**APRENDENDO JAVASCRIPT**

**Criando uma HTML**

- **HTML:** Cria uma área HTML completa para exibição no navegador, cujo é constituída de 2 partes principais: HEAD e BODY. Todos os comandos escritos dentro da área HTML, seja ela na seção head ou body, precisa ser colocado entre “< >”, sem exceção. E, praticamente todos os comandos precisam ser escritos 2 vezes, uma que indica a abertura do comando (<P>, por exemplo) e uma que indica o fechamento dele (</P>, por exemplo). Para o comando que indica o fechamento é necessário colocar uma **“/”** antes do comando em si. O texto a ser afetado pelo comando precisa estar entre o comando de abertura e o de fechamento, caso contrário, não irá funcionar. Ex: <P>Escrevendo um exemplo</P>.

**HEAD**

Dentro dessa área é onde se faz toda a formatação da página e onde se encontram os comandos:

**- Meta:** Utilizado para formatar acentuação dentre outras coisas.

**- Title:** Define o título que é exibido na aba da página.

**- Style:** Discutido mais para frente.

**BODY**

Nessa área é onde é colocado todo o conteúdo da página, desde o título, que é a primeira coisa a aparecer, até textos, imagens, links etc. Nessa área se encontram comandos como:

**Formatação de Títulos:**

Estes comandos definem que o texto digitado dentro deles será determinado o título, subtítulo e afins, da página. Começa pelo que deixa o título maior e vai diminuindo conforme o número.

1. **H1:** O maior de todos
2. **H2: ..**
3. **H3: ..**
4. **H4: ..**
5. **H5:** O menor de todos.

**Formatação de Quebra de linhas**

Estes comandos definem o tipo de quebra de linha que será feita, com o intuito de fazer uma formatação adequada para o conteúdo da página.

**- P:** Faz com que o texto digitado dentro fique separado em um parágrafo, ou seja, pule uma linha, escreva o texto e pule outra linha antes que o próximo seja escrito.

**- Br/:** Faz com que ao final do texto, o próximo, ou a parte subsequente, seja escrito na linha de baixo. Funciona como ponto final. Este comando, diferente dos outros, só é colocado no final, sem a necessidade da abertura e fechamento do comando, como a grande maioria deles.

**Criando Inputs**

O comando <input> é feito para criar diversos tipos de elementos como caixas para digitar textos ou números, botões e afins. Dentro deste comando existe a tag **type=” ”**, onde você determina o que será. Se for uma caixa de digitação, basta colocar qual o tipo de informação será colocado, como: number, string etc., se for um botão, basta digitar button, que automaticamente se torna um. Se for caixa de texto terão outras duas tags, são elas: name e id=” ”, onde você digita a identificação do input. Se for botão, haverá uma outra: value=” ”, onde você digita o que estará escrito no botão. Para criar a interação com ela adicione o evento que desejar como o onclick=” ”, e dentro das aspas a função relacionada, que será criada na área script.

**CSS**

**- STYLE:** É o comando que abre e dá início a uma área, onde se faz todo o design da página definindo os estilos, como: Tamanhos e Fontes das Letras, Cores das Mesmas e do Background etc. Esse comando entra na parte HEAD do HTML.

Dentro da área STYLE existem comandos, cujo, sempre há a necessidade de se colocar chaves (**{}**) logo após os comandos, para que se possa fazer as alterações de design dá página na área desejada. Os “comandos” são referentes às áreas que se quer alterar, portanto os comandos são:

**- Body {}:**

Abre uma área que interliga com o BODY do HTML e permite fazer alterações.

Os comandos digitados dentro das chaves afetarão todo o conteúdo do corpo da página.

**- H1 {}:**

Liga todos os comandos de estilos digitados dentro das chaves ao título da página, ou seja, ao texto digitado dentro do comando <H1> </H1> no body do HTML, uma vez que h1 atribui o texto digitado ao título.

**Comandos:**

**- Color:** Determina a cor das letras. Quando coloca o mouse em cima da cor selecionada abre uma janela de gama das cores permitindo que você selecione qual cor quer especificamente, junto com sua opacidade.

**- Background-color:** Determina a cor do fundo e tem o mesmo esquema de seleção de cor com o mouse que a função Color

**- Font:** Determina a Fonte, Tamanho e o tipo das letras (Normal, pequena, oblíqua...)

**- Width:** Define o comprimento de inputs por pixels. Coloca o valor depois dos dois pontos e “px” depois do valor.

**- Height:** Define a largura de imputs por pixels.

**- Line-height:** Define onde a linha de um determinado texto ficará localizado na vertical. Também determinado por pixels.

- **Text-align:** Define o alinhamento de um texto. Center, end, left, right...

**-Margin:** Define o tamanho de uma margem. Pode ser no top, botton... Tamanho definido por pixels.

**JAVASCRIPT**

Para criar um programa apenas JS sem o html, basta colocar a extensão “.js” no arquivo. Para executar o programa aperte f8.

**Comandos:**

**- Console:** Faz com que os comandos seguintes sejam executados no console.

**- Log(‘ ’):** Seguido do “console”, faz escrever o texto dentro das aspas.

**- Var**: Cria variável.

**- SCRIPT:** É a tag que quando colocada na área BODY do HTML, abre uma área onde se faz a programação em JS, é a área de interação da página com o cliente, onde você coloca comandos que permitem o usuário interagir. Os comandos podem ser usados juntamente com outros se separados por um **“.”**. Por exemplo: **window.alert(‘ ‘); window.prompt(‘ ’); document.write(‘’), etc.** Sendo eles:

**- Window:** Antes de exibir o conteúdo da página é mostrada uma janela no topo da página. Quando seguido de um “.” Esse comando pode fazer outras coisas como:

**- Alert(“ ”):** Exibe um alerta, sem nenhum tipo de interação com o cliente, exibindo a mensagem escrita dentro das aspas.

**- Confirm(“ ”):** Exibe uma janela com a mensagem escrita dentro das aspas e duas opções de interação abaixo dela: “Ok” e “Cancelar”

**- Prompt(“ “):** Exibe uma janela com a mensagem e uma área de digitação abaixo dela, permitindo que o usuário digite algo, como seu nome ou qualquer outra coisa. Sempre retorna string, independentemente de ter sido digitado um número.

**- Document:** Faz a ligação do comando com o documento.

**- Write(‘ ’):** Escreve o texto dentro das aspas, crases, apóstrofos na página. Pode usar artifícios de HTML dentro desse comando a fim de alterar o texto, como por exemplo, colocar o comando <h1> </h1> ou <h2> </h2> para alterar ele.

**- N.toFixed(n de casas):** Fixa um número em determinado número de casas. “N” substituído pela variável do número e dentro dos parênteses a quantidade de casas depois da “,” que serão exibidas.

**- Replace(‘ ’, ’ ’):** Troca uma coisa por outra, primeiro o que quer ser substituído no primeiro apóstrofo e o que irá substituir no segundo. Da para usar com o comando N.toFixed(n) – N.toFixed(n).replace(‘.’ , ‘,’) dessa maneira substituindo o ponto dos números reais para vírgula, como nós usamos no Brasil.

**- InnerText:** Pega o texto de um lugar e escreve no documento se quiser, mas sem formatação html, por exemplo, se alguma palavra do texto html estiver em negrito, o negrito não será pego junto.

**- InnerHTML:** Pega o texto com a formatação em HTML completa.

**VARIÁVEIS**

**Comandos Relacionados à Variáveis:**

**- Number.parseInt(n)/ParseInt(n):** Faz a conversão de uma variável qualquer para um número inteiro.

**- Number.parseFloat(n):** Faz a conversão de uma variável qualquer para um número real.

**- Number(n):** Faz com que o JS veja qual o tipo de variável que foi recebida e converta o valor número inteiro ou real sozinho. Não descarta os comandos anteriores. Existem casos em que tem a necessidade de se forçar a conversão de um número para inteiro ou para real.

**- String(n):** Converte uma variável em uma string, substituindo o “N”.

**- N.toString():** Converte uma variável em uma string, mas ela vem primeiro, substituindo o “N”.

**- S.length:** Exibe o tamanho de uma string, a quantidade de letras dela. A variável em que a string estiver irá substituir o “S”.

**- S.toUpperCase():** Joga tudo da string para letras maiúsculas. Variável da string substitui o “S”.

**- S.toLowerCase():** Joga tudo da string para letras minúsculas. Variável da string substitui o “S”.

**- N1.toLocaleString(‘pt-BR’, {style: ‘currency’, currency: ‘BRL’}):** Converte a variável n1 para a moeda do Brasil, colocando R$ antes do número. Ele diz que quer localizar a string para o Brasil e o estilo vai ser currency que é dinheiro, e o currency vai aparecer em BRL: real. Se quiser dólar é só trocar por USD, ou EUR para euro.

**- Sinal de +**: Significa concatenação. É ele que vai grudar uma estrutura com uma variável, como por exemplo: **window.alert(“Bem-vindo, ‘ + nome)**. Porém, o + também serve para adição, mas como diferenciar? O JS só vai fazer a concatenação, quando houver 2 strings (string + string: concatenação). E consequentemente, só faz as adições quando forem 2 numbers (number + number: adição).

Apenas um sinal de “=” significa recebe, e não igual.

Existem 3 formas de delimitar uma string (sequência de caracteres) no JS e cada uma tem uma função diferente, são elas:

- **“ ”:** Textos padrões;

- **‘ ‘:** Textos padrões;

**- ` `:** Usado com template String, quando se usa os place holders.

Para criar uma variável usa-se o comando “**var**” seguido do nome da variável (identificador); do ”=” (recebe) e do que ela irá receber, seja uma string ou um valor. Ex.: **var nome = window.prompt(‘Qual o seu nome?’)**

**Regras para determinar um identificador:**

1. Podem começar com **letra**, **$**  ou **\_**.
2. Não podem começar com **números**.
3. Pode usar **letras** ou **números**.
4. Pode usar **acentos** e **símbolos**.
5. Não podem conter **espaços**.
6. Não podem ser **palavras** **reservadas**.

**Dicas para escolha de identificadores:**

1. **Maiúsculas** e **Minúsculas** fazem diferença.
2. Use **nomes** **coerentes** para as variáveis: se for idade, escreva idade; salário, escreva salário...
3. Evite se tornar o **“programador alfabeto”** (todas as variáveis são só letras) ou o **“programador contador”** (todas as variáveis são n1, n2, n3...).

Variáveis são dados.

**Tipos de variáveis:**

**- Number:** São números, independentemente de serem floats (números flutuantes, ou seja, reais (tem esse nome por causa dos point floats, ou, pontos, que separa o número inteiro das casas decimais dos números (Nos USA usa-se ponto no lugar da virgula, daí o nome))) ou números inteiros.

**- String:** São sequências de caracteres. Tudo que é escrito. Podendo ser letras ou números, como, seu Nome, CPF, telefone, RG... São colocados dentro de aspas ( **“ ”** ), apóstrofo ( **‘ ‘** ) ou crases ( **` `** ).

**- Boolean:** Verdadeiro (true) e falso (false). Também chamados de valores booleanos.

Estes são os 3 tipos primitivos que vamos usar inicialmente, mas existem muitos outros, como por exemplo:

**- Infinity e NaN (Not a number):** Valores internos muito importantes do Number.

**- Null:** Significa que a variável recebeu um valor nulo, ou seja, valor nenhum.

**- Undefined:** Indefinido.

**- Object:** Tem um monte de outros tipos internos. Inclusive o null, que é considerado um object.

**- Array:** Usado para vetores. Tipo interno do object.

**- Function.**

Para trabalhar com todos esses tipos, existe o comando **typeof**. Ele exibe qual é o tipo da variável que está sendo trabalhada. Basta colocar o comando e o identificador da variável depois.

Quando se determina uma variável você não diz qual é o tipo, apenas que é uma variável. O próprio programa define o tipo dela. Você pode definir um número para uma variável e ela se torna o tipo number, mas se definir ela novamente com uma string, ela altera seu tipo para string.

**FORMATANDO STRINGS**

Ao invés de usar o + para fazer todas as concatenações, pode-se usar a estrutura de Template String – **${n}**, sempre dentro das crases quando for escrever um texto, deixando assim mais fácil e mais bem formatado para visualização e menos confusão durante a escrita. Quando for usar concatenação, pode-se colocar o texto dentro de aspas ou apostrofo também, além de crase, porém, para template string com os place holders, usa-se apenas as crases.

**FORMATANDO NÚMEROS**

**OPERADORES**

**Tipos de operadores:**

**- Aritméticos:** São usados para fazer operações binárias, para fazer cálculos.

1. **+:** Adição

**2. -:** Subtração

**3.** **\*:** Multiplicação

**4. /:** Para o JS é a divisão real, ou seja, 5 / 2 = 2.5

**5. %:** Divisão inteira. Aquela que não usa vírgula, ou seja, a divisão para quando precisar colocar ela. O resultado que é considerado é o resto da divisão, e não o divisor. Ex.: 5 % 2 = 1

**6. \*\*:** Potência. Ex.: 5 \*\* 2 = 25. Equivale ao **“ ^ ”**.

**Ordem de precedência:**

**- ();**

**- \*\*;**

**- \* e / e %.** Quando os 3 aparecem na mesma conta a ordem de precedência é da esquerda para direita, quem aparecer primeiro;

**- + e -;**

**- Atribuição:** Atribui o valor a uma variável. Ex.: **var n = 3.** Pode-se fazer atribuições complexas, como: var n = 4+5, ou, var n = 9 % 2, ou, var n = 4 + 5 \* 9 % 2. Ou pode-se fazer atribuições simples, como: **var n = 3, n = n + 4, n = n - 5**.... Essa mesma forma de atribuição pode ser alterada para **var n = 3, n += 4, n -= 5**... Significa a mesma coisa que o exemplo simples anterior. Esse tipo de simplificação funciona para grande maioria de linguagens de programação, sejam elas, C, C++, Java etc. Só funciona quando você está fazendo uma operação onde uma variável a recebe junto com a operação, seja ela soma, multiplicação etc.

**- Incremento:** Quando se quer fazer uma variável a recebe mais ou menos um, pode-se simplificar ainda mais, colocando dois sinais de + ou de – na frente da variável. Ex.: **x = x + 1**, fica: **x ++**. Ou então, **x = x – 1**, fica: **x --**. No nodejs, quando se faz esse tipo de operação o programa vai mostrar o valor de n atual e não o somado ou subtraído, para exibir o novo valor da variável precisa mandar ele exibir dnv, porém, pode colocar os sinais antes variável. Dessa forma, o valor mostrado será o resultado da operação. A maioria das linguagens de programação aceita esse tipo de operação.

**- Relacionais:**

1. **>:** Maior
2. **<:** Menor
3. **>=:** Maior ou igual
4. **<=:** Menor ou igual
5. **==:** Igual
6. **===:** Identidade. Testa se os 2 itens têm o mesmo valor e se são do mesmo tipo. Ex.: 5 == 5: true, 5 == ‘5’: true, 5 === ‘5’: false (mesmo valor, porém de tipos diferentes. Um é number e o outro é string), 5 === 5: true.
7. **!=:** Diferente/não igual.
8. **!==:** Desigual restrito, mesma coisa que o “===” só que para diferente.

Para toda expressão que tenha um operador relacional ligado a ela, o resultado vai sempre ser um valor booleano, do tipo verdadeiro ou falso. Ex.: 5 > 2: true, 7 < 4: false, 8 >= 8: true,9 <= 7: false, 5 == 5: true, 4 != 4: false.

**- Lógicos:**

1. **!:** Negação, quer dizer não. Ex.: Me dê uma caneta, desde que não seja azul. Pode ser qualquer cor, menos azul. Tratado como operador unário, ou seja, depois da ! ou eu coloco true ou false, ou uma expressão que seja true ou false. Ex: !true: false, pois uma coisa que não é verdadeira é falsa. Ele também se aplica à !false: true, pois uma expressão que não é false, é true.
2. **&&:** Conjunção, ou “E” lógico. Ex.: Me dê uma caneta azul e vermelha. Só fico feliz se receber as 2. Operador binário, ou seja, tem 2 operadores lógicos, um de cada lado e vão me dar o resultado lógico dessa expressão. Ex.: true && true: true, true && false: false, false &&t true: false...
3. **||:** Disjunção, ou “OU” lógico. Ex.: Me dê uma caneta azul ou vermelha. Se entregar as 2, ótimo, se entregar só uma das 2 está bom também. Se não receber nenhuma, aí não fico satisfeito. Operador binário, ou seja, tem 2 operadores lógicos, um de cada lado e vão me dar o resultado lógico dessa expressão. Ex.: true || true: true, true || false: true, false || true: true, false || false: false.

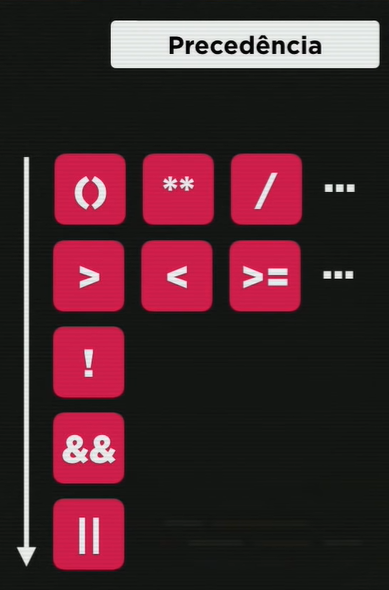
A ordem de execução é: primeiro o NÃO, depois o E, e por último o OU.

**- Ternário:** Tem esse nome por ter 3 partes: teste **?** (Teste lógico, true ou false) true **:** (O que vai acontecer quando esse teste lógico for verdadeiro) false (O que vai acontecer quando o teste lógico for falso). Ex.: **média >= 7.0 ? “Aprovado” : “Reprovado”**, se a média for maior ou igual a 7, irá aparecer “aprovado” escrito, se for menor, irá aparecer “reprovado”.

1. **?:**
2. **::**

Existem muitas outras famílias de operadores, mas as que iremos focar serão essas. Inclusive o **typeof** é um tipo de operador.

Primeiro são feitos todos os operadores aritméticos (Ordem de precedência: Parênteses, potência, divisão real, divisão inteira e multiplicação (estes 3 são lidos o que vier primeiro, da esquerda para a direita), soma e subtração), depois os relacionais (Ordem de precedência: O que vier primeiro. Leitura feita da esquerda para a direita), os lógicos (ordem de precedência: Não, E, OU) e por último o ternário.



**Entendendo o DOM**

É um acrônimo **D**ocument **O**bject **M**odel. Conjunto de objetos dentro do navegador que dará acesso aos componentes internos do website. Não funciona dentro do Nodejs. Está presente quando rodando JS dentro do navegador.

**Arvore DOM**

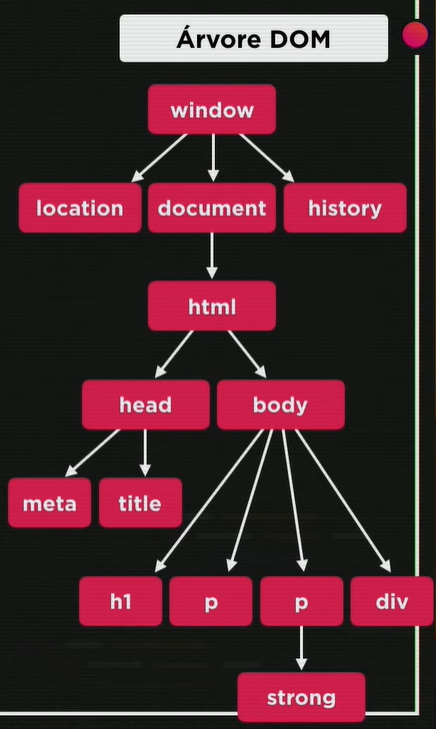
Começa da raiz dentro do navegador, chamada de window. Tudo dentro do JS está dentro desse objeto. A janela do navegador é um objeto DOM e se chama window.

Dentro do window existem outros 3 objetos:

**- Location:** Diz qual é a localização do site, a URL, qual é página atual, a página anterior.

**- Document:** Documento atual. Tem outro objeto dentro deste, o:

- HTML: Que é todo o conteúdo html do site. Dentro dele, tem outros 2: HEAD e BODY. São child, filhos do html, que por sua vez é um parent. Dentro de HEAD existem tags, que são filhos dele: Meta, title, dentre outros. E dentro de body existem outras tags, que são no documento atual (ex005 – aula 9): H1, P, P e um Div. O segundo parágrafo (P), por sua vez, também tem um filho, que é o strong. E assim por diante. Essa é a arvore hierárquica do site que acabamos de criar.



**- History:** Vai guardando da onde você veio, para onde vai etc. O que facilita a navegação dentro do site.

Existem muitos outros objetos, aqui só foram representados 3.

**Métodos de acesso para selecionar elementos na árvore DOM:**

1. **Por Marca:** Ou por Tag. Usa-se o comando **getElementsByTagName()**. Quando se usa esse comando, pode-se selecionar mais de 1 objeto, pois existem vários objetos do mesmo tipo, com mesma tag, por exemplo, no nosso código temos 2 P.
2. **Por ID:** Usa-se o comando: **getElementById()**. Adiciona-se ID colocando **id=”nome do ID”** dentro das tags do HTML como no ex005 – aula 09
3. **Por Nome:** Usa-se o comando: **getElementsByName()**.
4. **Por Classe:** Usa-se o comando: **getElementsByClassName()**.
5. **Por seletor:** Por css. Usa-se os comandos: **querySelector()** e **querySelectorAll()**(Plural). Toda div/ID é representada por uma # e toda classe por um ponto “.”. Função recente. Navegadores antigos que rodam ECMAScript versões mais antigas, não terão suporte a ele.

Quando se tem mais de 1 objeto da mesma classe, marca, nome... Coloca-se **[ ]** após o comando e o número referente ao objeto que se quer selecionar, sempre começando com o 0, ou seja, se tem 2 parágrafos, ao final do comando coloca-se [0] para pegar o primeiro, [1] para pegar o segundo e assim sucessivamente. Quando o comando estiver no singular não precisa colocar qual é o elemento depois do comando, mas quando estiver precisa.

**EVENTOS**

É tudo o que pode acontecer com essa div:

<DIV>

Por exemplo, os mais comuns são eventos de mouse, como mover o mouse até chegar dentro da div, quando isso ocorre dispara um evento: **mouseenter**, que é quando o mouse entra na div. Outro é quando continua movendo o mouse dentro da div, nesse momento seria disparado diversas vezes o evento: **mousemove**. Se clicar e segurar, tem o disparo do evento: **mousedown**. No momento em que solta, dispara: **mouseup**. Quando clica rapidamente: **click**. Quando o mouse sai da div: **mouseout**.

**FUNÇÃO**

Conjunto de códigos/linhas que vão ser executadas apenas quando os eventos acontecerem. Para executar um bloco somente quando um evento acontecer, ele primeiro tem que estar dentro de **{ }**, apenas aí ele pode ser chamado de bloco. Ele tem que ser nomeado como uma **Function** antes dos {}. No JS existem as funções anônimas, que são as que não tem nome, mas, para que o método funcione ela tem que ser nomeada, e geralmente, essas funções são ações que se podem fazer. Então, para resumir, tem que ficar assim: **Function ação(nome da ação que vai acontecer) (pode-se colocar parâmetros dentro destes parênteses. Seja 1 ou vários) {“bloco”}.**

**ERRO:** Quando o programa der erro o JS não vai falar qual foi, apenas não irá rodar. Para descobrir qual é: clique com o botão direito na página -> inspecionar. O devtools vai mostrar qual é o erro.

Bloco: conjunto de linhas

**COMANDOS**

**- AddEventListener (‘ ’, ):** Faz o programa ficar analisando se o evento dentro das ‘ ’ aconteceu, para daí executar a função escrita depois da vírgula. Ex.: **addEventListener(‘mouseenter’, entrar)**. Geralmente associada a uma variável que recebeu uma div.

**CONDIÇÕES**

**Condições simples:**

Basicamente usadas para dizer que se algo acontecer, faz isso se não, faz aquilo.

**- if (Condição) {}:** Igual Arduino. Condição simples

**Condições compostas e aninhadas:**

As condições aninhadas se tratam basicamente de várias condições dentro de outras condições, formando como se fosse um ninho. Por exemplo, se eu clicar num botão, ele muda o escrito, se eu continuar com o mouse apertado ele muda o botão para a cor azul, se não ele muda para vermelho.

**- Else if (Condição){}:** Comando colocado abaixo do if quando existem mais de 2 opções/caminhos a serem seguidos. Depois do if e *antes* do *else*.

**- Else {}:** ” “. Condição composta

**Condições Multiplas:**

Usada para valores fixos, super utilizada. Não serve muito para intervalo de valores.

Tem a possibilidade, além do sim e não, Muito útil para situações pontuais e específicas, analisando diversos pontos e convergendo eles a um único novamente.

**- switch (expressão){}:**

**- case** valor 1**:**

**break**

**- case** valor 2**:**

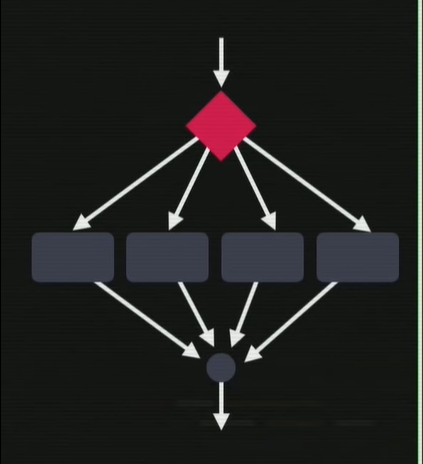
**break**

**- case** valor 3**:**

**break**

**- default:**

**break**



Nessa condição ele vai testar todos os valores antes de convergir para uma opção, se nenhuma delas for verdade ele vai executar o default. Todos os breaks são obrigatórios para o funcionamento do switch. O break serve para parar a execução dos códigos no bloco. O seu comando resultou em um dos valores, logo, ele irá executar todos os comandos do bloco contidos na case valor 2, por exemplo. Se o break estiver ali, ele irá diretamente para o fim da condição e seguir o programa, caso ele não esteja, irá executar todos os comandos do case valor 3/4/5 e assim por diante até que ele encontre um break.

