giá trị và việc thực thi các câu lệnh
  + Sử dụng debugger: một chương trình cho phép quan sát quá trình thực thi của một ứng dụng
  + Các IDE thông thường cũng tích hợp sẵn debugger
  + Các công cụ debug của trình duyệt

**Bài 5: Biến, kiểu dữ liệu và toán tử**

**1, Biến (variable)** là một tên gọi được gắn cho một vùng nhớ chứa dữ liệu

* Dữ liệu được lưu trữ trong vùng nhớ của biến được gọi là giá trị (value)
* Có thể truy nhập, gán hay thay đổi giá trị của biến
* Khi gán một giá trị mới thì giá trị cũ sẽ bị ghi đè lên

**2, Cách khai báo biến:**

* **Let:** variable name
* **Var:** variable name
* **Const:** variable name (biến hằng)

|  |  |
| --- | --- |
| **Let** | **Var** |
| * Không được khai báo cùng tên * Bắt buộc khai báo trước khi dùng * Phạm vi hoạt động ở trong block scope{} | * Được khai báo cùng tên * Không bắt buộc khai báo trước khi dùng * Phạm vị hoạt động nằm trong function scope{} |

* **Hoisting** là cơ chế mặc định của JavaScript để di chuyển tất cả các biến và hàm khi khai báo lên đầu scope trước khi chúng được thực thi.
* **Lưu ý** đối với cơ chế này nó chỉ di chuyển khai báo, còn việc gán giá trị thì giữ nguyên
* **Const (biến hằng):** là một tên gọi đại diện cho một giá trị cố định
  + Giá trị của hằng không thể thay đổi
  + Giá trị của hằng cần phải được gán tại thời điểm khai báo
  + Còn lại giống **Let**

**3, Quy tắt đặt tên biến:**

* Đặt tên có ý nghĩa, đúng ý nghĩa của biến
* Tên biến phải là **danh từ**
* Tên biến chỉ được phép bắt đầu với \_ hoặc $ hoặc a-z. Không được phép bắt đầu bằng những ký tự còn lại
* Đặt tên theo quy tắc CamelCase

**4, Kiểu dữ liệu (Data Type)**

* **Primitive (nguyên thủy)**
  + **Number:** số nguyên, số thực
  + **String:** chuỗi “”
  + **Boolean:** true and false
  + **Undefined:** kiểu dữ liệu undefined, giá trị undefined
  + **Null:** kiểu dữ liệu Object, giá trị null
  + **Symbol**
* **Reference – Object (đối tượng)**
  + **Object:** có 2 loại: - Do JS cung cấp sẳn (Date, Math) –Do lập trình viên tự tạo ra thông qua các class
  + **Array**
  + **Funtion**
* **Kiểm tra kiểu dữ liệu: typeOf**
* Chuyển đổi String => Số
  + paseInt: đổi sang số nguyên
  + paseFloat: đổi sang số thực
  + + or \*1

**5, Toán tử (Operator)** là các ký hiệu được sử dụng để thực hiện các thao tác trongcác biểu thức và sinh ra kết quả cuối

* Có nhiều loại toán tử khác nhau:
  + Toán tử toán học
  + Toán tử gán
  + Toán tử cộng chuỗi
  + Toán tử so sánh
  + Toán tử logic
  + Toán tử typeof
* **Lưu ý: ==** So sánh giá trị hình thức. === So sánh giá trị và kiểu dữ liệu
* **Độ ưu tiên toán tử: số học – so sánh – logic – gán.** Để thay thế độ ưu tiên sử dụng()

**Bài 6: Cấu trúc điều kiện**

**1, Câu lệnh điều khiển:**

* Một chương trình phần mềm thực thi các câu lệnh theo trật tự từ trên xuống dưới
* 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)

**2, Câu lệnh điều kiện:**

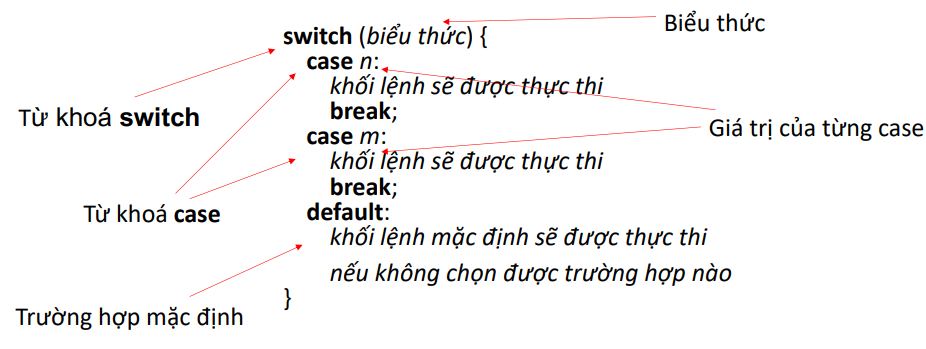
* Câu lệnh sẽ thực hiện các hành động khác nhau dựa trên các điều kiện đúng or sai
* JavaScript hỗ trợ các câu lệnh điều kiện:
  + **If**
  + **switch-case**

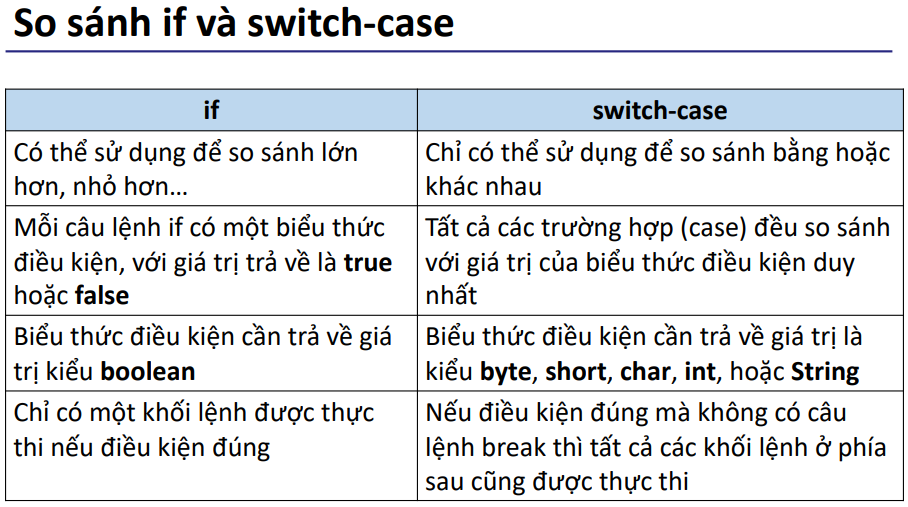
**3, Câu lệnh if:** Đánh giá một điều kiện và đưa ra lựa chọn thực thi một khối lệnh dựa trên điều kiện đó

* **If đơn: If (condition) {…}**
* **If đôi: If(condition) {…} else{…}**
* **If lồng nhau**
* **If bậc thang:** khi điều kiện có miền giá trị liên tiếp nhau

**4, Câu lệnh switch-case:** là một cấu trúc điều kiện cho phép lựa chọn thực thi các

khối lệnh khác nhau dựa trên kết quả của việc so sánh





**5, Biểu thức điều kiện (toán tử 3 ngôi)**

* Biểu thức điều kiện đánh giá một biểu thức dựa vào một điều kiện cho trước
* **Cú pháp:**

condition ? expression\_true\_case : expression\_false\_case

**Bài 7: Cấu trúc vòng lặp**

**1, Vòng lặp (loop):** 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
* Trong JS có 3 loại cơ bản: for, while và do-while

**2, Vòng lặp for:** thường được sử dụng khi biết trước số lần lặp

for (initial-action(1); loop-continuation-condition(2); action-after-each-iteration(4)) {

statement(s)(3); }

(1): Khối lệnh khởi động biến lặp

(2): Khối lệnh điều kiện

(3): Nội dung lặp

(4): Khối lệnh tang or giảm biến lặp

**3, Vòng lặp while:** thực thi lặp lại một khối lệnhnếu biểu thức điều kiện trả về giá trị đúng. while thường được sử dụng khi không biết số lần lặp.

* **Cú pháp: while** (loop-continuation-condition) { statement(s); }

**4, Vòng lặp do-while:** thực hiện ít nhất là một lần. Kể cả khi biểu thức đk trả về sai

* **Cú pháp:** do { statements(s); } while (loop-continuation-condition);

|  |  |
| --- | --- |
| **Break** | **Continue** |
| * Thoát vòng lặp chứa nó, bỏ qua tất cả các lệnh ở sau nó * Còn được dùng trong cấu trúc lựa chọn switch | * Nhảy qua một phần của khối lệnh và bắt đầu dòng tiếp theo * Khi gặp lệnh này trong các vòng lặp, chương trình sẽ bỏ qua phần còn lại trong vòng lặp và tiếp tục thực hiện lần lặp tiếp theo |

**Bài 8: Mảng**

**1, Mảng:** là một loại biến đặc biệt, có thể lưu được nhiều giá trị thay vì chỉ một giá trị như các biến thông thường

* Mỗi giá trị trong mảng được gọi là một phần tử
* Các phần tử được lưu trữ ở các vị trí kế tiếp nhau trong bộ nhớ
* Tên mảng: Tuân thủ theo quy tắc đặt tên của biến CamelCase
* Phần tử: Các giá trị được lưu trữ trong mảng
* Chỉ số: Vị trí của các phần tử trong mảng. Chỉ số bắt đầu từ 0.
* Độ dài: Số lượng các phần tử của mảng, độ dài có thể thay đổi được

**2, Khai báo Mảng:**

* Dùng []: let myArray = [];
* Dùng Array or New Array: let myArray = new Array();
* Lưu ý: new Array(5) => tạo mảng Array có 5 phần tử

**3, Phần tử và chỉ số của mảng**

* Mỗi phần tử (item) được xác định bằng một số thứ tự còn gọi là chỉ số (index) duy nhất trong mảng
* Chỉ số là một số nguyên dương
* Chỉ số của phần tử đầu tiên là 0
* Chỉ số của phần tử cuối cùng là n – 1, trong đó n là độ dài của mảng
* Có thể truy xuất đến phần tử của mảng thông qua chỉ số

**4, Truy xuất phần tử trong mảng**

* Mỗi phần tử trong mảng được thao tác giống như một biến
* Truy xuất các phần tử thông qua chỉ số đặt trong dấu []

**5, For in: for (let index in arr ) {…}:** duyệt index của mảng

**6, For of: for (let index of arr) {…}:** duyện value của mảng

|  |  |
| --- | --- |
| **For i** | **For in, For of** |
| * Có thể duyệt theo nhiều chiều khác nhau (từ đầu đến cuối or từ cuối đến đầu) | * Chỉ duyệt từ đầu đến cuối mảng |

**7, Các thuộc tính và phương thức của mảng**

* **Push()** them phần tử mới vào cuối mảng
* **Unshift()** them phần tử mới vào đầu mảng
* **Pop()** xóa phần từ cuối mảng
* **Shift()** xóa phần tử vào đầu mảng
* **Sort()** sắp xếp mảng theo một trật tự nhất định
* **Join()** chuyển tất cả các phần tử trong mảng thành chuỗi và nối chúng lại với nhau
* **Reverse()** đảo ngược các phần tử trong mảng
* **Concat()** nối phần tử của 2 mảng với nhau
* **toString()** chuyển toàn bộ các phần tử trong mảng sang dạng chuỗi
* **splice()** thêm or xóa phần tử trong mảng
* **include()** tìm kiếm trong mảng. có trả về true, không có trả về false
* **indexOf() tìm kiếm** có trả về index, không có trả về false
* **split()** chia một chuỗi thành 1 mảng các chuỗi con

**8, Mảng đa chiều:** là mảng có các phần tử là các mảng khác

* Có thể có mảng 2 chiều, 3 chiều… hoặc nhiều hơn
* Mảng càng nhiều chiều thì độ phức tạp khi xử lý càng cao
* Mảng hai chiều là mảng nhiều chiều được sử dụng phổ biến
* Mảng hai chiều bao gồm các phần tử, mỗi phần tử là một mảng một chiều.
* Mảng hai chiều là một bảng gồm n dòng và m cột: Số phần tử trong mảng là n \* m
* Index: các chỉ số dòng và cột

**Bài 9: Hàm**

**1, Hàm (function)** là một nhóm các câu lệnh thực hiện một nhiệm vụ nhất định

**2, Cú pháp của hàm**

**Function** name(parameter1, parameter2,…){body}

**Name**: là động từ, danh động từ, tuân thủ theo quy tắc đặt tên biến CamelCase

**Parameter**: tham số có thể có or không. Số lượng tham số không rang buộc

Mục đích sử dụng hàm: - tái sử dụng mã nguồn, code sạch đẹp, dễ bảo trì sữa chữa và mở rộng

* **Tham số**: là biến được khai báo ở phần header của function. Còn có tên gọi khác là tham số hình thức
* **Đối số(argument):** là giá trị được truyền vào lời gọi hàm và giá trị này được gọi tham số thực

**3, Có 2 loại hàm:-** Hàm không có giá trị trả về

* Hàm có giá trị trả về:
  + Dùng từ khóa **return** trong than hàm để trả về kết quả của 1 hàm
  + Mỗi hàm chỉ có 1 giá trị trả về và giá trị có thể là biến, số, chuỗi, mảng
  + Gặp từ khóa **return** thì sẽ thoát hàm hiện tại
  + Để nhạn giá trị trả về của hàm thì có thể dùng toán tử gán

**4, Phạm vi của biến**

* **Phạm vi (scope) của biến** là các vị trí trong chương trình mà một biến có thể được sử dụng
* **Biến cục bộ (local variable)** là biến được khai báo trong hàm, tham số của hàm. Biến cục bộ có phạm vi hoạt động ở trong hàm
* **Biến toàn cục(global variable)** là biến được khai báo ngoài hàm. Biến toàn cục có thể sử dụng ở trong or ngoài hàm

**5, Tham trị, tham chiếu**

* **Truyền tham trị (pass by value)** lấy giá trị và truyền vào hàm (gán cho tham số của hàm)
  + Áp dụng cho kiểu nguyên thủy
  + Giá trị của biến trước và sau khi gọi hàm không thay đổi giá trị
* **Truyền tham chiếu(pass by reference): truyền địa chỉ vào ô nhớ**
  + Áp dụng cho kiểu tham chiếu (Object,array)
  + Giá trị của biến sau khi gọi hàm có thể bị thay đổi

**Bài 10: OOP**

**1, Lập trình hướng đối tượng(Object oriented programming):** là một kỹ thuật lập trình cho phép lập trình tạo ra các đối tượng trong code, ánh xạ các đối tượng thực tế vào trong lập trình

* Đối tượng bao gồm:
  + **Thuộc tính(property)**: các dữ liệu, tính chất của đối tượng
  + **Hành vi(behavior):** các khả năng, hành động mà đối tượng có thể thực hiện
* Các đối tượng có thể có quan hệ với nhau
* Ưu điểm:
  + Khả năng tái sử dụng cao
  + Ổn định và dễ bảo trì
  + Chi phí giảm dần
* **Đối tượng (Object) Lớp (Class) Thuộc tính (Property) Phương thức (Method)/Hành vi (Behavior)/Hành động (Action)/Khả năng(Capability)**

**2, Đối tượng (Object)**

* Mỗi đối tượng có những thuộc tính hay những đặc điểm mô tả và những hành vi riêng nhằm phân biệt nó với các đối tượng khác.
* Thể hiện của **class** (là sản phẩm được tạo ra từ class)
* **Thuộc tính** là những đặc điểm đặc trưng của đối tượng, thể hiện thông qua những giá trị cụ thể.
* **Hành vi** là những cách thức mà qua đó đối tượng thể hiện sự hoạt động hay chức năng của chúng.

**3, Lớp (Class)**

* là khái niệm dùng để mô tả một loại đối tượng có những thuộc tính, hành vi và những mối quan hệ thông thường tương tự nhau
* **Class** là khuôn mẫu định nghĩa các thuộc tính và hành vi chung cho tất cả các đối tượng cùng lớp

**4, Constructor:** hàm dùng để khởi tạo thông tin ban đầu cho đối tượng

* Được gọi thông qua từ khóa new
* Trong JS chỉ có tối đa 1 constructor

**5, Khởi tạo class:**

* Dùng Function
* Dùng từ khóa Class
* Lưu ý: tên class bắt đầu chữ cái in hoa, danh từ
* Có 2 cách khởi tạo đối tượng:
  + Sử dụng object literal: let objectName = {key1:value1, key2:value2}
  + Sử dụng từ khóa new: let objectName = new NameClass (…)

**6, Tính chất lập trình OOP**

* **Tính bao gói (encapsulation)**
* **Tính đa hình (polymorphism)**
* **Tính trừu tượng (abstraction)**
* **Tính kế thừa (inheritance)**
* **Tính bao gói:** là khả năng cho phép truy cập vào các thành phần của đối tượng trong khi vẫn đảm bảo che dấu các đặc tính riêng tư bên trong của đối tượng
  + Triển khai tính bao đóng sử dụng getter setter
* **Trình trừu tượng:** loại bỏ những thuộc tính và hành vi không quan trọng của đối tượng. Chỉ giữ lại những thuộc tính và hành vi liên quan đến đối tượng đang giải quyết =>Loại bỏ sự phức tạp của đối tượng, ẩn các chi tiết triển khai, chỉ hiển thị tính năng/kết quả liên quan đến bài toán đang làm
* **Tính đa hình:** là khả năng đối tượng có thể hiện hành vi theo nhiều cách khác nhau tùy thuộc vào ngữ cảnh
  + Tính đa hình có thể hiện khi cùng 1 phương thức / hành vi nhưng có thể có các cách ứng xử khác nhau ở những đối tượng khác nhau
  + Triển khai: Overloading/ overriding
* **Tính kế thừa:** cho phép đối tượng chia sẽ, mở rộng các thuộc tính or phương thức mà không cần định nghĩa lại
  + **Kế thừa** cho phép lớp con (Child class,sub class, derived class) sử dụng lại các đặc điểm hành vi đã được định nghĩa trong lớp cha (super class, parent class, base class)
  + Dùng từ khóa **Extends** đến triển khai kế thừa