iscte
INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

Módulo 4: Introdução à programação em javascript

Aula 02
Setup do IDE, Variáveis e Funções

UPskill

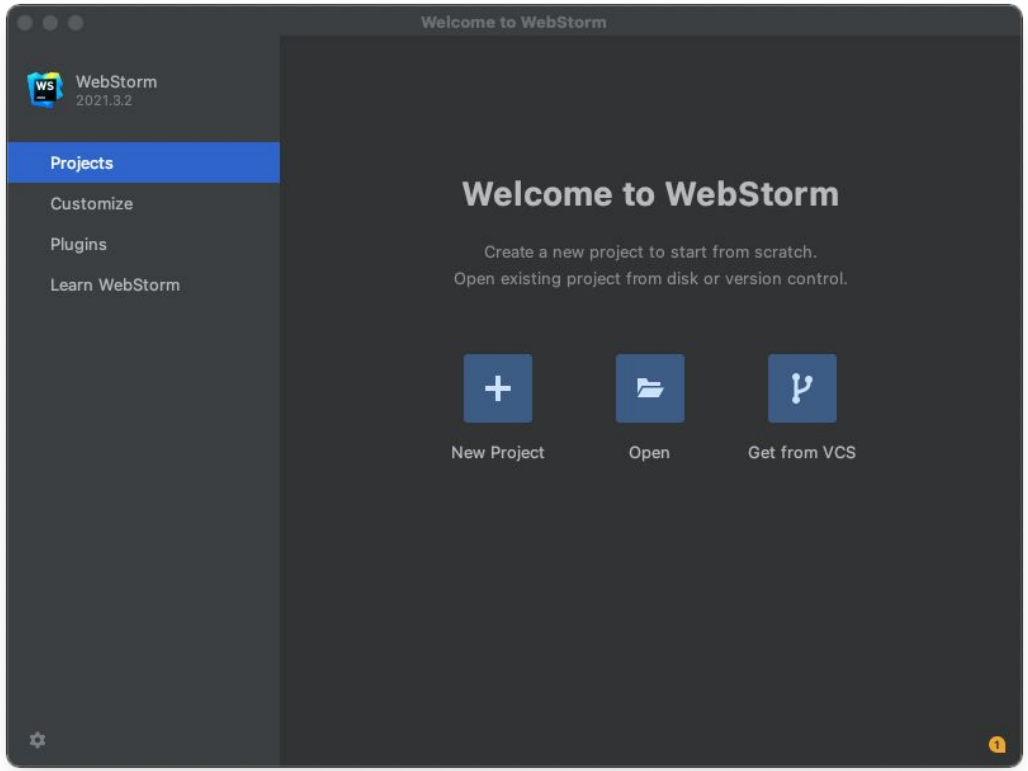
O IDE

- **Um IDE** (Ambiente de Desenvolvimento Integrado) é uma ferramenta que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. No fundo, **é onde escrevemos o código.**

- Existem muitos IDE: IntelliJ, NetBeans, Visual Studio Code, por exemplo. Muitos suportam várias linguagens de programação, sendo possível utilizar o mesmo IDE para vários projetos, em várias linguagens.

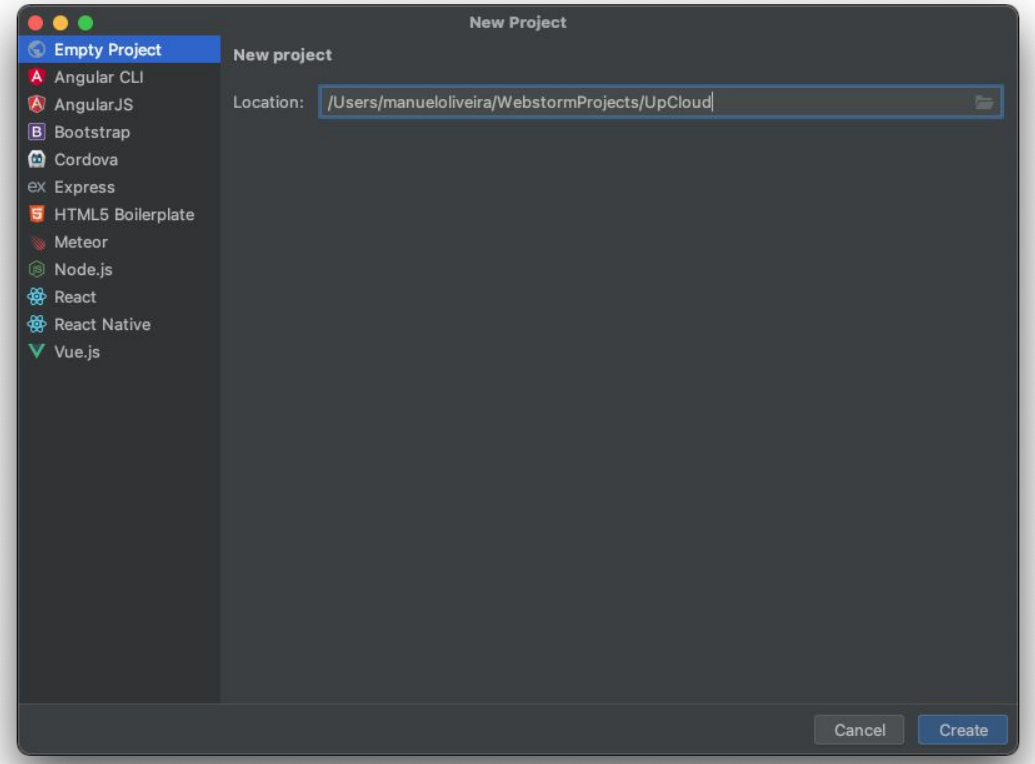
- **Neste módulo iremos utilizar o Webstorm (versão de *trial*):** https://www.jetbrains.com/webstorm/download

Criação do projeto



- Clicar em **New Project**

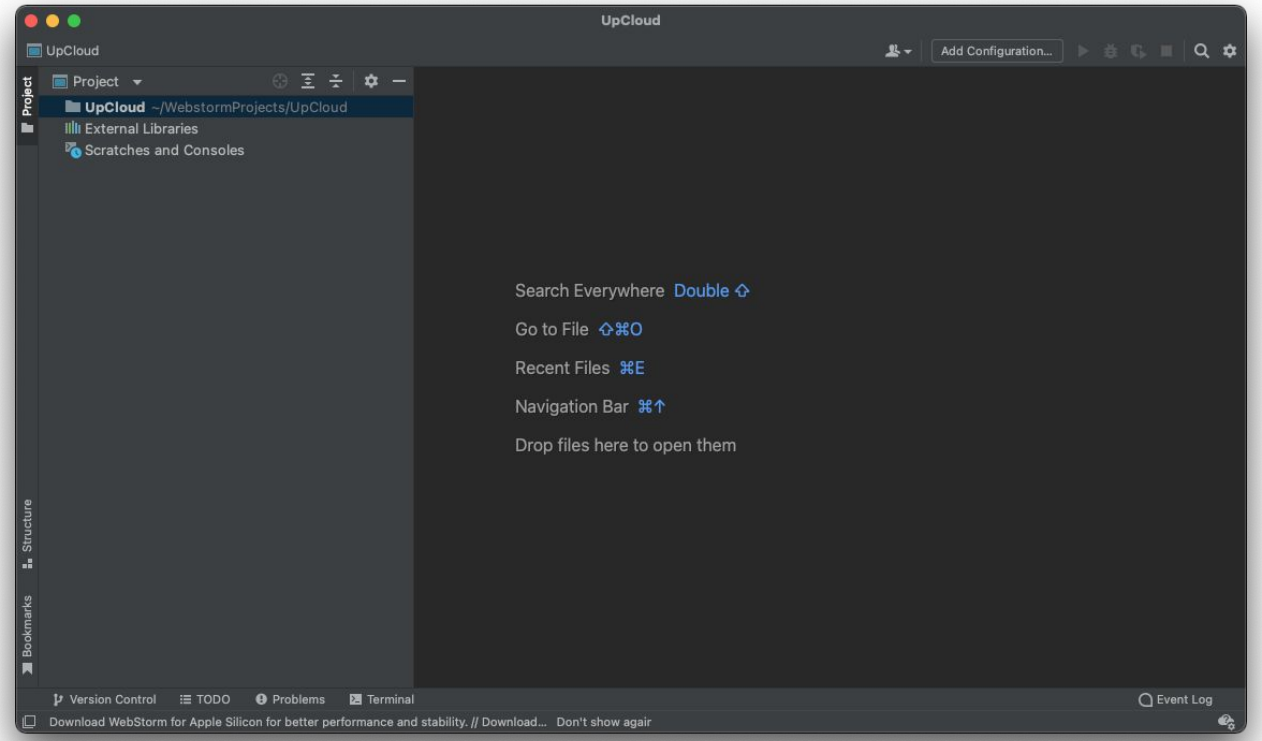
Criação do projeto



Localização:

- Escolher um sítio que seja fácil de encontrar no explorador de ficheiros (por exemplo no ambiente de trabalho);
- Dar logo o nome do projeto, p. ex: **upskill_m4**

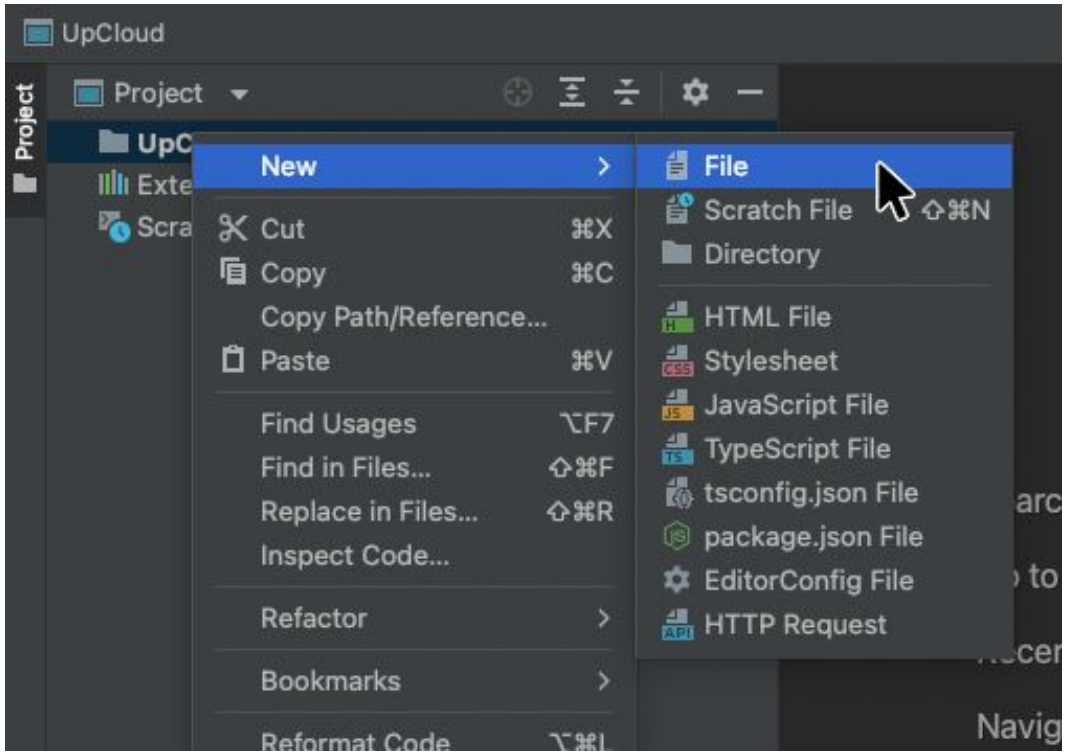
Criação do projeto



- IDE abre o novo projeto **vazio**;
- Do lado esquerdo encontram a navegação dentro do projeto, neste momento não tem lá nada.

iscte INSTITUTO UNIVERSITÁRIO DE LISBOA emprego digital

Criação de ficheiros



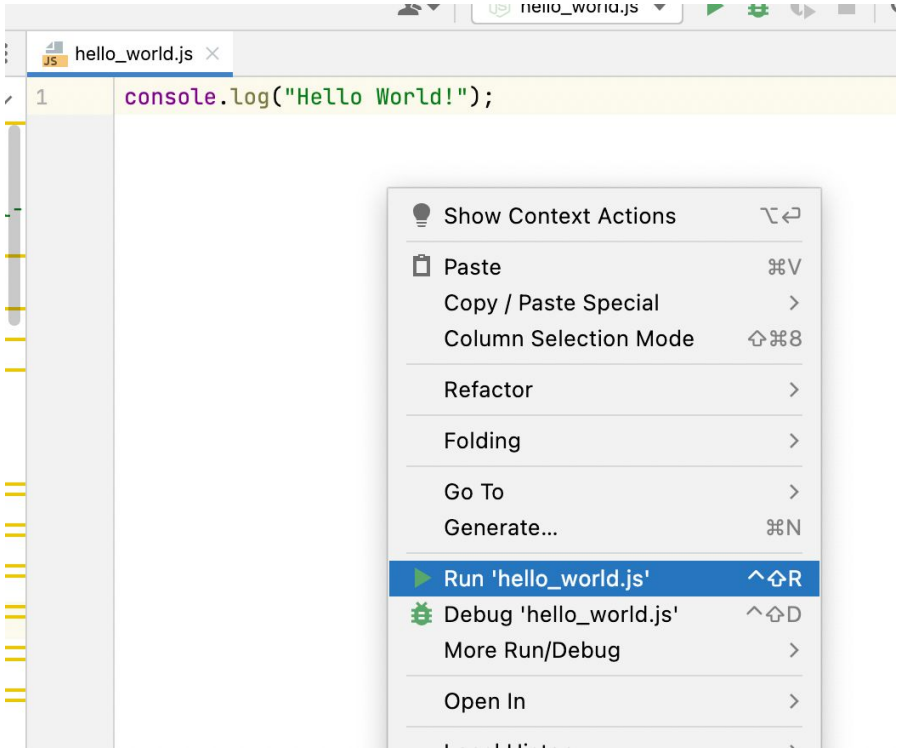
- Clicar com o **botão direito do rato** em cima do nome do nome do projeto nessa navegação;
- Dentro do sub-menu "**New**" escolher o "**File**"
- Colocar o nome "**hello_world.js**" (sem aspas);
- Clicar no **ENTER** para confirmar.

Pequena experiência

- Criem um ficheiro **hello_world.js** no Webstorm. Podem utilizar o vosso projeto habitual ou criar um projeto diferente se preferirem.
- Coloquem o código:

```
console.log("hello world");
```

- Para executar o código cliquem com o lado direito no espaço branco e escolham a opção “Run”



Pequena experiência

- O resultado da execução irá ser apresentado na parte inferior do ecrã e poderá voltar a ser executado clicando no triângulo verde.

Exemplo de algoritmo JavaScript

```
let ano_atual = 2022;
let ano_nascimento = 1996;

let idade = ano_atual - ano_nascimento;

console.log(idade);
```

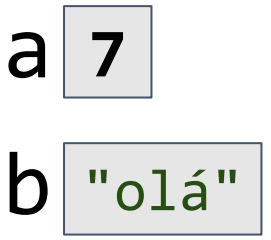
O que são variáveis?

- Uma variável pode ser vista como um espaço em memória onde um valor de determinado tipo pode ser guardado.
- As variáveis têm três características
    - **Tipo**
    - **Nome**
    - **Valor**
- Em JavaScript, ao definir uma variável estamos a indicar o seu **nome** e o seu **valor** inicial. O tipo é **inferido pelo valor**.

Variáveis em JavaScript

- Ao contrário de muitas outras linguagens, em JavaScript o tipo da variável é **flexível**, o que nos liberta para desenvolvermos o nosso código sem ter de pensar no tipo antes de definir a variável.

```
let a = 7;
let b = "olá";
```



Variáveis - Declaração

Existem três formas diferentes de declarar uma variável

var **let** **const**

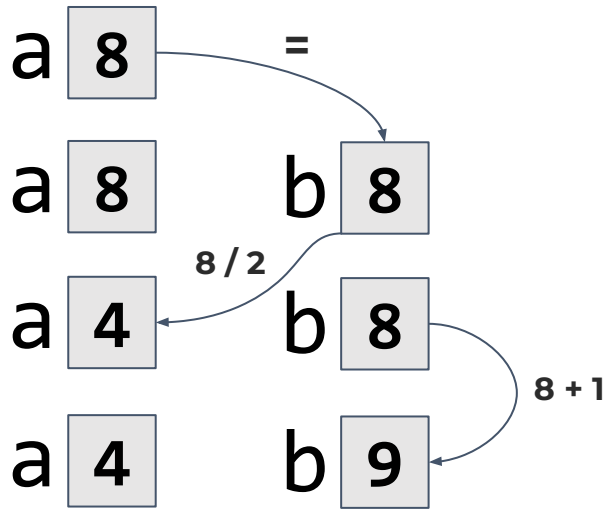
Mas cuidado! JavaScript é case-sensitive

Variáveis - Atribuição

- Usamos o operador de atribuição (**=**) para **definir/alterar o valor** das variáveis;

```
let a = 8;
let b = a;

a = b / 2;
b = b + 1;
```



Instruções e Blocos de Instrução

- Uma instrução é uma **ação na execução do programa** que pode mudar o seu estado (variáveis).

```
let a = 8;
```

- Um bloco de instruções é um **conjunto de instruções entre chavetas** que será executado sequencialmente.

```
{
  a = b / 2;
  b = b + 1;
}
```

Quantos “resultados” diferentes pode ter uma variável?

Variáveis - Valores

Non-zero value

0

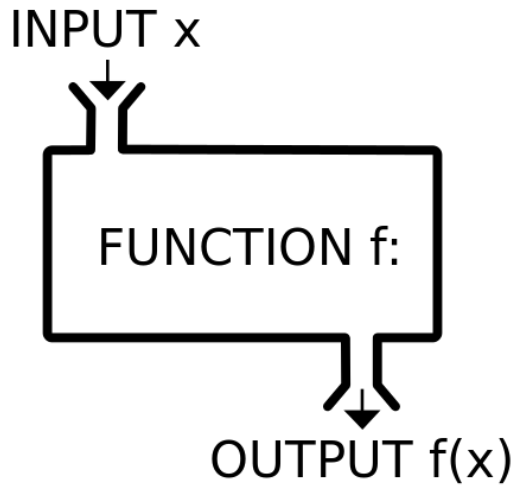
null

undefined

Funções

- Em Matemática, uma função **`f()`** associa a um argumento **`x`**, um valor **`y`**. Ou seja, **`y = f(x)`**.

  Se **`f(x) = x²`**
  então **`f(3) = 9`**



- Em JavaScript, o conceito de função é parecido:

```
function square(x) {
    return x * x;
}
```

- As funções são **implementações** de métodos de resolução de problemas.

Funções - Assinatura

- Para que seja possível utilizar uma função, tal como na Matemática, esta tem de ter um nome. O nome da função deve indicar aquilo que a função calcula.

- Uma função tem uma assinatura, onde se define o **nome** e os **parâmetros** da função (o que entra):

```
function somar(a,b) {
    return a + b;
}
```

Funções - Corpo

- O corpo de uma função é a **explicação ao computador** do método de resolução de um problema. Esta definição aparece a seguir à assinatura da função, entre chavetas.

```
function somar(a,b) {
    return a + b;
}
```

Funções - Parâmetros e Argumentos

- Os parâmetros indicam a que informação é que a função se aplica;
- Os argumentos são os valores que se dão aos parâmetros.

parâmetros

```
function somar(a, b) {
    return a + b;
}
```

argumentos

```
somar(3, 5);
>> 8
```

Funções - Retorno

- A instrução **`return`** indica qual o valor devolvido pela função.
- Neste caso, a expressão **`x * x`** calcula o quadrado de **`x`** e a instrução **`return`** devolve o valor calculado:

```
function square(x) {
    return x * x;
}
```

```
>  square(3);
>  return 3 * 3;
>  return 9;
>> 9
```

Funções - Invocação

- As funções na programação podem ser utilizadas numa determinada instrução pelo computador através de uma **invocação**.
- Uma invocação é composta pelo **nome** da função que se pretende invocar, seguida dos **argumentos** a passar nessa função:

```
let m = somar(5, 8);
```

Expressões Matemáticas

- Na programação, tal como na matemática, podemos declarar expressões que representam algoritmos e contas:

```
let m = 60 / 10 - 1;
```

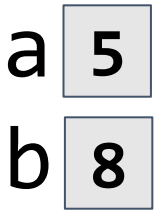
m 5

```
>  let m = 60 / 10 - 1;
>  m = 6 - 1;
>> m = 5;
```

Variáveis como Argumentos

- O valor de um argumento pode ser dado usando uma variável;
- O argumento será o **valor guardado na variável** no momento em que a função é invocada.

```
let a = 5;
let b = somar(a, 3);
```

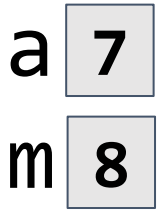



```
>  let a = 5;
>  let b = somar(a, 3);
>  b = somar(5, 3);
>  b = 5 + 3;
>> b = 8;
```

Expressões como Argumentos

- O valor de um argumento pode ser dado usando uma expressão;
- O argumento será o **valor da expressão** no momento em que a função é invocada:

```
let a = 7;
let m = somar(a - 5, 6);
```




```
>  let a = 7;
>  let m = somar(a - 5, 6);
>  m = somar(2, 6);
>  m = 2 + 6;
>> m = 8;
```

Resultados de Funções como Argumentos

- O valor de um argumento pode ser o valor devolvido por uma função:

```
let a = 7;
let m = somar(subtrair(a, 4), 6);
```

a 7

m 9

```
>  let a = 7;
>  let m = somar(subtrair(a, 4), 6);
>  m = somar(subtrair(7, 4), 6);
>  m = somar(7 - 4, 6);
>  m = somar(3, 6);
>  m = 3 + 6;
>> m = 9;
```

- As **invocações usadas como argumento são executadas primeiro**, de modo a que o valor devolvido possa ser utilizado como argumento.

Exercício 1

- Criar uma função que tenha 3 argumentos e retorne a soma dos primeiros dois, multiplicando depois pelo terceiro argumento

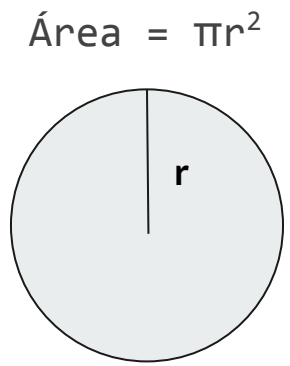
```
function somaMult(a, b, c) {
    ...
}
```

Bónus: E se quisermos multiplicar os primeiros dois e depois somar o terceiro?

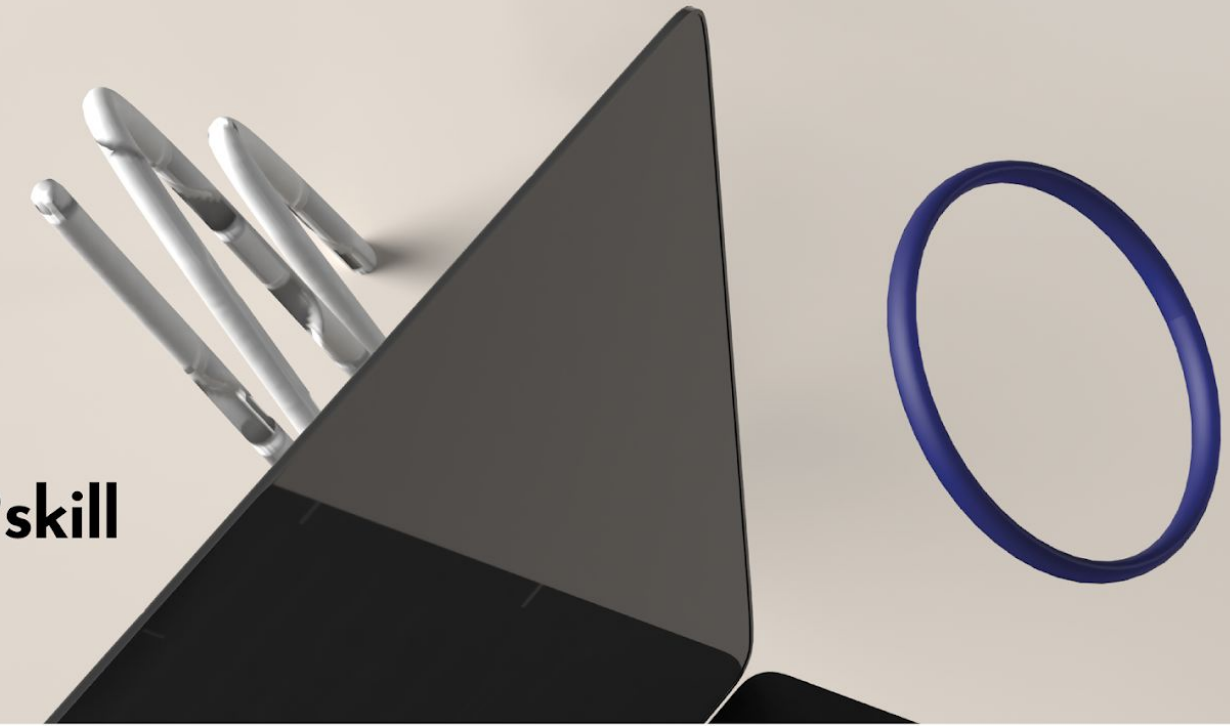
Exercício 2

- Criar uma função que retorne a área de um círculo com raio **r**
  - Dica: a expressão **`Math.PI`** devolve o valor de pi

```
function areaCirculo(r) {
    ...
}
```



O futuro profissional começa aqui



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

UPskill