**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 thực hiện một công việc nào đó (Javascript, Java,...).

- **Thuật toán** là tập hợp hữu hạn các bước để giải quyết 1 vấn đề (bài toán).

+ Mã giả(pseudo code)

+ Lưu đồ(flowchart)

+ Ngôn ngữ lập trình.

|  |  |
| --- | --- |
| **Pseudo code** | **Flowchart** |
| - Sd ngôn ngữ tự nhiên (thông thường là tiếng anh) để mô tả thuật toán..  - **Ưu điểm:** đơn giản, dễ sử dụng, dễ bắt đầu thuật toán, 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, khó kiểm tra và dễ gây hiểu nhầm giữa các bên liên quan | - Sd sơ đồ khối (mỗi khối là 1 hành động) để mô tả thuật toán từ đó giải quyết vấn đề.  - **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. |

-Ý nghĩa các hình khối trong Flowchart

+Elip: Begin/End

+Hình bình hành: Input/Output/Display

+Hình chữ nhật: tính toán

+Hình thoi: Điều kiện, rẽ nhánh

+Dấu mũi tên: Thể hiện luồng thực thi

**2. GIT & HTML** ☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺

- **Quản lí mã nguồn**: là công việc liên quan đến lưu trữ, chia sẻ, cộng tác nhóm với mã nguồn. Quản lí các phiên bản của mã nguồn, tránh xung đột khi làm việc nhóm.

- **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.

- **SCMs** **(Source code management system – Hệ thống quản lí mã nguồn)**: hỗ trợ các công việc liên quan đến lưu trữ, chia sẻ, cộng tác nhóm,…

- **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.

- **Repository:** là kho lưu trữ mã nguồn. Có hai loại là local repo và remote repo

+ **Local repo**: là kho lưu trữ mã nguồn trên máy lập trình viên

+ **Remote repo**: là kho lưu trữ mã nguồn ở server.

|  |  |
| --- | --- |
| Hệ thống quản lí tập trung | Hệ thống quản lí phân tán |
| Lấy file -> sữa chữa-> đẩy lên -> có lỗi k thể lấy lại dc | Lấy nguyên một source lớn khi có lỗi thì sẽ có thể sữa chữa và tránh xung đột khi có lỗi. hệ thống quản lí phân tán sẽ chia ra nhiều nhánh riêng, mỗi ng lm việc trên một nhánh, khi có lỗi thì dễ phát hiện và fix |

- **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 –** đẩy files từ local repository lên remote repository.

+ **git init –** tạo mới repository.

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

- **HTML** **– HyperText 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.

**<dl>,<dt>,<dd>** tạo danh sách từ điển.

**☺**

**3. FORM & TABLE**

- **Form** cho phép người dùng 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.

+ **Name** – tên của form có thể sử dụng để **truy xuất** đến thuộc tính của form

+ **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 hiển 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).

**<select>, <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>:** tạo hàng trong bảng

**<th></th>:** tạo ô tiêu đề

**<td></td>:** tạo ô dữ liệu

Rowspan= “...” gộp theo hàng <tr>

colspan= “..” gộp theo cột <td>

align=”…”, style=”…”, …

**4. JAVASCRIPT**

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

+ **Chức năng:**

- Thay đổi giao diện của web.

- Thay đổi nội dung 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**).

**\*Note:** + Nên nhúng vào cuối phần body. Mục đích để trang web sẽ được load nhanh hơn và tránh một số lỗi k đáng có.

+ Nên sử dụng external để dễ quản lí và có thể tái sử dụng code.

- **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 = …;

+ console.log()-> thường dùng để debug

- Chú thích: sử dụng chú thích cho các LTV khác biết được công dụng, tác dụng của đoạn code trong JS

+ // -> shift + / -> Tạo một dòng chú thích

+ /\* \*/ -> Ctrl + shift + / -> tạo một đoạn chú thích

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

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

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

+ Giá trị của biến có thể ghi đè.

+ Phải khai báo biến trc khi sd.

|  |  |
| --- | --- |
| **var** | **let** |
| - Có thể khai báo cùng tên (lấy giá trị sau)  - 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** nên phải khai báo trước khi sử dụng  - Phạm vi truy cập của biến nằm trong **block scope.** |

**- Const:** **const** NAME = value;

+ Phải gán giá trị khi khởi tạo ( giá trị này không thể thay đổi)

+ K thể ghi đè lại giá trị mới

+ Phạm vi truy cập trong cặp Block scope

+ Dùng để khai báo hằng số

+ Tên biến viết in hoa

- **Quy tắc đặt tên biến:**

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

+ Quy tắc **camel Case** (con lạc đà).(numberInArray, soHang,…)

+ Đặt tên biến phải có ý nghĩa.

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

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

String: kiểu chuỗi

Number: kiểu số (infinity: k xác định vô cực (5/0); NaN(not a number): k phải là số)

Boolean: true/false

Null: rỗng, giá trị k rõ

Underfined: biến chưa dc có giá trị

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

- **Toán tử** là các kí hiệu được sử dụng trong các biểu thức để sinh ra kết quả cuối

+ **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.

**- Để ép kiểu string sang number:**

+ **ParseInt():** ép sang kiểu số nguyên

+ **ParseFloat():** ép sang kiểu số thực

+ **Dùng dấu + hoặc (\*1)** Ex: let num=prompt(‘’); num=num\*1;

\*Note: **NaN**(not a number): k phải là số, **giá trị không xác định**, kiểu dữ liệu của NaN là **number**

**6. CONDITIONAL STATMENTS**

- **Câu lệnh điều khiển**: là câu lệnh sinh ra nhằm thay đổi luồng thực thi của chương trình

+ Câu lệnh điều kiện (conditional statement)

+ Câu lệnh lặp (loop statement)

+ Câu lệnh nhảy (jump statement)

- **Câu lệnh điều kiện:** là câu lệnh đưa ra các hành động thực thi dựa trên điều kiện cho trước

- **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. | Chỉ dùng trong **so sánh** bằng/ k bằng. |
| Biểu thức đk có giá trị trả về là kiểu **boolean.** | Biểu thức có thể là kiểu **string, number, boolean,** … |
| Chỉ có 1 khối lệnh đc thực thi khi mà đk đúng/ sai.  **(sử dụng tối đa 3 if)** | Nếu đk đúng mà k có break thì all các khối lệnh đằng sau đc thực thi. |
| Mỗi khối if chỉ có 1 biểu thức điều kiện trả về giá trị true/fasle | Một biểu thức điều kiện sẽ đc so sánh với tất cả các case |

**- Toán tử ba ngôi: Conditional statement ? Expression true case : Expression false case**

**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:** khi gặp từ khóa break thì chương trình thoát vòng lặp hiện tại

- **continue:** khi gặp từ khóa continue thì chương trình sẽ bỏ qua tất cả câu lệnh bên dưới và 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**.

+ Độ dài bằng tổng số phần tử có trong mảng

+ Mảng có thể lưu được nhiều kiểu dữ liệu khác nhau

- **Khai báo mảng:**

+ **Dùng [ ]:** let … = […];

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

+ **Dùng Array:** let … = Array(…);

+ Note: let … = new Array(13); //tạo mảng có 13 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).  - Giá trị trả về: index or value  - 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 duyệt mảng theo 1 chiều (từ đầu đến cuối, và duyệt hết mảng).  - Giá trị trả về: for-in trả về index(có thể lấy được value), for-of trả về value  - 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** là mảng chứa các phần tử là 1 mảng khác.

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

- **1 số hàm thông dụng:**

+ **array.length** độ dài mảng. (trả về giá trị)

+ **array.push(…)** thêm 1 hoặc nhiều phần tử vào cuối mảng (cách khác: **array[array.length] = “…”**).(trả về độ dài mới của mảng)

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

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

+ **array.unshift(…)** thêm phần tử vào đầu mảng.(trả về độ dài mới của 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.splice(**i, n**)** xóa n phần tử từ vị trí i. (trả về phần tử bị xóa)

**array.splice(**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 sử dụng mã nguồn và “chia để trị” và lm cho file code ngắn gọn dễ sử dụng).

- **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. |

- Sử dụng hàm có return khi cần sử dụng kết quả trả về của hàm để thực hiện các bước tiếp theo của bài toán. Khi gặp từ khóa return thì sẽ ngay lập tức thoát khỏi. Có thể trả về mọi kiểu dữ liệu trong JS. Trong hàm thì k có hoặc có vô số từ khóa return (nằm trong câu điều kiện) tuy nhiên tại mỗi thời điểm thì chỉ có 1 từ khóa return duy nhất được thực hiện.

- **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).

- Javascript chỉ truyền tham trị

|  |  |
| --- | --- |
| **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.  - Tên và giá trị của đối số(biến) đều được lưu ở vùng nhớ stack. | - 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.  - Tên được lưu ở vùng nhớ stack, giá trị được lưu ở vùng nhớ heap. |

**10. OOP** 🥹🥹🥹🥹🥹🥹🥹🥹

- **OOP** (**Object oriented programming**)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: (object)**

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

+ **Hành vi (behavior)**(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 **sản phẩm** của class (thể hiện cụ thể). Mỗi object có giá trị và hành vi khác nhau để phân biệt với các object khác.

- **Construction** là phương thức đặc biệt được sử dụng để khởi tạo đối tượng, còn được gọi là hàm khởi tạo, được gọi thông qua từ khóa “new”. Trong js chỉ có tối đa 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**. (tạo đối tượng trực tiếp)

- **Encapsulation** **(tính bao đóng)** bảo đảm toàn vẹn bảo mật thông tin của đối tượng. Không cho phép truy cập trực tiếp vào thông tin của đối tượng mà phải thông qua các phương thức trung gian.

=> Vd: viên thuốc.

=> Triển khai tính bao đóng thì dùng **getter** (lấy ra giá trị thuộc tính hiện tại của đối tượng )**/setter**(thiết lập lại giá trị cho thuộc tính của đối tượng)**.**

=> 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 => đơn giản hóa được bài toán, code ngắn gọn và logic hơn, tránh dư thừa k cần thiết. Chỉ quan tâm đầu vào và đầu ra chứ k quan tâm đến những xử lý bên trong chương trình

=> 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 kế thừa lại những thuộc tính và hành vi đã được khai báo mà k cần khai báo lại. Tính kế thừa cho phép lớp con có thể kế thừa và mở rộng thuộc tính đã có ở lớp cha

Super(): gọi lại constructor của lớp cha

=> Vd: các lớp bò sát, chim, thú… kế thừa các đặc điểm của ngành động vật có xương sống. win 10 kế thừa những cái tốt ở win 7 và phát triển hơn

=> Lớp con (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)** cùng một đối tượng nhưng thể hiện vai trò khác nhau ở mỗi thời điểm khác nhau, cùng một phương thức nhưng trả về các kết quả khác nhau tùy vào từng thời điểm.

=> Vd: A ở nhà là con ngoan, ở trường là học sinh. Đi học đi bằng xe máy, đi biển thì bơi.

=> Triển khai bằng: **Overloading** (chồng)**, Oversding**(ghi đè)**.** (phương thức).

SOME QUESTION 🤪🤪🤪🤪🤪

**1. What is the difference between Attributes and Property?**(Sự khác nhau giữa Attributes (Thuộc tính) và Property (Đặc tính) là gì?)

Attributes (Thuộc tính) – cung cấp thêm thông tin chi tiết về một phần tử như id, type, value, v.v.

Property (Đặc tính) – là giá trị được gán cho thuộc tính như type=”text”, value=’Name’, v.v.

**2. List out the different ways an HTML element can be accessed in a JavaScript code** (Liệt kê các cách khác nhau mà một phần tử HTML có thể được truy cập trong code JavaScript)

Dưới đây là các cách một phần tử HTML có thể được truy cập trong code Javascript:

-getElementById(‘idname’): Lấy một phần tử theo tên ID của nó

-getElementsByClass(‘classname’): Lấy tất cả các phần tử có tên lớp cho sẵn

-getElementsByTagName(‘tagname’): Lấy tất cả các phần tử có tên thẻ cho sẵn

-querySelector(): Hàm này nhận bộ chọn kiểu CSS (CSS style selector) và trả về phần tử được chọn đầu tiên.

**3. In how many ways a JavaScript code can be involved in an HTML file?**(Có bao nhiêu cách để thêm code JavaScript vào một tệp HTML?)

- Có 3 cách để thêm code JavaScript vào tệp HTML:

Inline

Internal

External

Hàm inline (hàm nội tuyến) là một hàm JavaScript được gán cho một biến được tạo trong runtime (môi trường thực thi). Bạn có thể phân biệt giữa hàm inline và hàm ẩn danh (anonymous function) vì một hàm inline được gán cho một biến và có thể dễ dàng sử dụng lại. Khi bạn cần JavaScript cho một tính năng nào đó, bạn có thể chọn cách tích hợp script trong trang bạn đang làm việc, hoặc có thể đặt nó trong một tệp riêng để gọi khi cần. Đây là sự khác biệt giữa internal script và external script.

**4. What are the ways to define a variable in JavaScript?**(Có những cách nào để xác định một biến trong JavaScript?)

**5. What is a Typed language?(**Typed language là gì?)

Typed language là ngôn ngữ lập trình mà trong đó các giá trị được liên kết với các giá trị chứ không phải với các biến. Nó có hai loại:

Dynamically Typed Language: biến có thể chứa nhiều kiểu; giống như trong JavaScript, một biến có thể nhận số, ký tự, etc.

Statically Typed Language: biến chỉ có thể chứa một kiểu; giống như trong Java, một biến được khai báo là chuỗi chỉ có thể nhận một tập ký tự và không có gì khác.

**6. What is the difference between Local storage & Session storage?**(Sự khác nhau giữa Local storage & Session storage là gì?)

Local storage – Dữ liệu không được gửi lại máy chủ cho mọi HTTP requets (HTML, hình ảnh, JavaScript, CSS, etc.) – giảm lưu lượng giữa máy khách và máy chủ. Nó sẽ lưu lại cho đến khi được xóa thủ công thông qua cài đặt hoặc chương trình.

Session storage – Tương tự như local storage; sự khác biệt duy nhất là dữ liệu được lưu trữ trong local storage không có thời gian hết hạn, còn dữ liệu được lưu trữ trong session storage sẽ bị xóa khi phiên của trang kết thúc. Session storage sẽ rời khỏi khi trình duyệt đóng.

**7. What is the difference between the operators ‘==‘ & ‘===‘?**(Sự khác biệt giữa 2 toán tử ‘==‘ và ‘===‘ là gì?)

Toán tử “==” chỉ kiểm tra giá trị và tự chuyển đổi nếu các giá trị có kiểu dữ liệu khác nhau trước khi so sánh.

Mặt khác, toán tử “===” kiểm tra cả giá trị và kiểu dữ liệu của giá trị. Nó không chuyển đổi các giá trị khi so sánh, nên nếu hai giá trị không cùng kiểu dữ liệu thì kết quả sẽ trả về là false.

Ví dụ: nếu bạn so sánh một số với một chuỗi có ký tự số, “==” cho phép điều đó, nhưng “===” không cho phép, bởi nó không chỉ kiểm tra giá trị mà còn kiểm tra kiểu của hai biến, nếu hai biến không cùng loại, “===” trả về false, trong khi “==” trả về true.

**8. What is the difference between null & undefined?**(Sự khác nhau giữa null và undefined là gì?)

Undefined là một biến đã được khai báo nhưng chưa được gán giá trị. Null là một giá trị gán. Nó có thể được gán cho một biến dưới dạng đại diện không có giá trị. Ngoài ra, undefined và null là hai kiểu riêng biệt: undefined là một kiểu riêng biệt (kiểu dữ liệu không xác định) trong khi null là một đối tượng (object).

**9. What is the difference between undeclared & undefined?**(Sự khác nhau giữa undeclared và undefined là gì?)

Biến undeclared là những biến không tồn tại trong chương trình và không được khai báo. Nếu chương trình cố gắng đọc giá trị của một biến chưa được khai báo, thì sẽ xảy ra runtime error.

Biến undefined là những biến được khai báo trong chương trình nhưng chưa được cung cấp bất kỳ giá trị nào. Nếu chương trình cố gắng đọc giá trị của một biến không xác định, một giá trị không xác định sẽ được trả về.

**10. Name some of the JavaScript Frameworks?(**Kể tên một số JavaScript Framework)

JavaScript Framework là những application framework được viết bằng JavaScript. Chúng khác với thư viện JavaScript ở luồng điều khiển (control flow). Có rất nhiều JavaScript Framework có sẵn nhưng một số framework được sử dụng phổ biến nhất là:

Angular

React

Vue

**11. What is the difference between window & document in JavaScript?** (Điểm khác nhau giữa window và document trong JavaScript là gì?)

| Window | Document |
| --- | --- |
| Là một đối tượng toàn cục (global object) chứa các biến, hàm, lịch sử, vị trí. | Nằm dưới window và có thể được coi là thuộc tính của window. |

**12. What is the difference between innerHTML & innerText?** (Sự khác nhau giữa innerHTML và innerText là gì?)

innerHTML – Sẽ xử lý một thẻ HTML nếu được tìm thấy trong một chuỗi

innerText – Không xử lý thẻ HTML nếu được tìm thấy trong một chuỗi

**13. What is an event bubbling in JavaScript?** (Event bubbling trong JavaScript là gì?)

 Event bubbling là một cách lan truyền event trong HTML DOM API, khi một event xảy ra trong một phần tử bên trong một phần tử khác và cả hai phần tử đã đăng ký một xử lý cho event đó. Với bubbling, event đầu tiên được phần tử trong cùng nắm bắt và xử lý, sau đó được truyền đến các phần tử bên ngoài. Việc thực thi bắt đầu từ event đó và chuyển đến phần tử mẹ của nó. Sau đó, việc thực thi chuyển đến phần tử cha của nó và cứ tiếp tục như vậy cho đến phần tử body.

**14 What is NaN in JavaScript?** (NaN trong JavaScript là gì?)

NaN là một dạng viết tắt của “Not a Number”. Vì NaN luôn so sánh không bằng với bất kỳ số nào, kể cả NaN, nó thường được sử dụng để chỉ ra điều kiện lỗi cho một hàm mà đáng ra phải trả về một số hợp lệ. Khi một chuỗi hoặc một cái gì đó khác đang được chuyển đổi thành một số và điều đó không thể được thực hiện, thì ta sẽ thấy NaN.

**15. How do JavaScript primitive/object types passed in functions?** (Các kiểu dữ liệu primitive/object trong JavaScript được truyền vào các hàm như thế nào?)

Một trong những điểm khác biệt giữa hai kiểu này là các kiểu dữ liệu nguyên thủy (Primitive Data Types) được chuyển theo giá trị (By Value) và các đối tượng (Objects) được chuyển theo tham chiếu (By Reference).

By Value nghĩa là tạo BẢN SAO của bản gốc. Hãy hình dung nó giống như những cặp song sinh: họ được sinh ra giống hệt nhau, nhưng người anh/chị không bị mất chân, trong khi người em bị mất một chân trong chiến tranh.

By Reference nghĩa là tạo ALIAS cho bản gốc. Nó giống như việc mẹ gọi bạn là “Pumpkin Pie” mặc dù tên của bạn là Margaret, điều này không đột nhiên sinh ra một bản sao của bạn. Bạn vẫn là một, nhưng bạn có thể được gọi bằng hai cái tên rất khác nhau này.

**16. How can you convert the string of any base to integer in JavaScript?** (Làm thế nào để chuyển đổi một chuỗi bất kỳ thành số nguyên trong JavaScript?)

Hàm parseInt() được sử dụng để chuyển đổi số giữa các base khác nhau. Nó nhận chuỗi được chuyển đổi làm tham số đầu tiên và tham số thứ hai là base của chuỗi đã cho.

**17. What would be the result of 2+5+”3″?** (Kết quả của 2+5+”3″ là gì?)

Vì 2 và 5 là số nguyên nên chúng sẽ được cộng theo dạng số. Vì “3” là một chuỗi nên việc nối bằng hàm concat (concatenation). Vì vậy, kết quả sẽ là 73. Dấu “” tạo ra tất cả sự khác biệt ở đây và khiến số 3 được coi là một chuỗi chứ không phải một số.

**18. Các cách tạo chuỗi chứa dấu nháy?**

- Các chuỗi sử dụng dấu nháy kép và dấu nháy đơn có hiệu quả giống nhau.

- Vì không có quy ước hoặc ưu tiên chính thức cho các chuỗi được trích dẫn đơn hoặc kép, tất cả những gì quan trọng là sự nhất quán trong các file code

**19. NaN là gì? NaN === NaN có đúng k? Vì sao?**

NaN là một dạng viết tắt của “Not a Number”. Vì NaN luôn so sánh không bằng với bất kỳ số nào, kể cả NaN, nó thường được sử dụng để chỉ ra điều kiện lỗi cho một hàm mà đáng ra phải trả về một số hợp lệ. Khi một chuỗi hoặc một cái gì đó khác đang được chuyển đổi thành một số và điều đó không thể được thực hiện, thì ta sẽ thấy NaN.

**20. So sánh null và rỗng?**

- null được sử dụng để chỉ không có gì, đối tượng chưa tồn tại  
- rỗng được sử dụng để chỉ một chuỗi duy nhất có độ dài bằng không

**21. Những phương thức nào cho phép tương tác với chuỗi?**

-Đếm số lượng ký tự của chuỗi

-Chuyển chuỗi về dạng chữ in hoa

-Chuyển chuỗi về dạng chữ thường

-Nối các chuỗi lại với nhau

-Xác định chỉ số của một chuỗi con

-Trích xuất một ký tự trong chuỗi

-Phương thức substring()

-Phương thức substr()

-Thay thế tất cả nội dung trùng khớp

-Thay thế nội dung không phân biệt trường hợp chữ hoa hay thường