**1. PSEUDO CODE & FLOWCHART**

- **Lập trình** là **quá trình** tạo ra **chỉ dẫn** (instruction) để ra lệnh cho máy tính hoàn thành 1 công việc (task).

- **Ngôn ngữ lập trình** là **phương tiện** lập trình viên tạo ra các chỉ dẫn để máy tính hiểu (Javascript, Java,...).

- **Thuật toán** bao gồm các **chỉ thị** (các bước) để giải quyết 1 vấn đề.

+ Mã giả.

+ Lưu đồ.

+ Ngôn ngữ lập trình.

|  |  |
| --- | --- |
| **Pseudo code** | **Flowchart** |
| - Sd ngôn ngữ tự nhiên để biểu diễn các bước (English).  - **Ưu điểm:** đơn giản, k cần nhiều kiến thức về cách biểu diễn (lưu đồ, ngôn ngữ lập trình,…).  - **Nhược điểm:** dài dòng, k cấu trúc, khó hiểu, khó diễn đạt được thuật toán. | - Sd sơ đồ khối (mỗi khối là 1 hành động) để mô tả thuật toán.  - **Ưu điểm:** quy định chặt chẽ về kí hiệu, thống nhất, dễ hiểu, trực quan.  - **Nhược điểm:** khó hiểu cho những ai k nắm đc các kí hiệu (bắt buộc phải học thuộc), khó gỡ lỗi. |

**2. GIT & HTML**

- **VCS** **(Version Control Systems – Hệ thống quản lý phiên bản mã nguồn)** - là 1 **hệ thống** lưu giữ các phiên bản [mã nguồn](https://vi.wikipedia.org/wiki/M%C3%A3_ngu%E1%BB%93n) của sp phần mềm, giúp các [lập trình viên](https://vi.wikipedia.org/wiki/L%E1%BA%ADp_tr%C3%ACnh_vi%C3%AAn) có thể dễ dàng lấy lại phiên bản mong muốn.

+ **Tập trung:** bao gồm 1 máy chủ chứa các tệp tin đc phiên bản hóa, các máy khách có quyền thay đổi các tệp tin này trên 1 máy chủ trung tâm đó.

+ **Phân tán:** các máy khách k chỉ sao chép về máy của mình phiên bản mới nhất của các tệp tin mới nhất, mà chúng còn sao chép toàn bộ kho chứa mã nguồn.

- **GIT** là **hệ thống** dùng để làm việc cộng tác với nhau trong dự án, giải quyết các vấn đề về quản lí mã nguồn (GIT là 1 VCS theo hình thức phân tán).

- **Github** là 1 dịch vụ quản lí mã nguồn online.

- **Các lệnh cơ bản của Git:**

+ **git clone –** sao chép remote repository (folder trên server Github) về local repository (folder trong máy tính cá nhân).

+ **git add . –** thêm file hoặc ghi nhận file có sự thay đổi để đưa lên remote repository.

+ **git commit –m”…” –** thực hiện tại 1 phiên bản, sao lưu all sự thay đổi tại thời điểm hiện tại.

+ **git push –** lấy mã nguồn từ local repository lên remote repository.

+ **git init –** tạo mới repository.

+ **git pull –** cập nhật từ remote repository về local repository.

- **HTML** **– Hyper Text Markup Language** (ngôn ngữ đánh dấu siêu văn bản), dùng để tạo **khung** ban đầu của trang web.

- **2 nhóm thẻ chính:**

+ **Inline** – khoảng k gian của thẻ vừa khớp với nội dung.

+ **Block** – khoảng k gian của thẻ chứa toàn bộ chiều rộng màn hình.

- **1 số thẻ cơ bản:**

**<head></head>** khai báo tên (title) của trang web, nhúng CSS, Javascript,…

**<body></body>** nội dung được hiển thị ở web.

**<h></h>** tiêu đề (h1, h2,... h6).

**<p></p>** văn bản (kiểu block).

**<span></span>** văn bản (kiểu inline).

**<img src="…">** nhúng hình ảnh (thuộc tính src).

**<a href="…"> </a>** nhúng trang web (thuộc tính href).

**<ul></ul>, <li></li>** danh sách k có thứ tự.

**<ol></ol>, <li></li>** danh sách có thứ tự.

**3. FORM & TABLE**

- **Form** cho phép người dừng nhập dữ liệu và gửi lên server (máy chủ web).

- **Thuộc tính:**

+ **Action** – mô tả **URL** của server, nơi mà dữ liệu đc gửi đến.

+ **Method** – xác định kiểu **phương thức** sử dụng để gửi dữ liệu (POST & GET).

|  |  |
| --- | --- |
| **GET** | **POST** |
| - Dữ liệu hiển thị trên URL.  - K bảo mật, k nên sd để lưu các dữ liệu nhạy cảm (mật khẩu).  - Có giới hạn dung lượng dữ liệu gửi. | - Dữ liệu k hiện thị trên URL.  - Bảo mật, đc sd để lưu các dữ liệu nhạy cảm.  - K giới hạn data khi gửi. |

**<input type=”…”>** tạo ô nhập liệu.

**<input type=”checkbox”>** tạo ô tích (nhiều lựa chọn).

**<input type=”radio”>** tạo ô tích (1 lựa chọn).

**<selec>, <option>** chọn 1 trong 1 danh sách.

**<input type="submit">** gửi thông tin đến liên kết.

**<input type="reset">** xóa các thiết lập vừa tạo.

**placeholder** giữ chỗ tạm thời.

**name** liên kết (link).

**<label><label>** cái nhỏ.

**<textarea></textarea>** tạo ô nhập văn bản kích thước thay đổi đc.

- **Table** là nơi lưu trữ thông tin dữ liệu các đối tượng theo dạng hàng & cột.

**<table** border="1" cellpadding="…" cellspacing="…"**></table>**

**<tr></tr>, <th></th>, <td></td>,** rowspan=…, colspan=…, align=”…”, style=”…”, …

**4. JAVASCRIPT**

- **Javascript** là 1 ngôn ngữ lập trình sd nhiều trên các website (có thể sd lập trình game, web-service, xử lí sự kiện người dùng, …).

+ **Chức năng:**

- Thay đổi giao diện.

- Thay đổi nd hiển thị.

- Xử lí các sự kiện của người dùng, tăng tính tương tác giữa người dùng và web.

+ **Các cách nhúng:**

- Nhúng trực tiếp (**inline**).

- Code js trong thẻ <script> (**internal**).

-Sd file **\*.js** (dùng thuộc tính “**src**” của thẻ <script> để khai báo đường dẫn) (**external**).

- **Hàm thông báo:**

+ **alert( )** thông báo thông tin đến người dùng.

+ **prompt( )** thông báo thông tin đến người dùng, đồng thời yêu cầu nhập cung cấp thông tin.

+ **confirm( )** thông báo xác nhận đúng (true), sai (false).

- **Biểu diễn dữ liệu đầu ra:**

+ document.write(“…”);

+ document.getElementById(“myId”).innerText = …;

+ document.getElementById(“myId”).innerHTML = …;

**5. VARIABLE, DATA TYPES & OPERATOR**

- **Biến** là tên gọi đặt cho 1 vùng nhớ dữ liệu.

+ Dữ liệu trong vùng nhớ đc gọi là giá trị.

+ Giá trị của biến có thể ghi đè.

+ Phải khai báo biến trc khi sd.

|  |  |
| --- | --- |
| **var** | **let** |
| - Có thể khai báo đè.  - Có cơ chế **hoisting** cho phép sử dụng trc khi khởi tạo.  - Phạm vi truy cập của biến nằm trong **function scope**. | - K khai báo lại.  - K có cơ chế **hoisting**.  - Phạm vi truy cập của biến nằm trong **block scope.** |

- **Quy tắc đặt tên biến:**

+ Bắt đầu là **\_** hoặc **$** hoặc **a**->**z**.

+ Quy tắc **camel Case** (con lạc đà).

+ Đặt tên biến phải có ý nghĩa.

- **Khai báo hằng số:** **const** name = value;

=> Khi khai báo phải gán giá trị cho hằng (giá trị này k thể thay đổi).

- **Kiểu dữ liệu** là cách phân loại dữ liệu giúp cho chương trình biết lập trình viên đang muốn sd dữ liệu phân loại nào.

+ **Kiểu nguyên thủy:** string, number, boolean, undefined, null, symbol, …

+ **Kiểu đối tượng:** object, array, …

- **Toán tử** là kí hiệu trong biểu thức để thực hiện các thao tác.

+ **Toán tử số học:** +, -, \*, /, %, ++, --

+ **Toán tử so sánh:** >, <, >=, <=, ==, ===, !=, !==

+ **Toán tử logic:** &&, ||, !

+ **Toán tử gán:** =, +=, -=, \*=, /=, %=

+ **Toán tử cộng chuỗi:** +

- Xác định kiểu dữ liệu của 1 biến: **typeof**.

- Độ ưu tiên: **số học** > **so sánh** > **logic** > **gán**. Dấu **( )** để thay đổi độ ưu tiên.

**6. CONDITIONAL STATMENTS**

- **Câu lệnh if**

+ If khuyết.

+ If đầy đủ.

+ If lồng nhau (nhiều biến).

+ If bậc thang (1 biến).

- **Câu lệnh switch-case**

**switch** (condition) **{**

**case** x: …

**break**;

**default**: …

**}**

|  |  |
| --- | --- |
| **if** | **switch-case** |
| - **So sánh** hơn, bằng/ k bằng.  - Biểu thức đk có giá trị trả về là kiểu **boolean.**  - Chỉ có 1 khối lệnh đc thực thi khi mà đk đúng/ sai.  - Nên sd tối đa 3 đk if. | - Chỉ dùng trong **so sánh** bằng/ k bằng.  - Biểu thức có thể là kiểu **string, number, boolean,** …  - Nếu đk đúng mà k có break thì all các khối lệnh đằng sau đc thực thi.  - Hỗ trợ kiểm tra nhiều đk. |

**7. LOOP STATEMENTS**

- **Vòng lặp** cho phép tự động thực hiện 1 khối lệnh lặp đi lặp lại nhiều lần dựa vào 1 đk cho trước.

- **for (**khối lệnh khởi tạo biến lặp; đk lặp; khối lệnh tăng giảm **) {**

nội dung lặp; **}**

(dùng khi biết trước số lần lặp).

- **while (**đk lặp**) {**

nội dung lặp; **}**

(dùng khi k biết trước số lần lặp, có thể k đc thực thi nếu k thỏa đk từ đầu).

- **do {**

nội dung lặp;

**} while (**đk lặp**)**

(k biết trước số lần lặp, đc thực hiện ít nhất 1 lần kể cả khi k thỏa đk).

- **break:** thoát vòng lặp hiện tại.

- **continue:** nhảy vòng lặp tiếp theo.

**8. ARRAY**

- **Mảng** là 1 biến đặc biệt (biến mảng), có thể lưu trữ đc nhiều giá trị.

+ Mỗi **giá trị** trong mảng là 1 **phần tử**.

+ Các phần tử của mảng đc lưu trữ ở các vị trí kế tiếp nhau trong ô nhớ.

+ Quy tắc đặt tên mảng phải tuân theo quy tắc đặt tên biến.

+ Chỉ số (**index**) – vị trí phần tử trong mảng, có vị trí bắt đầu từ **0** đến **length-1**.

- **Khai báo mảng:**

+ **Dùng [ ]:** let … = […];

**+ Dùng newArray:** let … = new Array(…);

+ **Dùng Array:** let … = Array(…);

+ Note: let … = new Array(13); //tạo mảng có 15 phần tử undefined.

- **Duyệt mảng 1 chiều:**

+ Cập nhật phần tử bằng **toán tử gán**.

+ Dùng **for, while, do-while** để duyệt.

+ **for-in:** hỗ trợ duyệt **index** của mảng.

**for** (let *element* **in** array) {

console.log(“element = “ + array[*element*];

}

+ **for-of:** hỗ trợ duyệt **value** của mảng.

|  |  |
| --- | --- |
| **for** | **for-in/ for-of** |
| - Duyệt mảng theo nhiều chiều (duyệt từ đầu đến cuối, cuối đến đầu, từ giữa ra 2 bên, có thể duyệt k hết mảng).  - Có khai báo và sd biến đếm.  - Cấu trúc gồm 4 phần (khối lệnh khởi tạo biến lặp, đk lặp, khối lệnh tăng giảm, nội dung). | - Chỉ duyết mảng theo 1 chiều (từ đầu đến cuối, và duyệt hết mảng).  - K cần khai báo và sd biến đếm.  - Cấu trúc chỉ gồm 2 phần (phần khai báo và phần thực thi khối lệnh). |

- **Mảng 2 chiều** thực chất là mảng 1 chiều, mà mỗi phần tử là 1 mảng 1 chiều khác.

**Truy xuất phần tử:** array2d[row-index][col-index].

- **1 số hàm thông dụng:**

+ **array.length** độ dài mảng.

+ **array.push(…)** thêm phần tử vào cuối mảng (cách khác: **array[array.length] = “…”**).

+ **array.pop( )** xóa phần tử cuối mảng.

+ **array.shift( )** xóa phần tử đầu tiên của mảng.

+ **array.unshift(…)** thêm phần tử vào đầu mảng.

+ **array.join( )** chuyển các phần tử sang chuỗi và nối chúng lại với nhau.

+ **array.toString( )** chuyển mảng sang chuỗi.

+ **array.reverse( )** đảo ngược mảng.

+ **array.sort( )** sắp xếp.

+ **array1.concat(**array2**)** nối 2 mảng với nhau.

+ **array.splite(**i, n**)** xóa n phần tử từ vị trí i.

**array.splite(**i, 0, a, b, c,…**)** thêm a,b,c,… từ vị trí i.

+ **array.includes(**i**)** kiểm tra giá trị có tồn tại hay k (trả về **true/ false**).

+ **indexOf (…)** kiểm tra giá trị có tồn tại hay k (có thì xuất ra **index**, k thì trả về **-1**).

**9. FUNCTION**

- **Hàm** là 1 nhóm các câu lệnh thực thi 1 nhiệm vụ nhất định. (mục đích: tái sự dụng mã nguồn và “chia để trị”).

- **Tham số** **(parameter)** là biến đc khai báo ở phần header of function (tham số hình thức).

- **Đối số** **(argument)** là giá trị được truyền vào khi gọi hàm (tham số thực).

|  |  |
| --- | --- |
| **Hàm có trả về** (return) | **Hàm k có trả về** |
| - Thân hàm ngắn, dễ hiểu, bên ngoài code nhiều hơn.  - Muốn sd ở 1 bài toán khác thì k cần phải tạo mới.  - Giá trị gán biến cho hàm = giá trị của lệnh return.  - Mỗi hàm chỉ có 1 giá trị trả về (biến, số, chữ, mảng…).  - Dùng toán tử gán để nhận giá trị trả về. | - Thân hàm code nhiều hơn, bên ngoài code ít hơn.  - K thể thay đổi nội dung thông báo khi sd, nếu muốn sd với những giá trị khác, nội dung thông báo khác thì phải tạo hàm mới.  - Giá trị gán biến cho hàm = undefined. |

- **Phạm vi của biến** là các vị trí trong chương trình mà 1 biến có thể sd đc.

- **Biến cục bộ** **(local variable)** là biến đc khai báo trong hàm, tham số của hàm (sd trong function).

- **Biến toàn cục** **(global variable)** là biến đc khai báo ngoài hàm (sd ngoài hoặc trong function).

|  |  |
| --- | --- |
| **Truyền tham trị** (pass by value) | **Truyền tham chiếu** (pass by reference) |
| - Lấy giá trị truyền vào hàm (gán cho tham số của hàm).  - Áp dụng cho kiểu nguyên thủy (number, string, boolean, symbol, undefined, null…).  - Giá trị của biến trc và sau khi gọi hàm k thay đổi.  - Khi gán cho nó 1 giá trị => lưu lại giá trị đó và tại 1 thời điểm chỉ lưu 1 giá trị. | - Áp dụng cho kiểu dữ liệu phức tạp hơn (array, object, function…).  - Giá trị của biến sau khi gọi hàm “có thể” thay đổi.  - Khi gán cho nó 1 giá trị => k lưu lại giá trị, mà chỉ lưu lại địa chỉ của ô nhớ lưu giá trị này. |

**10. OPP**

- **OPP** là kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code, ánh xạ các đối tượng trong thực tế vào ngôn ngữ lập trình (mô phỏng, mô hình hóa dữ liệu).

- **Đối tượng:**

+ **Thuộc tính** (atribute/property) – các dữ liệu về tính chất, đặc điểm của đối tượng.

+ **Hành vi** (function/method) – các khả năng, hành động của đối tượng.

- **Class** dùng để mô tả nhóm các đối tượng có thuộc tính và hành vi tương tự nhau, là **khuôn mẫu** định nghĩa các thuộc tính và hành vi chung cho all các đối tượng cùng loại.

- **Object** là thể hiện của class (sp đc tạo ra từ class). Mỗi object có giá trị và hành vi khác nhau để phân biệt với các object khác.

- **Construction** là hàm dùng để khởi tạo đối tượng, được gọi thông qua từ khóa “new”. Trong js chỉ có tối ta 1 construction trong 1 class.

- **Các cách khởi tạo:**

+ Sd function để tạo **class** (dùng từ khóa “new” để khởi tạo đối tượng).

+ Sd class để tạo **class** (dùng từ khóa “new” để khởi tạo đối tượng).

+ Sd object literals để tạo **object**.

- **Encapsulation** **(tính bao đóng)** 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 riêng tư bên trong của đối tượng.

=> Vd: viên thuốc.

=> Triển khai tính bao đóng thì dùng **getter/setter.**

=> Linh hoạt, dễ sữa đổi hơn những mã độc lập.

- **Abstraction** **(tính trừu tượng)** loại bỏ các thuộc tính và hành vi k quan trọng của đối tượng, chỉ giữ lại những thuộc tính và hành vi có liên quan đến đối tượng đang giải quyết.

=> Vd: Xe máy (ta k cần quan tâm đến cấu tạo của nó, chỉ quan tâm đến những bộ phận để sử dụng).

=> Giao diện đơn giản, loại bỏ sự phức tạp của đối tượng, ẩn các chi tiết triển khai, chỉ hiển thị các tính năng/kết quả liên quan đến người dùng.

- **Inheritance** **(tính kế thừa)** cho phép các đối tượng có thể chia sẻ, mở rộng các thuộc tính hoặc phương thức mà k cần tiến hành định nghĩa lại.

=> Vd: các lớp bò sát, chim, thú… kế thừa các đặc điểm của ngành động vật có xương sống.

=> Lớp (child class, sub class, derived class) sd lại các đặc điểm/hành vi đã đc định nghĩa trong lớp cha (supper class, parent class, base class).

=> Dùng từ khóa “**extends**” để triển khai tính kế thừa.

=> Tái sd, hạn chế dư thừa mã, dễ kiểm tra và gỡ lỗi.

- **Polymorphism** **(tính đa hình)** là khả năng 1 đối tượng có thể hiện/hành vi theo nhiều cách khác nhau tùy thuộc vào ngữ cảnh. Cùng 1 phương phức/hành vi nhưng có thể có các ứng xử khác nhau.

=> Vd: hành vi nói (mỗi loài động vật thể hiện 1 cách khác nhau).

=> Triển khai bằng: **Overloading, Oversding.**