|  |  |
| --- | --- |
|  | **1.PSEUDO CODE & FLOWCHART** |
| **Lập trình** | **Lập trình** là **quá trình** tạo ra các chỉ dẫn (instruction) để ra lệnh cho máy tính hoàn thành 1 công việc (task). |
| **Ngôn ngữ lập trình** | **Ngôn ngữ lập trình** là **phương tiện** lập trình viên tạo ra các chỉ dẫn để máy tính hiểu (javascript, java, …). |
| **Thuật toán** | * **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** | Sử dụng ngôn ngữ tự nhiên để biểu diễn các bước (sử dụng ngôn ngữ english)  **Ưu điểm:** Đơn giản, không cần nhiều kiến thức về cách biểu diễn như lưu đồ hay ngôn ngữ IT  **Nhược điểm:** Dài dòng, không cấu trúc, khó hiểu, khó diễn đạt được thuật toán. |
| **Flowchart** | Sử dụng sơ đồ khối (mỗi khối 1 hành động) để mô tả thuật toán  **Ưu điểm:** Quy định chặt chẽ về ký hiệu, thống nhất, dễ hiểu, trực quan  **Nhược điểm:** Khó hiểu cho những ai không nắm được các ký hiệu (bắt buộc phải học thuộc), khó gỡ lỗi |

|  |  |
| --- | --- |
|  | **2. GIT & HTML** |
| **VCS** | **VCS (Version control systems –** Hệ thống quản lý phiên bản mã nguồn) là 1 **hệ thống** lưu giữ các phiên bản của sp phần mềm, giúp các lập trình viên có thể dễ dàng lấy lại phiên bản mong muốn   * **Tập trung:** Bao gồm 1 máy chủ chứa các tệp tin được phiên bản hoá, các máy khác có quyền thay đổi các tệp tin này trên 1 máy chủ trung tâm đó * **Phân tán:** Các máy khách ko chỉ sao chép về máy tính của mình phiên bản mới nhất của các tệp tin mới nhất, mà chúng còn sao chép kho chứa mã nguồn |
| **GIT** | 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) |
| **Các lệnh cơ bản của Git** | **Git-clone:** Sao chép remote repository (folder trên Github) về local repository (folder trong máy tính các nhân)  **Git add:** Thêm file hoặc ghi nhận file có thay đổi để đưa lên rômte repository  **Git commit -m “…”:** Gửi 1 thông điệp đi kèm và sao lưu tất cả sự thay đổi tại thời điểm hiện tại  **Git push:** Đẩy mã nguồn từ local repository lên remote repository  **Git pull:** Cập nhập từ remote repository về local repository |
| **HTML** | **HTML – HyperText Markup Language** (Ngôn ngữ đánh dấu siêu văn bản), có chức năng giúp người dùng xây dựng và cấu trúc các thành phần trong trang web (HTML không phải là ngôn ngữ lập trình).  **2 nhóm thẻ chính:**   * **Inline –** Khoảng ko gian của thẻ vừa khớp với nội dung * **Block –** Khỏng ko gian của thẻ chứa toàn bộ chiều rộng của màn hình   **1 số thẻ cơ bản:**   * **<head></head>** Khai báo tên (title) của trang web, nhúng css,js,… * **<body></body>** Nội dung được hiển thị ở web * **<h></h>** Thẻ tiêu đề * **<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 ko có thứ tự * **<ol></ol>,<li></li>** Danh sách có thứ tự * **<dl>,<dt>,<dd>** Tạo danh sách từ điển |

|  |  |
| --- | --- |
|  | **3. FORM & TABLE** |
| **Form** | **Form** cho phép người dùng nhập dữ liệu và gửi lên sever (máy chủ web) |
| **Thuộc tính** | **Action –** mô tả URL của sever, nơi mà dữ liệu được gửi đến  **Method –** Xác định kiểu phương thức sử dụng để gửi dữ liệu (POST & GET) |
| **GET** | Dữ liệu hiển thị trên URL  Ko bảo mật, ko nên sử dụng để lưu các dữ liệu nhạy cảm (bảo mật)  Có giới hạn dung lượng dữ liệu gửi |
| **POST** | Dữ liệu ko hiển thị trên URL  Bảo mật, đc sử dụng để lưu các dữ liệu nhạy cảm  Ko giới hạn data khi gửi |
| **1 số thẻ cơ bản** | **<input type = “…”>** Tạo ô nhập liệu  **<select> <option>** Chọn 1 trong 1 danh sách  **<input type = “submit”>** Gửi thông tin đến liên kết  **<input type = “reset”>** Xoá các thiết lập vừa tạo  **name** liên kết (link)  **<textarea>** Tạo ô nhập văn bản  **Table** là nơi lưu trữ thông tin dữ liệu các đối tượng theo dạng hàng và cột |

|  |  |
| --- | --- |
|  | **4. JAVASCRIPT** |
| **Javascript** | **Javascript** là 1 ngôn ngữ lập trình sử dụng nhiều trên các website (có thể sử dụng lập trình web-service, game, xử lí sự kiện người dùng)   * **Chức năng:**   **+** Thay đổi giao diện  **+** Thay đổi nội dung hiển thị  **+** Xử lí các sự kiện người dùng, tăng tính tương tác giữa người dùng và trang web   * **Các cách nhúng:**   + Nhúng trực tiếp **(inline)**  + Code js trong thẻ <script> **(internal)**  + Sử dung file \*.js (dùng thuộc tính “src” của thẻ <script> để khái 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) và sai (false) |
| **Biểu diễn đầu ra** | document.write(“…”);  document.getElementById(“myId”).innerText = …;  document.getElementById(“myId”).innerHTML = …; |

|  |  |
| --- | --- |
|  | **5. VARIABLE, DATA TYPE & OPERATOR** |
| **Biến** | **Biến** là tên gọi đặt cho 1 vùng nhớ dữ liệu   * Dữ liệu được lưu trữ trong biến được gọi là giá trị (value) * Giá trị của biến có thể ghì đè (gán) * Phải khai báo biến trước khi sử dụng |
| **Var** | Có thể khai báo đè  Có cơ chế **hoisting** cho phép sử dụng trước khi khởi tạo  Phạm vi truy cập của biến nằm trong **Function scope** |
| **Let** | Ko thể khai báo lại  Ko có cơ chế **hoisting**  Phạm vi truy cập của biến nằm trong **Block scope** |
| **Const** | Khai báo giá trị hằng  Cơ chế giống với **let** |
| **Các quy tắc đặt tên, kiểu dữ liệu, toán tử** | **Quy tắc đặt tên biến:**   * Bắt đầu là \_ hoặc $ hoặc a->z * Quy tắc **camelCase** * 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 sử dụng dữ liệu phân loại nào   * **Kiểu nguyên thuỷ:** string, number, boolean, undefined, null, symbol * **Kiểu đối tượng:** object, array,…   **Toán tử** là ký 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 của 1 biến: **typeof** * **Độ ưu tiên:** số học > so sánh > logic > gán >. Dấu () để thay đổi độ ưu tiên |

|  |  |
| --- | --- |
|  | **6. CONDITIONAL STATMENTS** |
| **If** | **If khuyết**  **If đầy đủ**  **If lồng nhau (nhiều biến)**  **If bậc thang (1 biến)** |
| **Switch case** | **Switch** (condition){  **case** x: …  **break;**  **default**  } |
| **So sánh** | **If**   * So sánh hơn, bằng / ko bằng * Biểu thức điều kiện có giá trị trả về là kiểu boolean * Chỉ có 1 khối lệnh được thực thi khi mà điều kiện đúng / sai * Nên sử dụng tối đa 3 điều kiện if |
| **Switch case**   * Chỉ dùng trong so sánh bằng / ko bằng * Biểu thức có thể là kiểu **string, number, boolean,…** * Nếu điều kiện dùng mà ko có break thì tất cả khối lệnh đằng sau được thực thi liên tục * Hỗ trợ kiểm tra nhiều điều kiện |

|  |  |
| --- | --- |
|  | **7. LOOPS STATMENTS** |
| **Vòng lặp** | Vòng lặp cho phép tự động thực hiện 1 khối lệnh lặp đi lặp lại nhiều lần dựa vào 1 điều kiện cho trước |
| **for** | **for** (khối lệnh khởi tạo biến lặp; điều kiện lặp; khối lệnh tăng giảm){  Nội dung lặp;  }**(Sử dụng khi biết trước số lần lặp)** |
| **While** | **While** (Điều kiện lặp){  Nội dung lặp;  }  **Dùng khi ko biết trước số lần lặp, ko đc thực thi khi ko thoã mãn điều kiện đầu vào** |
| **Do While** | Do{  Nội dung lặp  }While (Điều kiện lặp)  **Dùng khi ko biết trước số lần lặp, thực hiện 1 lần trước khi đến điều kiện lặp** |
| **break và continue** | **break:** Thoát khỏi vòng lặp hiện tại  **continue:** Nhảy qua vòng lặp tiếp theo, đến điểm ++ |

|  |  |
| --- | --- |
|  | **8. ARRAY** |
| **Mảng** | **Mảng** là 1 biến đặc biệt (biến mảng), có thể lưu trữ được nhiều giá trị   * Mỗi giá trị trong mảng là 1 **phần tử** * Các phần tử của mảng được lưu trữ ở các vị trị kế tiếp nhau trong ô nhớ * Quy tắc đặt tên của mảng là camelCase * 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 … = newArray(…); * **Dùng array:** let … = Array(…); * **Note:** let … = newArray(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(“elenment = ” + array[elenment])  }   * **for-off** Hỗ trợ duyệt **value** của mảng |
| **For** | * Duyệt mảng theo nhiều chiều (duyệt từ đầu đến cuối, cuối đến đầu, từ giữa ra 2 bên, có thể duyệt ko hết mảng). * Có khai báo và sử dụng biến đếm. * Cấu trúc gồm 4 phần (khối lệnh khởi tạo biến lặp, điều kiện lặp, khối lệnh tăng giảm và nội dung) |
| **For-in**  **For-off** | * Chỉ duyệt mảng theo 1 chiều (từ đầu đến cuối và duyệt hết mảng) * Ko cần khai báo và sử dụng biến đếm * Cấu trúc chỉ gồm 2 phần (phần khai báo và phần thực thi khối lệnh) |
| **Mảng 2 chiều** | Mảng 2 chiều thực chất là mảng 1 chiều, mà mỗi phần tử là 1 mảng 1 chiều   * 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 mảng (các khác: arr[arr.length] = “…”  array.pop() : xoá phần tử cuối của mảng  array.shift() : xoá phần tử đầu tiên của mảng  array.unshift(): thêm phần tử vào đầu của mảng  array.join() : chuyển 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 mảng  array.concat() : nối 2 mảng lại với nhau  array.splice() : xoá hoặc thêm phần tử vào vị trị nhất định  array.includes(i) : kiểm tra giá trị có tồn tại hay ko (trả về true / false)  indexOf() : kiểm tra giá trị có tồn tại hay ko (có thì xuất ra index, ko thì -1) |

|  |  |  |
| --- | --- | --- |
|  | **9. FUNCTION** | |
| **Hàm** | **Hàm** là 1 nhóm các câu lệnh thực thi 1 nhiệm vụ nhất định. (Mục đích: tái sử dụng mã nguồn và “chia sẻ để trị”) | |
| **Tham số** | **Tham số (parameter)** là biến được khai báo ở phần header of function (tham số hình thức) | |
| **Đối số** | **Đối số (argument)** là giá trị được truyền vào khi gọi hàm (tham số thực) | |
|  | **Hàm có trả về (return)** | **Hàm ko có trả về** |
| **So sánh hàm có return và ko có return** | - Thân hàm ngắn, dễ hiểu, bên ngoài code nhiều hơn.  - Muốn sử dụng ở 1 bài toán khác thì ko 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, dữ liêu, mảng,…)  - Dùng toán tử gán để nhận giá trị trả về. | - Thân hàm code nhiều hơn, bên ngoài code ít hơn  - Ko thể thay đổi nội dung thông báo khi sử dụng, nếu muốn sử dụng 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** | Phạm vi của biến là vị trí trong chương trình mà 1 biến có thể sử dụng được | |
| **Biến cục bộ và biến toàn cục** | **Biến cục bộ (local variable)** là biến đc khai báo trong hàm, tham số của hàm (sử dụng trong function) | **Biến toàn cục (global variable)** là biến đc khai báo ngoài hàm (sử dụng trong hoặc ngoài function) |
|  | **Truyền tham trị** | **Truyền tham chiếu** |
| **So sánh tham trị và tham chiếu** | - Lấy giá trị truyền vào cho hàm (gán cho tham số hàm)  - Áp dụng cho kiểu nguyên thuỷ (number, string, …)  - Giá trị của biến trước và sau gọi hàm ko 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 => ko lưu lại giá trị, mà chỉ lưu lại địa chỉ của ô nhớ lưu giá trị này |

|  |  |
| --- | --- |
|  | **10. OOP** |
| **OOP** | **OOP** là kỹ thuật 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 hoá 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** | **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 cho tất cả các đối tượng cùng loại |
| **Object** | **Object** là **thể hiện** của class (sp đc tạo từ class). Mỗi object có giá trị và hành vi khác nhau để phân biệt với các object khác |
| **Construction** | **Construction** là hàm dùng để khởi tạo đối tượng, được gọi thông qua từ khoá “**new**”. Trong js chỉ có tối đa 1 construction trong 1 class |
| **Các cách khởi tạo** | * Sử dụng **function** để tạo **class** (dùng từ khoá “**new**” để khởi tạo đối tượng) * Sử dụng **class** để tạo **class** (dùng từ khoá “**new**” để khởi tạo đối tượng) * Sử dụng **object literals** để tạo **object** |
| **Encapsulation** | **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 tư bên trong của đối tượng |
| **Abstraction** | **Abstraction (tính trừu tượng)** loại bỏ các thuộc tính và hành vi ko quan trọng của đối tượng, chỉ giữ lại những thuộc tính và hành vi có liên quan của đối tượng đang giải quyết |
| **Inheritance** | **Inheritance (tính kế thừa)** cho phép các đối tượng có thể chia sẻ, mở rộng các thuộc tính hoặc phương thức mà ko cần tiến hành định nghĩa lại |
| **Polymorphism** | **Polymorphism (tính đa hình)** là khả năng 1 đối tượng có thể thể hiện / hành vi theo nhiều cách khác nhau tuỳ thuộc vào ngữ cảnh. Cùng 1 phương thức / hành vi nhưng có thể có các ứng xử khác nhau. |