PROGRAMAÇÃO ORIENTADA A OBJETOS

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB

POLITÉCNICO
DO PORTO
ESCOLA
SUPERIOR
DE MEDIA ARTES E
DESIGN

POLITÉCNICO DO PORTO
ESCOLA SUPERIOR DE MEDIA ARTES E DESIGN

P.PORTO

PROGRAMAÇÃO ORIENTADA A OBJETOS

TECNOLOGIAS E SISTEMAS DE INFORMAÇÃO PARA A WEB

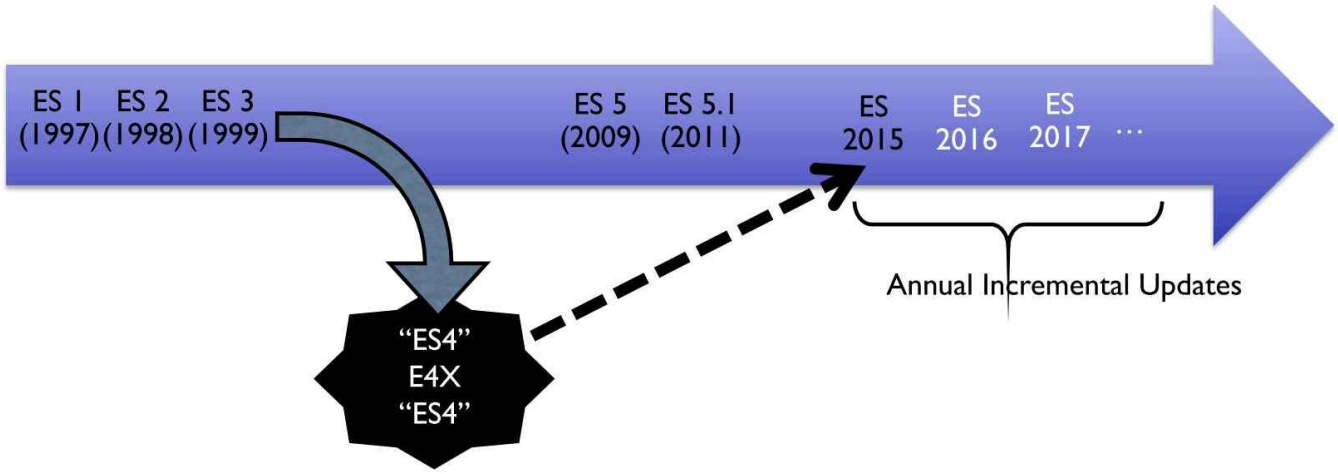
2017-18

MÓDULO II – INTRODUÇÃO AO JAVASCRIPT

ÍNDICE

1. INTRODUÇÃO AO JAVASCRIPT

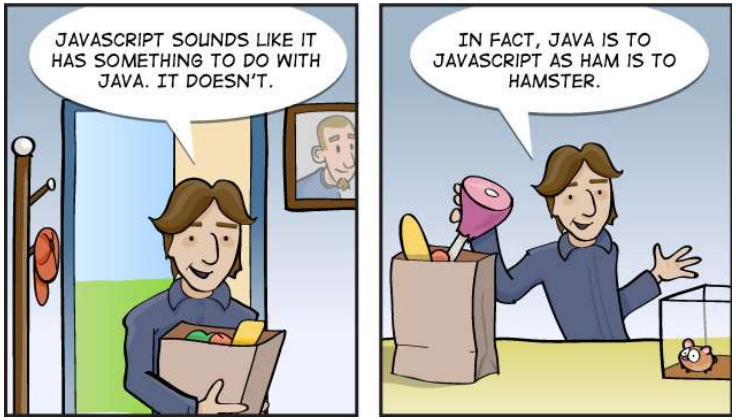
- ECMASCRIPT - ESPECIFICAÇÃO DE LINGUAGEM DE SCRIPT PADRONIZADA PELA **ECMASCRIPT INTERNATIONAL**



- A ESPECIFICAÇÃO ES2015 (6ª EDIÇÃO DO STANDARD ECMA-262) É UMA DAS MAIS POPULARES
- DESTA ESPECIFICAÇÃO NASCERAM VÁRIAS IMPLEMENTAÇÕES (JAVASCRIPT, JSCRIPT AND ACTIONSCRIPT)
- NESTA SEBENTA DESCREVE-SE A SUA IMPLEMENTAÇÃO EM **JAVASCRIPT**

1. INTRODUÇÃO AO JAVASCRIPT

- LINGUAGEM DE PROGRAMAÇÃO INTERPRETADA
- CRIADA POR **BRENDAN EICH** (PROGRAMADOR DA NETSCAPE) EM 1995
- COMEÇOU POR SE CHAMAR MOCHA, DEPOIS LIVESCRIPT, E FINALMENTE, **JAVASCRIPT**



- ORIGINALMENTE IMPLEMENTADA PARA SER USADA EM BROWSERS (LADO CLIENTE)
- HOJE EM DIA JÁ É USADA
  - NO SERVIDOR WEB (NODE.JS)
  - PARA DESKTOPS (ELECTRON, …)
  - PARA MOBILE (REACT NATIVE, …)

1. INTRODUÇÃO AO JAVASCRIPT

INTEGRAÇÃO NUMA PÁGINA HTML

- PARA USAR JAVASCRIPT NUMA PÁGINA HTML USE O ELEMENTO **<SCRIPT>**
  - DE FORMA DIRETA NO ELEMENTO **<HEAD>** OU **<BODY>**

```
<script>
  // código JavaScript
</script>
```

  - COMO REFERÊNCIA A FICHEIRO EXTERNO (FICHEIRO .JS)

```
<script src="myscript.js" />
```

    - SCRIPT VAI COMPORTAR-SE COMO SE ESTIVESSE ONDE A TAG SCRIPT ESTÁ LOCALIZADA
    - O FICHEIRO EXTERNO NÃO PODE CONTER O ELEMENTO <SCRIPT>
    - VANTAGENS:
      - SEPARA O HTML DO CÓDIGO
      - TORNAM O HTML E O JAVASCRIPT MAIS FÁCEIS DE LER/MANTER
      - PERMITE AO BROWSER FAZER CACHE DOS FICHEIROS JS ACELERANDO O CARREGAMENTO DAS PÁGINAS

1. INTRODUÇÃO AO JAVASCRIPT

DEPURAÇÃO DE DADOS

- ESCRITA PARA A CONSOLA DO BROWSER
  - ESCREVA O SEGUINTE TRECHO DE CÓDIGO NO ELEMENTO **<HEAD>** OU **<BODY>**

```
<script>
  console.log(5 + 6)
</script>
```

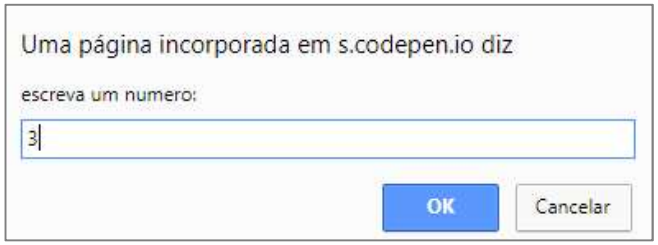
  - ABRA UM BROWSER (CHROME)
  - ATIVE AS FERRAMENTAS DO PROGRAMADOR (NO CHROME É **CTRL+SHIFT+I**)
  - SELECIONA O SEPARADOR **CONSOLE**
  - VISUALIZE OS RESULTADOS QUE SURGEM NA CONSOLA

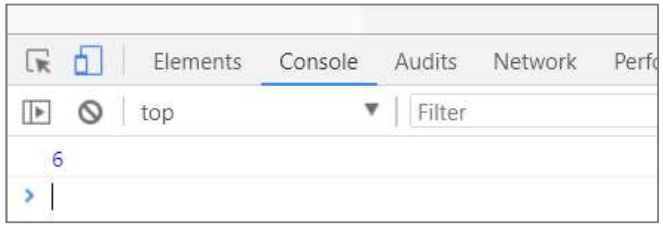
1. INTRODUÇÃO AO JAVASCRIPT

DEPURAÇÃO DE DADOS

- EXIBIÇÃO DE MODAL ATRAVÉS DE FUNÇÃO **PROMPT**
  - ESCREVA O SEGUINTE TRECHO DE CÓDIGO NO ELEMENTO **<HEAD>** OU **<BODY>**

```
<script>
  var num = prompt(“escreva um numero:”)
  console.log(parseInt(num) * 2)
</script>
```





ÍNDICE

2. SINTAXE

- CONJUNTO DE REGRAS PARA ESCREVER PROGRAMAS
- CADA ESPECIFICAÇÃO DE LINGUAGEM DEFINE A SUA PRÓPRIA SINTAXE
- CÓDIGO JAVASCRIPT ORGANIZADO POR INSTRUÇÕES EXECUTADAS NA ORDEM PELA QUAL FORAM DE DEFINIDAS

```
var price = 5
console.log(price)
```

- SE VÁRIAS NA MESMA LINHA, ESTAS DEVEM SER SEPARADAS POR PONTO E VÍRGULA (;)

```
var price = 5; console.log(price)
```

- UM PROGRAMA DE JAVASCRIPT PODE SER COMPOSTO POR:
  - **VARIÁVEIS** - BLOCO DE MEMÓRIA COM NOME QUE PODE ARMAZENAR VALORES PARA O PROGRAMA
  - **LITERAIS** - VALORES CONSTANTES / FIXOS
  - **OPERADORES** - SÍMBOLOS QUE DEFINEM COMO OS OPERANDOS SERÃO PROCESSADOS
  - **KEYWORDS** - PALAVRAS COM SIGNIFICADO ESPECIAL NO CONTEXTO DUMA LINGUAGEM (VAR, FOR, ETC.)
  - **MÓDULOS** - BLOCOS DE CÓDIGO QUE PODEM SER REUTILIZADOS EM DIFERENTES PROGRAMAS / SCRIPTS
  - **COMENTÁRIOS** - USADO PARA MELHORAR A LEGIBILIDADE DO CÓDIGO
  - **IDENTIFICADORES** - NOMES DADOS AOS ELEMENTOS DUM PROGRAMA, COMO VARIÁVEIS, FUNÇÕES, ETC.

2. SINTAXE

COMENTÁRIOS

- USADOS PARA TORNAR O CÓDIGO MAIS LEGÍVEL
- USADOS TAMBÉM PARA PREVENIR A EXECUÇÃO DE CÓDIGO QUANDO TESTANDO OUTRAS ALTERNATIVAS

- ÂMBITOS:
  - LINHA SIMPLES (TEXTO A PARTIR DE // ATÉ AO FINAL DA LINHA É IGNORADO)

```
// um comentário
var y = 5
var z = y + 1 // outro comentário
```

  - LINHAS MÚLTIPLAS (AGREGADO POR /* E */)

```
var y = 5
/*
var z = y + 1
console.log(z)
*/
```

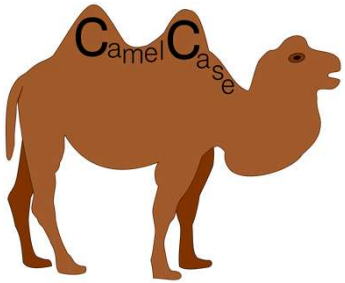
ÍNDICE

2. SINTAXE

VARIÁVEIS

- SÃO CONTENTORES PARA ARMAZENAMENTO DE VALORES

- OS NOMES DAS VARIÁVEIS SÃO CHAMADOS DE **IDENTIFICADORES:**
  - PODEM SER NOMES CURTOS (COMO X E Y), OU NOMES MAIS DESCRITIVOS (IDADE, SOMA, TOTALVOLUME)
  - CONVENÇÕES DE NOMEAÇÃO:
    - camelCase
    - snake_case
    - spinal-case



  - PODEM CONTER LETRAS, NÚMEROS, UNDERSCORES E SINAIS DE DÓLAR
  - NÃO PODEM COMEÇAR COM UM NÚMERO
  - SÃO SENSÍVEIS A MAIÚSCULAS E MINÚSCULAS (y E Y SÃO VARIÁVEIS DIFERENTES)
  - NÃO PODEM SER USADAS PALAVRAS RESERVADAS (**KEYWORDS** EM JAVASCRIPT)

2. SINTAXE

VARIÁVEIS

- SÃO CONTENTORES PARA ARMAZENAMENTO DE VALORES

- **DECLARAÇÃO:**
  - USO DA PALAVRA-CHAVE **VAR** PARA DECLARAÇÃO DE UMA VARIÁVEL
  - APÓS A DECLARAÇÃO, A VARIÁVEL NÃO TEM VALOR (TECNICAMENTE ELA TEM O VALOR **UNDEFINED**)

```
var price
```

- **ATRIBUIÇÃO:**
  - PARA ATRIBUIR UM VALOR A UMA VARIÁVEL , USE O OPERADOR DE ATRIBUIÇÃO =

```
var price = 5
console.log(price)
----------------------------------------
5
```

2. SINTAXE

VARIÁVEIS

- UMA VARIÁVEL JAVASCRIPT PODE CONTER UM VALOR DE QUALQUER TIPO DE DADOS
- AO CONTRÁRIO DE OUTRAS LINGUAGENS, NÃO PRECISA DIZER AO JAVASCRIPT DURANTE A DECLARAÇÃO DE VARIÁVEL QUAL O TIPO DE VALOR QUE A VARIÁVEL IRÁ MANTER
- O TIPO DE VALOR DE UMA VARIÁVEL PODE MUDAR DURANTE A EXECUÇÃO DUM PROGRAMA E O JAVASCRIPT CUIDA DISSO AUTOMATICAMENTE
- ESTE RECURSO É DENOMINADO COMO **TIPAGEM DINÂMICA**

```
var price = 5
console.log(price)
price = "car"
console.log(price.toUpperCase())
-------------------------------------------------
5
CAR
```

2. SINTAXE

VARIÁVEIS

- O **ESCOPO DE UMA VARIÁVEL** É A REGIÃO DO PROGRAMA ONDE É DEFINIDA

- TRADICIONALMENTE, O JAVASCRIPT DEFINE APENAS DOIS ESCOPOS:
  - **GLOBAL** - VARIÁVEL PODE SER ACEDIDA A PARTIR DE QUALQUER PARTE DO CÓDIGO JAVASCRIPT
  - **LOCAL** - VARIÁVEL PODE SER ACEDIDA A PARTIR DE UMA FUNÇÃO ONDE É DECLARADA

```
var num = 10
function test() {
   var num = 100
   console.log(num)
}
console.log(num)
test()
console.log(num)
-------------------------------------------------
10
100
10
```

2. SINTAXE

VARIÁVEIS

- **HOISTING**
  - A DECLARAÇÃO DUMA VARIÁVEL EM RUNTIME É MOVIDA PARA O TOPO DO ÂMBITO ONDE FOI DEFINIDA

```
var idade = 20
if(idade == 30) {
  var teste = “ola”
  console.log(teste)
}
console.log(teste)
-------------------------
undefined
```

```
var idade = 20
var teste  // hoisting
if(idade == 30) {
  teste = “ola”
  console.log(teste)
}
console.log(teste)
-------------------------
undefined
```

2. SINTAXE

VARIÁVEIS

- O **ES6** ADICIONOU UM NOVO ESCOPO: **BLOCO**

- O **ESCOPO DE BLOCO** RESTRINGE O ACESSO DE UMA VARIÁVEL AO BLOCO NO QUAL ELE É DECLARADO
  - **VAR** ATRIBUI UM **ESCOPO DE FUNÇÃO À VARIÁVEL**
  - **LET** PERMITE QUE O SCRIPT RESTRINJA O ACESSO À VARIÁVEL **AO BLOCO ENVOLVENTE MAIS PRÓXIMO**

```
var idade = 20
if(idade == 30) {
  let teste = “ola”
  console.log(teste)
}
console.log(teste)
-------------------------
Uncaught ReferenceError: teste is not defined
```

```
var idade = 30
if(idade == 30) {
  let teste = “ola”
  console.log(teste)
}
console.log(teste)
-------------------------
ola
Uncaught ReferenceError: teste is not defined
```

2. SINTAXE

VARIÁVEIS

- O **ES6** ADICIONOU UM NOVO TIPO DE DECLARAÇÃO DE VARIÁVEIS: **CONST**
  - **CONST** CRIA UMA REFERÊNCIA SOMENTE DE LEITURA PARA UM VALOR
  - VALOR DUMA CONSTANTE NÃO PODE MUDAR POR REATRIBUIÇÃO E NÃO PODE SER REDECLARADO
  - EM TERMOS DE ESCOPO COMPORTA-SE COMO A KEYWORD LET

```
let idade = 72
const MAX = 60
if(idade > MAX) {
  console.log(“maior de idade”)
}
MAX = 65
-------------------------
maior de idade
Uncaught TypeError: Assignment to constant variable.
```

ÍNDICE

2. SINTAXE

OPERADORES

- **OPERADORES ARITMÉTICOS**
  - SÃO USADOS PARA REALIZAR OPERAÇÕES ARITMÉTICAS EM NÚMEROS (LITERAIS OU VARIÁVEIS)
  - EXEMPLOS:
    - +, -, *, / - OPERAÇÕES BÁSICAS
    - % - RESTO
    - ++ E -- - INCREMENTO E DECREMENTO

- **OPERADORES DE ATRIBUIÇÃO**
  - ATRIBUEM VALORES A VARIÁVEIS
  - EXEMPLOS:
    - = - ATRIBUIÇÃO
    - +=, -=, *=, /=, %= - OPERAÇÃO E ATRIBUIÇÃO

- **OPERADORES STRING**
  - + - CONCATENAÇÃO

```
let name = “Rui”
let surname = “Silva”
let fullName = name + “ “ + surname
console.log(fullNname)
------------------------------
Rui Silva
```

2. SINTAXE

OPERADORES

- **OPERADORES STRING (CONT.)**
  - ADIÇÃO DE UM NÚMERO COM UMA STRING RESULTA NUMA STRING

```
let x = 5 + 5
let y = "5" + 5
let z = "ola" + 5
console.log(x)
console.log(y)
console.log(z)
--------------------------------
10
55
ola5
```

- **OPERADORES RELACIONAIS**
  - == IGUAL A
  - != DIFERENTE
  - > MAIOR
  - < MENOR
  - >= MAIOR OU IGUAL A
  - <= MENOR OU IGUAL A

2. SINTAXE

OPERADORES

- **OPERADORES LÓGICOS**
  - && (CONJUNÇÃO)
  - || (DISJUNÇÃO)
  - ! (NEGAÇÃO)

- **OPERADOR CONDICIONAL**
  - DEFINE TESTES SIMPLES
  - SINTAXE:
    TEST ? EXPR1 : EXPR2

```
let num = 2
let result = num > 0 ? “positivo” : “negativo”
console.log(result)
-------------------------------
positivo
```

- **OPERADOR DE TIPO**
  - OPERADOR UNÁRIO **TYPEOF**
  - RETORNA O TIPO DE DADOS DO OPERANDO
  - LISTA OS TIPOS DE DADOS E OS VALORES RETORNADOS PELO OPERADOR TYPEOF EM JAVASCRIPT
    - NUMBER: “number”
    - STRING: “string”
    - BOOLEAN: “boolean”
    - OBJECT: “object”

```
let num = 12
console.log(typeof num)
-------------------------------
number
```

ÍNDICE

2. SINTAXE

ESTRUTURAS DE DECISÃO

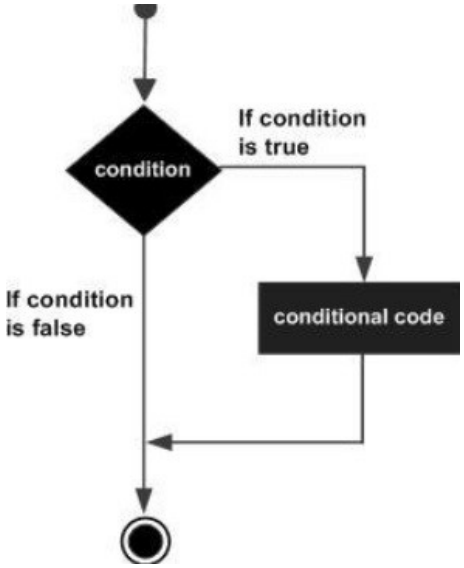
- EXECUTAM DIFERENTES AÇÕES BASEADAS EM DIFERENTES CONDIÇÕES
- EM JAVASCRIPT TEMOS AS SEGUINTES ESTRUTURAS:
  - **IF** - PARA ESPECIFICAR BLOCO DE CÓDIGO A SER EXECUTADO, SE DETERMINADA CONDIÇÃO É VERDADEIRA
  - **ELSE** - PARA ESPECIFICAR BLOCO DE CÓDIGO A SER EXECUTADO, SE A MESMA CONDIÇÃO É FALSA
  - **ELSE…IF** - PARA ESPECIFICAR UMA NOVA CONDIÇÃO PARA TESTAR, SE A PRIMEIRA CONDIÇÃO É FALSA
  - **SWITCH** - PARA ESPECIFICAR BLOCOS DE CÓDIGO ALTERNATIVOS A SEREM EXECUTADOS

2. SINTAXE

ESTRUTURAS DE DECISÃO

- **IF** - PARA ESPECIFICAR BLOCO DE CÓDIGO A SER EXECUTADO SE DETERMINADA CONDIÇÃO É VERDADEIRA

```
let num = 5
if(num > 0) {
  console.log("número é positivo!")
}
-----------------------------------
número é positivo!
```

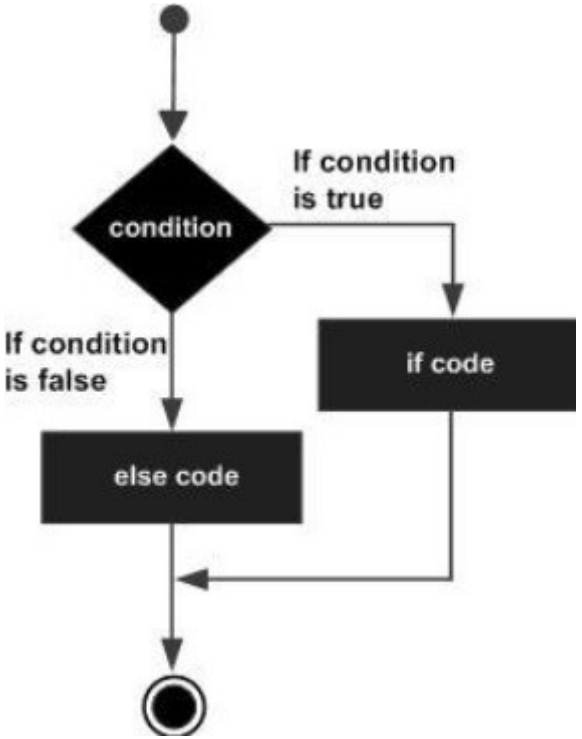


2. SINTAXE

ESTRUTURAS DE DECISÃO

- **ELSE** - PARA ESPECIFICAR BLOCO DE CÓDIGO A SER EXECUTADO, SE A MESMA CONDIÇÃO É FALSA

```
let num = 12
if(num % 2 == 0) {
  console.log(“par”)
} else {
 console.log(“ímpar”)
}
-----------------------------------
par
```



2. SINTAXE

ESTRUTURAS DE DECISÃO

- **ELSE…IF** - PARA ESPECIFICAR TESTES MÚLTIPLOS

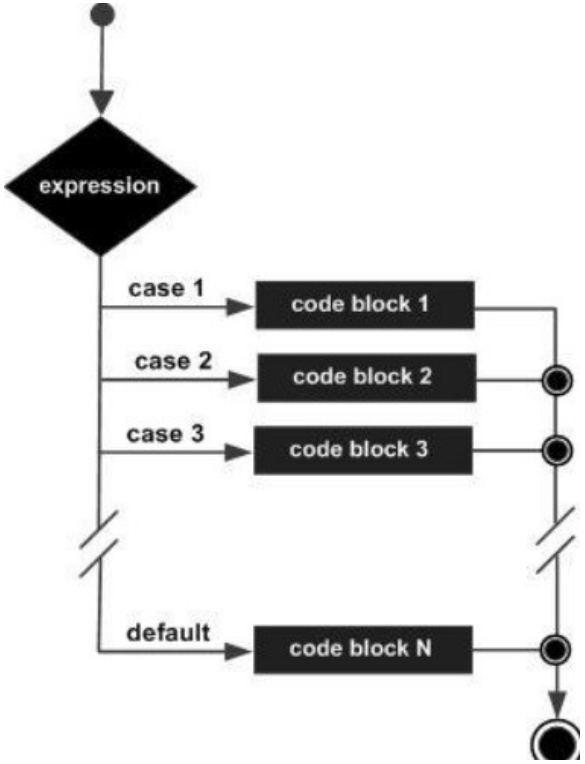
```
let num = 8
if(num >= 10) {
  console.log(“positiva”)
} else if(num >= 7) {
  console.log(“oral”)
} else {
  console.log(“negativa”)
}
-----------------------------------
oral
```

2. SINTAXE

ESTRUTURAS DE DECISÃO

- **SWITCH** – AVALIA UMA EXPRESSÃO, FAZ O MATCHING DO VALOR DA EXPRESSÃO PARA UMA CLÁUSULA **CASE** E EXECUTA AS INSTRUÇÕES ASSOCIADAS A ESSE CASO

```
let grade = "C"
switch(grade) {
 case "A": console.log("excelente"); break
 case "B": console.log("bom"); break
 case "C": console.log("médio"); break
 case "D": console.log("fraco"); break
 default: console.log("escolha inválida"); break
}
----------------------------------
médio
```

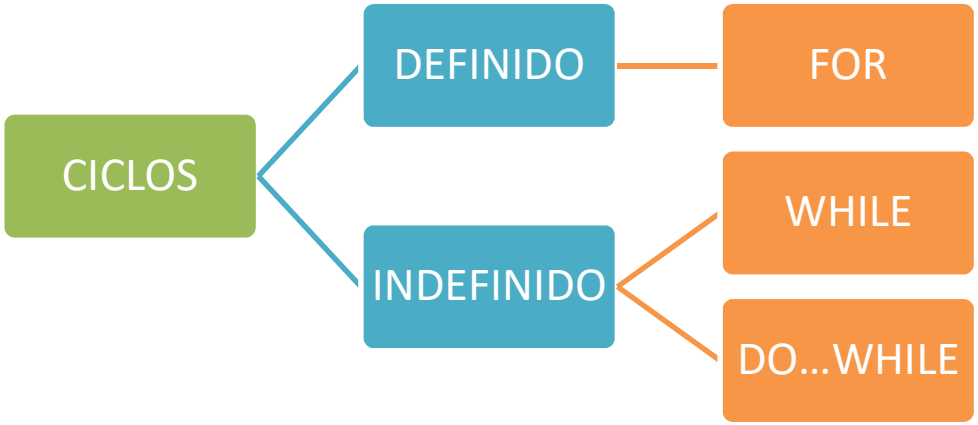


ÍNDICE

2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

- ÀS VEZES, CERTAS INSTRUÇÕES REQUEREM EXECUÇÃO REPETIDA
- OS **CICLOS** SÃO UMA MANEIRA IDEAL DE REPRODUZIR ESSE EFEITO
- UM CICLO REPRESENTA UM CONJUNTO DE INSTRUÇÕES QUE DEVEM SER REPETIDAS
- NO CONTEXTO DE UM CICLO, UMA REPETIÇÃO É DENOMINADA COMO UMA **ITERAÇÃO**
- CLASSIFICAÇÃO DE CICLOS:



2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

CICLO DEFINIDO

- UM CICLO CUJO NÚMERO DE ITERAÇÕES SÃO DEFINIDAS / FIXADAS
- O CICLO **FOR** É UMA IMPLEMENTAÇÃO DE UM CICLO DEFINIDO
- TRÊS VARIANTES:
  - **FOR** - EXECUTA O BLOCO DE CÓDIGO POR UM NÚMERO ESPECÍFICO DE VEZES
  - **FOR…IN** - É USADO PARA PERCORRER AS PROPRIEDADES DE UM OBJETO
  - **FOR…OF** - É USADO PARA ITERAR SOBRE ITERÁVEIS EM VEZ DE LITERAIS DE OBJETO

```
for (let i = 0; i < 5; i ++) {
 console.log(i)
}
-----------------------------------
0
1
2
3
4
```

2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

CICLO INDEFINIDO

- É USADO QUANDO O NÚMERO DE ITERAÇÕES NUM CICLO É INDETERMINADO OU DESCONHECIDO

- DUAS VARIANTES:
  - **WHILE** - EXECUTA AS INSTRUÇÕES CADA VEZ QUE A CONDIÇÃO ESPECIFICADA É AVALIADA COMO VERDADEIRA
  - **DO…WHILE** - É SIMILAR AO ANTERIOR, EXCETO QUE NÃO AVALIA A CONDIÇÃO PELA 1ª VEZ QUE O CICLO É EXECUTADO

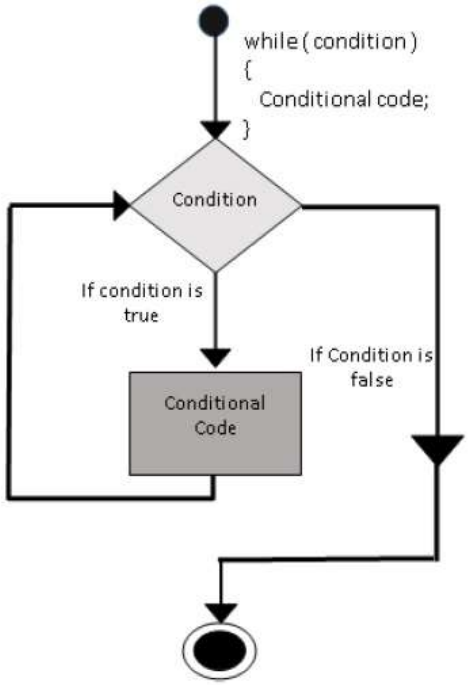
2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

CICLO WHILE

- EXECUTA AS INSTRUÇÕES SEMPRE QUE A CONDIÇÃO ESPECIFICADA É VERDADEIRA

```
let num = 5
let factorial = 1
while (num >= 1) {
 factorial = factorial * num
 num--
}
console.log("O factorial é " + factorial);
-------------------------------------------
O factorial é 120
```



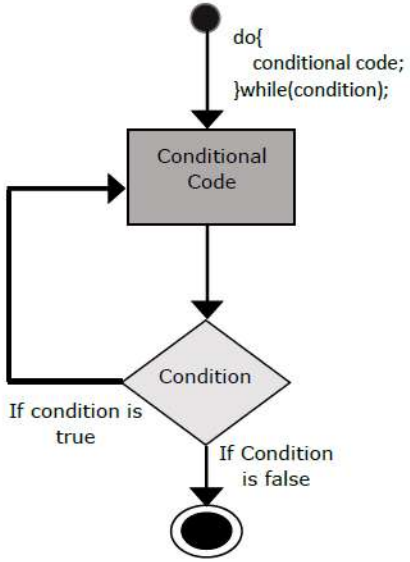
2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

CICLO DO…WHILE

- É SEMELHANTE AO CICLO WHILE, EXCETO QUE O CICLO **DO … WHILE** NÃO AVALIA A CONDIÇÃO PELA 1ª VEZ QUE O CICLO É EXECUTADO
- NO ENTANTO, A CONDIÇÃO É AVALIADA PARA AS ITERAÇÕES SUBSEQUENTES
- EM SUMA, O BLOCO DE CÓDIGO SERÁ EXECUTADO PELO MENOS UMA VEZ NESTE TIPO DE CICLOS

```
let n = 5
do {
 console.log(n)
 n--
} while(n >= 0)
-------------------------
5
4
3
2
1
0
```



2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

- OS CICLOS POSSUEM KEYWORDS DE **CONTROLO DE FLUXO**:
  - **BREAK** – USADO PARA SAIR DE UM CICLO
  - **CONTINUE -** IGNORA AS DECLARAÇÕES SUBSEQUENTES NA ITERAÇÃO ATUAL E LEVA O CONTROLO DE VOLTA AO INÍCIO DO CICLO

```
let i = 1
while(i <= 10) {
 if (i % 5 == 0) {
  console.log("O 1º múltiplo de 5 entre 1 e 10 é " + i)
  break     // Sai do ciclo se o 1º múltiplo é encontrado
 }
 i++
}
-------------------------
O 1º múltiplo de 5 entre 1 e 10 é 5
```

2. SINTAXE

ESTRUTURAS DE REPETIÇÃO

- OS CICLOS POSSUEM KEYWORDS DE **CONTROLO DE FLUXO**:
  - **BREAK** – USADO PARA SAIR DE UM CICLO
  - **CONTINUE -** IGNORA AS DECLARAÇÕES SUBSEQUENTES NA ITERAÇÃO ATUAL E LEVA O CONTROLO DE VOLTA AO INÍCIO DO CICLO

```
let count = 0
for(let num = 0; num <= 20; num++) {
 if (num % 2 == 0) {
  continue
 }
 count++
}
console.log(“A contagem de nºs ímpares entre 0 e 20 é “ + count)
-------------------------
A contagem de nºs ímpares entre 0 e 20 é 10
```

ÍNDICE

2. SINTAXE

FUNÇÕES

- SÃO OS BLOCOS DE CONSTRUÇÃO DO CÓDIGO LEGÍVEL, SUSTENTÁVEL E REUTILIZÁVEL
- SÃO DEFINIDAS USANDO A PALAVRA-CHAVE **FUNCTION**

```
// definição da função
function test() {
  console.log("isto é um teste")
}
test() // invocação da função
-------------------------------------------------
isto é um teste
```

- FUNÇÕES PODEM SER CLASSIFICADAS COMO
  - FUNÇÕES **COM RETORNO**
  - FUNÇÕES **PARAMETRIZADAS**

2. SINTAXE

FUNÇÕES COM RETORNO

- RETORNAM UM VALOR DE VOLTA A QUEM A INVOCOU
- USO DA KEYWORD **RETURN**

```
function saudacao() {
  return("olá mundo!")
}
let msg = saudacao()
console.log(msg)
-------------------------------------------------
olá mundo!
```

- CARACTERÍSTICAS PRINCIPAIS:
  - DEVE TERMINAR COM UMA DECLARAÇÃO DE RETORNO
  - PODE RETORNAR NO MÁXIMO UM VALOR. NOUTRAS PALAVRAS, PODE HAVER APENAS UMA DECLARAÇÃO DE RETORNO POR FUNÇÃO
  - A DECLARAÇÃO DE RETORNO DEVE SER A ÚLTIMA DECLARAÇÃO NA FUNÇÃO

2. SINTAXE

FUNÇÕES PARAMETRIZADAS

- OS PARÂMETROS
  - SÃO UM MECANISMO PARA PASSAR VALORES PARA FUNÇÕES
  - FORMAM PARTE DA ASSINATURA DA FUNÇÃO
  - OS SEUS VALORES SÃO PASSADOS PARA A FUNÇÃO DURANTE A SUA INVOCAÇÃO

- A MENOS QUE EXPLICITAMENTE ESPECIFICADO, O NÚMERO DE VALORES PASSADOS PARA UMA FUNÇÃO DEVE CORRESPONDER AO NÚMERO DE PARÂMETROS DEFINIDOS

```
function add(n1, n2) {
  let sum = n1 + n2
  return sum
}
console.log(add(3,4))
-----------------------------------------------
7
```

2. SINTAXE

NOVOS TIPOS DE FUNÇÕES

- COM O ES6 APARECEREM NOVAS FORMAS DE USAR FUNÇÕES:
  - **FUNÇÕES COM PARÂMETROS POR OMISSÃO**
  - **FUNÇÕES COM PARÂMETROS REST**
  - **FUNÇÕES ANÓNIMAS**
  - **FUNÇÕES LAMBDA (ARROW FUNCTIONS)**
  - **IMMEDIATELY INVOKED FUNCTION EXPRESSION (IIFE)**
  - **FUNÇÕES GERADORAS**

```
let foo = (x) => 5 + x
console.log(foo(10))
----------------------------
15
```

2. SINTAXE

NOVOS TIPOS DE FUNÇÕES

- **FUNÇÕES COM PARÂMETROS POR OMISSÃO**
  - PERMITE QUE OS PARÂMETROS SEJAM INICIALIZADOS COM VALORES PADRÃO, SE NENHUM VALOR FOR PASSADO OU NÃO FOR DEFINIDO

```
function add(n1, n2 = 3) {
  let sum = n1 + n2
  return sum
}
console.log(add(3))
console.log(add(3,4))
----------------------------------------------------
6
7
```

2. SINTAXE

NOVOS TIPOS DE FUNÇÕES

- **FUNÇÕES COM PARÂMETROS REST**
  - NÃO RESTRINGEM O Nº DE VALORES QUE PODE PASSAR PARA UMA FUNÇÃO. NO ENTANTO, OS VALORES PASSADOS DEVEM SER TODOS DO MESMO TIPO
  - NOUTRAS PALAVRAS, OS PARÂMETROS REST ATUAM COMO ESPAÇOS RESERVADOS PARA VÁRIOS ARGUMENTOS DO MESMO TIPO

```
function fun(...params) {
  console.log(params.length)

}
fun()
fun(5)
fun(5, 6, 12)
-------------------------------------------------
0
1
3
```

  - NOTA: OS PARÂMETROS REST DEVEM SER OS ÚLTIMOS DE UMA LISTA DE PARÂMETROS.

2. SINTAXE

NOVOS TIPOS DE FUNÇÕES

- **FUNÇÕES ANÓNIMAS**
  - NÃO VINCULADAS A UM IDENTIFICADOR (NOME DA FUNÇÃO)
  - DECLARADAS DINAMICAMENTE NO TEMPO DE EXECUÇÃO
  - PODEM ACEITAR INPUTS E RETORNAR OUTPUTS
  - GERALMENTE NÃO É ACESSÍVEL APÓS A SUA CRIAÇÃO INICIAL

  - AS VARIÁVEIS PODEM SER ATRIBUÍDAS A UMA FUNÇÃO ANÔNIMA
  - ESSA EXPRESSÃO É CHAMADA DE **EXPRESSÃO DE FUNÇÃO**

```
let f = function() { return “olá” }
console.log(f())
-----------------------------------
olá
```

```
let f = function(x, y) { return x * y }
function product() {
  let result
  result = f(10, 20)
  return result
}
console.log(product())
-----------------------------------
200
```

2. SINTAXE

NOVOS TIPOS DE FUNÇÕES

- **FUNÇÕES LAMBDA (OU FUNÇÕES ARROW)**
  - SÃO UM MECANISMO CONCISO PARA REPRESENTAR **FUNÇÕES ANÓNIMAS (FA)**
  - TAMBÉM CHAMADAS COMO **FUNÇÕES ARROW (=>)**
  - DUAS VARIANTES
    - **EXPRESSÕES LAMBDA** - EXPRESSÃO DE FA QUE APONTA PARA UMA ÚNICA LINHA DE CÓDIGO
    - **DECLARAÇÕES LAMBDA** - EXPRESSÃO DE FA QUE APONTA PARA UM BLOCO DE CÓDIGO

```
let foo = (x) => 5 + x
console.log(foo(10))
----------------------------
15
```

```
let msg = () => {
 console.log("função invocada")
}
msg()
----------------------------
função invocada
```

  - NOTAS:
    - PARÊNTESIS OPCIONAIS PARA PARÂMETRO ÚNICO
    - CHAVETAS OPCIONAIS PARA INSTRUÇÃO ÚNICA
    - PARÊNTESIS VAZIOS PARA 0 PARÂMETROS

ÍNDICE

2. SINTAXE

EVENTOS

- O JAVASCRIPT ADICIONA INTERATIVIDADE ÀS PÁGINAS HTML ATRAVÉS DE **EVENTOS:**
  - SÃO PARTE DO NÍVEL 3 DO **DOM (DOCUMENT OBJET MODEL)** ONDE CADA ELEMENTO HTML CONTÉM UM CONJUNTO DE EVENTOS QUE PODEM DESENCADEAR CÓDIGO JAVASCRIPT
  - AÇÃO OU OCORRÊNCIA RECONHECIDA PELO SOFTWARE
  - PODE SER ACIONADO POR UM UTILIZADOR OU PELO SISTEMA

- EXEMPLOS:
  - UTILIZADOR CLICAR NUM BOTÃO OU NUM LINK
  - CARREGAMENTO DA PÁGINA WEB

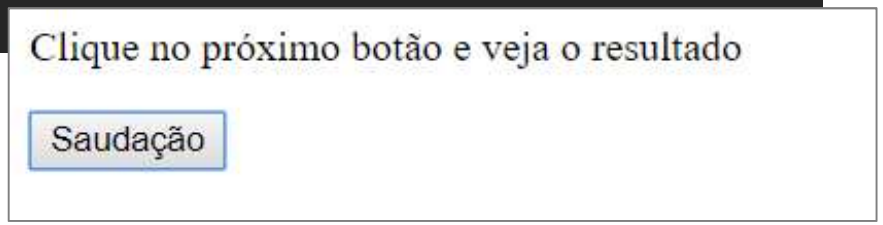
- PODEMOS DEFINIR COMO OS EVENTOS SERÃO PROCESSADOS EM JAVASCRIPT USANDO **MANIPULADORES DE EVENTOS (EVENT HANLERS)**
- PRINCIPAIS TIPOS DE EVENTOS:
  - **ONCLICK**
  - **ONSUBMIT**
  - **ONMOUSEOVER/ONMOUSEOUT**
  - **ONLOAD**
  - **ONKEYPRESS**
  - **…**

2. SINTAXE

EVENTOS

- TIPOS DE EVENTO **ONCLICK**

```
<html>
 <head>
  <script type = "text/javascript">
   function sayHello() {
    console.log(“Olá mundo!”)
   }
  </script>
 </head>
 <body>
  <p>Clique no próximo botão e veja o resultado</p>
  <input type="button" onclick="sayHello()" value=“Saudação" />
 </body>
</html>
```



2. SINTAXE

EVENTOS

- TIPOS DE EVENTO **ONSUBMIT**

```
<html>
 <head>
  <script type = "text/javascript">
   function validate() {
     // validação aqui
     return true // ou false
   }
  </script>
 </head>
 <body>
  <form method=“POST” action=“run.php” onsubmit=“return validate()”>
    ...
    <input type=“submit" value=“submeter" />
  </form>
 </body>
</html>
```

ÍNDICE

3. TIPOS DE DADOS

TIPOS DE DADOS

- **PRIMITIVOS:** VALOR DE DADOS SIMPLES SEM PROPRIEDADES E MÉTODOS ADICIONAIS

```
console.log(typeof 3.14)
console.log(typeof “esmad”)
console.log(typeof true)
console.log(typeof x)
-------------------------------------
number
string
boolean
undefined
```

- **COMPLEXOS:** VALOR DE DADOS COMPLEXOS COM PROPRIEDADES E MÉTODOS ADICIONAIS

```
console.log(typeof {name: “Ricardo”, age: 42})
console.log(typeof [1,2,3,4])
console.log(typeof null)
console.log(typeof function myFunc(){})
-------------------------------------
object
object
object
function
```

ÍNDICE

3. TIPOS DE DADOS

NUMBER

- O JAVASCRIPT SUPORTA APENAS UM TIPO DE NÚMERO
- UM NÚMERO PODE TER OU NÃO CASAS DECIMAIS

```
let x = 34.02
let y = 12
```

- TIPO DE DADOS NÚMERO TEM PRECISÃO ATÉ 15 DÍGITOS
- **INFINITY** - VALOR QUE JAVASCRIPT RETORNA SE CALCULAR UM NÚMERO ACIMA DO MAIOR NÚMERO POSSÍVEL

```
let x = 2 / 0
let y = -2 / 0
console.log(x)
console.log(y)
-----------------------------------
Infinity
-Infinity
```

- INFINITY É UM NÚMERO: **TYPEOF** DE INFINITY DEVOLVE **NUMBER**

```
console.log(typeof Infinity)
-----------------------------------
number
```

3. TIPOS DE DADOS

NUMBER

- **NAN** É UMA PALAVRA RESERVADA QUE INDICA UM VALOR NÃO NUMÉRICO
- TENTANDO FAZER ARITMÉTICA COM UMA STRING NÃO NUMÉRICA RESULTARÁ EM **NAN (NOT A NUMBER)**

```
let x = 100 / “esmad”
console.log(x)
-----------------------------------
NaN
```

- CONTUDO, SE A STRING CONTÉM UM VALOR NUMÉRICO, O RESULTADO SERÁ UM NÚMERO:

```
let x = 100 / “10”
console.log(x)
-----------------------------------
10
```

- PODE-SE USAR A FUNÇÃO GLOBAL **ISNAN** PARA AFERIR SE O VALOR É UM NÚMERO:

```
let x = 100 / “esmad”
console.log(isNaN(x))
-----------------------------------
true
```

3. TIPOS DE DADOS

NUMBER

- **PRINCIPAIS MÉTODOS**
  - **NUMBER.ISINTEGER(N)** - DETERMINA SE O VALOR PASSADO É UM NÚMERO INTEIRO

```
let x = 32.2
console.log(Number.isInteger(x))
-----------------------------------
false
```

  - **TOFIXED(X)** - CONVERTE UM NÚMERO NUMA STRING, MANTENDO UM Nº ESPECÍFICO DE DECIMAIS

```
let num = 5.53789
let newNum = num.toFixed(2)
console.log(newNum)
-----------------------------------
“5.54”
```

3. TIPOS DE DADOS

NUMBER

- **CONVERSÕES**
  - **TOSTRING(NUM) -** RETORNA UM NÚMERO COMO UMA STRING

```
let x = 123
let str = x.toString()
console.log(str)
-----------------------------------
“123”
```

  - **PARSEINT(STR)** - CONVERTE UMA STRING NUM NÚMERO

```
let str = ”123”
let x = parseInt(str)
console.log(x)
-----------------------------------
123
```

  - SE TENTAR CONVERTER UMA STRING USANDO ESTE MÉTODO, DEVOLVE **NAN**

3. TIPOS DE DADOS

NUMBER

- **REFERÊNCIAS ONLINE**

  MDN

  HTTPS://DEVELOPER.MOZILLA.ORG/PT-PT/DOCS/WEB/JAVASCRIPT/REFERENCE/GLOBAL_OBJECTS/NUMBER

  W3SCHOOLS

  HTTPS://WWW.W3SCHOOLS.COM/JSREF/JSREF_OBJ_NUMBER.ASP

ÍNDICE

3. TIPOS DE DADOS

STRINGS

- USADAS PARA ARMAZENAR E MANIPULAR TEXTO (SEQUÊNCIA DE CARACTERES)
- UM VALOR STRING DEVE ESTAR ENTRE ASPAS OU PLICAS, PODENDO-AS COMBINAR

```
let school = “esmad”
let name = ‘ricardo’
let restaurant = “McDonald’s”
```

- TAMANHO DE UMA STRING USA-SE A PROPRIEDADE **LENGTH**

```
let school = “esmad”
console.log(school.length)
-----------------------------------
5
```

- CARACTERES ESPECIAIS (USO DE CARACTER DE ESCAPE \)

```
let y = "We are the \\Vikings\' \n from the north."
console.log(y)
-----------------------------------
"We are the \Vikings'
 from the north."
```

3. TIPOS DE DADOS

STRINGS

- **PROCURA DE TEXTO**
  - **INDEXOF(STR)** - DEVOLVE A POSIÇÃO DA PRIMEIRA OCORRÊNCIA DE UM TEXTO NUMA STRING
  - **LASTINDEXOF(STR)** - DEVOLVE A POSIÇÃO DA ÚLTIMA OCORRÊNCIA DE UM TEXTO NUMA STRING

```
let msg = “Eu gosto de Vila do Conde e gosto da Póvoa do Varzim”
let firstPos = msg.indexOf(“gosto”)
let lastPos = msg.lastIndexOf(“gosto”)
console.log(firstPos)
console.log(lastPos)
----------------------------------
3
28
```

- AS POSIÇÕES DE CARACTERES NUMA STRING INICIAM-SE EM 0
- AMBOS OS MÉTODOS DEVOLVEM -1 CASO O TEXTO NÃO SEJA ENCONTRADO
- UM PARÂMETRO EXTRA PODE SER ADICIONADO INDICANDO A POSIÇÃO ONDE DEVE INICIAR A PROCURA (POR OMISSÃO INICIA-SE EM 0)

3. TIPOS DE DADOS

STRINGS

- **EXTRAÇÃO DE TEXTO**
  - **SLICE(START, END) -** EXTRAI UMA PARTE DA STRING (INICIADA EM START E TERMINADA EM END) E DEVOLVE A PARTE NUMA NOVA STRING

```
let str = "Apple, Banana, Kiwi"
let res = str.slice(7, 13)
console.log(res)
-----------------------------------
"Banana"
```

  - **SUBSTR(START, LENGTH)** - IGUAL AO ANTERIOR, MAS O SEGUNDO PARÂMETRO INDICA O Nº DE CARACTERES A EXTRAIR

```
let str = "Apple, Banana, Kiwi"
let res = str.substr(7)
console.log(res)
-----------------------------------
"Banana"
```

  - **SUBSTRING(START)** – EXTRAI DADOS DE UMA POSIÇÃO ATÉ AO FIM DA STRING

3. TIPOS DE DADOS

STRINGS

- **SUBSTITUIÇÃO DE TEXTO**
  - **REPLACE(LOCSTR, NEWSTR) -** PROCURA UMA PARTE DA STRING (LOCSTR) E SUBSTITUI POR OUTRA (NEWSTR)

```
let str = “Eu gosto da ESEIG!”
let newStr = str.replace(”ESEIG”, “ESMAD”)
console.log(newStr)
-----------------------------------
“Eu gosto da ESMAD!”
```

  - POR OMISSÃO, A SUBSTITUIÇÃO É FEITA APENAS NA PRIMEIRA OCORRÊNCIA
  - PARA PROPAGAR A SUBSTITUIÇÃO EM TODAS AS OCORRÊNCIAS USE A EXPRESSÃO REGULAR **/G**

```
let str = “Eu gosto da ESEIG e da ESEIG!”
let newStr = str.replace(/ESEIG/g, “ESMAD”)
console.log(newStr)
-----------------------------------
“Eu gosto da ESMAD e da ESMAD!”
```

3. TIPOS DE DADOS

STRINGS

- **ITERAÇÃO DE TEXTO**
  - **CHARAT(POSITION) -** RETORNA O CARACTER NO ÍNDICE ESPECIFICADO NUMA STRING

```
let str = "ESMAD"
for(let i = 0; i < str.length; i++) {
  console.log(str.charAt(i))
}
-----------------------------------
E
S
M
A
D
```

3. TIPOS DE DADOS

STRINGS

- OS **LITERAIS TEMPLATE (OU TEMPLATES STRING)** SÃO LITERAIS DE STRINGS QUE PERMITEM EXPRESSÕES INCORPORADAS
- USAM ACENTROS GRAVES – BACK-TICKS (``) EM VEZ DE ASPAS SIMPLES OU DUPLAS
- EXEMPLO DE UMA STRING TEMPLATE

```
let str = `ESMAD`
```

- PODEM USAR MARCADORES DE POSIÇÃO PARA SUBSTITUIÇÃO DE STRINGS USANDO A SINTAXE **${}**

```
let name = ”Ricardo”
console.log(`Olá, ${name}!`)
-----------------------------------
Olá, Ricardo!
```

- SUPORTE PARA EXPRESSÕES

```
let a = 10; let b = 10
console.log(`A soma de ${a} com ${b} é ${a+b}`)
-----------------------------------
A soma de 10 com 10 é 20
```

3. TIPOS DE DADOS

STRINGS

- OS **LITERAIS TEMPLATE (OU TEMPLATES STRING)** SÃO LITERAIS DE STRINGS QUE PERMITEM EXPRESSÕES INCORPORADAS

- SUPORTE PARA FUNÇÕES

```
function fn() { return “Olá Mundo"; }
console.log(`Mensagem: ${fn()} !!`);
-----------------------------------
Mensagem: Olá Mundo !!
```

3. TIPOS DE DADOS

STRINGS

- **REFERÊNCIAS ONLINE**

  MDN

  HTTPS://DEVELOPER.MOZILLA.ORG/PT-PT/DOCS/WEB/JAVASCRIPT/REFERENCE/GLOBAL_OBJECTS/STRING

  W3SCHOOLS

  HTTPS://WWW.W3SCHOOLS.COM/JSREF/JSREF_OBJ_STRING.ASP