# HTML

## Các loại links - liên kết

### Dùng ảnh làm link: nhét <img> làm nội dung của </a>

|  |
| --- |
| <a href=”đường-link” target=”\_blank”>  <img src=”link-ảnh” alt=”mô tả” width=”…” height=”…”>  </a> |

## Hình ảnh

### Tạo ảnh click được nhiều chỗ </map> <area> shape coords

a. Tạo ảnh và đăng ký map trong <img>

|  |
| --- |
| <img src… usemap=”#Name-map”> |

b.Tạo map

|  |
| --- |
| <map name=”Name-map”>  <area shape=”rect” coords=”x1, y1, x2, y2” alt=”Mô tả” href=”Link-vùng”>  (vùng hình chữ nhật, x là khoảng từ cạnh trái, y là từ top)  <area shape=”circle” coords=”x, y, radius” alt=”Mô tả” href=”link-của-vùng”>  (vùng hình tròn, x-y là tọa độ trái-top, radius là bán kính)  </map> |

### Tạo ảnh lv figure có caption </figure>

|  |
| --- |
| <figure>  <img … >  <figcaption>*Caption cho figure này*</figcaption>  </figure> |

### Tạo ảnh lv picture </picture>

|  |
| --- |
| <picture>  <source media=”min-width:…px” srcset=”link-ảnh-1”>  <source …>  <img …> (ảnh dự phòng)  </picture> |

## Video/audio

### Chèn audio </audio>

|  |
| --- |
| <audio controls>  <source src=”….mp3” type=”audio/mp3”>  <source…>  Trình duyệt của bạn không hỗ trợ audio này  </audio> |

### Chèn video </video>

|  |
| --- |
| <video width=”…” height=”…” controls>  <source src=”….mp4” type=”video/mp4”>  <source webm…>  Trình duyệt của bạn không hỗ trợ video này  </video> |

### Chèn video Youtube

|  |
| --- |
| <iframe width=”…” height=”…”  src=”https://www.youtube.com/embed/idYoutube”>  </iframe> |

## List - Danh sách

### Tạo danh sách chi tiết có mục lớn nhỏ description list <dl>

|  |  |
| --- | --- |
| <dl>  <dt>Đồ uống</dt>  <dd>Cafe</dd>  <dt>Đồ ăn</dt>  <dd>Bánh</dd>  ….  </dl> | CSS (</dl> cho cả list)  line-height:  margin/padding |

## Bảng

### Tạo bảng

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <table> DỌC  !<caption> … </caption>!  <tr>  <th> Header 1 </th>  <th> Header 2 </th>  </tr>  <tr>  <td> #1 Header 1 </td>  <td> #1 Header 2 </td>  </tr>  </table>  <table> NGANG  !<caption> … </caption>!  <tr>  <th> Header 1 </th>  <td> #1 Header 1 </td>  <td> #2 Header 1 </td>  …  </tr>  <tr>  <th> Header 2 </th>  <td> #1 Header 2 </td>  <td> #2 Header 2</td>  …  </tr>  </table> | |  |  | | --- | --- | | Header 1 | Header 2 | | #1  Header 1 | #1  Header 2 | | Header 1 | #1 Header 1 | #2 Header 1 | | Header 2 | #1 Header 2 | #2 Header 2 |   <td / th colspan=”x”> nhập x ô column  <td / th rowspan=”x”> nhập x ô row  tr:hover {background-color: …;)  tr:nth-child(even/odd) ***{background-color***: …;}  table {***float***: right;} (lên cạnh bên phải block trước)  table {***margin***: auto;} (căn giữa)  table {border-spacing:…px;  div {***overflow-x***:auto |

## Form

|  |
| --- |
| HTML  1. Code theo chiều row, lần lượt từ cột nhỏ (cho label) đến cột lớn (cho input)  2. Mỗi row là 1 div chung class=”row”, chứa các ô bên column  3. Mỗi ô bên column là 1 div, class=”col-label” (label) và class=”col-input” (input)  VD:  <div class=”overall”>  <form>  <div class=”row”>  <div class=”col-label”>  <label for=”tên-id”>*First name*</label>  </div>  <div class=”col-input”>  <input type=”text” id=”tên-id” name=”fisrtname” placeholder…>  </div>  </div>  <div class=”row”>  …  </div>  </form>  </div> |

### Tạo danh sách xổ xuống </select> + </option>

|  |
| --- |
| <form>  <label for=”tên-id”>*Chọn sau đây*</label>  <select id=”tên-id” name=”tên-nhóm”>  (<optgroup label=”A-C”>)  <option value=”A”>*Phương án A*</option>  <option value=”B”>*Phương án B*</option>  ….  </select>  </form> |

### Tạo input ô lớn: </textarea>

|  |
| --- |
| <label for=”Tên-id”>*Review của bạn:*</label>  <textarea id=”Tên-id” name=”...” rows=”…” cols=”…”>*Nơi điền data*</textarea> |

### Kết nối các input rời rạc vào 1 form

|  |
| --- |
| - VD: có 1 input Họ, ở dưới là 1 vài text rồi mới đến input Tên  - Ở input rời ra cho thêm form="id-của-form-đích”  VD: <input type="text" id="lname" name="lname" form="form1"> |

### Tạo option action khác cho 1 form formaction=”…php”

|  |
| --- |
| - VD 1 form có action là gửi về server của nhân viên, có thể tạo 1 action khác là gửi cho quản lý (bằng 1 nút “gửi quản lý”, vẫn cùng 1 form)  - Tạo thêm 1 submit với formaction=”/action\_page2.php” value=”Gửi quản lý” |

### Tạo viền + mô tả 1 khu form lớn </fieldset> </legend>

|  |
| --- |
| <form>  <fieldset name=”tên-nhóm”>  <legend>Thông tin đăng nhập</legend>  …input label submit…  </fieldset>  </form> |

# CSS

## CSS Selector

### Chọn nhiều selector

|  |
| --- |
| - Cách nhau bằng dấu phẩy ,  VD: Element, .tên-class, #tên-id { |

### Chọn toàn bộ Element hậu duệ nằm trong 1 Element gốc

|  |
| --- |
| - Cách nhau bằng dấu cách  VD: ElementAncestor ElementDesendant { |

### Chọn các Element con ruột của 1 Element parent

|  |
| --- |
| - Cách nhau bằng Parent > Child  VD: ElementParent > ElementChild { |

### Chọn 1 và chỉ 1 Element B ngang cấp và nằm ngay sau 1 Element A

|  |
| --- |
| - Cách nhau bằng dấu cộng +  VD: Element A + Element B { |

### Chọn các Element … ngang cấp nằm sau 1 element A

|  |
| --- |
| - Cách nhau bằng dấu ngã ~  VD: Element A ~ Element B { |

### Chọn các Element có 1 [tên-attribute] nào đó

|  |
| --- |
| VD: Chọn các </a> có attribute target là \_blank a[target="\_blank"] {css}  Element / .tên-class / #tên-id[attribute=”value”] { } Các attr có value cụ thể  Element / .tên-class / #tên-id[attribute~=”abc”] { } Các attr có value có chữ abc (space/-)  Element / .tên-class / #tên-id[attribute|=”abc”] { } Các attr có value bắt đầu = chữ abc (space/-)  Element / .tên-class / #tên-id[attribute^=”abc”] { } Các attr có value bắt đầu = chữ abc  Element / .tên-class / #tên-id[attribute$=”abc”] { } Các attr có value kết thúc = chữ abc  Element / .tên-class / #tên-id[attribute\*=”abc”] { } Các attr có value có chứa chữ abc |

## :PSEUDO-CLASS & ::PSEUDO-ELEMENT

### Trạng thái nhất định của một element :pseudo-class

|  |
| --- |
| - Syntax: viết liền  Element / .tên-class / #tên-id:pseudo-class { |

### 4 pseudo-class của link </a>

|  |  |
| --- | --- |
| a:link {css} | link mới nguyên seal |
| a:visited {css} | link đã xem |
| a:hover {css} | link khi hơ chuột lên |
| a:active {css} | link khi được bấm vào |
| !!! Chú ý thứ tự link -> visited -> hover -> active | |

### Chọn input ở trạng thái focus

|  |
| --- |
| - Syntax: input[type=text] / #tên-id .tên-class:focus {css} |

### Chọn 1 Element là first child của 1 Element #tên-id .tên-class :first-child

|  |
| --- |
| - Syntax: Element #tên-id .tên-class elementChild:firstchild { }  vd: chọn element p nào là first child của #demo:  #demo p:first-child {  /\* nếu bỏ trống Element #tên-id .tên-class thì sẽ target mọi first child \*/ |

### Chọn một phần cụ thể của một element ::pseudo-element

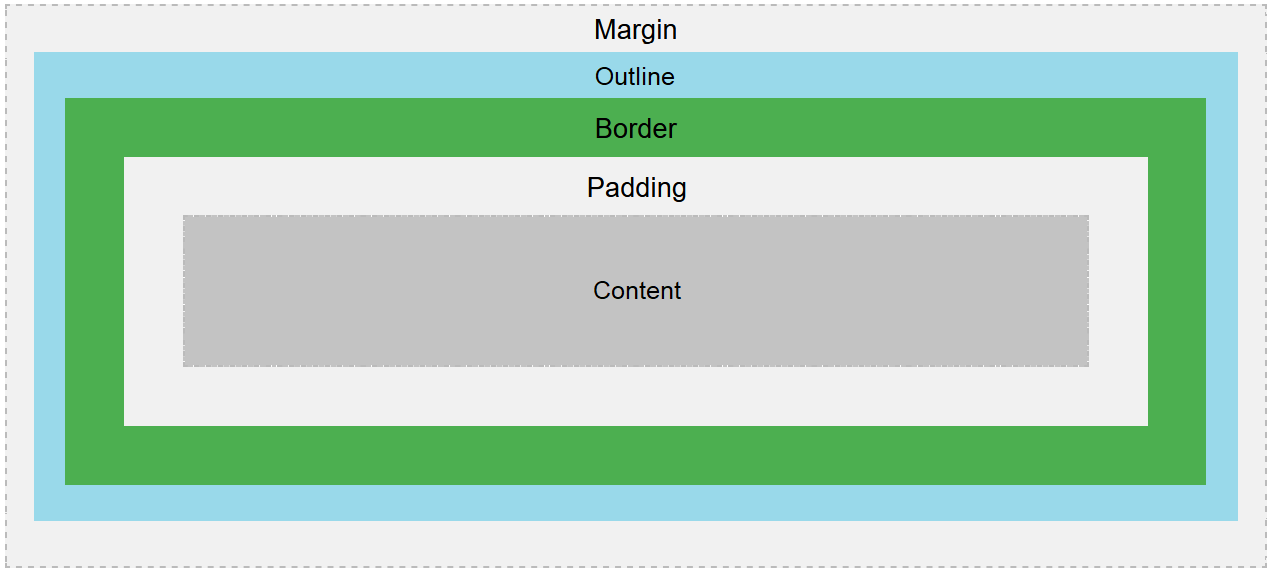
|  |  |
| --- | --- |
| Element / .tên-class / #tên-id::first-line {css}  Element / .tên-class / #tên-id::first-letter {css}  Element / .tên-class / #tên-id::before {content:}  Element / .tên-class / #tên-id::after {content:}  Element / .tên-class / #tên-id::selection {css} | Chọn dòng đầu tiên  Chọn chữ đầu tiên  (thêm content) vào trước element  (thêm content) vào sau element  Chọn các element được bôi đen |

## CSS BACKGROUND

### Các CSS chỉnh background-…

|  |  |
| --- | --- |
| Element / .tên-class / #tên-id { | |
| ***opacity***: 0-1; | độ trong suốt |
| ***background-blend-mode***: multiply; | dung hợp ảnh nền và màu\*\* |
| ***background-image***: url(“link-ảnh”); | Dùng ảnh làm nền |
| ***background-repeat***: repeat-x / repeat-y / no-repeat; | Lặp ảnh theo chiều ngang/dọc |
| ***background-position***: right left top bottom center; | Vị trí nền |
| ***background-attachment***: scroll / fixed ; | Nền di chuyển/cố định |
| ***background***: màu url(”link-ảnh”) repeat-x scroll top; | màu-ảnh-lặp-định-điểm |
| ***background-size***: auto / …px …px / cover / contain / inherit; | |

## CSS BOX MODEL



## CSS Text

### Trang trí text

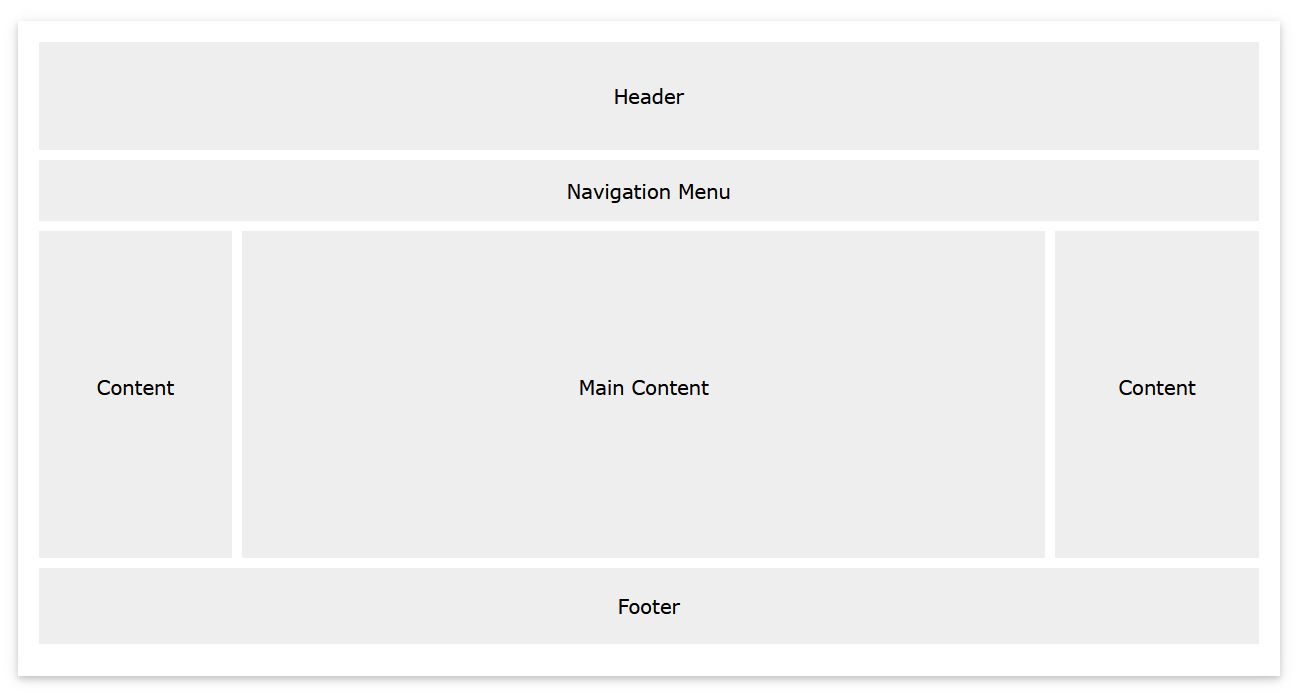
|  |  |
| --- | --- |
| Element / .tên-class / #tên-id { | |
| text-transform: capitalize / lowercase / uppercase ; | viết hoa/thường |
| text-indent: …px; | gạch đầu dòng đầu tiên |
| letter-spacing: ±…px; | khoảng cách giữa chữ cái |
| line-height: …px; | khách cách giữa các dòng |
| word-spacing: …px; | khoảng cách giữa các từ |
| vertical-align: top / middle / bottom ; | căn chiều ngang |

### Font chữ

|  |  |
| --- | --- |
| Element / .tên-class / #tên-id { | |
| <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia"> | Google fonts |
| font: italic bold weight size/height family; | |

## CSS LAYOUT

### Mẫu layout



|  |  |
| --- | --- |
| Header  <div class=”header”>  <h1>HEADER</h1>  </div> | .header {}  background-color:…;  padding: …;  text-align: …; |
| Top navigation bar  <div class=”topnav”>  <a href=”#”>Mục 1</a>  <a href=”#”>Mục n</a>  …  </div> | .topnav {  overflow: hidden;  background-color…  }  .topnav a {  float: left;  display: block;  color/padding/align…  }  .topnav a:hover {  css  } |
| Content  Chữ cột  <div class=”overall”>  <div class=”column”>  content 1…  </div>  <div class=”column”>  content 2…  </div>  </div> | .column {  float: left;  width: 100/số cột%; Chia cột bằng nhau  padding/margin: …;  }  .overall:after {  content: "";  display: table; Xử lý float  clear: both;  } |
| Header  <div class=”footer”>  <h1>FOOTER</h1>  </div> | .footer {}  background-color:…;  padding: …;  text-align: …; |

### Chỉnh cách hiển thị 1 element: ***display***

|  |  |
| --- | --- |
| display: inline; | Giữ trong dòng |
| display: block; | Tạo thành khối, xuống dòng |
| display: inline-block | Tạo thành khối, giữ trong dòng |
| display: none; | Tạm xóa khỏi flow |
| visibility: hidden; | Ẩn hiển thị, vẫn chiếm chỗ trong flow |

### Căn phải một element bằng float

|  |
| --- |
| Element / .tên-class / #tên-id {  float: right;  !!!Clear fix hack: set element gốc class=”clearfix”  .clearfix {  overflow: auto;  } |

### Kiểm soát vị trí của element: position

|  |
| --- |
| position: static; (default)  position: relative; (có thể di chuyển 4 hướng, relative với vị trí bt, element khác không đắp vào)  position: fixed; (có thể di chuyển 4 hướng, relative với viewport + không bị scroll)  position: absolute; (có thể di chuyển 4 hướng, relative với **positioned** element gốc hoặc body)  position: sticky; (thường là top: 0, relateive cho đến khi chạm top: 0  *!!! Khi positioned, element sẽ overlap các element khác, kiểm soát = z-index*  z-index: +/- …; (Nếu không set thì element được code sau sẽ ở trên) |

### Lồng chữ vào ảnh

|  |
| --- |
| <div class="Tổng"> /\*div này để ***position***: relative để trở thành *positioned* element\*/  <img src….>  <div class="tên-class">…</div> /\*div này để ***position***: absolute để relative theo div tổng\*/  </div> |

### Element quá lớn: xử lý với overflow

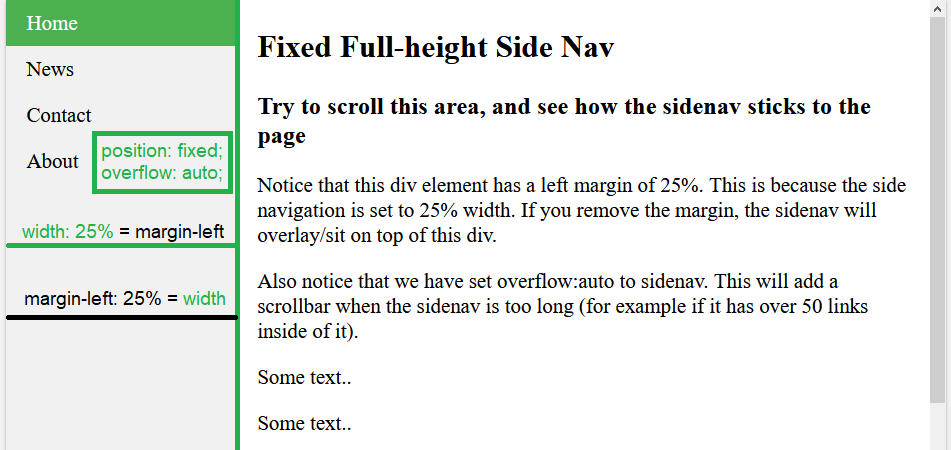
|  |  |
| --- | --- |
| ***overflow***: visible;  ***overflow***: hidden;  ***overflow***: scroll;  ***overflow***: auto;  ***overflow-x:***  ***overflow-y:*** | default, content tràn ra ngoài size chỉ định, trùng element khác  content thừa sẽ ẩn, không trùng element khác  content thừa sẽ có scroll  content thừa sẽ có scroll POWERED BY AI  chỉ tùy chỉnh cho left-right  chỉ tùy chỉnh cho top-bot |

### Kiểm soát vị trí các element bằng float

|  |  |
| --- | --- |
| ***float***: none;  ***float:*** right;  ***float:*** left;  ***float:*** inherit; | default, code đến đâu hiện đến đấy  nổi lên bên trái element gốc  nổi lên bên phải element gốc  kế thừa ***float*** của element gốc |

### Cho phép các element khác float xung quanh hay không: clear

|  |  |
| --- | --- |
| ***clear***: none;  ***clear***: left;  ***clear***: right;  ***clear***: both;  ***clear***: inherit;  ***box-sizing***: border-box;  .clearfix::after {  ***content***: "";  ***clear***: both;  ***display***: table;  } | default, cho float ở 2 bên  cấm float ở bên trái  cấm float ở bên phải  cấm float 2 bên  kế thừa ***clear*** của element gốc |



### **Tạo drop-down menu**

HTML

|  |
| --- |
| Mỗi </li> có id, a và </div class=”dropdown”> chứa các </a> phụ lục  <ul>  <li id=”mục-chính-1”>  <a href=”link-home”>Home</a>  <div class=”dropdown-1”>  <a href=”link-phụ-lục-1”>Phụ lục 1</a>  <a href=”link-phụ-lục-n”>Phụ lục n</a>  …  </div>  </li>  <li id=”tên-id-n”>  …  </ul> |

CSS

|  |
| --- |
| ul {  ***list-style-type***: none;  ***padding***: 0; ***margin***: 0;  ***overflow***: hidden;  ***background-color***: màu cho bav bar;  }  li {  ***float***: left;  }  a {  ***text-decoration***: none;  ***color***: màu chữ;  ***display***: block;  ***padding***:…px;  }  #dropdown-1, #dropdown-n, … {  ***position***: absolute;  ***color***, ***background-color***, ***z-index***, ***width***,….  ***display***: none;  }  #mục-chính-1:hover #dropdown-1 {  ***display***: block;  } |

### Dùng ảnh làm nav bar

HTML

|  |
| --- |
| Tạo </ul> có id riêng |

CSS

|  |
| --- |
| #tên-id-ul {  position: relative; (Để áp dụng absolute cho các </li>)  }  li {  position: absolute;  }  li, a {  height:…px; (chiều cao chung của ảnh)  display: block;  }  #tên-id-mục {  left: … px (cách lề trái)  width: …px (phần hiển thị tính từ lề trái ảnh, kết hợp height)  background: url(link-ảnh) … … ;  } |

### Ảnh nền là nhiều ảnh lồng vào nhau

|  |
| --- |
| background-image: url(link-ảnh-1st), url(link-anh-2nd);  background-position: right bottom/…, right bottom/…;  background-repeat: repeat/no-repeat/…, : repeat/no-reapeat/…;  hoặc  background: url(link-ảnh-1st) right bottom no-reapeat, url(link-anh-2nd)… ;  background-size: (width)…px (height)…px/contain/cover;  hoặc 1 loạt ảnh  background-size: …, …, …; |

### Ảnh nền cho cả page

|  |
| --- |
| html {  background: url(link-ảnh) no-repeat center fixed;  background-size: cover;  } |

### Ảnh dạng cover có nội dung bên trong: div nội dung có background là ảnh

|  |
| --- |
| HTML  <div class=”hero-image”>  <div class=”hero-text”>  <p>Content</p>  nút nếu có  </div>  </div> |
| CSS  body {  margin: 0;  font…;  }  .hero-image {  background: url(link-anh-hero) no-repeat center;  background-size: cover;  height:…px;  position: relarive;  }  .hero-text {  text-align: …;  position: absolute;  top:…  left:…  transform: translate(-50%, -50%);  } |

## TRANSFORMER

### Dịch chuyển vị trí 1 element

|  |
| --- |
| transform: translate(…, …); (…về bên phải, …xuống dưới); |

### Xoay element theo kim đồng hồ (negative để ngược kim đồng hồ)

|  |
| --- |
| transform: rotate(1-360deg); |

### Thay đổi kích thước element

|  |
| --- |
| transform: scale(…, …); (…lần width gốc, …lần height gốc);  transform: scaleX(…); (…lần width gốc);  transform: scaleY(…); (…lần height gốc); |

### Nghiêng 1 element

|  |
| --- |
| transform: skew(…deg, …deg);  transform: skewX(…deg);  transform: skewY(…deg); |

### Tổng tài thay đổi

|  |
| --- |
| transform: matrix(scaleX(), skewY(), skewX(), scaleY(), translateX(), translateY()) |

## TRANSITION TIMING

### Transitions-timing: Nhịp độ thay đổi

|  |  |
| --- | --- |
| transition-timing-function: linear;  transition-timing-function: ease;  transition-timing-function: ease-in;  transition-timing-function: ease-in-out;  transition-timing-function: ease-out; | Chạy đều  Ban đầu chậm, nhanh dần, kết thúc chậm  Ban đầu chậm, kết thúc nhanh  Ban đầu chậm, kết thúc chậm  Ban đầu nhanh, kết thúc chậm |

### Transition-delay

|  |
| --- |
| transition-delay: …s; |

### Transitions: Tổng

|  |
| --- |
| transition: property1 …s timing delay, property2 …s timing delay, …; |

## Animations:

|  |
| --- |
| 1. Đăng ký tên animation  animation-name: tên-anime;  animation-duration: …s;  animation-delay:…s; (Để số âm để start từ đoạn…s)  animation-iteration-count: …/infinite;  animation-direction: reverse/alternate/alternate-reverse;  animation-timing-function: ease/linear/ease-in/ease-out/ease-in-out;  animation-fill-mode: none/forwards/backwards/both;  Tổng tài:  animation: name time speed delay count direction  2. Lập @keyframes cho animation  @keyframes tên-anime { (nếu di chuyển cần position: relative)  0% {property-muốn-đổi-1: giá-trị-mới; property-muốn-đổi-2: giá-trị-mới;… }  25% {property-muốn-đổi: giá-trị-mới;…}  …  100% {property-muốn-đổi: giá-trị-mới;…}  } |

## FLEXBOX - Container

### Setup Flexbox container

|  |
| --- |
| - Tạo 1 div “container” bao trùm, display: flex; |

### Chọn hướng và warp flexbox ***flex-direction + flex-warp = flex-flow***

|  |
| --- |
| - Chọn hướng ***flex-direction***: (thứ tự theo code HTML)  ***flex-direction***: row; /\* default, xếp các box theo dàng ngang từ trái sang phải  ***flex-direction***: column; /\*xếp các box theo dàng dọc từ trên xuống  - Chọn xem có warp flexbox không ***flex-warp***: 2 kiểu (thứ tự theo code HTML)  ***flex-warp***: nowarp /\*default, các item sẽ không bị warp, sẽ cố nhét theo 1 line  ***flex-warp***: warp /\*các item sẽ bị warp, sẽ trải ra nhiều lines, fullsize  - Tổng tài hướng warp: ***flex-flow***  ***flex-flow:*** hướng warp; |

### Căn flex lines-spacing a.k.a căn cả group items (áp dụng cho warp) ***align-content***

|  |
| --- |
| - 5 kiểu căn flex lines:  ***align-content:*** stretch; /\*default, kéo giãn các items chiếm hết lines-spacing (-margin, padding)  ***align-content:*** center; /\*căn giữa group về tâm container  ***align-content:***; /\*căn lề group theo hướng trên, flex  ***align-content:*** flex-end; /\*căn lề group theo hướng dưới, flex  ***align-content:*** space-between; /\*khoảng giữa các lines sẽ đều nhau, 2 lề trên dưới = 0 (+margin) |

### Căn lề các items theo hướng flex ***justify-content***

|  |
| --- |
| - 6 kiểu căn lề items theo hướng flex:  ***justify-content:*** flex-start; /\*default, căn lề về đầu hướng flex  ***justify-content:*** flex-end; /\*căn lề về cuối hướng flex  ***justify-content:*** center; /\*căn giữa hướng flex  ***justify-content:*** space-between; /\*khoảng giữa các item sẽ đều nhau, 2 lề = 0 (+margin)  ***justify-content:*** space-evenly; /\*khoảng giữa các item sẽ đều nhau, trục giữa  ***justify-content:*** space-around; /\*khoảng cách giữa item và các lề sẽ bằng nhau |

### Căn lề các items theo hướng cross-flex ***align-items***

|  |
| --- |
| - 4 kiểu căn lề theo hướng cross-flex:  ***align-items:*** stretch; /\*default, kéo giãn item theo hướng cross-flex  ***align-items:*** flex-start; /\*căn lề về đầu hướng cross-flex a.k.a bên trên  ***align-items:*** flex-end; /\* căn lề về cuối hướng cross-flex a.k.a bên dưới  ***align-items:*** center; /\* căn giữa hướng cross-flex  - Căn lề một vài item chỉ định theo hướng cross-flex: ***align-self*** |

### Perfect center:

|  |
| --- |
| - Căn giữa tuyệt đối: căn giữa cả hướng flex và hướng cross-flex  ***justify-content:*** center; /\*căn giữa hướng flex  ***align-items:*** center; /\* căn giữa hướng cross-flex |

## FLEXBOX - Child Items

### Setup Flexbox child items

|  |
| --- |
| - Child items tự động display: flex |

### Độ grow/shrink và độ dài child items ***flex-grow*** / ***flex-shrink*** / ***flex-basis***

|  |
| --- |
| - Kiểm soát mức grow/shrink của child items:  child items {  ***flex-grow:*** number…; /\*theo tỷ lệ, nếu các items không lấp đủ container thì sẽ grow lên để lấp  ***flex-shrink:*** number…; /\*theo tỷ lệ,  }  - Độ dài của child items:  child items {  ***flex-basis:*** length…;  }  - Tổng tài:  ***flex:*** grow shrink length…; |

### Căn lề item cụ thể

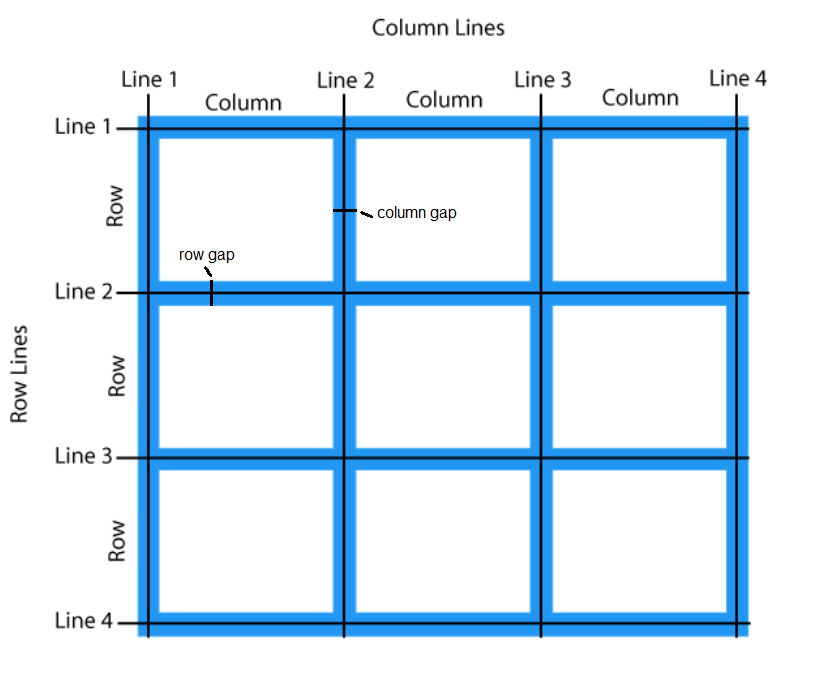
|  |
| --- |
| - Chỉnh ***align-self*** của items cụ thể sẽ override giá trị align-items của container  ***align-self:*** stretch; /\*default, kéo giãn 1 vài item cụ thể theo hướng cross-flex  ***align-self:*** flex-start; /\*căn lề 1 vài item cụ thể về đầu hướng cross-flex a.k.a bên trên  ***align-self:*** flex-end; /\* căn lề 1 vài item cụ thể về cuối hướng cross-flex a.k.a bên dưới  ***align-self:*** center; /\* căn giữa 1 vài item cụ thể hướng cross-flex |

# CSS GRID

### Setup Grid container

|  |
| --- |
| - Tạo 1 div “container” bao trùm, display: grid /\* Grid dạng block\*/  - Các element con đẻ của container sẽ là các items  - Khai báo width (của mỗi cells):  grid-template-columns: width1 width2 width3 width…  /\* nếu số items vượt quá số column thì sẽ auto tạo row để nhét thêm \*/  - Khai báo height (của mỗi cells):  grid-template-rows: height1 height2 height3 height…  /\* nếu số items vượt quá số column thì sẽ auto tạo row để nhét thêm \*/ |

### Sơ đồ 1 grid



### Tạo layout bằng tên

|  |
| --- |
| - Chỉnh trong .container  - Syntax:  grid-template-areas:  “tên-cell1-1 tên-cell1-2 . tên-cell1-4”/\*tạo row 1st có 4 cell: 1-1, 1-2, n/a, 1-4\*/  “tên-cell1-1 tên-cell2-2 tên-cell2-3 . ”/\*tạo row 2nd có 1+3 cell: 1-1(span) + 2-2, 2-3, n/a\*/  …;  VD:  grid-template-areas:  “header header header header” /\*tạo row 1st có 4 cells, header chiếm cả 4 cells \*/  “menu main main aside1”; /\*tạo row 2nd có 4 cells, menu - mainx2 cells giữa - aside1\*/  “menu main main aside2”; /\*tạo row 3rd có 4 cells, menu - mainx2 cells giữa - aside2\*/  “footer footer footer footer”; /\*tạo row 4th có 4 cells, footer chiếm cả 4 cells \*/  - Sau đó mang tên kia apply vào .itemX  .header { grid-area: header; }  .main { grid-area: main; }  … |

### Chỉnh grid gap

|  |
| --- |
| - Chỉnh trong .container  - Syntax: grid-gap: row-gap column-gap; (hoặc 1 cho cả hai) |

### Căn vị trí các items: trái -> phải (tổng width grid < container)

|  |
| --- |
| - Chỉnh trong .container, căn theo trái -> phải  justify-content: start /\* Căn lề sang trái tất cả items\*/  justify-content: end /\* Căn lề sang phải tất cả items\*/  justify-content: center /\* Căn giữa tất cả items\*/  justify-content: space-around /\* Các items có margin bằng nhau (2 margin chập lại)\*/  justify-content: space-between /\* Căn sang 2 lề và ở giữa gap bằng nhau\*/  justify-content: space-evenly /\* Căn gap giữa các items bằng nhau\*/ |

### Căn vị trí các items: trên -> dưới (tổng height grid < container)

|  |
| --- |
| - Chỉnh trong .container, căn theo trên -> dưới  align-content: start /\* Căn lề cạnh trên tất cả items\*/  align-content: end /\* Căn lề cạnh dưới tất cả items\*/  align-content: center /\* Căn giữa tất cả items\*/  align-content: space-around /\* Các items có margin bằng nhau (2 margin chập lại)\*/  align-content: space-between /\* Căn sang 2 cạnh và ở giữa gap bằng nhau\*/  align-content: space-evenly /\* Căn gap giữa các items bằng nhau\*/ |

### Chọn vị trí và span items theo lines

|  |
| --- |
| - Chỉnh trong .item cần chọn  - Syntax theo column lines (trái -> phải): grid-column:…  Kiểu 1: grid-column: 2 / 5; /\*bắt đầu từ c.lines 2, kéo dài và dừng lại tại c.lines 5\*/  Kiểu 2: grid-column: 1 / span 3; /\*bắt đầu từ c.lines 2, kéo dài ra trong 3 ô\*/  - Syntax theo row lines (trên -> dưới): grid-row:…  Kiểu 1: grid-row: 2 / 5; /\*bắt đầu từ r.lines 2, kéo dài và dừng lại tại r.lines 5\*/  Kiểu 2: grid-row: 1 / span 3; /\*bắt đầu từ r.lines 2, kéo dài ra trong 3 ô\*/  - Syntax Tổng tài: chọn 1 area nhiều cells: grid-area: … / … / … / …;  grid-area: row 1st / col 1st /row 2nd / col 2nd  VD1: grid-area: 1 / 2 / 5 / 6; /\*start từ top-left: row1/col2, end tại bottom-right: row5/col6\*/  VD2: grid-area: 1 / 2 / span 4 / span 4; /\*start từ tọa độ: row1/col2, span 4 row, span 4 col\*/ |

## Responsive Media Queries

Tổng quan

|  |
| --- |
| - Dùng để apply các style khác nhau trên các loại device khác nhau  - Syntax:  @media not|only loạimedia and (mediaFeature) and |or|not (mediaFeature) {css}  +not: làm ngược lại nội dung @media này  +only: support browser cũ  - Có thể dùng external CSS cho @media, nhét vào </link>  <link rel="stylesheet" media="screen and (max-width: 600px)" href="smallScreen.css">  <link rel="stylesheet" media="screen and (min-width: 900px)" href="wideScreen.css"> |

### Dựa trên kích thước width:

|  |
| --- |
| - Màn hình cỡ nhỏ (phones) : < 600px  @media only screen and (max-width: 600px) {...}  - Màn hình cỡ trung bình nhỏ (big phones, portraits tablets) : 600px +  @media only screen and (min-width: 768px) {...}  - Màn hình cỡ trung bình (landscape tablets) : 768px +  @media only screen and (min-width: 768px) {...}  - Màn hình cỡ trung bình lớn (laptop) : 992px +  @media only screen and (min-width: 992px) {...}  - Màn hình cỡ lớn (desktop) : 1200px +  @media only screen and (min-width: 1200px) {...} |

### Dựa trên kiểu landscape:

|  |
| --- |
| - Áp dụng khi device xoay landscape:  @media only screen and (orientation: landscape) {...} |

# 

# BOOSTRAP 4

## Default

|  |
| --- |
| - Padding: trái phải 15px;  - font-family: "Helvetica Neue", Helvetica, Arial, sans-serif  - font-size: 16px;  - line-height: 1.5; |

## Containers

### .container

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * - Sẽ set cứng max-width cho element, sẽ responsive theo các breakpoints:  |  |  |  |  |  | | --- | --- | --- | --- | --- | | Extra small  < 576px | Small  ≥ 576px | Medium  ≥ 768px | Large  ≥ 992px | Extra large  ≥ 1200px | | 100% | 540px | 720px | 960px | 1140px |  * - Default các .container sẽ có padding trái phải 15px * - Responsive .containers: sử dụng các hậu tố -sm, -md, -lg, -xl, max-width sẽ:  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | .container-*suffix* | Extra small  < 576px | Small  ≥ 576px | Medium  ≥ 768px | Large  ≥ 992px | Extra large  ≥ 1200px | | -sm (default)  .container chỉ active với màn > 576 | 100% | 540px | 720px | 960px | 1140px | | -md  container chỉ active với màn > 768 | 100% | 100% | 720px | 960px | 1140px | | -lg  container chỉ active với màn > 992 | 100% | 100% | 100% | 960px | 1140px | | -xl  container chỉ active với màn > 1200 | 100% | 100% | 100% | 100% | 1140px | |

### .container-fluid

|  |
| --- |
| * - Sẽ set fix width cho element luôn = 100% view width * - Default các .container sẽ có padding trái phải 15px (trên dưới k có) |

### Responsive containers

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - Có thể dùng thêm các hậu tố -sm, -md, -lg, -xl vào .container để responsive  - Max-width sẽ thay đổi như sau: (màu xám là lúc suffix từ 100% về 9x%)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | .Class-*suffix* | Extra small  <576px | Small  ≥576px | Medium  ≥768px | Large  ≥992px | Extra large  ≥1200px | | -sm (default)  container chỉ 100% với màn < 576 | 100% | 540px | 720px | 960px | 1140px | | -md  container chỉ 100% với màn < 768 | 100% | 100% | 720px | 960px | 1140px | | -lg  container chỉ 100% với màn < 992 | 100% | 100% | 100% | 960px | 1140px | | -xl  container chỉ 100% với màn < 1200 | 100% | 100% | 100% | 100% | 1140px | |

## Grid boostrap

### 5 class .col- cho 5 khoảng màn hình

|  |
| --- |
| - 5 class cho layout:  .col- (áp dụng cho mọi màn extra small, < 576px)  .col-sm- (áp dụng cho mọi màn từ small trở lên, >= 576px)  .col-md- (áp dụng cho mọi màn từ medium trở lên, >= 768px)  .col-lg- (áp dụng cho mọi màn từ large trở lên, >= 992px)  .col-xl- (áp dụng cho mọi màn từ extra large trở lên, >= 1200px)  - Các class này cộng dồn, nên nếu muốn set wid giống nhau cho màn từ medium trở lên và từ extra large trở lên thì chỉ cần set medium |

### Cú pháp điều chỉnh grid

|  |  |
| --- | --- |
| - Đầu tiên phải tạo 1 row để chứa các items 1 hàng class=”row”   |  | | --- | | col-\*-n  \* là điều kiện để active n: 1 trong 4 khoảng màn (omit sẽ = mọi màn)  n là khoảng không nó chiếm trên 12 cột (omit sẽ chia fr theo số lượng omit)  VD: col-sm-3 col-sm-3 col-sm-3 col-sm-3  1) Khoảng màn áp dụng: mọi màn từ 576 trở lên  2) 12/3 = 4 nên row sẽ chia làm 4 cột  => Với mọi màn > 576px thì chia 4 cột (thấp hơn thì default: 4 hàng 100%)  VD: col-sm-3 col-sm-3 col-sm-3 col-sm-3  1) Khoảng màn áp dụng: mọi màn từ 576 trở lên  2) 12/3 = 4 nên row sẽ chia làm 4 cột  => Với mọi màn > 576px thì chia 4 cột (thấp hơn thì default: 4 hàng 100%) | |

## Typography

### Font chung:

|  |
| --- |
| - font-family: "Helvetica Neue", Helvetica, Arial, sans-serif  - font-size: 16px;  - line-height: 1.5; |

### Các loại đề mục

|  |
| --- |
| - <h1> - <h6>: Các headings sẽ bold và font lớn hơn  - .display-1 -> 4: Lighter nhưng font lớn hơn <h1> - <h6>:  - <small>: Tạo chữ thứ cấp nhỏ hơn headings  - <mak>: hightlight text màu vàng |

## Màu sắc

### Màu chữ

|  |
| --- |
| - Các class “bối cảnh”: text- + muted, danger, important, success, secondary ... |

## Alerts

### Tạo alert message

|  |
| --- |
| - Dùng .alert và .alert-context  - Dùng .alert-links lên thẻ <a> trong alert để đồng bộ màu |

### Nút tắt alert

|  |
| --- |
| - Thêm .alert-dismissible vào alert contianer  - Thêm .close và data-dismiss="alert" vào element dùng để đóng (button or link)  - Thêm .fade và .show vào alert container để có hiệu ứng mờ dần khi tắt  VD: alert: thành công, tắt bằng 1 button có dạng dấu &times; và tắt đi mờ dần  <div class="alert alert-success alert-dismissible fade show">  <button type="button" class="close" data-dismiss="alert">&times;</button>  <strong>Success!</strong> Should <a href="#" class="alert-link">read this </a>.  </div> |

## Buttons

### Đăng ký button

|  |
| --- |
| - Dùng .btn để đăng ký 1 button  - Có thể đăng ký button cho cả link và input với .btn  VD:  <a href="#" class="btn btn-info" role="button">Link Button</a>  <button type="button" class="btn btn-info">Button</button>  <input type="button" class="btn btn-info" value="Input Button">  <input type="submit" class="btn btn-info" value="Submit Button"> |

### Màu button và màu viền button

|  |
| --- |
| - Dùng .btn-context để tạo màu cho button  - Dùng .btn-outline-context để tạo màu viền cho button |

### Active/Disabled button

|  |
| --- |
| - Dùng .active để button giữ trạng thái đã được nhấn  - Dùng attribute disabled để button không thể click được (thẻ <a> dùng .disabled)  VD:  <button type="button" class="btn btn-primary active">Active</button>  <button type="button" class="btn btn-primary" disabled>Disabled</button>  <a href="#" class="btn btn-primary disabled">Disabled Link</a> |

### Nhóm các button lại với nhau

|  |
| --- |
| - Bọc các button trong 1 div.btn-group để nhóm các button ngang  - Bọc các button trong 1 div.btn-group-vertical để nhóm các button dọc  - .btn-group cũng dùng để bọc 1 nhóm dropdown  - Các .btn-group default sẽ là “inline” |

## Nhãn dán (badge)

### Tạo 1 dán nhãn nhỏ bên cạnh element

|  |
| --- |
| - Gói nhãn trong 1 thẻ <span> nằm cuối element cần dán  - Dùng .badge và badge-context lên thẻ <span> đó  - Dùng .badge-pill để tạo badge dạng viên thuốc nằm ngang |

## Progress bar

### Tạo thanh mô phỏng tiến trình

|  |
| --- |
| - Dùng .progress lên container để đăng ký  - Tạo 1 div con để làm bar, dùng .progress-bar lên div đó  - Chỉnh mức progress = CSS width lên .progress-bar  - |

### Chỉnh kích thước progress bar

|  |
| --- |
| - Chỉnh chiều cao progress = CSS width lên .progress-bar VÀ .progress = nhau  - Chỉnh chiều rộng progress = CSS width lên .progress |

### Label, màu mè, trang trí progress bar

|  |
| --- |
| - Label: Viết text vào trong .progress-bar để làm label  - Dùng .bg-context lên .progres-bar để chỉnh màu  - Dùng .progress-bar-striped lên .progres-bar để tạo sọc  - Dùng .progress-bar-animated lên .progres-bar đã có sọc để chạy animation |

### Stacked progress bar

|  |
| --- |
| - Dùng nhiều .progress-bar trong 1 .progress container |

## Spinner/loader

### Tạo spinner/loader

|  |
| --- |
| - Dùng .spinner-border để tạo spinner xoay tròn  - Dùng .spinner-grow để tạo loader phóng to  - Chỉnh màu = .text-context  - Chỉnh kích cỡ = các class kích thước  - Dùng spinner/loader lên 1 thẻ <span> để nhúng vào các element |

## Pagination (thanh đếm trang <ul>)

### Tạo thanh đếm trang từ 1 cụm thẻ <ul>

|  |
| --- |
| - Dùng .pagination lên <ul>  - Dùng .page-item lên các <li>  - Dùng .page-link lên các <a> |

### Trạng thái active/disabled

|  |
| --- |
| - Dùng .active lên <li> hiện đang active  - Dùng .disabled lên <li> không thể click |

### Tinh chỉnh thanh đếm trang

|  |
| --- |
| - Căn lề thanh đếm bằng các class căn lề dùng lên .pagination  - Chỉnh kích thước thanh đếm bằng các class kích thước |

### Tạo thanh đếm dạng Breadcrumbs: Devices/Phones/Samsung

|  |
| --- |
| - Dùng .breadcrumb lên <ul> và .breadcrumb-item lên các <li>  - Bắt buộc dùng .active lên <li> trong cùng (page hiện tại) |

## List Groups (<ul>)

### Tạo list thường, không phải là list chứa link

|  |
| --- |
| - Dùng .list-group lên <ul>  - Dùng .list-group-item lên các <li> |

### Tạo list chứa link (dùng div + a thay cho ul + li)

|  |
| --- |
| - Dùng .list-group lên div  - Dùng .list-group-item lên các <a>  - Dùng thêm .list-group-item-action lên các a .list-group-item để tạo hover |

### Active, disabled

|  |
| --- |
| - Dùng .active lên ul/div .list-group để thể hiện active  - Dùng .disabled lên li/a .list-group-item để khóa item |

### List ngang

|  |
| --- |
| - Dùng .list-group-horizontal lên ul/div .list-group để list nằm ngang |

### Trang trí, thêm thắt

|  |
| --- |
| - Dùng .list-group-item-context lên các item để đổ màu context  - Dùng .list-group-flush lên ul/div .list-group để bỏ 4 borders bao ngoài  - Dùng badges: <span class="badge">Nhãn<span> ở cuối item để tạo nhãn  - Căn chỉnh nội dung và badges = ultilities |

## Cards (posts, panels)

### Cards cơ bản

|  |
| --- |
| - Dùng .card lên 1 div để tạo container cho card, card có 3 phần (3 div con):  1. Dùng .card-header lên 1 div con để tạo header cho card  2. Dùng .card-body lên 1 div con để tạo main content cho card (padding)  3. Dùng .card-footer lên 1 div con để tạo footer cho card |

### Trang trí, thêm thắt

|  |
| --- |
| - Dùng .bg-context lên .card để đổ màu context cho card |

### .card-body

|  |
| --- |
| - Dùng .card-title lên 1 heading h1 - h6 để làm headings (đặt trên cùng)  - Dùng .card-subtitle lên 1 heading h1 - h6 nhỏ hơn để làm subtitle  - Dùng .card-text lên <p> cuối cùng/duy nhất để bỏ margin cuối  - Dùng .card-link lên các thẻ <a> để đổ màu link xanh và hover |

### Card images

|  |
| --- |
| - Đặt thẻ img ra ngoài .card-body để không bị padding  - Dùng .card-img-top hoặc .card-img-bottom lên img để căn trên/dưới ảnh  - Dùng .card-img-overlay lên 1 div ngang cấp thẻ img. Thẻ img sẽ thành ảnh nền và content của .card-img-overlay sẽ đè lên trên img  VD:  <div class="card">  <img class="card-img-top" src="”>  <div class="card-img-overlay">  <h4 class="card-title">TITLE</h4>  <p class="card-text">TEXT</p>  </div>  </div> |

### Card layout dạng Pinterest: .card-columns

|  |
| --- |
| - Dùng .card-columns lên 1 div container chứa nhóm thẻ  - Sẽ tự responsive về 1 cột với màn extra small  - Dùng CSS property column-count cho .card-columns để chỉnh số cột @media |

### Card layout dạng grid các cột đều nhau: .card-deck

|  |
| --- |
| - Dùng .card-deck lên 1 div container chứa nhóm thẻ  - Các thẻ sẽ thành các cột cùng chiều rộng (chia đều) và chiều cao (theo thẻ cao nhất), cách nhau 1 khoảng grid gap đều nhau  - Sẽ tự responsive về 1 cột với màn extra small |

### Card layout dạng các cột sát và đều nhau : .card-group

|  |
| --- |
| - Dùng .card-deck lên 1 div container chứa nhóm thẻ  - Các thẻ sẽ thành các cột cùng chiều rộng (chia đều) và chiều cao (theo thẻ cao nhất), nằm sát nhau (không có gap)  - Sẽ tự responsive về 1 cột với màn extra small (sẽ có gap như columns và deck) |

## Dropdowns

### Dropdown thông thường

|  |
| --- |
| - Bao gồm 4 lớp thành phần:  1. dropdown wrapper: Dùng .dropdown lên 1 wrapper (div, btn-group, nav-item)  2. dropdown toggle: Dùng .dropdown-toggle và data-toggle="dropdown" lên trigger hiện dropdown (button, <a>)  3. dropdown menu: Tạo 1 div làm menu và dùng .dropdown-menu  4. dropdown items: Tạo các <a> trong menu và dùng .dropdown-item |

### Căn lề phải cho dropdown (default là căn lề trái)

|  |
| --- |
| - Dùng .dropdown-menu-right lên .dropdown-menu để căn lề phải |

### Hướng Drop phải/trái/trên

|  |
| --- |
| - Dùng .dropright lên wrapper .dropdown để đổ menu sang phía bên phải  - Dùng .dropleft lên wrapper .dropdown để đổ menu sang phía bên trái  - Đổi wrapper thành .dropup để đổ menu từ dưới lên trên |

### Dropdown trong 1 btn-group

|  |
| --- |
| - Đổi .dropdown thành .btn-group (toàn bộ dropdown sẽ là 1 .btn-group), 4 tp:  1. dropdown wrapper: Dùng .btn-group lên wrapper (là 1 div bọc lấy toggle button trigger và dropdown menu, ngang hàng các button gốc)  2. dropdown toggle: Dùng .dropdown-toggle và data-toggle="dropdown" lên toggle button  3. dropdown menu: Tạo 1 div làm menu và dùng .dropdown-menu  4. dropdown items: Tạo các <a> trong menu và dùng .dropdown-item |

## Collapse

### Collapse cơ bản

|  |
| --- |
| - Dùng data-toggle="collapse" và data-target="#Target-id" lên trigger  (hoặc dùng href thay cho data-target nếu trigger là <a>)  - Dùng .collapse lên target cần collapse (link = Target-id)  - Dùng thêm .show lên target để hiện sẵn (default là collapse sẵn) |

### Accordion (nhóm collapse mở cùng lúc chỉ 1)

|  |
| --- |
| - Dùng data-parent=”#Tên-id-accordion” lên các target. Các target có cùng parent sẽ chỉ mở 1 cái tại 1 thời điểm (cái nào show thì các cái khác sẽ collapse)  VD:  <div class="container" id="accordion">  <div class="card">  <div class="card-header">  <a href="#collapse-1" data-toggle="collapse">Item #1</a>  </div>  <div class="collapse show" id="collapse-1" data-parent="#accordion">  <div class="card-body">  Lorem ipsum dolor sit amet, consectetur adipisicing elit,  </div>  </div>  </div>  <div class="card">  <div class="card-header">  <a href="#collapse-2" data-toggle="collapse">Item #2</a>  </div>  <div class="collapse show" id="collapse-2" data-parent="#accordion">  <div class="card-body">  Lorem ipsum dolor sit amet, consectetur adipisicing elit,  </div>  </div>  </div>  </div> |

## Nav

### Nav menu cơ bản

|  |
| --- |
| - Dùng .nav lên 1 <ul>  - Dùng .nav-item lên các <li>  - Dùng .nav-link lên các <a> |

### Căn lề .nav (default là căn trái)

|  |
| --- |
| - Dùng .justify-content-center lên .nav để căn giữa  - Dùng .justify-content-end lên .nav để căn phải |

### .nav dọc

|  |
| --- |
| - Dùng .flex-column lên .nav |

### . active và .disabled nav link

|  |
| --- |
| - Dùng .active lên .nav-link hiện tại  - Dùng .disabled lên .nav-link muốn khóa |

### nav dạng tab

|  |
| --- |
| - Dùng .nav-tabs lên .nav  - Căn đều các tabs: dùng .nav-justified lên .nav |

### .nav dạng pills

|  |
| --- |
| - Dùng .nav-pills lên .nav  - Căn đều các pills: dùng .nav-justified lên .nav |

### Dropdown tabs/pills

|  |
| --- |
| - 4 lớp thành phần  1. dropdown wrapper: là .nav-item chứa toggle .nav-link, dùng .dropdown  2. dropdown toggle: là toggle .nav-link, dùng dropdown-toggle và data-toggle="dropdown"  3. dropdown menu: là 1 div chứa menu nằm dưới toggle, dùng .dropdown-menu  4. dropdown item: là các thẻ <a> trong dropdown menu, dùng .dropdown-item |

### Dynamic tabs/pills

|  |
| --- |
| 2 phần: chỉnh .nav-link và tạo pane  1.1 Dùng thêm data-toggle="tab" / data-toggle="pill" lên mọi .nav-link  1.2 Các .nav-link sẽ có href trỏ đến các #pane-id tương ứng  2.1 Tạo 1 div chứa các pane nằm ngoài .nav, dùng .tab-content  2.2 Tạo các pane div, dùng .tab-pane chung và id riêng (thêm .active và .fade) |

## Navbar

### Navbar cơ bản

|  |
| --- |
| - 4 lớp thành phần  1. navbar: là <nav>, dùng .navbar và .navbar-expand-@media để control  (nếu omit .navbar-expand-@media thì sẽ luôn là navbar dọc)  2. navbar nav: là <ul>, dùng .navbar-nav  3. nav item: là các <li>, dùng .nav-item  4. nav link: là các <a>, dùng .nav-link |

### Logo cho navbar

|  |
| --- |
| - Dùng .navbar-brand lên một <a>/button trước <ul> (có thể nhúng ảnh) |

### Navbar có collapse

|  |
| --- |
| - 6 lớp thành phần (+ 1 nút toggler và 1 wrapper phần còn lại)  1. navbar: là <nav>, dùng .navbar và .navbar-expand-@media để control  (nếu omit .navbar-expand-@media thì sẽ luôn là navbar dọc)  2. collapse toggler: là 1 button, dùng 3 thứ:  .navbar-toggler  data-toggle="collapse"  data-target="#navbar-nav-id" (#id của <ul>)  *\* hamburger: <span class="navbar-toggler-icon"></span>*  3. navbar nav wrapper: là 1 div chứa content, dùng 2 thứ:  .collapse navbar-collapse  id=”navbar-nav-id”  4. navbar nav: là <ul>, dùng .navbar-nav  5. nav item: là các <li>, dùng .nav-item  6. nav link: là các <a>, dùng .nav-link |

### Form trong navbar

|  |
| --- |
| - Dùng .form-inline lên form (form có thể gói cả các button) |

### Vị trí navbar

|  |
| --- |
| - Dùng .fixed-top lên .navbar để fix trên top  - Dùng .fixed-bottom lên .navbar để fix dưới bottom  - Dùng .sticky-top lên .navbar để fix trên top khi scroll đến navbar (IE 11+) |

## Form

### .form-group cho stacked form (full-width)

|  |
| --- |
| - Bọc input và label vào từng div, dùng .form-group lên các div đó |

### .form-inline cho inline form (chỉ màn > 576px)

|  |
| --- |
| - Dùng .form-inline lên <form>, không cần bọc div từng input |

### Validation: check trước, báo lỗi trước

|  |
| --- |
| - Dùng .was-validated lên <form>  - Thêm 1 div.valid-feedback chứa nội dung khi valid, đặt sau input  - Thêm 1 div.invalid-feedback chứa nội dung khi invalid, đặt sau input |

### Validation: check sau, báo lỗi sau

|  |
| --- |
| - Dùng .needs-validated và novalidate lên <form>  - Thêm 1 div.valid-feedback chứa nội dung khi valid, đặt sau input  - Thêm 1 div.invalid-feedback chứa nội dung khi invalid, đặt sau input  - Thêm JQuery |

## Inputs

### .form-control cho đa số <input>

|  |
| --- |
| - Dùng .form-control lên các <input> để đồng bộ style: padding, 100% width  - Ngoại trừ:  1. checkbox, radio dùng .form-check-input  2. range dùng .form-control-range  3. file dùng .form-control-file (dùng thêm border attribute để hiện viền) |

### .form-check-\* cho checkbox, radio

|  |
| --- |
| - 3 lớp thành phần:  1. Bọc label (chứa check/radio) bằng 1 div, dùng .form-check lên div đó  (muốn các check/radio cùng hàng thì dùng .form-check-inline)  2. Bọc check/radio bằng label, dùng .form-check-label lên label đó  3. Dùng .form-check-input lên check/radio |

### Select list

|  |
| --- |
| - Bọc <select> và <label> vào 1 div, dùng .form-group lên div đó  - Dùng .form-control lên <select> |

### .form-control-plaintext để bỏ viền

|  |
| --- |
| - Upgrade .form-control thành .form-control-plaintext để bỏ viền <input> |

## Input group

### .input-group để upgrade input bằng help text

|  |
| --- |
| - label đặt bên ngoài input group (chú ý for và id)  - 4 lớp thành phần:  1. input group wrapper: Bọc cả input và help text wrapper bằng 1 div:  1.1 dùng .input-group lên div đó  2. help text wrapper: Bọc help text bằng 1 div:  2.1 dùng .input-group-prepend để đặt help text trước input  2.2 dùng .input-group-append để đặt help text sau input  3. help text: là 1 <span> help text (text, check, button), dùng .input-group-text  (Có thể dùng nhiều help text trong 1 help text wrapper)  \* help text là dropdown: ko dùng <span>. Dropdown chỉ 3 tp, ko cần wrapper  a. dropdown toggler: Tạo 1 button/<a> làm toggler,  a.1 dùng .dropdown-toggle  a.2 dùng attribute data-toggle="dropdown”  b. dropdown menu: Tạo 1 div làm menu và dùng .dropdown-menu  c. dropdown items: Tạo các <a> trong menu và dùng .dropdown-item  4. input: dùng .form-control để đồng bộ style  (Có thể dùng nhiều input trong 1 input group wrapper) |

## Customized form

### Custom checkbox (SWITCH edition) và custom radio

|  |
| --- |
| - Bắt buộc tách riêng <input> và <label>. Input phải đứng trước label  - 3 bước  1. Custom wrapper: Bọc cả input và label vào 1 div,  1.1 checkbox: dùng .custom-control VÀ .custom-checkbox  \* SWITCH edition: .custom-control VÀ .custom-switch  1.2 radio: dùng .custom-control VÀ .custom-radio  (nếu muốn inline: dùng thêm .custom-control-inline lên wrapper)  2. Custom input: Dùng .custom-control-input lên <input>  3. Custom label: Dùng .custom-control-label lên <label> |

### Custom select, range:

|  |
| --- |
| - Dùng .custom-select lên <select>  - Dùng .custom-range lên range input |

### Custom file input:

|  |
| --- |
| - 4 bước  1. Custom wrapper: Bọc cả input và label vào 1 div, dùng .custom-file lên div đó  2. Dùng .custom-file-input lên <input>  3. Dùng .custom-file-label lên <label>  4. Dùng Jquery:  $(".custom-file-input").on("change", function() {  var fileName = $(this).val().split("\\").pop();  $(this).siblings(".custom-file-label").addClass("selected").html(fileName);  }); |

## Carousel

### Carousel cơ bản (có indicator và control)

|  |
| --- |
| -  1. Carousel wrapper: Tạo 1 div bọc toàn bộ, có id VD “wrapper-id”  1.1 Declare 1 carousel: dùng .carousel  1.2 Tạo hiệu ứng trượt: dùng .slide  1.3 Page load: dùng attribute data-ride="carousel"  2. Indicators: minimap bên dưới  2.1 indicators: là 1 thẻ <ul>, dùng .carousel-indicators  2.2 indicators <li>:  2.2.a Hướng trỏ: dùng attribute data-target="#wrapper-id"  2.2.b Vị trí silde dưới dạng index[0]: data-slide-to=”index”  2.2.c Nếu là slide khởi đầu: dùng thêm .active  3. Slide inner: là 1 div chứa content(ảnh)  3.1 Inner wrapper: dùng .carousel-inner  3.2 Carousel items: bọc các <img> và caption vào trong các div,  3.2.a Đánh dấu item: dùng .carousel-item lên các div trên  3.2.b Nếu là item khởi đầu: dùng thêm .active  3.2.c Caption wrapper: Bọc caption vào 1 div, dùng carousel-caption  4. Controls: là 2 nút điều khiển 2 bên  4.1 Nút bên trái (previous): là một thẻ <a> chứa icon (là 1 <span>)  4.1.a Đánh dấu nút prev: dùng .carousel-control-prev  4.1.b Hướng trỏ: href=”#wrapper-id”  4.1.c Hướng trượt: dùng attribute data-slide=”prev”  4.1.d Icon prev: chứa <span class="carousel-control-prev-icon"></span>  4.2 Nút bên phải (next): là một thẻ <a> chứa icon (là 1 <span>)  4.1.a Đánh dấu nút next: dùng .carousel-control-next  4.1.b Hướng trỏ: href=”#wrapper-id”  4.1.c Hướng trượt: dùng attribute data-slide=”next”  4.1.d Icon next: chứa <span class="carousel-control-next-icon"></span |

## Modal (BS4 alert)

### Modal cơ bản

|  |
| --- |
| - 2 thành phần: (5 thành phần nhỏ)  1. Modal toggler: 2 attribute: data-toggle=”modal” và data-target=”#modal-id”  2. Modal: (3 wrapper)  2.1 Modal wrapper: bọc modal vào 1 div, dùng .modal + .fade và id=”modal-id”  2.1.1 Modal dialog: bọc tiếp 1 lớp div, dùng .modal-dialog  (nếu muốn perfect center modal thì dùng thêm .modal-dialog-centered)  (nếu content dài, cần scroll thì dùng thêm .modal-dialog-scrollable)  2.1.1.1 Modal content: bọc tiếp 1 lớp div nữa, dùng .modal-content  2.1.1.1.a Modal header: là 1 div, dùng .modal-header  2.1.1.1.a.1 Modal title: là 1 <h1-6>, dùng .modal-title  2.1.1.1.b Modal body: là 1 div, dùng .modal-body  2.1.1.1.b.1 Content cho body…  2.1.1.1.c Modal footer: là 1 div, dùng .modal-footer  2.1.1.1.c.1 Content cho footer, thường là nút close:  <button type="button" class="btn" data-dismiss="modal">Close</button> |

## Tooltip

### Tooltip cơ bản

|  |
| --- |
| - Cần dùng Jquery:  $(document).ready(function(){  $('[data-toggle="tooltip"]').tooltip();  });  - Dùng attribute: data-toggle="tooltip" lên target  - Dùng attribute: title=”nội dung tooltip” |

### Vị trí tooltip

|  |
| --- |
| - Dùng thêm attribute: data-placement=”top/bottom/left/right” |

## Popover (clickable, tooltip2.0)

### Popover cơ bản

|  |
| --- |
| - Cần dùng Jquery:  $(document).ready(function(){  $('[data-toggle="popover"]').popover();  });  - Dùng attribute: data-toggle="popover" lên target  - Dùng attribute: title=”header cho popover”  - Dùng attribute: data-content=”nội dung popover”  - Dùng attribute: data-trigger=”focus” để tắt popover từ mọi chỗ  \*cosplay tooltip a.k.a tooltip2.0:  - Dùng attribute: data-trigger=”hover” |

## Toast (popover2.0)

### Toast cơ bản

|  |
| --- |
| - Cần Jquery: (chỉ đích danh id toggler)  $(document).ready(function(){  $("#toggler-id").click(function(){  $('.toast').toast('show');  });  });  - 3 thành phần:  1. Toast wrapper: Gói toàn bộ toast vào 1 div, dùng .toast  (nếu muốn hiện sẵn khi load thì dùng thêm attribute: data-autohide=”false”)  1.1 Toast header: tạo 1 div header, dùng .toast-header  (nếu hiện sẵn thì thêm nút tắt toast:  <button type="button” data-dismiss="toast">&times;</button> )  1.2 Toast body: tạo 1 div body, dùng .toast-body |

## Scrollspy

### Scrollspy navbar

|  |
| --- |
| -  1. Scroll area: chính là <body>,  1.1 set position: relative  1.2 Dùng attribute: data-spy=”scroll”  1.3 Dùng attribute: data-target=".navbar"  (1.4 Dùng attribute: data-offset=”n” để khớp pixel)  2. Đồng bộ href của .nav-link và id của section tương ứng  (hoặc href của .dropdown-item) |

## Ultilies

### Borders

|  |
| --- |
| - Thêm/xóa border:  a. Thêm 4 borders: .border  b. Xóa 4 borders: .border và .border-0  c. Chỉ xóa border top/botto/right/left: .border và .border-hướng-0  - Màu border:  a. Chỉnh màu: .border và .border-context  - Border radius:  a. Bo tròn 4 góc: .rounded  b. Bo tròn element: .rounded-circle |

### Căn lề trái/phải/giữa

|  |
| --- |
| - Căn trái/phải với float:  1. Bọc (các) element cần float vào 1 div, dùng .clearfix lên div đó  2. Dùng .float-right hoặc .float-left lên (các) element  (nếu muốn responsize, dùng .float-cỡ-right hoặc .float-cỡ-left  - Căn giữa (margin:auto): dùng .mx-auto |

### Chỉnh width, height

|  |
| --- |
| - Width: Dùng .w-25 / .w-50 / .w-75 / .w-100 / .mw-100  - Height: Dùng .h-25 / .h-50 / .h-75 / .h-100 / .mh-100 |

### Spacing

|  |
| --- |
| - Cú pháp: (có thể dùng 2 tổ hợp cho padding và margin)  1. class=”{property}{sides}-{breakpoints}-{size}”  => set margin/padding hướng t/b/l/r/x/y/omit tại màn … là 0-5/auto  1.1 property:  m hoặc p (set margin hoặc padding)  1.2 sides:  t/b/l/r hoặc x/y (trái-phải/trên-dưới) hoặc omit (4 hướng)  1.3 breakpoints:  omit: < 576  sm: >576px  md: >768px  lg: >992px  xl: >1200px  1.4 size:  0 = 0  1 = 0.25rem (4px)  2 = 0.5rem (8px)  3 = 1rem (16px)  4 = 1.5rem (24px)  5 = 3rem (48px)  auto = margin auto |

### Shadows

|  |
| --- |
| - Dùng .shadow để tạo bóng  - Chỉnh kích thước: dùng .shadow-sm/lg/none |

### Fixed/sticky vị trí top/bottom

|  |
| --- |
| - Dùng .fixed-top / .fixed-bottom để fix  - Dùng .sticky-top lên .navbar để fix trên top khi scroll đến navbar (IE 11+) |

### Nút close

|  |
| --- |
| - Là một button, dùng .close:  <button type="button" class="close">&times;</button> |

### Biến element thành block

|  |
| --- |
| - Dùng .d-block để biến element thành block  - Dùng .d-breakpoints-block để chọn màn mà element thành block |

## Flex

### Biến 1 container thành flex, inline-flex

|  |
| --- |
| - Dùng .d-flex để biến container thành flex (100% width)  - Dùng .d-inline-flex để biến container thành flex, inline (không 100% width)  - Responsive: .d-màn-flex / .d-màn-inline-flex |

### Flex column

|  |
| --- |
| - Dùng .flex-column để flex thành cột  - Responsive: .d-màn-column |

### Căn theo main-axis flex

|  |
| --- |
| - Dùng .justify-content-end/center/between/around  - Responsive: .justify-content-màn-end/center/between/around |

### Auto margin các items

|  |
| --- |
| - Có thể dùng margin-left / right để đẩy hết các item khác về kịch trái/phải,  - Dùng .mr-auto để đẩy các item còn lại kịch phải  VD: 3 items, dùng .mr-auto lên item 2 sẽ đẩy item 3 kịch phải, item 1+2 kịch trái  - Dùng .ml-auto để đẩy các item còn lại kịch trái  VD: 3 items, dùng .mr-auto lên item 2 sẽ đẩy item 1 kịch trái, item 2+3 kịch trái |

### Kích thước flex items

|  |
| --- |
| - Dùng .flex-fill lên các (mọi) flex items để các (mọi) items đó rộng bằng nhau  - Dùng .flex-grow-1 lên 1 flex item để force nó grow hết width còn lại  - Responsive: .flex-màn-fill và flex-màn-grow-1 |

### Wrap và hệ lụy align

|  |
| --- |
| - Default là no wrap, item sẽ chỉ theo 1 hàng:  1. Default là .align-items-stretch để mọi item cao đều nhau, full hight  2. Dùng .align-items-start để mọi item kịch về start cross-axis  3. Dùng .align-items-end để mọi item kịch về end cross-axis  4. Dùng .align-items-center để mọi item kịch về giữa cross-axis  5. Dùng .align-items-baseline để mọi item kịch theo baseline  \* Dùng .align-self để tự căn lẻ 1 item, như .align-items  - Dùng .flex-wrap, items sẽ chia theo các hàng:  1. Dùng .align-content-end để các hàng đều kịch dưới cross axis  2. Dùng .align-content-center để các hàng nhóm về giữa cross axis  3. Dùng .align-content- stretch để các hàng cao đều nhau, full hight |

## Mediabox (comments)

### Mediabox cơ bản

|  |
| --- |
| - 3 lớp tp  1. media wrapper: là 1 div dùng .media  1.1 media img: là 1 img (giới hạn width, dùng .align-self để căn lên/xuống)  1.2 content wrapper: là 1 div dùng .media-body  (thứ tự img và content wrapper sẽ có ảnh hưởng vị trí) |

### Mediabox nested

|  |
| --- |
| 1. big-media wrapper: là 1 div dùng .media  1.1 big-media img: là 1 img (giới hạn width, dùng .align-self để căn lên/xuống)  1.2 big-content wrapper: là 1 div dùng .media-body  1.2.1 small- media wrapper: là 1 div dùng .media  1.2.1.1 small-media img: là 1 img (giới hạn width, .align-self)  1.2.1.2 small-content wrapper: là 1 div dùng .media-body |

### Vị trí tooltip

|  |
| --- |
|  |

### Vị trí tooltip

|  |
| --- |
|  |

### Vị trí tooltip

|  |
| --- |
|  |

### Vị trí tooltip

|  |
| --- |
|  |

# JAVASCRIPT

## Data types

### 9 loại bao gồm 6 Primitives và 3 loại khác

|  |
| --- |
| - The Six Primitives, có thể check bằng operator *typeof :*  1. undefined  + Là loại data chưa được xác định (VD gọi ra 1 variable chưa được declare)  2. Boolean  + Là loại data chỉ có 2 giá trị: true hoặc false  3. Number  + Là loại data có dạng các số, NaN, ±Infinity  4. String  + Là loại data có dạng chuỗi các chữ cái nằm trong ngoặc ‘abc’  5. BigInt  +  6. Symbol  +  - 3 loại còn lại:  7. null  + Là data rỗng  8. Object  + Là loại cấu trúc chứa nhiều cặp properties (data dưới dạng name:value)  9. Function:  + Là loại cấu trúc chứa các tác vụ (kích hoạt khi được call) |

### Data types methods

## Module

### Khai báo module

|  |  |
| --- | --- |
| - Có thể share JS codes bằng cách export hoặc import 1 phần code  - Đoạn </script> cần có type=”module” để có thể export/import   |  | | --- | | <script type=”module” src…> code </script> |   . |

### Export code block

|  |  |  |  |
| --- | --- | --- | --- |
| - Có 2 cách để export các biến (đã được gán value là code): 1 biến or nhiều biến  1. Export 1 biến: thêm export vào trước declaration   |  | | --- | | export const add = (x, y) => { return x + y; } |   2. Export nhiều biến: dùng 1 export statement chung   |  | | --- | | export { varName1, funcName1, … } |   - Đặc biệt: khi chỉ export 1 value (không phải 1 biến) và làm fallback default   |  | | --- | | export default funtion(x, y) { code… } //không tên  export default funtion myFunc(x, y) { code… } //có tên |   . |

### Import code block

|  |  |  |  |
| --- | --- | --- | --- |
| - Import một vài thứ   |  | | --- | | import { varName1, funcName1, … } from ‘./index.js’ |   .  - Import toàn bộ content file, gán thành 1 variable   |  | | --- | | import \* as newVarName from ‘./index.js’ |   - Nội dung from phải gói trong 1 string  - Lưu ý vị trí file (./) và loại file (.js)  - Nếu import \*, refer tên variable mới để dùng content của file được import  VD: có 1 file string\_functions.js chứa các function liên quan đến string  import \* as stringFunctions from './string\_functions.js'  .  stringFunctions.method1…  - Đặc biệt: import 1 default export:   |  | | --- | | import newVarName from ‘./index.js’ //không có { } | |

## FUNCTION

|  |
| --- |
| - function là một khối code dùng để thực hiện 1 số tác vụ nào đó khi call function  - function khi nằm trong 1 object được gọi là method  - function khi dùng để design khung object: được gọi là Constructor |

### 2 cách tạo function: **declaration** & **expression**

|  |  |  |
| --- | --- | --- |
| - Có 2 cách dạo function:  1. Declare 1 global function bằng keyword function (**declaration**)   |  | | --- | | function functionName(a, b) { //a, b là *parameters*, giá trị thực nhận vào là *arguments*  code…;  } //không kết thúc với dấu ; vì function k phải là 1 statement |   Call ra: functionName(argument of a, argument of b, …)  Tự call: ( functionName(argument of a, argument of b, …) ) (); //bọc = () và thêm () ở sau  2. Tạo 1 biến, gán value = 1 function. Dùng biến đó như 1 function (**expression**)  function này gọi là **anonymous function,** call bằng: tên biến(arguments)   |  | | --- | | var x = function (a, b, …) { //a, b là *parameters*, giá trị thực nhận vào là arguments  code…;  }; //kết thúc với dấu ; vì function này là 1 phần của 1 statement (var x = …) |   Call ra: x(argument of a, argument of b, …)  Tự call: thêm () vào cuối expression a.k.a lúc declare var x |

### Truy xuất danh sách các arguments của function

|  |
| --- |
| - Trong các funtion, có một built-in object là arguments  - {arguments} là 1 array chứa các arguments được truyền vào khi call function  VD: myFunc(2, 3, 4) thì {arguments} = [2, 3, 4] |

### Arrow function

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - Dùng arrow cho những function chỉ để làm arguments, làm value cho biến, không re-use, không cần name  - Syntax:   |  |  | | --- | --- | | Normal: | var x = function(a, b) {return “hello” + a + b}; | | Arrow: | var x = (a, b) => {return “hello” + a + b}; |   -Syntax nếu chỉ có 1 statement return   |  |  | | --- | --- | | Arrow thường: | var x = (a, b) => { return “hello”+ a + b }; | | Arrow ngắn: | var x = (a, b) => “hello” + a + b; //bỏ {} và return | | Arrow ngắn  chỉ có 1 para: | var x = a => “hello” + a; //bỏ () {} và return |   - Nếu có | |
| - Syntax chung:  var x = () => {  return ‘…’;  }  - Syntax có parameter:  var x = (a) => {  return “Hi “ + a;  } | - Syntax cho function chỉ có 1 statement:  var x = () => ‘…;  - Syntax chỉ có 1 parameter:  var x = a => “Hi “ + a; |

### Return ra 1 object

|  |  |  |
| --- | --- | --- |
| - Syntax: gói vào **{}** và đặt **ngay sau return**   |  | | --- | | function(parameters) {  return {  firstName: “Harry”, lastName: “Potter”  }  } |   - Syntax arrow: gói vào **{}** rồi gói tất cả vào **()**   |  | | --- | | (parameters) => (  { firstName: “Harry”, lastName: “Potter” }  ) |   . |

### Rest parameter: …rest

|  |  |
| --- | --- |
| - Với function không rõ sẽ có bao nhiêu arguments, đặt dấu … **trước parameter cuối cùng** sẽ gói mọi arguments kể từ đó vào 1 array giả lập.  (Parameter cuối cùng đó gọi là *rest parameter*)  - *rest parameter* sẽ đượccoi như 1 array, có thể dùng các arrayMethod  - Syntax:   |  | | --- | | function myFunc(a, b, …*rest*) { code…} |   - Sau đó, khi ở bên trong function có thể refer *rest* như 1 array bình thường  VD: function myFunc(a, …rest) {  console.log(a);  console.log(rest);  }  myFunc(1, 2, 3, 4) // log ra a và log ra array rest: 1 [2, 3, 4] |

### Promise

|  |  |
| --- | --- |
| - Promise là 1 Constructor function nên dùng keyword new để khởi tạo  - Nhận 1 function làm argument. Function đó có 2 parameters: *resolve* và *reject*  - Syntax:   |  | | --- | | const myPromise = new Promise( (resolve, reject) => {  let x  if (x === true) {  resolve(‘Success!’)  } else {  reject(‘Error!’)  }  }  ); |   .  - 1 Promise có 3 trạng thái: pending, fulfilled, rejected  + pending: là khi declare  + fulfilled: là khi resolve  + rejected: là khi reject  - Promises được dùng khi có 1 process không biết cụ thể khi nào hoàn thành, VD như: khi tạo 1 server request sẽ mất một khoảng thời gian |

### Promise: fulfilled

|  |  |
| --- | --- |
| - Nếu promise đạt trạng thái **fulfilled**, dùng method **then()** để chạy **result**   |  | | --- | | myPromise.then( result => {do something with the result} ); |   - VD: myPromise đạt fulfilled  myPromise.then( result => { console.log(result) } ); //log ra argument của resolve: Success! |

### Promise: rejected

|  |  |
| --- | --- |
| - Nếu promise bị rejected, dùng method **catch()** để chạy **error**   |  | | --- | | myPromise.catch( error => {do something with the error} ); |   - VD: myPromise bị rejected  myPromise.catch( error => { console.log(error) } ); //log ra argument của reject: Error! |

## OBJECTS

|  |
| --- |
| - Objects là 1 dạng variables chứa nhiều hơn 1 value, bao gồm properties và methods  - Syntax:  var person = { //objectName  firstName: ”Harry”, //property Name1 : property value1  lastName:”Potter”, //property Name2 : property value2    fullName: function() { //methodName1(): 1 function  return this.firstName + “ “ + this.lastName; //this refers to owner của function  }  }  - Có 3 loại properties: own, prototype và constructor |

### Access object properties

|  |
| --- |
| - Cách 1: objectName.propertyName  - Cách 2: objectName[‘propertyName’] |

### Access object methods()

|  |
| --- |
| - Syntax:  objectName.methodName()  - VD: Trong 1 object person:  person = {  fullName: function () { //fullName là 1 property, fullName() là method  …  }  objectName.methodName()  - Note: dùng nhiều method móc nhau được vì method trước return this để method sau áp lên |

### Loop cho objects: loop for/in

|  |
| --- |
| - Xem tại phần loop |

### Thêm properties, methods vào Objects

|  |
| --- |
| - Thêm properties: dùng pseudo-access để tạo props, gán luôn key và value  VD: person.nationality = "English" //key là nationality : value là “English”  - Thêm methods: dùng pseudo-access để tạo methods, gán luôn function cần  VD: person.fullname = function() {  return this.fName + this.lName; //dù nằm ngoài object, this vẫn refers to method owner  }; |

### Xóa properties của object

|  |  |
| --- | --- |
| - Syntax:   |  | | --- | | delete objectName.propName | |

### Object methods: 3 cách thay đổi values của properties

|  |
| --- |
| - Có 3 cách để thay đổi values:  1. **Object.defineProperty**  2. Object Accessors: **Getter**  3. Object Accessors: **Setter** |

### Object methods: thay đổi value part 1. **Object.defineProperty**

|  |  |
| --- | --- |
| - Syntax:   |  | | --- | | **Object.defineProperty**( objectName, “propertyName”, {value : ‘new value’} ) |   VD:  var person = {  firstName: "Harry",  language : "EN"  }  **Object.defineProperty**(person, “language”, {value : “Vietnamese”}) //thay đổi value  - Cũng có thể thêm properties/methods bằng cách pseudo-change này:  **Object.defineProperty**( objectName, “lastName”, {value : “Potter”} ) //thêm property  **Object.defineProperty**( objectName, “fullName”,  { value : function() {return this.firstName + this.lastName } //thêm method  );  { writable : true/ false }  //viết được  { enumerable : true/ false }    //liệt kê được  { configurable : true/ false }   //cấu hình được |

### Object methods: thay đổi value part 2. Object Accessors: **Getter**

|  |
| --- |
| - Dùng getter để get, modify value của property, hoặc để tạo method.  - Sau đó có thể access getter giống như access propetry  - Getter KHÔNG thay đổi value gốc: access property đó sẽ vẫn trả ra value cũ,  chỉ khi access CHÍNH getter mới trả ra value mới  1. Tạo getter và nội dung get:  get getterName() { //tạo getter tên là getterName  return this.propName1; //nội dung get: return ra value của propName1  }  2. Access getter để return ra nội dung get:  accessMethod… objectName.getterName; //access getter sẽ return ra value của propName1  - Tạo method bằng getter hay function?  + Khi access function method: access như 1 function //vd: person.fullName()  + Khi access getter method: access như 1 property //vd: person.fullName |

### Object methods: thay đổi value part 3. Object Accessors: **Setter**

|  |
| --- |
| - Dùng setter để set (thay đổi) value của property:  - Setter CÓ thay đổi value gốc: access property đó sẽ trả ra kq mới sau khi set  1. Tạo setter và nội dung set  set setterName(newValue) { //tạo setter, có parameter là newValue  return this.propName1 = newValue; //khi gán value mới vào setter, sẽ gán luôn propName1  }  2. Gán value mới cụ thể cho setter, lúc này value của prop liên quan sẽ bị đổi  objectName.setterName = ‘real new value’  3. Access property liên quan sẽ return value mới  accessMethod… objectName.propName1; //return ‘real new value’  --> setter đã set value của propName1 thành newValue |

### Object methods: liệt kê toàn bộ các property names

|  |
| --- |
| - Syntax: **Object.getOwnPropertyNames**(objectName); |

### Object methods: liệt kê toàn bộ các property names được cho phép liệt kê

|  |
| --- |
| - Các properties được liệt kê là có {enumerable : true}, vốn là default  - Syntax: **Object.keys**(objectName); |

### Object methods: Access properties, methods với method get

|  |  |
| --- | --- |
| - Syntax:  + Setup get:  get tênHàm() { *getter*  return … ;  }  + Access get:  objectName.tênHàm  //Access tênHàm như access 1 property | - VD:  + Setup set:  get fullName() {  return this.firstName + this.lastName;  }  + Access get:  objectName.fullName  //Access tênHàm như access 1 property, không có () |

### Object methods: Thay đổi value với method set (sang 1 value mới bên ngoài object)

|  |  |
| --- | --- |
| - Syntax:  + Setup get:  set tênHàm(value) {  this.nameNeedChange = value;  }  + Bên ngoài object: setup value mới  objectName.tênHàm = “value mới”  + Kết quả:  value của nameNeedChange đã đổi | - VD: person = {firstName = “Harry” …}  + Setup get:  set newFirstName(value) {  this.firstName = value;  }  + Bên ngoài object: setup value mới  person.newFirstName = “James”  + Kết quả:  person = {firstName = “James” …} |

### Object methods: Khóa 1 object với method Object.freeze ()

|  |  |
| --- | --- |
| - Dùng để protect/khóa data 1 object, không thể sửa/thêm/xóa nội dung   |  | | --- | | Object.freeze(tên-object) | |

### Hiển thị 1 object

|  |
| --- |
| - 3 cách hiển thị các values:  1. Hiển thị các values của từng properties như các ‘string’ nối nhau:  person.firstName + “, ” + person.lastName + “, ” + … ;  2. Dùng loop for/in để loop ra các values của từng properties:  var x;  for (x in person) { //loop start: *gán x thành từng propName* của person.  document.write(person[x] + “,”)  };  3. Convert các value of properties của 1 object thành array và hiển thị array đó:  var person = {…}  var myArray = Object.values(person);  document.write(myArray);  - Hiển thị toàn bộ object ngoại trừ các methods (chỉ các key:value)  1. Dùng JSON.stringify()  var person = {…}  var myJsonString = JSON.stringify(person);  document.write(myJsonString);  \* JSON.stringify() sẽ:  + convert date thành ‘string’  + không convert function nên sẽ không convert method.  Fix: objectName.methodName.toString() để convert method thành ‘string’ trước  + có thể dùng cho cả [array] |

### Tạo class chứa Constructor

|  |  |
| --- | --- |
| - Syntax tạo 1 Constructor a.k.a 1 cái khung để setup các objects:  class ConstructorName {  constructor (a, b, …) { //nên đặt tên parameters tương hợp, dễ hiểu với propName  this.propName1 = a, //argument của para a sẽ là value cho propName1 của this (obj được tạo)  this.propName2 = b, //argument của para b sẽ là value cho propName2 của this (obj được tạo)  this.method1 = function() { //method sẽ không có tham số đại diện như a, b. Vẫn access được  }  }  -> đã tạo được 1 “khung object”: một function ConstructorName(a, b, …)  - Call class ConstructorName :   |  | | --- | | const x = new ConstructorName(arguments) |   VD:  class Person { //tạo class Person  constructor(first, last, age) { //setup 3 parameters: first, last, age cho class Person  this.firstName = first; //argument của first sẽ thành value cho prop firstName của this (obj được tạo)  this.lastName = last; //argument của last sẽ thành value cho prop lastName của this (obj được tạo)  this.age = age; //argument của age sẽ thành value cho prop age của this (obj được tạo)  } //\* this trong class là 1 phần khung, sẽ refer object được khởi tạo từ class (vd: playerOne.age)  const myDad = new Person(‘Tho’, ‘Nguyen’, 58); //declare 1 obj bằng cách call class Person  displayMethod myDad // { firstName: ‘Tho’, lastName: ‘Nguyen’, age: 58 } |

### Chỉnh sửa Constructor:

|  |
| --- |
| - Không thể chỉnh sửa, thêm bớt properties/methods từ bên ngoài Constructor  VD: ConstructorName.propName3: tham số c //sai  - Có 2 cách:  1. Chỉnh sửa trực tiếp trong Constructor  2. Dùng prototype:  VD:  ConstructorName.prototype.propName3: tham số c;  ConstructorName.prototype.fullName = function() {  return this.firstName + … ;  }; |

### Tạo 1 method dùng được cho nhiều objects:

|  |
| --- |
| - Tạo 1 object khuôn mẫu, tạo method trong đó (dùng this)  - Call method đó với object khác (a.k.a properties khác) bằng syntax:  objectOrigin.methodName.call(objectName)  - VD:  var person = { //tạo object gốc  fullName: function(a, b) { //tạo method như bthg, có thể có parameters a, b. Nên đặt a b dễ hiểu  return this.firstName + " " + this.lastName + “ “ + a + “ “ + b;  }  }  var person1 = { firstName:"Harry", lastName: "Potter"}  alert( person. fullName.call(person1, “London”, “England”) ); //call method fullName  --> Harry Potter London England //ko cần declare trong person1 vẫn có a b vì declare khi call() |

### Keyword: this

|  |
| --- |
| - This trong 1 method thường: refers đến đối tượng gọi hàm  - This trong 1 method thường viết với arrow function: refers đến đối tượng bên ngoài (tìm this gần nhất)  - This khi method được tự gọi, không qua đối tượng: globalThis  +Normal: window (cả normal/arrow)  +Strict: undefined (normal) / window (arrow)  - VD:  var person = {  firstName: "Harry",  lastName : "Potter",  fullName : function() {  return this.firstName + " " + this.lastName; //this này = person (method owner)  }  };  - Dùng trong 1 function độc lập: refers to global owner  - Dùng trong 1 function độc lập ‘strict mode’: undefined  - Dùng trong event handlers: refers to HTML element nhận event: |

### Events

|  |  |
| --- | --- |
| - Một page load xong  - User nhập input  - User nhấn nút  - Hover  - Hover out | onload  onchange  onclick  onmouseover  onmouseout |
| - Tạo function() chứa {JS code} và để event call function() đó | |

### E6: Destructuring assignment

|  |  |  |  |
| --- | --- | --- | --- |
| - Unpack properties của objects (hoặc values của arrays) ra làm các variables:  - Có thể đặt tên variables trong lúc unpacking  VD: let objectName = { name: Harry, age: 30 }  +Syntax cũ (dot notation từng variables):   |  | | --- | | let hisName = objectName.name; //Harry  let hisAge = objectName.age; //30 |   + Syntax mới (destructuring assignment một loạt):   |  | | --- | | let { name: hisName, age: hisAge } = objectName;  //ra 2 variable hisName = Harry và hisAge = 30 |   - Nếu muốn extract 1 objectSmall nested trong 1 objectBig (hoặc 1 array):   |  | | --- | | let { objectSmall: {name: yourVarName, age: yourVarAge} } = objectBig; | |

### E6: Dùng D.A để extract properties 1 object làm parameters

|  |  |
| --- | --- |
| - Có thể dùng 1 object làm argument, bằng cách set parameters là properties của object đó. Các parameters này được extract ra từ object bằng D.A  - Syntax:   |  | | --- | | let x = ( {propName1, propName2, …} ) => {  //sử dụng các propName  } |   - VD:  let person1 = { //object  firstName = “James”,  lastName = “Bond”  }  let x = ( {firstName, lastName} ) => { //tạo biến x là function có 2 parameters là 2 prop  firstName + “ “ + lastName  }  displayMethod x(person1) //function x sẽ truy xuất 2 prop tương ứng của object person1 |

## STRINGS

|  |
| --- |
| - “Strings” thường là values đơn giản như  VD: var firstName = “Harry”  nhưng cũng có thể là objects với Keyword **new** |

### Strings method: tìm vị trí của 1 “string” trong 1 “string”

|  |
| --- |
| - Tìm vị trí của 1 “string” trong 1 “string” : indexOf() / lastIndexOf //đối tượng đầu / cuối  VD: var x = “ABCDEF” //Tạo variable x với value là 1 “strings” “ABCDEF”  var y = x.indexOf(“B”); //Tạo variable y với value là vị trí của “B” trong value của x  docu… = y; //Tạo variable y với value là vị trí của “B” trong value của x  hoặc docu… = x.indexOf(“B”) //Trực tiếp không cần tạo y  - Tìm vị trí của 1 “string” trong 1 “string” : search()  nt |

### Strings methods: cắt 1 đoạn “string”

|  |  |  |
| --- | --- | --- |
| 1. Cắt 1 đoạn của string: từ điểm a đến trước điểm b   |  | | --- | | slice(start, end)  //dùng index 0, cắt đến end nhưng k tính end, nếu bỏ end sẽ cắt đến hết string |   2. Cắt 1 đoạn của string: từ a cho đến x điểm (tính từ a)   |  | | --- | | substr(start, length)  //dùng index 0, cắt từ điểm start và cắt đi length điểm | |

### Strings method: replace()

|  |  |
| --- | --- |
| - Sẽ thay thế 1 đoạn (1 vài giá trị) của 1 string bằng 1 đoạn mới  - Nếu pattern lặp lại nhiều lần trong string, chỉ replace pattern đầu tiên  - Sẽ trả ra 1 string mới có content đã replace, không change string gốc   |  | | --- | | string.replace(searchPattern, newValue) |   - Thay thế 1st đoạn “abc” của “stringA” với 1 đoạn mới “xyz” bằng “stringB” mới  1. Tạo function abc() {} //để call trong onclick button  2. Tạo variable x, gán value = content của #id-output //content này là “stringA”  3. Tạo variable y, gán value = x.replace(“Old content”, “New content”); //Tạo “stringB”  4. Thay value content của #id-output thành y  VD:  <tag id=”id-output”>This is Old content</tag>  <script>  function abc() {  var x = document.getElementById(“id-output”).innerHTML;  var y = x.repalce(“Old contents”, “New contents”);  document.getElementById(“id-output”).innerHTML = y;  }  </script>  - Thay tất cả các đoạn “abc” với 1 đoạn mới “xyz” bằng “stringB” mới /global  var y = x.replace(/Old contents/g, “New contents”); |

### Method charAt(): Truy cập vị trí cụ thể trong string

|  |  |
| --- | --- |
| - Sử dụng hệ thống index[0] - index[.length - 1]  - Return character ở index cần tìm   |  | | --- | | str.charAt(index) | |

### String thành [array]

|  |
| --- |
| 1. Tạo variable x = “string”, ngăn cách trong “string” bằng space hoặc , hoặc |  2. Tạo variable y = x.split(“ ” hoặc “,” hoặc “|”) |

### E6: Tạo strings bằng Template Literals

|  |
| --- |
| -  VD: log ra các values của 1 array dưới dạng các </li> bằng Template Literals  const colors = [‘red’, ‘blue’, ‘green’];  function myFunc(arrayName) {  const kq = [ ];  for (i = 0; i < arrayName.length; i++) {  kq.push( ` <li> ${ arrayName[i] } </li> ` );  }  return kq;  } |

## NUMBERS

|  |
| --- |
| - Number chỉ có 1 loại: 64 bit floating point  - Adding và concatenation:  “String” + Number1 + Number2 = “StringNumber1Number2”  Number1 + Number 2 + “String” = sum(Number1, Number 2)”String”  -> Các số sẽ +- bình thường cho đến khi có “String”  - Nếu biểu thức chỉ chứa “Number” (Nubmer dạng “String”) thì kq sẽ là Number  - Xác định xem có phải là NaN không (không phải Number đúng không)  var x = 100 / “apple”  docu…isNaN(x) //kq sẽ là true -> x là NaN, không phải là số |

### Numbers method: Chuyển ra strings / number - toString() / valueOf()

|  |
| --- |
| - toString(): return từ số, biểu thức hoặc biến để cho ra 1 “string”  var x = 123;  docu… = x.toString(); //return “123” từ variable x  docu… = (123).toString(); //return “123” từ số 123  docu… = (100 + 23).toString(); //return “123” từ biểu thức 100 + 23  - valueOf(): return từ số, biểu thức hoặc biến để cho ra 1 number  var x = 123;  docu… = x. valueOf(); //return 123 từ variable x  docu… = (123). valueOf(); //return 123 từ số 123  docu… = (100 + 23). valueOf(); //return 123 từ biểu thức 100 + 23 |

### Numbers method: toFixed() và toPrecision()

|  |
| --- |
| - toFixed() sẽ return ra 1 “string” là số được chọn làm tròn còn…chữ số  var x = 3.1415;  docu… = x.toFixed(0); //return “3” vì không có decimal  docu… = x.toFixed(1); //return “3.1” vì có 1 decimal  docu… = x.toFixed(2); //return “3.14” vì có 2 decimals  - toPrecision() sẽ return ra 1 “string” là số dài…chữ số  var x = 3.1415;  docu… = x. toPrecision (); //return “3.1415” vì default  docu… = x. toPrecision (2); //return “3.1” vì length = 2  docu… = x. toPrecision (4); //return “3.141” vì length = 4 |

### Global method:

|  |
| --- |
| - Dùng được cho mọi loại data  1. Number() method dùng để chuyển hóa 1 variable thành 1 number  docu… = Number(tênVariable)  VD: docu… = Number(“ 10 “) //return number 10  docu… = Number(“10, 33, John “) //return NaN  2. parseInt() method dùng để check 1 “string” và return number đầu tiên check được  docu… = parseInt(“string”)  VD: docu… = Number(“ 10.1 20 years “) //return number 10  docu… = Number(“years 10 “) //return NaN vì 10 đứng sau “string” -> “string” hóa  3. parseFloat() method dùng để check 1 “string” và return nguyên 1 number đầu tiên  docu… = parseFloat(“string”)  VD: docu… = Number(“ 10.1 20 years “) //return number 10.1  docu… = Number(“years 10 “) //return NaN vì 10 đứng sau “string” -> “string” hóa |

### Number properties: MIN\_VALUE và MAX\_VALUE

|  |
| --- |
| 1. MIN\_VALUE return số bé nhất  var x = Number.MIN\_VALUE  docu… = x; //return kq  2. MAX\_VALUE return số bé nhất  var x = Number. MAX\_VALUE  docu… = x; //return kq |

### Phân biệt số thập phân và số nguyên

|  |
| --- |
| - Số nguyên chia cho chính nó sẽ dư 0  VD: 5 % 5 //sẽ bằng 0 (5/5 = 1 dư 0)  - Số thập phân khi làm tròn lên/xuống rồi chia cho chính nó sẽ dư khác 0  VD: 5.5 % Math.floor(5.5) //sẽ bằng 0.5 (5.5/5 = 1 dư 0.5) |

## ARRAYS

### Tổng quan

|  |
| --- |
| - Là một cấu trúc dữ liệu,  - Là 1 dạng object có cấu trúc: [ index: “value], index hệ 0 |

### 

### Array property: độ dài của 1 array: **.length**

|  |
| --- |
| - Là thuộc tính được update mỗi khi thay đổi phần tử  - Không phải là độ dài, mà là index cao nhất + 1  - Có thể thay đổi giá trị (khác string), sẽ update array theo thay đổi  - Update length cao hơn: sẽ sinh ra các empty, giá trị là undefined  - Update = 0 : xóa mảng  - For in sẽ loop cả những prop ngầm như length… nên không dùng  z- Property length return số lượng element của [array]  var fruits = [“Banana”, “Orange”, “Apple”, “Mango”]  docu… = fruits.length //return kq là 4 |

### Array method: Có phải là array không? **Array.isArray()**

|  |  |
| --- | --- |
| - Sẽ return true nếu đúng là array, false nếu không phải là array   |  | | --- | | Array.isArray( testObjectName hoặc [chuỗicần test] ) | |

### Array method: chuyển thành strings bằng toString() hoặc join()

|  |  |
| --- | --- |
| - Chuyển 1 array thành 1 “string” chứa các value   |  | | --- | | arrayName.toString();  hoặc  arrayName.join(“cách ngăn cách”); //defaut chỉ có dấu ‘,’ liền nhau | |

### Array mothod: thêm/bớt item vào đầu-cuối array:

|  |  |  |
| --- | --- | --- |
| - Thêm item vào đầu/cuối array: .unshift(“new”) / .push(“new”)   |  | | --- | | arrayName.unshift(new); //thêm item new vào đầu array  arrayName.push(new); //thêm item new vào cuối array |   - Bớt item cuối/đầu của array: .shift() / .pop()   |  | | --- | | arrayName.shift(); //bớt item đầu tiên của array  arrayName.pop(); //bớt item cuối cùng của array |   …. |

### Array method: xóa/thay thế values của array

|  |  |  |
| --- | --- | --- |
| - Có thể thêm/bớt items vào array với .splice()   |  | | --- | | arrayName.splice(a, b, c, d…);  //a là index bắt đầu - required  //b là số index sẽ cắt ra (nếu không cắt thì = 0)  //c, d, … là nội dung thay thế vào (nếu không thêm thì omit) |   .  - splice() sẽ return **1 array mới** (chứa phần cắt + thêm nếu có, nếu ko có thì = [])  - splice() **có modify** array gốc, nếu định modify thì truy cập lại array gốc  \* Khác biệt vs slice:  + slice() **không modify** array gốc |splice() **có modify** array gốc  + slice sẽ chỉ **extract** ra 1 đoạn | splice() sẽ **cắt** ra 1 đoạn/thêm đoạn  VD:let arr = [‘one’, ‘two’, ‘three’]  arr.splice(1, 2, ‘four’, ‘five’, ‘six’] //start tại [1], cắt 2 index ([1][2]), thay = ‘four’, ‘five’, ‘six’  arr: [‘one’, ‘four’, ‘five’, ‘six’]  - Có thể xóa values bằng delete   |  | | --- | | delete arrayName[index]  //sau đó dùng filter(Booleans) để lọc lỗ | |

### Array method: extract 1 đoạn với slice()

|  |  |
| --- | --- |
| - Có thể extract 1 đoạn trong array với .slice()   |  | | --- | | arrayName.slice(a, b);  //a là **index** bắt đầu extract, b là **index** kết thúc extract (không tính b) |   .  - slice() sẽ **không** modify array gốc và tạo 1 array mới chứa phần được extract  - Có thể dùng chay slice() để copy 1 array: arr.slice()  \* Khác biệt vs splice:  + slice() **không modify** array gốc, splice() **có modify** array gốc  + slice sẽ chỉ **extract** ra 1 đoạn, splice() sẽ **cắt** ra 1 đoạn/thêm đoạn  VD: let baseArray = [‘one’, ‘two’, ‘three’, ‘four’]  let sliceArray = arr.slice(1, 3)  -> baseArray sẽ giữ nguyên là [‘one’, ‘two’, ‘three’]  -> sliceArray sẽ là [‘two’, ‘three’] |

### Array method: Change item

|  |  |
| --- | --- |
| - Thay đổi 1 item bằng cách đăng ký lại, thông qua index của item đó   |  | | --- | | arrayName[index] = “new item”; |   . |

### Array method: Sát nhập các thứ vào 1 array: concat()

|  |  |
| --- | --- |
| - Có thể sát nhập thêm numbers, strings hoặc các arrays khác vào 1 array:   |  | | --- | | arrayName1.concat( arrayName2, arrayName3, ‘string1’, number1… ) |   - Sẽ tạo ra 1 array mới chứa tổng các elements, không đổi các array cũ |

### Array method: Sắp xếp primitive value elements

|  |
| --- |
| - Method sort() sẽ nhận vào 1 compare function(a, b)  - Compare function sẽ so sánh từng value theo cặp a, b và return ra 1 số  + Nếu return kq < 0 thì a sẽ đứng trước b  + Nếu return kq = 0 thì a b không đổi vị trí  + Nếu return kq > 0 thì b sẽ đứng trước a  - sort() **có modify** array, nên copy bằng slice trước  1. Sắp xếp “strings” theo alphabet:  arrayName.sort( function(a, b) {  return ( (a === b) ? 0 : ( (a < b) ? -1 : 1) ) //nếu a = b thì….  }  );  2. Đảo ngược kết quả sắp xếp:  ( (a === b) ? 0 : ( (a < b) ? 1 : -1) )  3. Sắp xếp number theo số thứ tự:  arrayName.sort( function(a, b) {  return (a - b) // ? negative : 0 : positive  }  );  - Sau khi sort, có thể suy ra value min/max của 1 [array]:  + index[0] sẽ là min value  + index[arrayName.length - 1] sẽ là max value |

### Array method: Sắp xếp objects elements

|  |
| --- |
| - Sắp xếp theo thứ tự của một object element có value là số  var cars = [  {type:"Volvo", year:2016},  {type:"Saab", year:2001},  {type:"BMW", year:2010}  ];  1. Hiển thị [array] gốc = function  a. Tạo function display() chứa cách hiển thị [array] gốc  function display() {  docu… = //vị trí hiển thị  cars[0].type + “ “ + cars[0].year //Hiển thị type 0 type 0  + “<br>” + //Xuống dòng  cars[1].type + “ “ + cars[1].year //Hiển thị type 1 type 1  +…;  }  b. Hiển thị [array] gốc với function hiển thị trên  display() //hiển thị function display() a.k.a vật chứa cách hiển thị [array] gốc  2. Tạo onlick function sort() chứa function compare (a, b) và hiển thị [array] gốc v2  function myFunc() { //tạo myFunc để cho vào event  cars.sort(function(a, b) {return a.year - b.year}); //tạo sort theo function (a, b)  display(); //hiển thị lại [array] gốc v2.0  } |

### Array method: indexOf() tìm index của 1 value

|  |  |  |
| --- | --- | --- |
| - Method indexOf() sẽ start từ đầu or 1 value cụ thể cho đến hết [array]  - Return -1 nếu không có (có thể dùng để check value có thuộc array không)  - Nếu xuất hiện nhiều lần thì chỉ trả kq vị trí đầu tiên tìm được  - Syntax:   |  | | --- | | arrayName.indexOf(“value”, điểm start nếu cần) |   - Syntax tìm từ cuối lên: Tạo var chứa kq và hiện thị var đó   |  | | --- | | arrayName.lastIndexOf(“value”, điểm start nếu cần) |   . |

### Array method: reverse() đảo ngược thứ tự item

|  |  |
| --- | --- |
| - reverse() sẽ đảo ngược thứ tự các item  - reverse() CÓ MODIFY  - Syntax:   |  | | --- | | arrayName.reverse() | |

### Array method: trả ra kq là value của element đầu tiên tìm thấy đủ trình

|  |
| --- |
| 1. Tạo function theo value và return ra bài test  function findTest(value) {  return value > or < or …  }  2. Tạo var chứa kq thu được từ find() method và function  var kq = arrayName.find(findTest); //trả ra value của element đầu tiên đủ trình  3. Hiển thị kq |

### Array method: trả kq là index# của element đủ trình đầu tiên tìm được

|  |
| --- |
| 1. Tạo function theo value và return ra bài test  function findIndexTest(value) {  return value > or < or …  }  2. Tạo var chứa kq thu được từ findIndex() method và function  var kq = arrayName.findIndex(findIndexTest); //trả ra index của element đầu tiên đủ trình  3. Hiển thị kq |

### Array method: tóm gọn các items về một giá trị: **reduce()**

|  |  |  |  |
| --- | --- | --- | --- |
| - Method **reduce()** dùng để tóm gọn các items của 1 array về 1 giá trị duy nhất thông qua việc chạy 1 *reducer function* lên từng items  - Value này có thể là tổng/hiệu/tích/thương của các items ban đầu  - Phép tính được thực hiện theo chiều trái -> phải //dùng **reduceRight()** để ngược lại  - Syntax cơ bản:   |  | | --- | | arrayName.reduce( *reducerFunc, initialValue* ) |   - Syntax cụ thể:   |  | | --- | | arrayName.reduce( *reducerFunc*(*accumulator*, *currentValue*), *initialValue* ) |   - Cấu hình cho *reducer function*:  + *reducer function* có 4 parameters: (*accumulator*, *currentValue, index, array)*  + Nếu có declare *initialValue*:  *accumulator = initialValue*  *currentValue = value đầu tiên* của arrayName  reduce() sẽ chạy từ index 0 của arrayName  + Nếu không declare *initialValue*:  *accumulator = value đầu tiên* của arrayName  *currentValue = value thứ hai* của arrayName  reduce() sẽ chạy từ index 1 của arrayName  + Áp dụng arrow function cho *reducer function:*  (*acc*, *cur, index, array)* => acc + - \* / cur  - Syntax cuối cùng:   |  | | --- | | arrayName.reduce( *acc* +-\*/ *cur, initialValue* ) |   VD: làm tròn xuống và tính tổng các value sau: 5.4, 10.5, 15  Cách 1: Tạo 1 array chứa tập số, gán array đó vào 1 biến và dùng reduce()  let arr = [5.1, 10.5, 15]  arr.reduce( (a, b) => Math.floor(a) + Math.floor(b) );  Cách 2: Tạo 1 function có rest parameters, truyền tập số làm arguments và call nó  let sum = (...args) => { //…args chỉ dùng trong function parameters  args.reduce( (a, b) => Math.floor(a) + Math.floor(b) );  } |

### Array iteration method: Gọi ra từng value của từng element (kq là ‘string’)

|  |
| --- |
| 1. Tạo 1 var chứa kết quả gốc:  var ketqua = ““; //kết quả gốc này sẽ rỗng  2. Tạo function gọi value  function valueCallingFunc(value) { //có thể gọi thêm index / array  ketqua = ketqua + value + “<br>”; //ketqua gốc rỗng nên value sẽ là kq chính  }  3. Áp dụng function gọi value trên cho mỗi element  arrayName.forEach(valueCallingFunc); //Chạy valueCallingFunc cho từng element của arrayName  4. Hiển thị kết quả |

### Array iteration method: .map() : áp 1 function lên từng values để tạo array mới

|  |  |  |
| --- | --- | --- |
| - .map() sẽ call 1 function lên từng values của 1 array gốc  - .map() sẽ tạo ra 1 array mới, ko change array gốc  - Syntax 1:   |  | | --- | | arrayName.map( function(val) { code cho từng values } ) |   - Syntax 2: khai báo riêng 1 named function rồi pass vào (only tên function):   |  | | --- | | function myFunc(val) { code cho từng values }  arrayName.map(myFunc) |   . |

### Array iteration method: lọc những element đủ trình

|  |
| --- |
| 1. Tạo function theo value, return ra các value đủ điều kiện nào đó  function theFilterTest(value) {  return value > or < or = or >= or <= …;  }  2. Tạo var chứa kết quả với iteration method .filter() // là [array] mới chứa value mới  var ketqua = arrayName.filter(theFilterTest); được return từ theFilterTest()  3. Hiển thị kết quả |

### Array iteration method: reduce các values về 1 value

|  |  |
| --- | --- |
| - Dùng reduce() để call 1 function lên từng values từ trái -> phải, tóm gọn các values về 1 value  - reduce() **không thay đổi** array gốc   |  | | --- | | arrayName.reduce( function(accumulator, currentValue), initialValue )  //accumulator: nơi chứa result, sẽ được update mỗi khi reduce chạy sang value mới  //currentValue: đại diện cho các value  //initialValue: giá trị khởi đầu cho accumulator, nếu omit thì accumulator = null |   - Tiến trình: reduce sẽ khởi đầu với accumulator = null (hoặc = initialValue), sau đó tùy nội dung function mà accumulator sẽ tương tác với currentValue. Cứ thế accumulator sẽ được update giá trị mới và tiếp tục với currentValue tiếp theo.  VD:  let const users = [  { name: 'Harry', age: 34 },  { name: 'Ron', age: 20 },  { name: 'camperCat', age: 10 }  ]; |

### Array iteration method: Test xem mọi value có đủ trình không

|  |
| --- |
| 1. Tạo function theo value để return ra bài test  function everyTest(value) {  return value … //nội dung bài test  }  2. Tạo var chứa kết quả với iteration method .every() //là một boolean  var ketqua = arrayName.every(everyTest);  3. Hiển thị kết quả |

### Array iteration method: Test xem có value nào đủ trình không

|  |
| --- |
| 1. Tạo function theo value để return ra bài test  function someTest(value) {  return value … //nội dung bài test  }  2. Tạo var chứa kết quả với iteration method .some() //là một boolean  var ketqua = arrayName.some(someTest);  3. Hiển thị kết quả |

### Dates: Tạo ra date objects

|  |
| --- |
| - Hiển thị thời gian: new date() //sẽ tự động thành 1 ‘string’  - Có 3 cách tạo Date objects:  1. Thời điểm hiện tại (static, không đổi sau đó)  var x = new Date()  2. Thời điểm chỉ định bằng 6 đơn vị:  var x = new Date(year, month(0-11), day, hour, minute, second);  vd var x = new Date(2020, 11, 24, 23, 59, 59);  -> Thu Dec 24 2020 23:59:59 GMT+0700 (Indochina Time)  3. Thời điểm chỉ định bằng 1 ‘string’  var x = new Date("June 27, 2020 14:53:00") |

### Dates: Input format

|  |
| --- |
| - ISO Date:  1. "year-month-day" //relative theo local timezone  vd: var x = new Date(“2020-06-27”)  2. “year-month-dayThours-minutes-secondsZ”  vd: var d = new Date("2020-06-27T15:05:00Z"); 3. “month-day-year” //short dates  vd: var d = new Date("06/27/2020"); |

### Dates GET methods

|  |
| --- |
| 1. |

### Math object: Làm tròn (lên xuống) 1 value

|  |
| --- |
| - Syntax: Math.round(value) //làm tròn  vd: Math.round(4.5) //sẽ làm tròn lên = 5  - Syntax: Math.ceil(value) //làm tròn lên  vd: Math.ceil(4.5) //sẽ làm tròn lên = 5  - Syntax: Math.floor(value) //làm tròn xuống  vd: Math.floor(4.5) //sẽ làm tròn xuống = 4 |

### Math object: Lùy thừa (Nhân mũ) 1 value

|  |
| --- |
| - Syntax: Math.pow(value, power)  vd: Math.pow (5, 2) //= 52 = 25 |

### Math object: Căn bậc hai của 1 value

|  |
| --- |
| - Syntax: Math.sqrt(value)  vd: Math.sqrt(25) //= √25 = 5 |

### Math object: giá trị positive của 1 value

|  |
| --- |
| - Syntax: Math.abs(-value)  vd: Math.abs(-25) //= 25 |

### Math object: tìm min/max trong số nhiều value

|  |
| --- |
| - Syntax: Math.min(value1, value2, …)  Math.max(value1, value2, …)  vd: Math.min(1, -20, 30) //= -20 |

### Math object: random 1 value

|  |
| --- |
| - Syntax: Math.random()  vd: Math.random() // 0.0 < ketqua < 1.0  - Syntax: dùng function  1. Tạo function  function getRndInteger(min, max) {  return Math.floor(Math.random() \* (max - min + 1) ) + min;  }  2. Hiển thị trong button |

### Booleans

|  |
| --- |
| - Boolean chỉ có 2 kq: true hoặc false  - Boolean function:  (tình huống) //sẽ return ra true/false |

### Comparison operators

|  |  |
| --- | --- |
| == | equal to |
| === | equal value and equal type |
| != | not equal to |
| !== | not equal value and not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |

### Logical operators

|  |  |  |
| --- | --- | --- |
| && | and | x = 1, y = 2 : (x > 0 && y <3) //true nếu cả 2 đều đúng, false nếu 1 trong 2 sai |
| || | or | x = 1, y = 2 : (x > 1 && y <3) //true nếu 1 trong 2 đúng, false nếu cả 2 sai |
| ! | not | x = 1, y = 2 : !(x == y) //true nếu phủ nhận này đúng, false nếu phủ nhận này sai |

### E6: Spead operator …arrayName

|  |  |
| --- | --- |
| - Dùng spread operator lên 1 array sẽ return ra array đó phiên bản unpacked a.k.a chỉ return ra các value của array, không có dấu [] và dấu phẩy  - Syntax   |  | | --- | | …arrayName |   - Dùng để copy values của 1 array cho:  + Làm arguments cho function vd: myFunc(…arrayName)  + Nest 1 array vào array khác (tại bất kỳ index nào) vd: [a, b, …arrayName, c]  VD:  let arrayName = [1, 2, 3, 4]  console.log(…arrayName) //return 1 2 3 4 |

### E6: Destructuring Assignment

|  |  |
| --- | --- |
| - Nếu như Spead operator …arr sẽ unpack toàn bộ array thì Destructuring Assignment sẽ cho phép chọn value để lấy ra  - Syntax   |  | | --- | | let a, b, c  [a, b, , , c] = [1, 2, 3, 4, 5, 6]  //lấy ra a = 1, b = 2, c = 5 |   - Dùng dấu phẩy , để bỏ qua |

## Conditional operators

|  |
| --- |
| - Nếu một variable có value đạt/không đạt điều kiện nào đó, sẽ có phản ứng  - Syntax:  varriableName = (điều kiện) ? value1:value2  vd: var ageCondition = (age < 18) ? “Young”:”Old” |

### Conditions

|  |
| --- |
| - Conditional statements: thực hiện các hành động khác nhau tùy tình huống khác nhau  1. if sẽ chỉ ra 1 đoạn code sẽ được thực hiện nếu 1 điều kiện nào đó là true  2. else sẽ chỉ ra 1 đoạn code sẽ được thực hiện nếu điều kiện trên là false  3. else if để chỉ ra một điều kiện mới để test, nếu điều kiện gốc đã false  4. switch để chi ra các đoạn code dự phòng khác để thực hiện |

### Conditional statement: **IF - ELSE**

|  |  |
| --- | --- |
| - Statement if chỉ ra 1 đoạn code được thực hiện nếu 1 điều kiện nào đó là true  - Statement else chỉ ra 1 đoạn code được thực hiện nếu điều kiện trên là false  - Có thể chèn keyword break vào code block nếu muốn ngắt. VD như check nếu true thì break, không hành động gì rồi dùng else | |
| - Syntax:    if (điều kiện) {  code cần thực hiện  } else {  code cần thực hiện  } | -VD: var time = new Date().getHours() //lấy giờ của now  var ketqua; //nhớ sau đó hiển thị docu  if (time < 18) {  ketqua = “Good day”  } else {  ketqua = “Good evening”  } |

### Conditional statement: **ELSE - IF**

|  |  |
| --- | --- |
| - Statement else if chỉ ra một điều kiện 2 để test, nếu điều kiện 1 đã false | |
| - Syntax:    if (điều kiện 1) {  code nếu điều kiện 1 true //text, </Element>…  } else if (điều kiện 2) {  code nếu điều kiện 1 false, điều kiện 2 true  } else {  code nếu điều kiện 1 và 2 đều false  } | -VD: var timeNow = new Date().getHours()  var ketqua; //nhớ sau đó hiển thị docu  if (timeNow < 12) {  ketqua = “Good morning”  } else if (timeNow < 18) {  ketqua = “Good afternoon”  } else {  ketqua =”Good evening”  } |

### Conditional statement: **SWITCH**

|  |  |
| --- | --- |
| - Statement switch để chọn một (trong nhiều đoạn code) đạt điều kiện  - Dùng khi có quá nhiều trường hợp (if else sẽ rối)  - (expression) là giá trị cần test  - Thường đây sẽ là 1 tác vụ của 1 function. Function sẽ có parameter là x, và switch dùng để test x đó: switch(x), case…  - Case có thể là number hoặc ‘string’  - Sau mỗi case cần có break để chốt kq nếu 1 case nào đó đúng, nếu k sẽ chạy mãi. | |
| - Syntax:  switch(expression) {  case value1:  code cho case 1;  break;  case value2:  case value3:  code cho case 2, 3;  break;  default: //nếu k case nào đạt  code cho default;  break;  } | - VD: var ketqua;  switch(newDate().getDay()) { //sẽ return kq từ 0 đến 6  case 1:  ketqua = “Monday”  break;  case 6:  case 0:  ketqua = “Weekend”  break;  default:  ketqua = “None”  break;  } |
| - Để 1 case đạt, cần thỏa mãn (===)  - Nếu nhiều case cùng đạt, sẽ chọn case đầu tiên  - Không case nào đạt, sẽ chọn case default | |

## Loop

|  |
| --- |
| - Loop for có 3 kiểu: for, for/in và for/of  - Loop while có 2 kiểu: while và do/while |

### **LOOP FOR**

|  |
| --- |
| - Loops for sẽ chạy một đoạn code nhiều lần |
| - Syntax:  for (*intitial* *expression*; *condition true*; *increment expression*) {  looping  }  //*intitial* *expression* sẽ được chạy 1 lần trước khi bắt đầu loop  //*condition true* chỉ ra điều kiện, nếu còn true thì còn loop  //*increment expression* sẽ được chạy sau mỗi lần loop  -VD: var ketqua = “”;  var i;  for (i = 0 ; i < 5; i++) {  ketqua += “Đáp án là “ + i //+ “<br>”;  }  //*intitial* *expression*: i = 0 sẽ gán giá trị variable i = 0 trước khi bắt đầu loop  (có thể omit nếu declare luôn var i trước for loop)  //*condition true*: i < 5 sẽ đặt điều kiện là variable i < 5 chừng nào true thì còn loop  (có thể omit nếu kết quả vẫn ra true/false nhưng trong loop phải có break)  // *increment expression*: i++ sẽ tăng giá trị của i lên 1 sau mỗi lần loop  (có thể omit nếu giá trị tăng/giảm được cho vào cuối của loop code) |

### **LOOP FOR/IN**

|  |
| --- |
| - Loop for/in sẽ chạy qua mỗi property của 1 object một lần và lấy ra property value  - Không dùng cho [array]  - Syntax:  1. Declare một variable x  var x; //sẽ gán x vào từng name  2. Áp variable x đó vào loop for/in  for (x in objectName) { //loop start: *gán x thành từng propertyName*. Loop ends khi xong name cuối  kq += objectName[x] ; //mỗi lần x được gán vào một propertyName, gọi values của propertyName đó  } |

### **LOOP FOR/OF**

|  |
| --- |
| - Loop for/of sẽ chạy qua mỗi value của 1 iterable object một lần  - Dùng cho [arrays], ‘strings’, Maps, Nodelists,…  - Syntax:  1. Declare một variable x  var x; //sẽ gán x vào từng value của iterableObjectName  2. Áp variable x đó vào loop for/in  for (x of iterableObjectName) { //loop: *gán x thành từng value*. Loop ends khi xong value cuối  ketqua += x ; //sẽ chạy ra các values của iterableObjectName  } |

### **LOOP WHILE**

|  |  |
| --- | --- |
| - Loop while sẽ loop 1 đoạn code cho tới chừng nào một điều kiện nào đó vẫn là true  - Loop while chính là loop for phiên bản omit đi statement 1 và 3 (declare chỗ khác) | |
| - Syntax:  while (conditions) {  đoạn code  } | VD: var i = 0;  while (i < 10) {  ketqua += i;  i++;  } |

### **LOOP DO/WHILE**

|  |  |
| --- | --- |
| - Loop do/while là biến thể của loop white  - Loop do/while sẽ chạy đoạn code 1 lần, sau đó check condition, nếu true sẽ chạy tiếp  - Ít nhất sẽ chạy đoạn code 1 lần đầu, kể cả sau đó condition bị false | |
| - Syntax:  do {  đoạn code  }  while (conditions); | VD: var i = 0;  do {  ketqua += i;  i++;  }  while (i < 10); |

### Loop break

|  |  |
| --- | --- |
| - Loop break sẽ hủy bỏ loop khi đạt điều kiện/event nào đó  - Trong đoạn code, các code khác sau loop (nếu có) vẫn sẽ chạy tiếp  - Syntax: if ( điều kiện ) { break; ) | |
| VD: // sẽ ra 1 2  var ketqua = ‘’;  var i;  for (i = 0; i <10; i++) {  if (i === 3) {  break;  }  ketqua += i;  } | VD: // sẽ ra 1 2  var ketqua = ‘’;  var i = 0  while (i <10) {  if (i === 3) {  break;  }  ketqua += i;  i++;  } |

### Loop continue

|  |  |
| --- | --- |
| - Loop continue sẽ tạm skip loop khi đạt điều kiện/event nào đó  - Sau khi skip giá trị đó, loop vẫn sẽ chạy tiếp  - Syntax: if ( điều kiện ) { continue; ) | |
| VD: // sẽ ra 1 2 4 5 …10 (skip 3)  var ketqua = ‘’;  var i;  for (i = 0; i <10; i++) {  if (i === 3) {  continue;  }  ketqua += i;  } | VD: // sẽ ra 1 2 4 5 …10 (skip 3)  var ketqua = ‘’;  var i = 0  while (i <10) {  if (i === 3) {  continue;  }  ketqua += i;  i++;  } |

## /Regular Expression/

### Regular Expressions:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - Regexp   |  | | --- | | /pattern/modifiers  //sẽ chọn kết quả đầu tiên valid pattern |   - Modifier: i   |  | | --- | | /pattern/i  //sẽ chọn kết quả đầu tiên valid pattern, bất kể case |   - Modifier: g   |  | | --- | | /pattern/g  //sẽ chọn mọi kết quả valid pattern |   - Chấp nhận nhiều kiểu pattern: |   |  | | --- | | /pattern1|pattern2|pattern3/  //regex này có 3 pattern, chỉ cần valid 1 trong 3 là match |   - Wildcard a.k.a . = bất kỳ ký tự nào: số, dấu, chữ,… (trừ xuống dòng)   |  | | --- | | /ab./ //pattern: ab và bất kỳ ký tự nào |   - n?: có ký tự n hay không cũng được   |  | | --- | | /ab?c/ //pattern: a + có b hay không cũng ok + c |   VD: /colo**u?**r/ //pattern sẽ match cả color và colour  - a+ : cụm nào có ít nhất 1 char a (nhiều a liền nhau thì lấy cả cụm a đó)   |  | | --- | | /a+/ //pattern: có ít nhất 1 character a (nhiều hơn 1 thì lấy cả cụm a) |   \*Thích hợp để loại hết các white space  VD1: “zzz”, “abczzzzzabc” và “zba” .match /z+/ = “zzz”, “zzzzz” và “z”  VD2: “abc”, “def” sẽ match /z+/ là null  - a\* : có từ 0 lần a trở lên   |  | | --- | | /a\*/ //pattern: có từ 0 lần a trở lên (nhiều hơn 0 thì lấy cả cụm a) |   VD: "Hellooo World!” .match /lo\*/ = “l” , “loo”, “l” //mọi chữ l có 0 hoặc nhiều hơn o ở sau  - Cụ thể a lặp lại mấy lần/tối thiểu mấy lần/trong khoảng mấy lần   |  | | --- | | /a{x}/ //pattern: a lặp lại đúng x lần  /a{x,}/ //pattern: a lặp lại tối thiểu x lần  /a{x,y}/ //pattern: a lặp lại từ x đến y lần |   Dùng trong: x char liên tiếp  VD1: /aargh/, /aaaargh/ .match /a{3}/ là null //vì 2 và 4 lần không valid: chỉ lặp đúng 3 lần  VD2: /argh/, /aargh/ .match /a{3,}/ là null //vì 1 và 2 lần không valid: lặp tối thiểu 3 lần  VD3: /argh/, /aargh/ .match /a{3,5}/ là null //vì 1 và 2 lần không valid: lặp từ 3 đến 5 lần  - \s và \S để chọn: space VS non-space   |  | | --- | | /\s/ //pattern: white spacing  /\S/ //pattern: mọi thứ trừ white spacing |   \* dùng .length để đếm số space/ký tự không bao gồm space  - \w và \W để chọn: chữ cái + số + \_ VS các loại dấu khác   |  | | --- | | \w //pattern: [A-Z|a-z|0-9|\_] a.k.a mọi chữ cái , số, \_  \W //pattern: [^A-Za-z0-9\_] a.k.a mọi thứ còn lại |   - \d và \D để chọn: số 0 - 9 VS mọi thứ không phải số   |  | | --- | | \d //pattern: mọi số từ 0-9  \D //pattern: mọi thứ không phải là số |   - Lookaheads: (?=x) và (?!=x) : có x sau đó và không có x sau đó   |  | | --- | | /a(?=x)/ //pattern: các a có x ở sau (ko cần liền kề)  /a(?!=x)/ //pattern: các a không có x ở sau (ko cần liền kề)  /stand-alone/ //pattern: cả string này ( có x | không có x ) | |

### Dùng [charset]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| - Có thể dùng [charSet] để search ra **MỘT** kq khớp 1 trong các [charSet]   |  | | --- | | /a[charSet]b/ //pattern: a + [ any thuộc charSet ] + b |   VD: /b[aiu]g/ //pattern có thể là: /bag/, /big/, /bug/  - Nếu [charset] quá rộng, dùng range để search kq khớp 1 trong range   |  | | --- | | /a[range1-range1range2-range2]b/  //pattern: a + [ bất kỳ thuộc range1 hoặc range2] + b |   VD: /x[a-c0-2]y/ //pattern có thể là: /xay/, /xby/, /xcy/, /x0y/, /x1y/, /x2y/  - Vì [charset] chỉ ra **MỘT** kq, khi search cụm **nhiều chữ cái** dùng **nhiều** [charSet]   |  | | --- | | /[charSet1][charSet1]/ //pattern: [ bất kỳ thuộc charSet1 ] + [bất kỳ thuộc charSet2 ] |   VD1: /[ab][01]/ //pattern có thể là: /a0/, /a1/, /b0/, /b1/  VD2: /[a-c][0-2]/ //pattern có thể là: /a0/, /a1/, /a2/, /b0/, /b1/, /b2/, /c0/, /c1/, /c2/  - Có thể **negative search** với **^**: không nhận các kq trong [charSet]:   |  | | --- | | /[^charSet]/ // pattern: bất kỳ, miễn **không** thuộc charSet |   VD: /x[^a-c0-2]y/ //pattern: x + [ bất kỳ, **ngoại trừ**: /xay/, /xby/, /xcy/, /x0y/, /x1y/, /x2y/ ] + y |

### Regex method: test( ) true or false

|  |  |
| --- | --- |
| - test() là method của regex, argument là string  - Return 1 boolean: regex thuộc string phải không?   |  | | --- | | regexName.test(stringName); //nếu regex thuộc string: true, otherwise: false |   - Dùng với /^pattern/ hoặc /pattern$/ //có “pattern” ở đầu hoặc cuối string phải không?  - Dùng để valid input (vd: required pattern có thuộc string input ko, nếu ko thì error) |

### Regex opposite method: match( )

|  |  |
| --- | --- |
| - match() là opposite method, là **method của string**, argument chính là regex  - Sẽ extract từ string ra những kq match với pattern tùy chọn (tạo ra = /pattern/)   |  | | --- | | stringName.match(regexName);  //return ra 1 array chứa những kq match |   - Các kq sẽ có dạng string, và làm values cho 1 array : [‘a’, ‘1’, ‘z’, …]  - Default sẽ greedy match, dùng dấu ? để lazy match  VD: cho string “giganic”, regex /g[a-z]\*i/, pattern sẽ là g + any char + i  greedy (default) return “gigani” vì khớp pattern g + igan + i và là dài nhất có thể  lazy (/g[a-z]\*?i/) return “gi” vì khớp pattern g + + i và là ngắn nhất có thể |

### (Capture Groups)

|  |  |
| --- | --- |
| - Dùng khi pattern sẽ xuất hiện nhiều lần   |  | | --- | | /(repeat pattern1)(repeat pattern2)\1/  // “$1…$2” vị trí, ngăn cách của $1 và $2 sẽ tương ứng vào 2 (group) |   . |

### Regex opposite method: replace(a, b)

|  |  |
| --- | --- |
| - replace(a, b) là **method của string**, arguments: (regex đích, new value)   |  | | --- | | stringName. replace(a, b);  //a là regex chứa pattern mục tiêu  //b là giá trị mới, có thể là strings hoặc functions hoặc “$index” |   Có thể gián tiếp xóa a bằng cách để trống b (b bằng “” sẽ replace a = rỗng) |

|  |
| --- |
| - Search() method tìm theo value trong 1 ‘string’ và return vị trí  var string = “ABC XYZ”;  var ketqua = string.search(“XY”);  docu … = ketqua;  - Search() method tìm theo /Regular Expression/  var string = “ABC XYZ”;  var ketqua = string.search(/xy/i); // i là case-insensitive  docu … = ketqua;  -> /Regular Expression/ sẽ search mạnh hơn “Value” thông thường  - Các modifiers cho /Regular Expression/ :  /Regular Expression/i : Case-insensitive matching  /Regular Expression/g : Global matching  /Regular Expression/m : Beginning of each line matching  - Các Brackets cho /Regular Expression/:  /[abc]/ : Tìm những chữ cái có trong brackets (thứ tự)  /[0-9]/ : Tìm những số có trong brackets (thứ tự)  /[abc] / : Beginning of each line matching |

## Test và chỉ ra lỗi

|  |  |
| --- | --- |
| - Statement try {đoạn code} sẽ kiểm tra đoạn code này xem có lỗi không  - Statement catch {đoạn code} sẽ chỉ ra lỗi nếu có | |
| - Syntax:  try {  đoạn code cần test  ==  isNan(x)  > … hoặc < …  }  catch(err) {  hiển thị lỗi  }  finally {  code bắt buộc chạy  } | - VD: var ketqua = docu…”message”; //Gán ketqua = </p> message  ketqua.innerHTML = “”; //Tạm thời giấu hiển thị ketqua  var x = docu…input.value; //gán giá trị nhận được từ input là x  try {  if(x == "") throw ‘is empty’; //nếu x để trống thì throw…  if(isNaN(x)) throw ‘is not a number’; //nếu x kp là số thì throw…  if(x > 10) throw ‘is too high’; //nếu x > 10 thì throw…  if(x < 5) throw ‘is too low’; // nếu x < 5 thì throw…  }  catch(err) { //kích hoạt hành động khi có những trường hợp if kia  ketqua.innerHTML = “That is “ + err //đổi giá trị ketqua (đang ẩn)  }  finally {  docu… .value = ““ //xong xuôi thì xóa đi value đang ở trong input  } |
| - kq lỗi trên thực tế là JS throw ra 1 cái Error object bao gồm: name và message  - Có thể tự custom cách throw (ra 1 ‘string’, number, boolean, hoặc object)  - Trên thực tế, tự gán attribute vào </Element> | |

### Keyword this trong 1 method

|  |
| --- |
| - Trong 1 method (a.k.a trong function() thuộc 1 object)  this = owner của method đó a.k.a objectName  vd: var person = {  firstName = “Harry”,  lastName = “Potter”,  fullName = function() {  return this.firstName + “ “ + this.lastName;  }  }  docu… = person.fullName(); |

### Keyword this trong 1 HTML event handler

|  |
| --- |
| - Trong 1 HTML event handler, this chính là HTML element nhận event đó |

### Keyword let

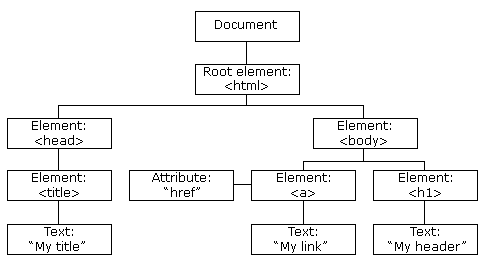
|  |
| --- |
| - let sẽ declare variable có block scope a.k.a chỉ hiện hình trong nội nộ 1 block (vd như loop) giúp declare này không bị xung đột global |

### Data Validation

|  |
| --- |
| - Data Validation là phương pháp đảm bảo input thu được dễ hiểu, hợp lý và có ích  - Validation tasks:  User đã fill hết required fields chưa?  User đã input một valid value?  - Client side validation: browser sẽ kiểm chứng trước khi đưa input cho server  - Server side validation: server sẽ kiểm chứng sai khi nhận input từ browser  - HTML5 hỗ trợ một số validation ràng buộc (vd required)  - VD:  <input id="input1" type="number" min="100" max="300" required> //tạo input và required  <button onclick="myFunction()">OK</button> //tạo nút OK, event onclick  <p>If the number is < 100 or > 300, an error message will be displayed.</p>  <p id="message"></p> //tạo chỗ hiển thị message  <script>  function myFunc() { //tạo function cho onclick event  var theInput = document.getElementById("input1"); //gán 1 var vào input để lấy data  if ( theInput.checkValidity() ) { //nếu “bài kiểm tra .checkValidity() ra true” a.k.a data hợp lệ  document.getElementById("message").innerHTML = "Input OK"; //thì hiển thị ok  } else {  document.getElementById("message").innerHTML = inputObject.validationMessage;  }  }  </script> |

### K

# HTML DOM



### HTML DOM

|  |  |
| --- | --- |
| - Trong DOM, mọi HTML elements được coi như các objects  - HTML DOM methods là các hành động có thể áp lên HTML elements  VD: xóa, access hoặc thêm 1 element  - HTML DOM properties là giá trị của HTML Elements (mà ta có thể set or thay đổi)  VD: content, style, attribute cụ thể nào đó…  -Syntax mẫu:   |  | | --- | | objectParent.accessMethod.property = "new property value"; |   - VD: document.getElementById(“tên-id”).innerHTML = “Hello”  document là objectParent a.k.a phạm vi để access  getElementById(“tên-id”) là 1 accessMethod tìm đến element có id là “demo”  innerHTML là 1 property: content của element đó  -> dịch ra là: content của element “demo” = … (thay đổi content element “demo”) |

### DOM methods: access 1 object

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| - Return ra element dưới dạng 1 object: có thể truy cập các prop, sd các method  - Nên gán các object này thành các biến để rút gọn cú pháp truy cập  1. Access theo id:   |  | | --- | | objectParent .getElementById(“tên-id”) |   2. Access theo tag name của element VD: chọn all element “main”, “img”   |  | | --- | | objectParent.getElement**s**ByTagName(“tên-tag”) |   + Kết quả return sẽ là 1 collection (1 list dạng [array]). Muốn access 1 element cụ thể nào thì dùng [index] theo thứ tự từ trên xuống trong HTML  3. Access theo class. VD: chọn mọi đối tượng có class=”demo”   |  | | --- | | objectParent .getElement**s**ByClassName(“tên-class”) |   + Kết quả return sẽ là 1 collection  4. Access theo CSS selector:   |  | | --- | | objectParent.querySelector("CSS Selector") *//chọn obj đầu tiên*  objectParent.querySelectorAll("CSS Selector") *//chọn toàn bộ match* |   + Kết quả return sẽ là 1 object hoặc 1 collection. |

### DOM methods: tạo /xóa /thêm / thay thế một element

|  |
| --- |
| 1. Tạo một element  document.createElement(element)  2. Xóa một element  document.removeChild(element)  3. Thêm một element  document.appendChild(element)  4. Thay thế một element  document.replaceChild(new, old) |

### DOM methods: thêm 1 đoạn text

|  |
| --- |
| document.write(‘text’) |

### DOM methods: tạo/thay đổi attribute

|  |
| --- |
| accessMethod.setAttribute(attribute, “value”)   * Không nên dùng |

### DOM properties: giá trị của attribute của một object

|  |
| --- |
| accessMethod.tên-attribute (= … để đổi value của attribute đó)  VD: document.getElementById(“image”).src = “picture.jpg” //đổi value của src |

### DOM properties: giá trị của content HTML của 1 object

|  |
| --- |
| accessmethod.innerHTML (= … để đổi nội dung HTML của object đó)  VD: document.getElementById(“text”).innerHTML = “new text” //đổi content mới |

### DOM properties: giá trị của một thuộc tính css của 1 object

|  |
| --- |
| accessMethod.style.cssProperty (= … để đổi style của object đó)  VD: accessmethod.style.color = “red” //đổi màu  accessmethod.style.fontSize = “40px” //prop dài thì camelCase  - .style.cssProperty sẽ chỉ return ra kq nếu như prop này là inline-css hoặc sau khi prop được đẩy vào = javascript. |

### DOM methods: addEventListener(“event”, func, bubble? || capture?)

|  |  |  |  |
| --- | --- | --- | --- |
| - Có 3 cách tạo event:  1. Nested các event attributes: onclick, onmouseover, onmouseup,… vào element  1.1: Tạo 1 function() chứa event  1.2: Nest function() này vào 1 event attributes   |  | | --- | | <tag onclick=”1 function được declare = JS”> |   2. Dùng HTML DOM: .onclick,…  2.1: Tạo 1 function() chứa event  2.2: Access nút event handlers = DOM methods và thêm DOM Events vd: .onclick   |  | | --- | | accessMethod.events = function; //function không có () |   3. Dùng DOM methods: EventListener (có thể dùng nhiều lần cho 1 element)     |  | | --- | | accessMethod.addEventListener(“event”, func, useCapture?); |   3.1: parameter “event”: DOM event bất kỳ vd: “click”, … (bỏ “on” ở đầu)    3.2: parameter func: function cần call khi event xảy ra, có thể nested(){} or call tên  Nếu cần argument: dùng “anonymous func” call, vd: function(){ myFunction(p1, p2);}  3.3: parameter useCapture: true or false? a.k.a Event Capturing or Event Bubbling?  - Nếu có 1 element con nằm trong 1 or nhiều lớp element parents, khi xảy ra event vd như “click”, element nào sẽ được active trước?  + Event **Bubbling**: active từ element trong cùng ra ngoài  + Event **Capturing**: active từ element ngoài cùng vào trong  - useCapture nếu k set gì thì default là **false** a.k.a dùng **bubbling** - trong cùng trước  - useCapture nếu set **true** thì dùng **capturing -** ngoài cùng trước  - dùng argument này ở element container (vd: div) |

### DOM methods: removeEventListener(parameters của add listener)

|  |  |
| --- | --- |
| - Xóa event listener, yêu cầu khớp accessMethod và các parameters của add listener   |  | | --- | | accessMethod.removeEventListener(“event”, func, useCapture?); | |

### DOM Navigation

|  |  |
| --- | --- |
| - Mọi thứ đều là nodes: element nodes, text nodes (text bên trong elements), …  - Relationships: parent, next siblings, previous siblings, first child, last child  - Node properties để navigation:   |  | | --- | | let x = accessMethod.innerHTML; //truy cập content của…  =  let x = accessMethod.firstChild.nodeValue; //truy cập value của firstChild của…  =  let x = accessMethod.childNodes[0].nodeValue; //truy cập value của firstChild của… | |

### DOM Property nodeName

|  |
| --- |
| - nodeName property chỉ ra tên của 1 node:  - nodeName của element node chính là uppercase tagname của element đó  - nodeName của attribute node chính là tên attribute  - nodeName của text node là #text  - nodeName document node là #document |

### DOM Property nodeValue

|  |
| --- |
| - nodeValue property chỉ ra value của 1 node  - nodeValue của element node là null  - nodeValue của text node là đoạn text đó  - nodeValue của attribute node là value của attribute đó |

### DOM Property nodeType

|  |
| --- |
| - nodeType property sẽ return loại node, giá trị này là read-only |

### DOM methods: tạo node mới: create và appendChild()

|  |
| --- |
| - Để tạo element trong DOM: tạo nodes rồi dùng 1 element có sẵn nhận nuôi nó  VD:  let pElement = document.createElement(“p”); //tạo element </p> (trong document)  let pContent = document.createTextNode(“Content of p”); //tạo text content (cho </p>)  pElement.appendChild(pContent); //cho </p> nhận nuôi pContent a.k.a nhét content vào </p>  accessMethod.appendChild(pElement) //cho 1 node to hơn nhận nuôi </p> (nếu cần)  - VD trên dùng appendChild() sẽ tạo ra last child, nếu muốn có vị trí nuôi cụ thể:  let x = accessMethod… //tạo biến x gán vào element sẽ đứng sau child mới  accessMethod.insertBefore(pElement, x) //cho 1 node to nhận nuôi </p>, đặt đứng trước x |

### DOM methods: xóa node với removeChild(child)

|  |
| --- |
| - Xóa 1 node: target parent và dùng .removeChild(child)  VD: //thường dùng 1 function  function myFunc() {  let parent = accessMethod đến parent của element cần xóa  let child = accessMethod đến element cần xóa  parent.removeChild(child);  }  hoặc childToBeRemoved.parentNode.removeChild(childToBeRemoved); |

### DOM methods: replace node với replaceChild(newChild, oldChild)

|  |
| --- |
| - Thay thế 1 node: target parent và dùng .replaceChild()  VD:  parent.replaceChild(newChild, oldChild) |

# Browser BOM

### Window History: back và forward

|  |
| --- |
| - history.back() để back lại url trước  - history.forward() để forward url tiếp theo |

### Window Alert Box: back và forward

|  |
| --- |
| - Alert Box:  alert(“content”);  - Confirm box: Sẽ ra 1 box chọn giữa OK và Cancel  confirm("content ") //chọn ok sẽ return true, chọn cancel sẽ return false  - Prompt Box: Sẽ ra 1 trường input, sau đó chọn OK hoặc Cancel  prompt(“Content”, “placeholder”); //chọn ok sẽ return content nhập, chọn cancel sẽ return null  \*Trong “content” có thể dùng \ để xuống dòng |

### Window Timing: setTimeOut() và setInterval()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| - Khi muốn delay 1 action nào đó sau 1 khoảng timing:   |  | | --- | | .setTimeout(function, milliseconds); |   Khi đang delay, nếu muốn hủy action đó:   |  | | --- | | clearTimeout(intervalVar) //gán timeout method vào 1 var intervalVar |   - Khi muốn repeat 1 action nào trong 1 khoảng timing:   |  | | --- | | .setInterval(function, milliseconds); |   Khi đang delay, nếu muốn hủy action đó:   |  | | --- | | clearTimeout(timeOutVar) //gán timeout method vào 1 var timeOutVar | |

### Cookies

|  |  |
| --- | --- |
| - Cookies là data được lưu dưới dạng các file text nhỏ.  - VD: Lần đầu truy cập user sẽ nhập tên, từ lần sau sẽ welcome user  1. Tạo function set 1 cookie value:   |  | | --- | | function setCookie(cname, cvalue, exdays) { //tạo func(name=value;expire date)  var d = new Date(); //tạo 1 date variable là d  d.setTime(d.getTime() + (exdays\*24\*60\*60\*1000)); //lấy time  var expires = "expires="+ d.toUTCString(); //chuyển sang UTC time  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";  } | |

# AJAX

### AJAX

|  |
| --- |
| - **A**synchronous **J**avaScript **A**nd **X**ML  - AJAX bao gồm 2 phần:  + XMLHttpRequest object (để request data từ server)  + JavaScript và HTML DOM (để hiển thị hoặc dùng data trên)  - Quy trình:  1. Một event được kích hoạt (page được load or bấm nút, …)  2. Tạo 1 XMLHttpRequest object bằng JavaScript  3. XMLHttpRequest object sẽ gửi request đến server  4. Server sẽ tiến hành request đó  5. Server trả kết quả về page  6. Đọc kết quả này bằng JavaScript  7. Một hành động nào đó sẽ được thực hiện bằng JavaScript |

### AJAX p2- Tạo một XMLHttpRequest object

|  |  |
| --- | --- |
| - XMLHttpRequest object dùng để trao đổi data với server  - Syntax:   |  | | --- | | let XHR = new XMLHttpRequest() | |

### AJAX p3- Gửi request đến server bằng open() và send() methods

|  |  |  |
| --- | --- | --- |
| - Có 2 loại request: “GET” và “POST”  1. “GET” request:   |  | | --- | | XHR.open("GET", "url", async!); //, file path, **true** (asynchronous) || false (synchronous)  XHR.send(); |   2. “POST” request:   |  | | --- | | XHR.open("POST ", "url", async!); //, file path, **true** (asynchronous) || false (synchronous)  XHR.send(); |   - url parameter: là địa chỉ của 1 file trên server, có thể là .txt, .xml, .asp, .php  - async parameter: nên là true |

### AJAX p5- Gửi request đến server bằng open() và send() methods

|  |
| --- |
| - Có thể tạo 1 function để chạy khi request được answer  - Function này được tạo với property onreadystatechange của XHR object  - Function này sẽ chạy khi mà property readyState thay đổi |

# CÁC BÀI TOÁN

### Tính số lớn nhất trong 1 tập hợp số p1

|  |
| --- |
| - Math.max(a, b, c) |

### Tính số lớn nhất trong 1 tập hợp số p2

|  |
| --- |
| - Sử dụng arguments object của funtion để tạo 1 array chứa các arguments  1. tạo 1 var kq = -infinity;  2. loop i từ 0 đến hết arguments object, so sánh với var kq kia;  3. Dùng if ( Nếu index nào lớn hơn kq kia ) thì { được gán cho kq };  4. loop xong thì index lớn nhất sẽ được làm kq;  5. return kq;  - Syntax  function findMax() { //truyền tập hợp số vào làm arguments  var kq = -Infinity; //kq là số bé nhất có thể, để đảm bảo index[0] pass loop đầu tiên và làm base để loop  var i; //tạo biến i cho loop for  for (i = 0; i < arguments.length; i++) { //arguments là [real arguments], dùng i để loop các index ra  if (arguments[i] > kq) { //index sẽ được loop dần từ 0, nếu index nào lớn hơn kq  kq = arguments[i] //thì index đó sẽ được làm kq, cứ thế loop  }  } //kết thúc loop, lúc này kq đã là index lớn nhất có thể  return kq; //trả ra kết quả là kq  } |

### Tính tổng các số trong 1 tập hợp số p1: [dùng arguments] để loop +=

|  |
| --- |
| - Khi truyền tập số đó vào parameters thì sẽ được 1 array ngầm tên là [arguments]  - [arguments] này tượng trưng cho 1 array chứa các arguments, có thể dùng index như bth  - Loop i làm index của [arguments] : arguments[i],  - += từng value của từng index vào 1 var kq rồi return kq  VD:  function sum(1, 2, 3) { //truyền tập hợp số vào làm arguments  var kq = 0;  var i; //tạo biến i cho loop for  for (i = 0; i < arguments.length; i++) { //arguments là [array] ngầm, a.k.a 1 array chứa arguments  kq += arguments[i] //+= lần lượt value của từng index vào kq, thông qua loop từ index 0 đến hết  } //kết thúc loop, lúc này kq đã là tổng tất cả các index  return kq; //trả ra kết quả là kq  } |

### Tính tổng các số trong 1 tập hợp số p2: method reduce()

|  |
| --- |
| VD: làm tròn xuống và tính tổng các value sau: 5.4, 10.5, 15  Cách 1: Tạo 1 biến chứa [array] tập hợp số, reduce() biến đó  var numbers = [5.4, 10.5, 15]  function sum() { //tạo function chứa code  displaymethod… = numbers.reduce( (total, num) => total + Math.floor(num) )  }  button…. onclick=”sum()”… //gán function vào event  Cách 2: Tạo 1 function nhận rest parameters, reduce() rest parameters đó  let sum = (...rest) => { // rest parameters chỉ dùng trong function  displaymethod… = rest.reduce( (total, num) => total + Math.floor(num) );  }  button…. onclick=”sum(5.1, 10.5, 15)”… //gán function vào event |

### Tạo số random trong 1 khoảng min-max:

|  |  |
| --- | --- |
| function randomMinMax(min, max) {   |  | | --- | | Math.floor( Math.random() \* (max - min + 1) ) + min |   }  //1. để kq tối thiểu = min: Trong TH random ra 0 thì tất cả sẽ = 0, nên phải + min để phòng TH này  + min  //2. để kq tối đa = max: + 1 vào max để round down = floor sẽ ra max  Math.floor( Math.random() \* (max + 1) ) + min  //3. để tránh TH random ra số lớn quá: max phải trừ đi min  Math.floor( Math.random() \* (max - min + 1) ) + min |

### Cắt đi một số values tùy chọn của 1 array

ES6: Use Destructuring Assignment with the Rest Parameter to Reassign Array Elements

|  |
| --- |
| - Tiến trình:  1. Gán array gốc vào 1 biến tên tùy chọn  2. Tạo 1 function có parameter tên tùy chọn (argument nhận vào sẽ là biến chứa array gốc) function này sẽ return ra phần còn lại sau cắt.  3. Áp dụng Destructuring Assignment a, b, và rest parameter …rest lên array gốc:  3.1 Trong function, declare 1 array giả lập có Destructuring Assignment a, b, và  rest parameter …rest. Tùy số values cần cắt mà tạo số lượng a, b phù hợp  3.2 Gán array giả lập đó vào parameter của function  3.3 mà parameter chính là array gốc -> đã áp dụng a, b, và …rest lên array gốc  4. Cho function return ra …rest (a.k.a chỉ return phần còn lại sau các a, b cần cắt)  - VD: Cắt đi 3 values đầu tiên của array sau: [1, 2, 3, 4, 5]  const arrayName = [1, 2, 3, 4, 5] //bước 1: gán  function cutTheFirstThree(inputArray) { //bước 2: tạo function  const [a, b, c, …rest] = inputArray; //bước 3: áp DA và …rest để cắt  return rest; //bước 4: return ra phần còn lại  } |

### Tạo pattern bắt buộc của username

|  |  |
| --- | --- |
| Yêu cầu:  I. Chỉ được dùng số và chữ cái (không ký tự)  II. Phải bắt đầu bằng chữ cái  III. Có thể có số hoặc không, nếu có thì phải đặt ở cuối  IV. Có phân biệt viết hoa-thường  V. Độ dài tối thiểu là 2 characters,  VI. Nếu chỉ có 2 char thì đó phải là 2 chữ cái     |  | | --- | | **/ ^[a-z] ( [0-9][0-9]+ | [a-z]+\d\* )$ /** |   Tiến trình:  1. điểm bắt đầu của input: ^  2. bắt đầu phải là chữ cái: ^[a-z]  sau đó chia làm 2 khả năng để kết thúc: (1 trong 2 đều được)  3.1.1 : có thêm 1 hoặc nhiều hơn chữ cái: [a-z]+ //để thỏa mãn: V, trong TH có VI  3.1.2: và có thể kết thúc bằng số hoặc không: \d\* //để thỏa mãn: III  hoặc thỏa mãn:  3.2. : kết thúc bằng 2 hoặc nhiều hơn số: [0-9] [0-9]+ //để thỏa mãn: V trong TH loại bỏ VI  4. bao trọn phần 3 bằng dấu ngoặc tròn () để làm kết thúc input: (phần 3)$ |

### Tạo function tìm độ dài của string dài nhất trong 1 chuỗi strings

|  |
| --- |
| - Tiến trình:  1. Cần biến chuỗi strings đó thành 1 array chứa các strings bằng split()  2. Tạo 1 var nền để chứa kq = 0  3.1 Loop một lượt các index của array đó,  3.2 Khi loop, cứ gặp index nào > var nền thì lại assgin đè lên var nền, cho đến hết  VD:  function myFunc(str) {  let arr = str.split(“ “); //1. biến string str thành 1 array, các value ngăn cách bằng space  let kq = 0; //2. tạo var nền = 0  for ( let i = 0; i < arr.length; i++ ) { //3.1 loop lần lượt các index[i] của arr  if ( kq < arr[i].length ) {  kq = arr[i].length //3.2 Khi loop, cứ gặp index nào dài hơn var nền thì var nền lấy giá trị đó  }  }  return kq //return ra kq lúc này đã có value cao nhất sau loop  } |

### Lọc ra values lớn nhất của mỗi array con nằm trong 1 array lớn (kq là 1 array)

|  |
| --- |
| - Tiến trình:  1. Tạo 1 var kq chứa 1 array rỗng làm results  2.1 Loop ngoài: loop các array con bằng for [i]  2.2 Tạo 1 var biggest đại diện cho số lớn nhất mỗi array, gán nó vào đầu mỗi array con bằng [i][0]  2.3.1 Loop trong: loop trong các array con bằng for [j], loop từ index [1] trở đi  2.3.2 Khi loop trong, nếu có value nào > biggest thì update nó vào b  2.4 Gán biggest vào mỗi array con [i] của array kq (vẫn trong loop ngoài)  3. Return kq  VD: cho arr = [ [1, 2, 3], [4, 5, 6], [-7, -8, -9] ]  function myFunc(arrayName) {  var kq = [];  for (let i = 0; i < arrayName.length, i++) {  var biggest = arrayName[i][0];  for (let j = 1; j < arrayName[i].length; j++) {  if (biggest < arrayName[i][j]) {  biggest = arrayName[i][j]  }  }  kq[i] = biggest;  }  return kq  } |

### Tạo 1 function nhận arguments là 1 array và 1 func, tìm value đầu tiên pass func

|  |
| --- |
| - Y/c: myFunc( [array], func )  - Tiến trình:  1. Tạo 1 var x để test kq với func  2. Loop 1 lượt array, gán x vào các values của array  3. Nếu func nhận x làm argument mà true thì return x  4. Nếu sau loop không có values nào pass thì return undefined |

### Viết hoa chữ cái đầu tiên của mỗi từ trong string

|  |
| --- |
| - Tiến trình:  1. Split string thành 1 array, lowercase cả string  2. Dùng map để áp 1 function viết hoa lên từng values (từng từ)  3. Function viết hoa: replace chữ index [0] thành chữ đó viết hoa  4. Join các values lại thành 1 string và return  VD: string1 = “Harry you’re a wizard”  function upperFirstChar(str) {  let x = str.toLowerCase().split(‘ ‘); //1. split và lowercase all  let kq = x.map( function(val) { //2. Dùng map để áp function viết hoa  return val.replace( val.charAt(0), val.charAt(0).toUpperCase() ) //3. replace index 0  } );  return kq.join(‘ ‘);  } |

### Lọc ra các unique value trong mỗi array

|  |
| --- |
| Tiến trình:  - Loop từng value của 1 array để check bằng indexOf hoặc includes  + Nếu indexOf == -1 thì value đó unique  + Nếu includes == false thì value đó unique |

### Tạo dãy Fibonacci

|  |
| --- |
| - Dãy Fibonacci bắt đầu với hai số 1, sau đó từ số thứ ba trở đi, sẽ bằng tổng 2 số liền trước.  VD: 1, 1, 2, 3, 5, 8, 13, 21,…  - Yêu cầu: Tạo dạy Fibonacci dài *num* số.  VD 5 số: 1, 1, 2, 3, 5  - Tiến trình:  + Tạo 1 array kq rỗng để chứa dãy số;  + Tạo 1 biến curr để tiến trình, bắt đầu = 1 (số đầu tiên của dãy)  + Tạo 1 biến prev sẽ là số đứng trước của cur, bắt đầu = 0 (trước curr ko có gì)  + Tạo vòng lặp while, chừng nào dãy số vẫn dài *num* số  + Nội dung lặp: cho curr += prev để tiến đến số tiếp theo  cài giá trị mới cho previous, chính là current này - previous  push current mới vào array kq |

### Giả lập browser tab management

|  |
| --- |
| - Mô tả:  + Quản lý các window dưới dạng các array  + Quản lý các tab dưới dạng các values thuộc các array (window)  + Tạo chức năng:  1. Tạo window mới  2. Mở tab mới  3. Đóng tab chỉ định  4. Nhập các window lại với nhau  + Test cả 4 chức năng trên cùng 1 lúc  - Code:  var Window = function(tabs) { //tạo Constructor *Window*, nhận argument là 1 array  this.tabs = tabs //Window’s instance sẽ có property *tabs*, values của *tabs* là array từ argument  } //hết chức năng 1  Window.prototype.open = function(newTab) { //tạo 1 prototype Method cho Window: open  this.tabs = this.tabs.push(newTab); //openTab push argument vào prop tabs của Window’s instance  return this; //trả ra Window’s instance, phiên bản đã thêm tab  }  Window.prototype.close = function (targetTab) { //tạo 1 prototype Method cho Window: close  let beforeTarget = this.tabs.slice(0, target); //tạo 1 var chứa các tabs trước target tab  let afterTarget = this.tabs.slice(target + 1); //tạo 1 var chứa các tabs sau target tab  this.tabs = beforeTarget.concat(afterTarget); //nối 2 var đó làm prop tabs của Window’s instance  return this; //trả ra Window’s instance, phiên bản đã đóng tab  }w  Window.prototype.join = function (targetWindow) { //tạo 1 prototype Method cho Window: join  this.tabs = this.tabs.concat(targetWindow.tabs); //nối prop tabs của Window’s instance  return this; và prop tabs target Window’s instance rồi return  } |

# OOP

### Introduction

|  |
| --- |
| Về bản chất, phần mềm giải quyết vấn đề bằng cách tính toán. Quy trình này bao gồm: xác định vấn đề và tìm giải pháp. Lập trình **hướng đối tượng** là 1 trong nhiều giải pháp chính trong công cuộc phát triển phần mềm  OOP sắp xếp code theo các objects. Đôi khi còn được gọi là classes. Data là attributes của 1 object, và hành vi (hàm) là methods  Cấu trúc hướng đối tượng khiến mọi thứ linh hoạt hơn. Các objects có thể trao đổi info bằng cách call và truyền data vào các methods của các objects khác. Thêm vào đó, các class có thể nhận. kế thừa mọi tính năng từ 1 base hoặc 1 parent class, giảm lặp code |

### Thành phần của 1 object

|  |
| --- |
| - 1 object objectName có thể bao gồm:  + Các properties có dạng propName: value  + Các methods có dạng methodName: function() {content}  - Access:  + Access properties: objectName.propName  + Access methods: objectName.methodName()  - Có thể thay thế objectName = this khi assign sẵn 1 khung, chưa có tên cụ thể cho obj |

### Constructor: Bộ khuôn mẫu tạo objects

|  |
| --- |
| - Constructor là 1 function đặc biệt dùng để tạo ra các object theo 1 khuôn mẫu  - Constructor() khác các function ở chỗ: không return  - Dùng parameters để xác định values cho props của các objects được khởi tạo  - Dùng this để refer tên object sẽ được tạo ra từ Constructor  VD:  let Person = function(fname, lname) {  this.firstName = fname;  this.lastName = lname;  this.fullName = function() {  return this.firstName + ‘ ‘ + this.lastName  };  } |

### Chỉnh sửa Constructor

|  |
| --- |
| - Không thể tự ý chỉnh sửa, thêm bớt props/methods từ bên ngoài Constructor  VD: ConstructorName.propName3: tham số c //sai  - Phải dùng .prototype để chỉnh sửa từ bên ngoài  VD:  ConstructorName.prototype.newProp: value cho newProp;  ConstructorName.prototype.newMethod = function() {  return … ;  }; |

### Constructor: Cách tạo objects (instance) từ khuôn

|  |  |  |
| --- | --- | --- |
| - Các objects được tạo ra từ Constructor được gọi là **instance** của Constructor đó  - Dùng keyword new để tạo 1 instance từ 1 Constructor   |  | | --- | | let x = new ConstructorName(arguments); |   VD:  let someone = Person(“Harry”, “Potter”);  //someone sẽ có mọi props/methods từ ConstructorName, các values nhận vào là 2 arguments “Harry”, “Potter”  console.log(someone);  // { firstName : “Harry”, lastName: “Potter”, fullName: “Harry Potter” }  - Có thể access, modify các props/methods mà instance nhận được từ Constructor  VD: someone.firstName = “James”;  - Có thể kiểm tra xem 1 object A có phải được tạo ra từ 1 Constructor X hay không  (kiểm tra xem 1 object A có phải là instance của 1 Constructor X hay không)   |  | | --- | | instanceName instanceOf ConstructorName;  // return true/false | |

### Constructor: dùng prototype props để apply lên mọi instance

|  |
| --- |
| - Constructor có 1 loại properties ẩn là prototype  - prototype props của 1 Constructor sẽ được share cho mọi instance  - Khác với việc khai báo cụ thể 1 prop trong Constructor: khi log 1 instance sẽ KHÔNG  hiển thị các prototype prop thừa kế, nhưng vẫn access được các prototype prop đó |

### 3 loại properties: own properties

|  |
| --- |
| - Là các prop được defined trực tiếp bên trong 1 object.  - Defined khi khởi tạo 1 object lẻ, hoặc defined trong Constructor để làm khuôn.  - Có thể check được bằng .hasOwnProperty() |

### 3 loại properties: prototype properties

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| `- Là các prop được defined **sau khi khởi tạo** với “**prototype** mode” (.prototype)  - Hầu hết mọi object đều có prototype, và 1 prototype của 1 object cũng là 1 object,  nên có prototype chain (Object -> Supertype -> Constructor -> object lẻ)  - Thường sẽ defined cho Supertype|Constructor, defined riêng bên ngoài hoặc:  + Define nhúng object: A. Nhét các prototype prop, methods vào 1 SuperObject:   |  | | --- | | Supertype.prototype = { các prototype props, methods }  //Lv 1: khai báo prototype vào 1 Supertype để truyền xuống Constructors, object lẻ ở lv dưới  hoặc  ConstructorName.prototype = { các prototype props, methods }  //Lv 2: khai báo prototype vào 1 Constructor để truyền xuống các object lẻ ở lv dưới  //sẽ override constructor prop, cần khai báo constructor prop = ConstructorName |   + Define nhúng object: B. Tạo Constructor từ SuperObject trên   |  |  | | --- | --- | | 1  2 | ConstructorName.prototype = Object.create(Supertype.prototype)  // cho prototype của Constructor thừa kế prototype của Supertype  //sẽ override constructor prop, sau khi kế thừa cần khai báo lại:  ConstructorName.prototype.constructor = ConstructorName  // sau khi kế thừa, Constructor có thể tạo thêm prototype riêng | override prototype kế thừa  let ObjectName = new ConstructorName()  //tạo object lẻ từ Constructor (Constructor này thừa kế prototype từ Supertype)  hoặc  let ObjectName = Object.create(Supertype.prototype)  //tạo object lẻ lấy proptotype trực tiếp từ Supertype | |  |   - Sau khi defined, các instance của Constructor đều được share Prototype props  - Access với Constructor: .prototype.propName  - Access với instance: .propName như bth, nhưng sẽ ko hiển thị khi log instance ra  - Kiểm tra xem prototype của 1 object A có phải thừa kế từ 1 Constructor X không:   |  | | --- | | ConstructorName.prototype.**isPrototypeOf**(objectName)  //return true | false | |

### 3 loại properties: constructor property

|  |  |
| --- | --- |
| - Là 1 prop đặc biệt, value trả về sẽ show ra Constructor của 1 instance:   |  | | --- | | objectName.constructor  //return tên Constructor của object |   - Hầu hết mọi object đều có 1 primitive Constructor (Number, Object, String, …) nên có thể dùng prop này để biết đây là loại data gì |

### Dùng closure để khóa properties tránh bị modify từ bên ngoài

|  |  |
| --- | --- |
| - Nếu defined trực tiếp 1 prop, prop đó có thẻ bị access và modify từ unexpected code  - Closure là thao tác gán giá trị của 1 prop vào 1 biến thông qua keyword let, sau đó tạo 1 method để return ra giá trị của biến trên   |  | | --- | | function ConstructorName {  let closure = x; //dùng closure tạo 1 biến private chứa giá trị x  this.privilegedMethodName = function() {  return x; //tạo 1 method đặc quyền, khi call nó sẽ trả ra x  }; //x giờ đã được khóa, chỉ có thể access bằng cách call method trên  } | |

### Dùng mixin để chứa methods cho unrelated objects

|  |  |
| --- | --- |
| - Có những lúc không thể dùng kế thừa để lấy props vì dù có chung 1 loại props nhưng chúng không giống nhau.  VD: 1 Constuctor là Airplanes và 1 object là parrot đều có thể fly() nhưng unrelated  - Lúc này có thể dùng 1 function mixin để chứa các methods chung đó   |  | | --- | | let mixin = function(obj) {  obj.methodName = function() {…}  } |   - Mỗi khi cần gán các method đó vào 1 object, chỉ cần call mixin(objectName)  VD: mixin(parrot) //object parrot đã có thêm các methods chứa trong mixin |

### IIFE: Immediately invoked function expression

|  |  |
| --- | --- |
| - 1 IIFE function sẽ được chạy ngay sau khi declare, không cần chờ call  - IIFE không có tên, không được gán vào biến   |  | | --- | | ( function() { code… } ) ()  //gói toàn bộ function vào trong () và kết thúc với một () nữa |   - Dùng để nhóm các chức năng có liên quan thành 1 module  VD:  let movingMethodObject =  **(**function() **{** return { mixin1: function(obj) { obj.med1 = function() {..} }, mixin2: function(obj) { obj.med2 = function() {..} } **}) ()**  //hoặc viết thường:  let movingMethodObject = ( function() {  return {  mixin1: function(obj) {  obj.method1 = function() {…}  } ,  mixin2: function(obj) {  obj.method2 = function() {…}  }  }  } ) ()  //sau đó, có thể dùng:  movingMethodObject.mixin1(objectName);  objectName.method1() |

# Lập trình hàm

### Introduction

|  |
| --- |
| - Lập trình hàm tuân thủ các nguyên tắc:  + INPUT -> PROCESS -> OUTPUT  + Các hàm sẽ **độc lập** với tình trạng của program hoặc các biến global. Các hàm sẽ chỉ **phụ thuộc vào** **arguments** được truyền vào trong hàm  + Functions try to limit any changes to the state of the program and avoid changes to the global objects holding data  + Các hàm sẽ phải hạn chế tác động đến tình trạng của program và tránh thay đổi các global objecst có chứa data  + Lập trình hàm sẽ chia program thành các phần nhỏ, có thể test được |

### Gọi tên các kiểu function

|  |
| --- |
| - Callbacks: là function được truyền làm argument | được return ra từ function khác  - Higher order: là các function nhận các function khác làm arguments  - Lambda: như callbacks |