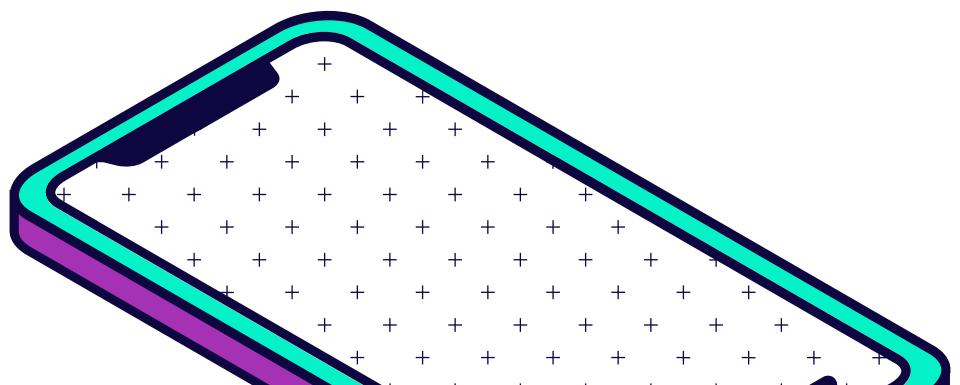
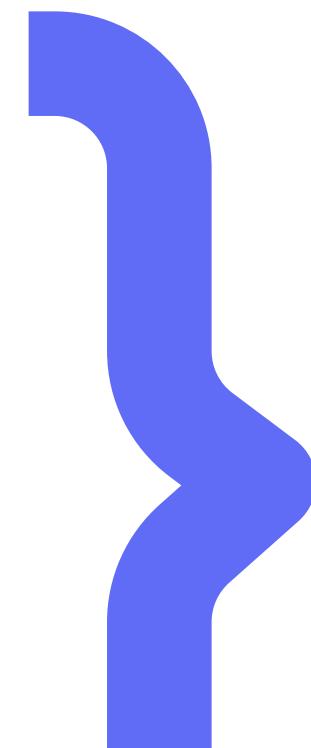




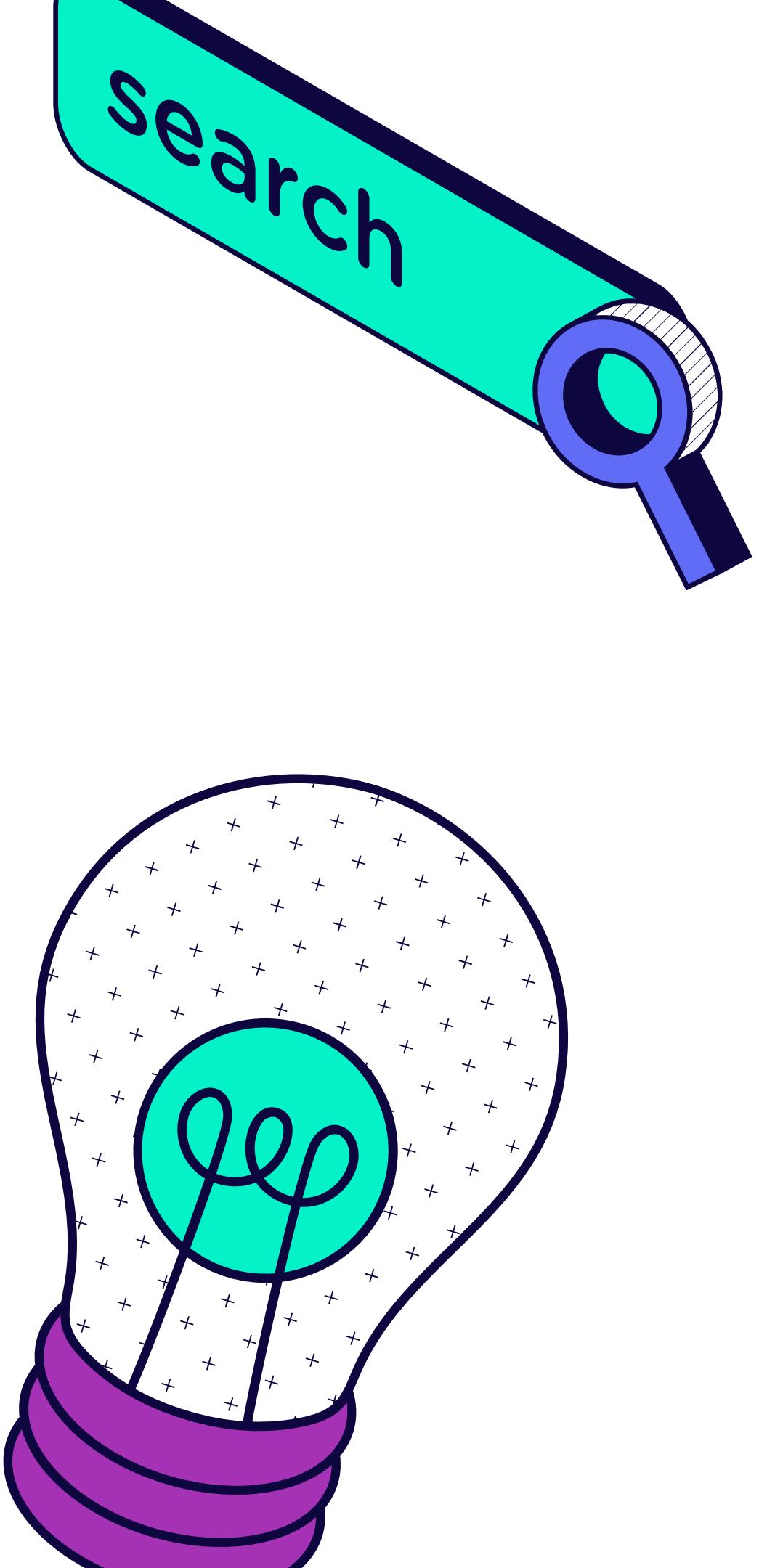
{ MINICURSO }

# Python



# O que é um algoritmo?

---



# O que é um algoritmo?

Algoritmos são um conjunto de instruções finitas para resolver um problema



# Algoritmo: atravessar a rua



# Algoritmo: atravessar a rua



# Algoritmo: atravessar a rua



# Algoritmo: atravessar a rua



# Algoritmo: atravessar a rua



# Algoritmo: receita de bolo simples



## INGREDIENTES

- 2 xícaras (chá) de açúcar
- 3 xícaras (chá) de farinha de trigo
- 4 colheres (sopa) de margarina
- 3 ovos
- 1 e 1/2 xícara (chá) de leite
- 1 colher (sopa) bem cheia de fermento em pó



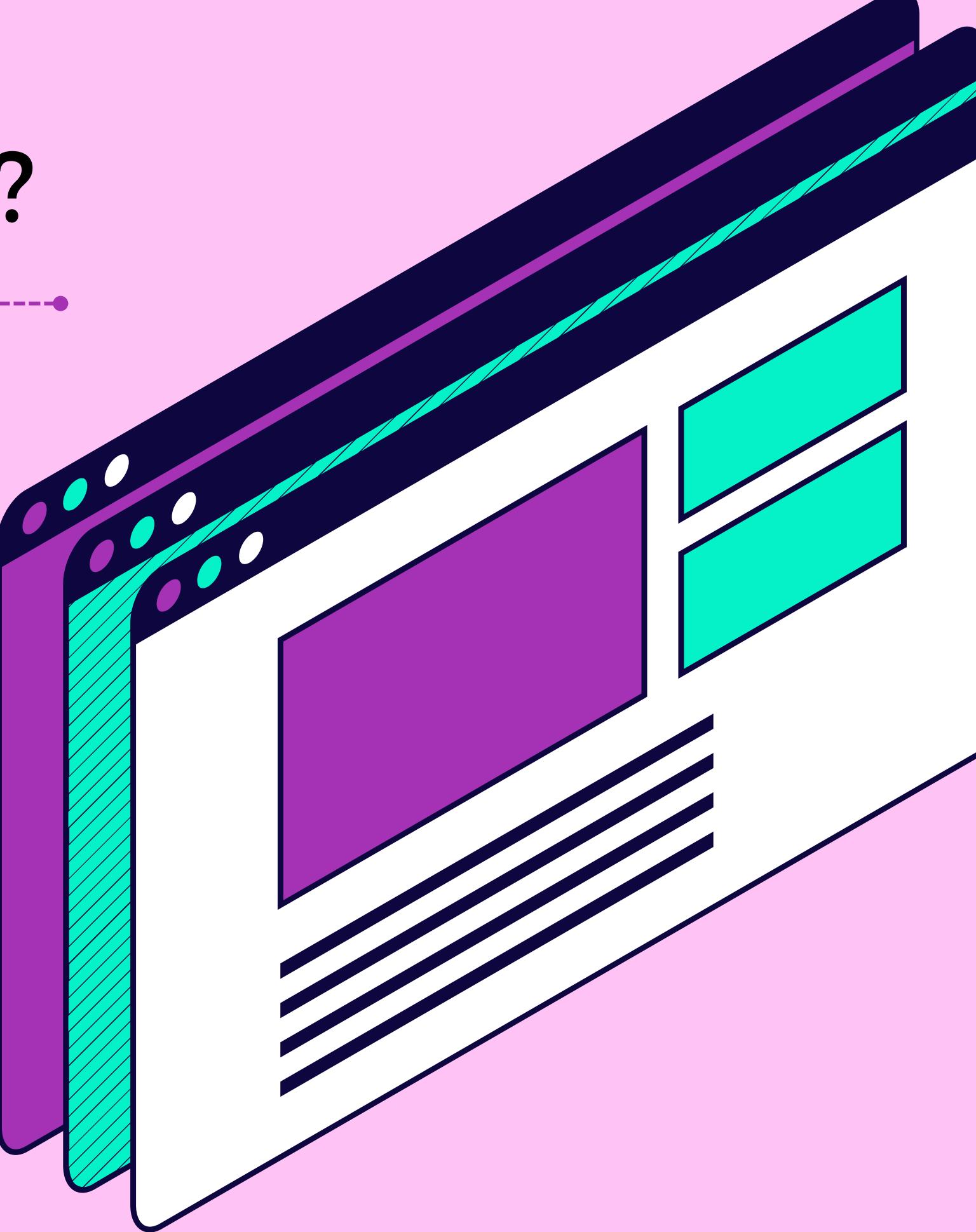
## MODO DE PREPARO

1. Bata as claras em neve e reserve.
2. Misture as gemas, a margarina e o açúcar até obter uma massa homogênea.
3. Acrescente o leite e a farinha de trigo aos poucos, sem parar de bater.
4. Por último, adicione as claras em neve e o fermento.
5. Despeje a massa em uma forma grande de furo central untada e enfarinhada.
6. Asse em forno médio 180 °C, preaquecido, por aproximadamente 40 minutos ou ao furar o bolo com um garfo, este saia limpo.

# Já ouviu falar em "Software"?

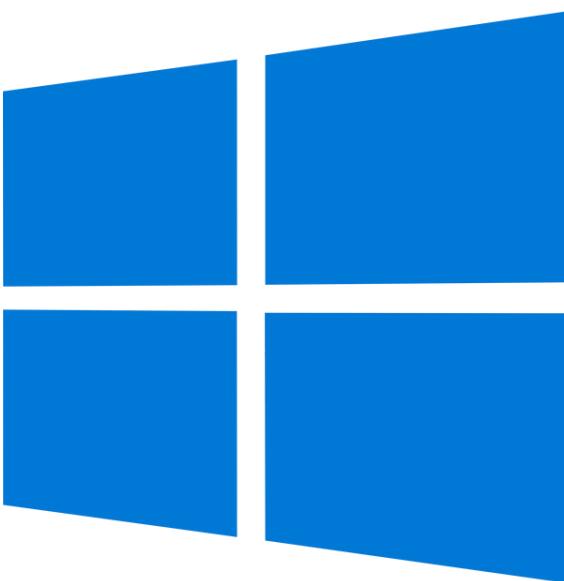
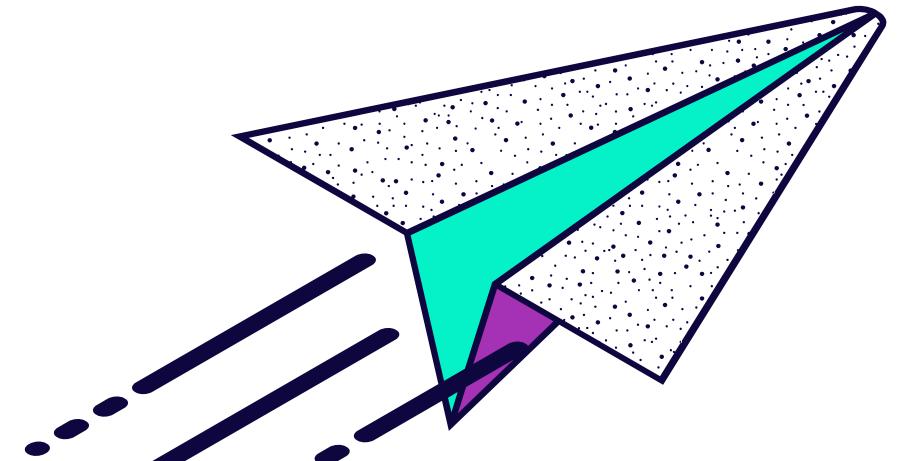
---

**Podemos considerar  
um software como um  
conjunto de vários  
algoritmos.**



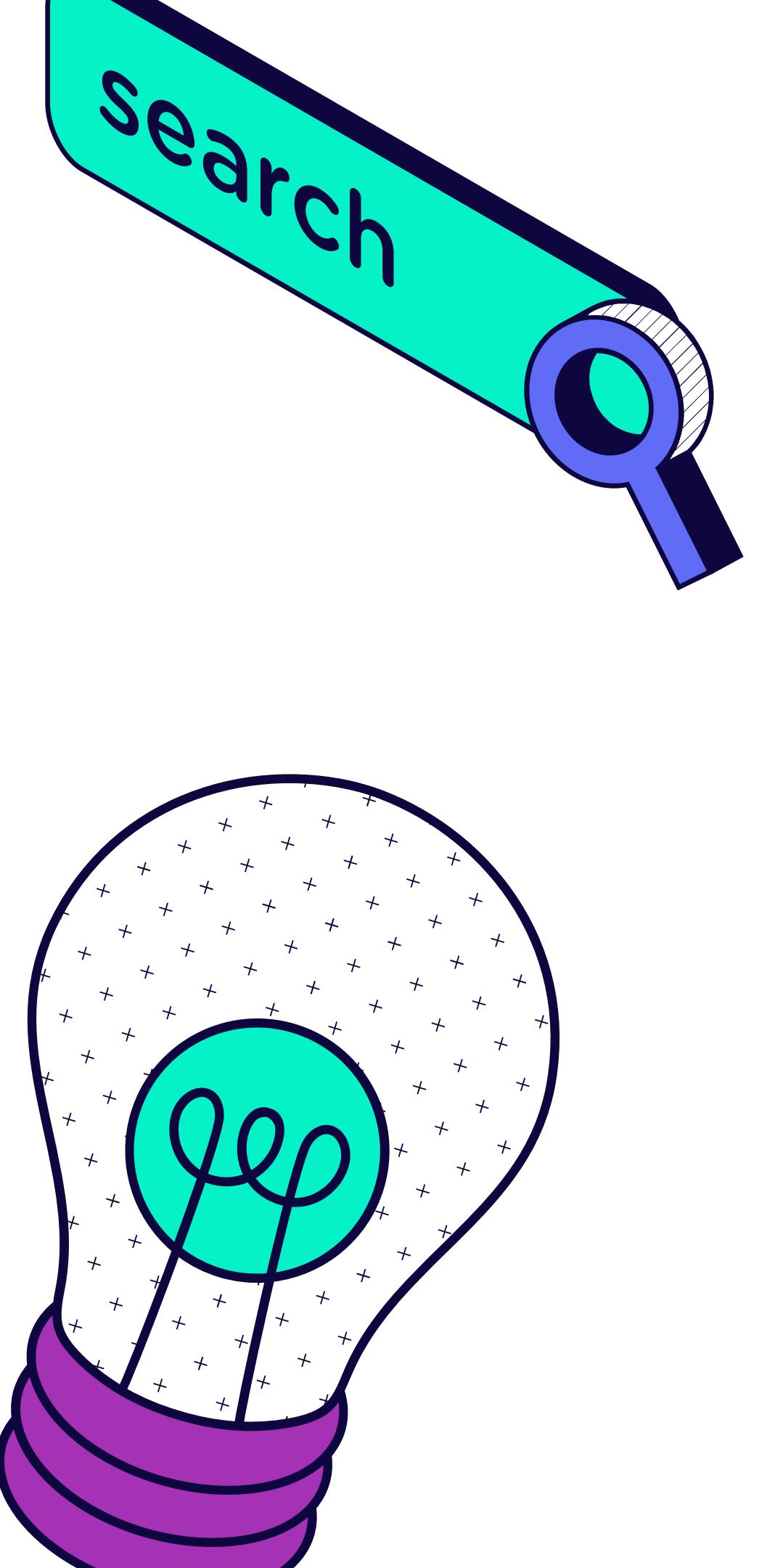
**Estamos  
cercados  
por eles**

Alguns exemplos de  
Software que você  
provavelmente conhece:



**Você sabe como  
os algoritmos  
são feitos?**

---



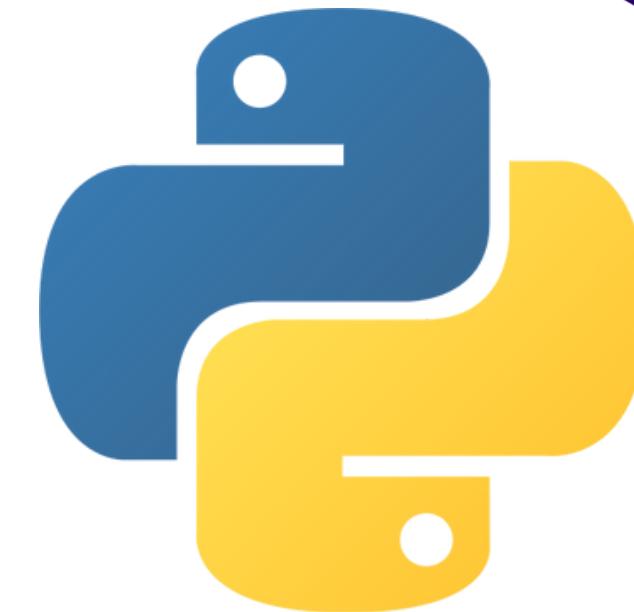
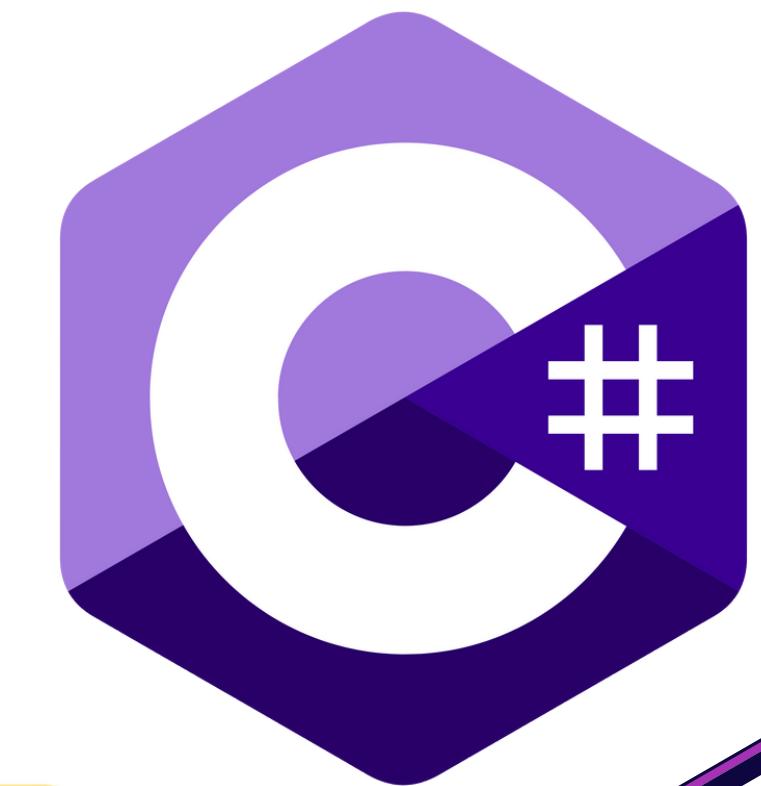
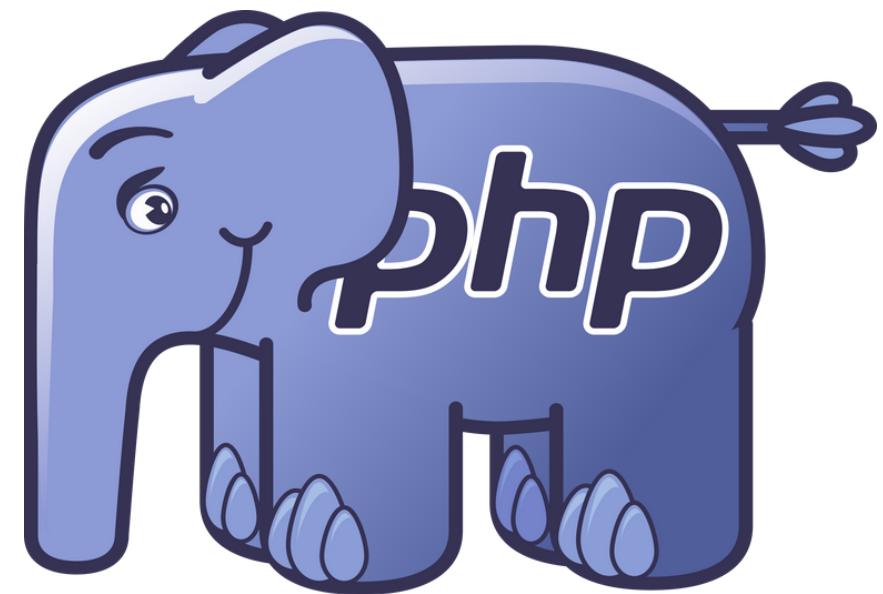
Algoritmos são feitos por meio de código escrito  
nas mais diversas linguagens de programação



Java™



JS





Linguagens de programação  
podem ser muito diferentes uma  
da outra

---



Cada uma tem seus  
escopos e seus pontos  
fortes e fracos

# No entanto, todas elas tem a mesma base

Um desenvolvedor que aprende a "pensar algoritmo" tem muito mais facilidade em aprender a sintaxe básica e os conceitos ao se deparar com uma nova linguagem





# Python

## Principais benefícios

- Sintaxe simples, semelhante ao inglês;
- Linguagem de alto nível;
- Open Source;
- Comunidade enorme de desenvolvedores;
- Outras características mais “técnicas”.



# Onde vou usar?

---

- Uma das principais linguagens no Back-End (Django, Flask...)
- Muito popular na área de ciência de dados
- Recentemente, o PyScript surgiu para levar o Python ao Front-End!

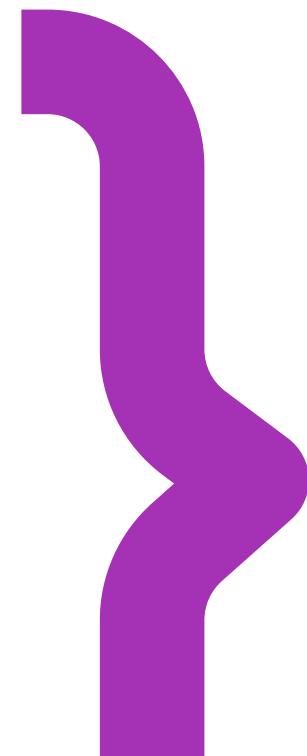


**Linguagens de programação  
podem ser muito diferentes uma  
da outra**



**Cada uma tem seus  
escopos e seus pontos  
fortes e fracos**

```
print("Hello World!")
```



# Hello World

Como uma tradição no mundo da programação, é dito que seu primeiro código numa nova linguagem deve ser sempre mostrar “Hello World” na tela.

Vamos começar disso, então:

hello\_world.py

```
1 print("Hello World!")
```

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
#include <stdio.h>  
int main() {  
    // printf() displays the string inside quotation  
    printf("Hello, World!");  
    return 0;  
}
```

# A função “print()”

A parte mais importante do código anterior é a função “print()”.

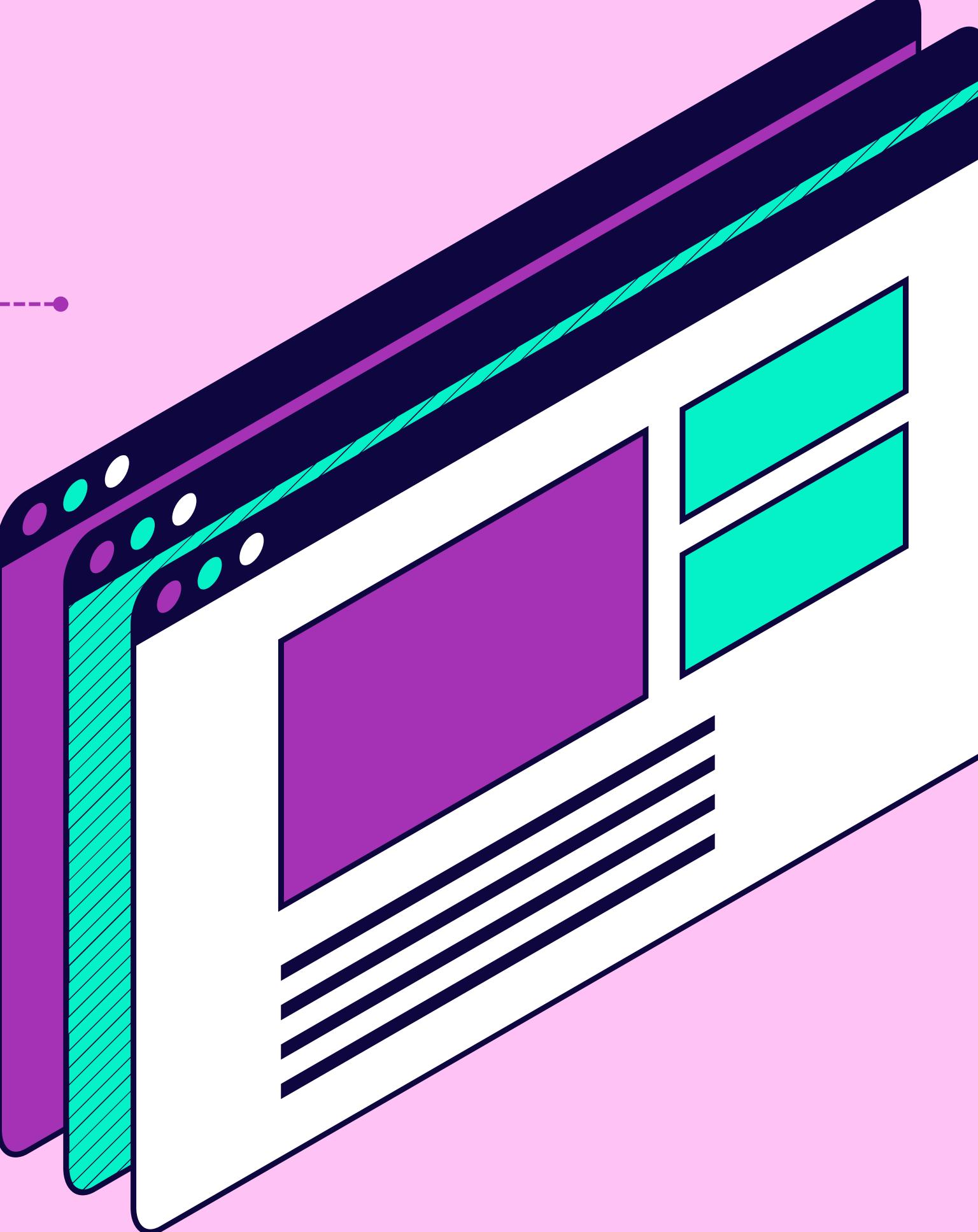
Mais tarde falaremos mais sobre o que exatamente é uma função. Por agora, perceba que qualquer coisa entre os parênteses será mostrada (ou “impressa”) no terminal.

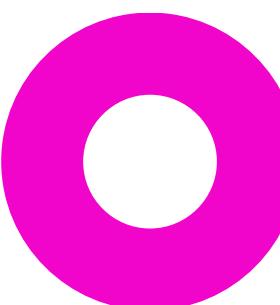
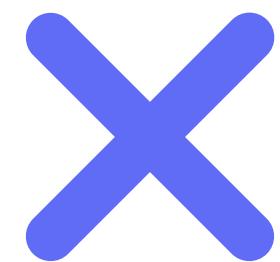
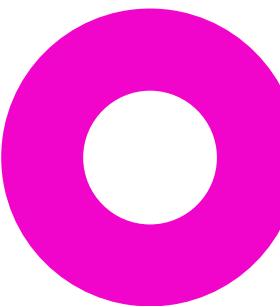
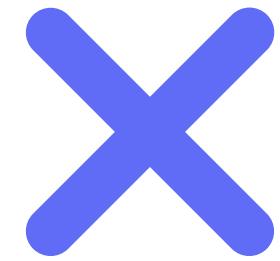
Essa função é muito importante em nosso aprendizado de uma linguagem de programação, principalmente porque nos permite ver os resultados do código sem a necessidade de uma interface gráfica.

# Strings e variáveis

---

**Você pode ter notado que colocamos aspas duplas ("") na expressão Hello World. Que tal tirá-las e ver o que acontece?**





“Hello” não  
está definido

## O que significa?

---

O erro ocorre porque o termo Hello está sendo interpretado como uma variável, mas ela não foi definida.

As aspas servem para identificar cadeias **indexadas** de caracteres, que chamamos de String. Usamos String quando queremos utilizar texto no código, como no caso do “Hello World”.

# Variáveis



Na programação, variáveis servem para guardar dados em tempo de execução, para que possámos utilizá-las em algum outro ponto do código.

Para tal, precisamos declará-las com um “rótulo” para que o Python saiba como encontrá-la depois.

Depois de declarada, podemos consultar ou substituir os dados que ela guarda quando quisermos, por meio desse “rótulo”.

# Variáveis

**A declaração de uma variável segue algumas regras:**

---

Estrutura:

`nome = valor`

Em Python, não é possível declarar uma variável sem atribuir um **valor** a ela;

No **nome**, podemos usar apenas letras, números e underline. Em Python, podemos colocar acentos, mas não é recomendável;

O **nome** não pode iniciar com um número;

Não pode ser uma palavra reservada (conheceremos algumas em breve);

Em Python, é possível alterar o tipo do **valor** de uma variável, devido a sua tipagem dinâmica.

# A função “`input()`”

Essa função também será importante no nosso minicurso. Com ela, podemos solicitar entradas de usuário pelo terminal e guardar esse valor em uma variável.



# Tipos de dados

Embora isso seja extremamente abstraído no Python, todo valor de uma variável tem um tipo associado a ela. Os mais importantes são:

**String**

valores unicode (caracteres)

**int**

números inteiros

**float**

números de ponto flutuante (números reais)

**bool**

valores booleanos (verdadeiro / falso)

**list**

listas

# Operadores Aritméticos

A matemática é um dos grandes pilares da computação.

- + adição // divisão inteira
- subtração % resto
- \* multiplicação < menor que
- / divisão > maior que
- \*\* potência <= menor ou igual
- == igualdade >= maior ou igual
- != diferença

# Manipulação de Strings

---

```
nome = "João"  
idade = 18  
  
print("{} tem {} anos!".format(nome, idade))  
  
print(f"{nome} tem {idade} anos!")
```

Assim como as listas, também podemos manipular Strings:

- `+`: quando estamos trabalhando com Strings, o símbolo `+` é tratado como concatenação;
- Se quisermos colocar valores de variáveis numa String, podemos usar:
  - o método `format`
  - f-Strings

Também temos alguns métodos interessantes para manipulação de Strings, como:

- `capitalize`: torna a primeira letra da String maiúscula;
- `upper`: torna todas as letras da String maiúsculas;
- `lower`: torna todas as letras da String minúsculas;
- `find`: busca o índice de um caractere na String;

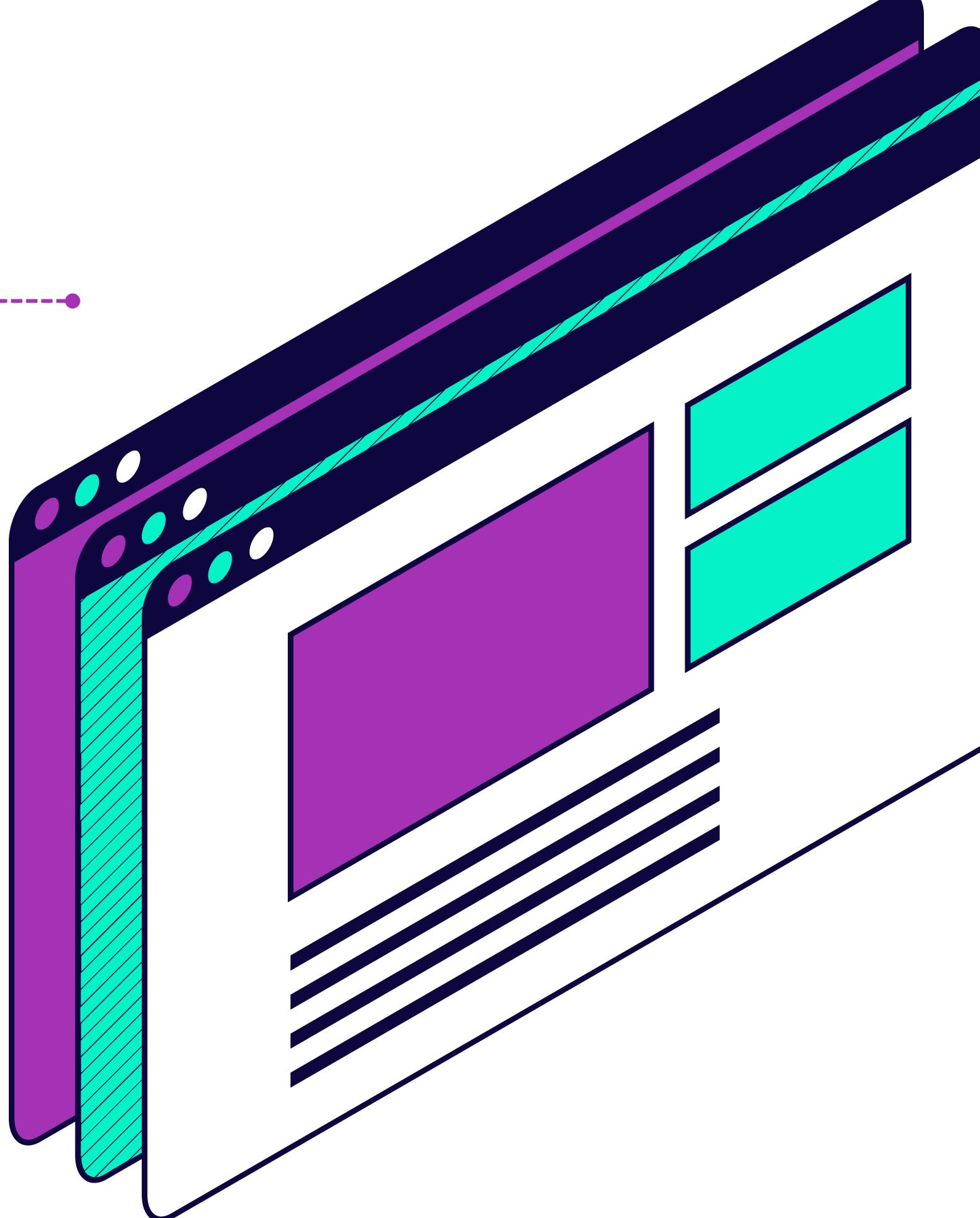
# Exercício 1

---

**Crie um algoritmo que solicite ao usuário a largura e a altura de um retângulo e calcule sua área e perímetro**

Exemplo de Saída esperada:

```
>> Digite a largura do retângulo: 5  
>> Digite a altura do retângulo: 3  
    >> Área do retângulo: 15  
>> Perímetro do retângulo: 16
```



# Listas

---

Listas são coleções de valores **indexados**. Podemos utilizá-las para guardar diversos valores numa só estrutura, o que é útil para a resolução de diversos problemas.

Existem muitos métodos importantes para manipular listas, como:

- **append**: adiciona um novo valor a uma lista;
- **pop**: remove um valor da lista pelo índice;
- **remove**: remove um valor da lista pelo próprio valor;
- **sort**: ordena a lista.

Além disso, podemos acessar um valor em uma determinada posição da lista utilizando seu índice, no formato `Lista[índice]`. Nesse ponto, como as Strings são, de certa forma, uma “lista de caracteres”, também podemos fazer isso com elas.



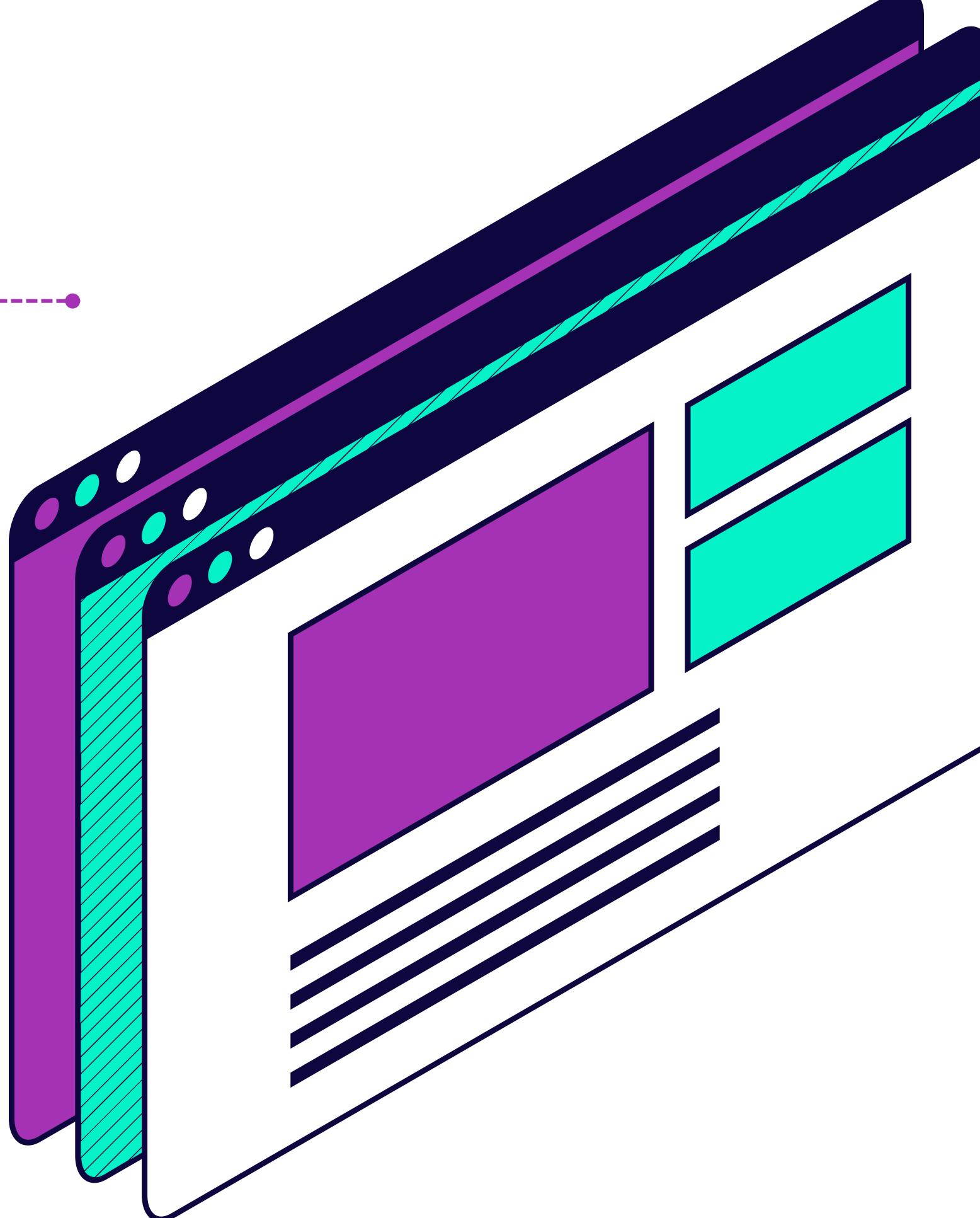
# Exercício 2

---

**Crie um algoritmo que solicite ao usuário 3 números e os exiba em ordem crescente**

Exemplo de Saída esperada:

```
>> Digite o primeiro número: 5  
>> Digite o segundo número: 3  
>> Digite o terceiro número: 2  
>> Lista ordenada: [2, 3, 5]
```





# Hora da Lógica

---

Podemos fazer um Hello World sem sabermos nada de lógica, mas não podemos ir muito além disso sem ela.

Por isso, todas as linguagens de programação possuem operadores lógicos, que conheceremos a seguir.

---

# Verdadeiro ou Falso

Nas expressões lógicas que veremos a seguir, perceberemos que seus valores lógicos sempre resultam em um dos dois estados: verdadeiro ou falso.

Para uma linguagem de programação, muitas vezes esse resultado significa prosseguir ou não com a execução de uma parte do código, a depender da expressão utilizada.



# Condicional

(**se / então**)

- A expressão “se ... então ...” representa o conectivo lógico condicional.
- Utilizamos isso quando queremos expressar que uma coisa depende de outra:
  - Se** nasceu em Alagoas, **então** é alagoano.
  - Se** não come carne, **então** é vegetariano.
  - Se** tem 18 anos ou mais, **então** é adulto.
  - Se** tem carro, **então** tem CNH.
- Em Python, utilizamos a seguinte estrutura para essa expressão:

```
if condição :  
    consequência
```

# Condicional

(se / então)

É importante deixar claro que, por mais óbvia que possa parecer a afirmação, precisamos explicitá-la. Por exemplo:

Se é divisível por 2, então é par

Se tomarmos o número 3 como exemplo, podemos afirmar que ele não é par, porém não podemos afirmar que ele é ímpar, pois não definimos nenhuma expressão para isso.

Podemos resolver isso criando mais uma expressão, como:

Se não é divisível por 2, então é ímpar

Porém, em problemas binários (como par/ímpar), temos uma outra expressão para resolver isso.

# Condicional

(se não)

Podemos utilizar o “se não” para complementar o “se / então” em alguns casos. Por exemplo:

Se é divisível por 2, então é par  
Se não, então é ímpar

Em Python, escrevemos:

```
if condição:  
    consequência  
else:  
    consequência
```

# Condicional

(**se / então + se não**)

Também podemos misturar ambos, como por exemplo:

Se tem 18 anos ou mais, então é adulto

Se não se tem 12 anos ou mais, então é adolescente

Se não, então é criança

Em Python, temos o **elif** (**else + if**) para isso:

```
if condição:  
    consequência  
elif condição:  
    consequência
```

# Conectivos Lógicos

Um conectivo lógico é um símbolo ou palavra usado para conectar duas ou mais sentenças. Sem perceber, os utilizamos constantemente em nossas falas no dia a dia.

A seguir, apresentaremos alguns desses conectivos lógicos:

Negação, Conjunção, Disjunção Inclusiva e Disjunção Exclusiva, Condicional e Bicondicional

Vamos ao Python!



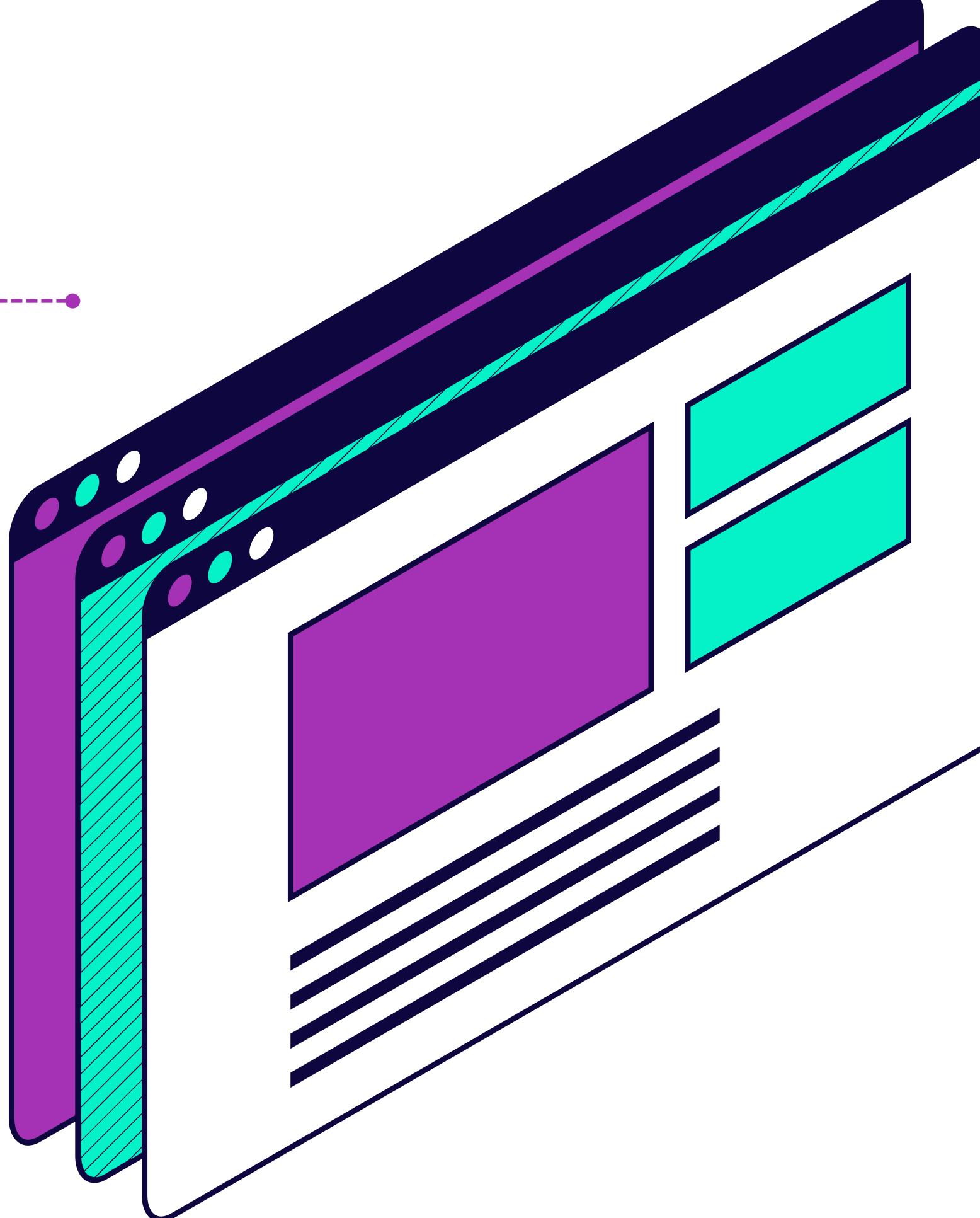
# Exercício 3

---

**Crie um algoritmo que solicite ao usuário 2 valores e mostre o maior entre eles.**

Exemplo de Saída esperada:

```
>> Digite o primeiro valor: 5  
>> Digite o segundo valor: 3  
    >> Maior valor: 5
```



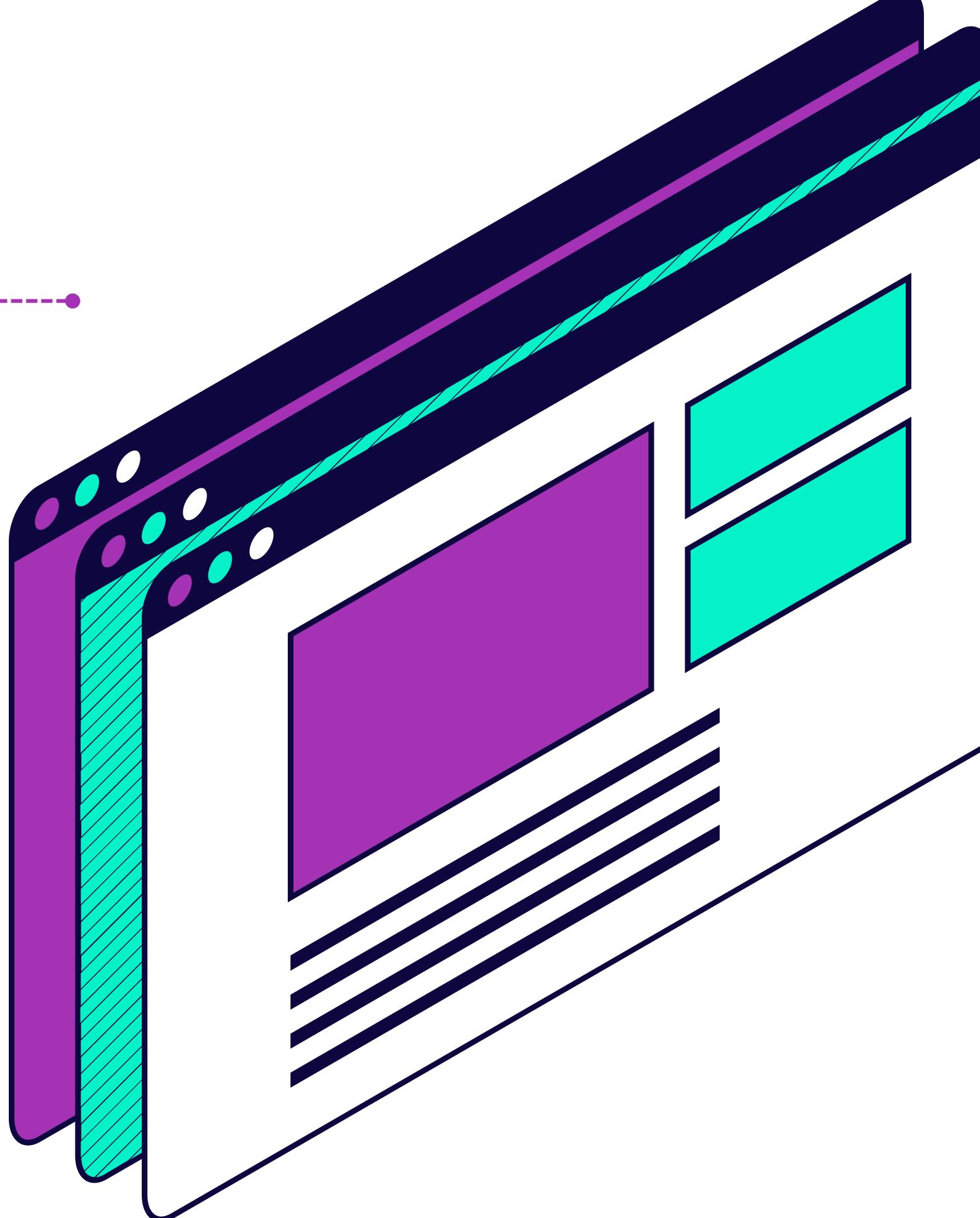
# Exercício 4

---

**Escreva um programa que  
leia as medidas dos lados  
de um triângulo e  
escreva se ele é Equilátero,  
Isósceles ou Escaleno.**

Exemplo de Saída esperada:

```
>> Digite a medida do lado A: 5
>> Digite a medida do lado B: 3
>> Digite a medida do lado C: 2
>> O triângulo é escaleno!
```



```
while n < 10:  
    print(n)  
    n = n + 1
```

```
for n in range(10):  
    print(n)
```

# Estruturas de repetição

Muitas vezes, precisamos que parte do código se repita até que alcancemos uma condição de parada. Para isso, usamos as estruturas de repetição. Temos duas formas de fazer isso: `for` e `while`.

Percorrer listas é um exemplo comum de utilidade das estruturas de repetição.

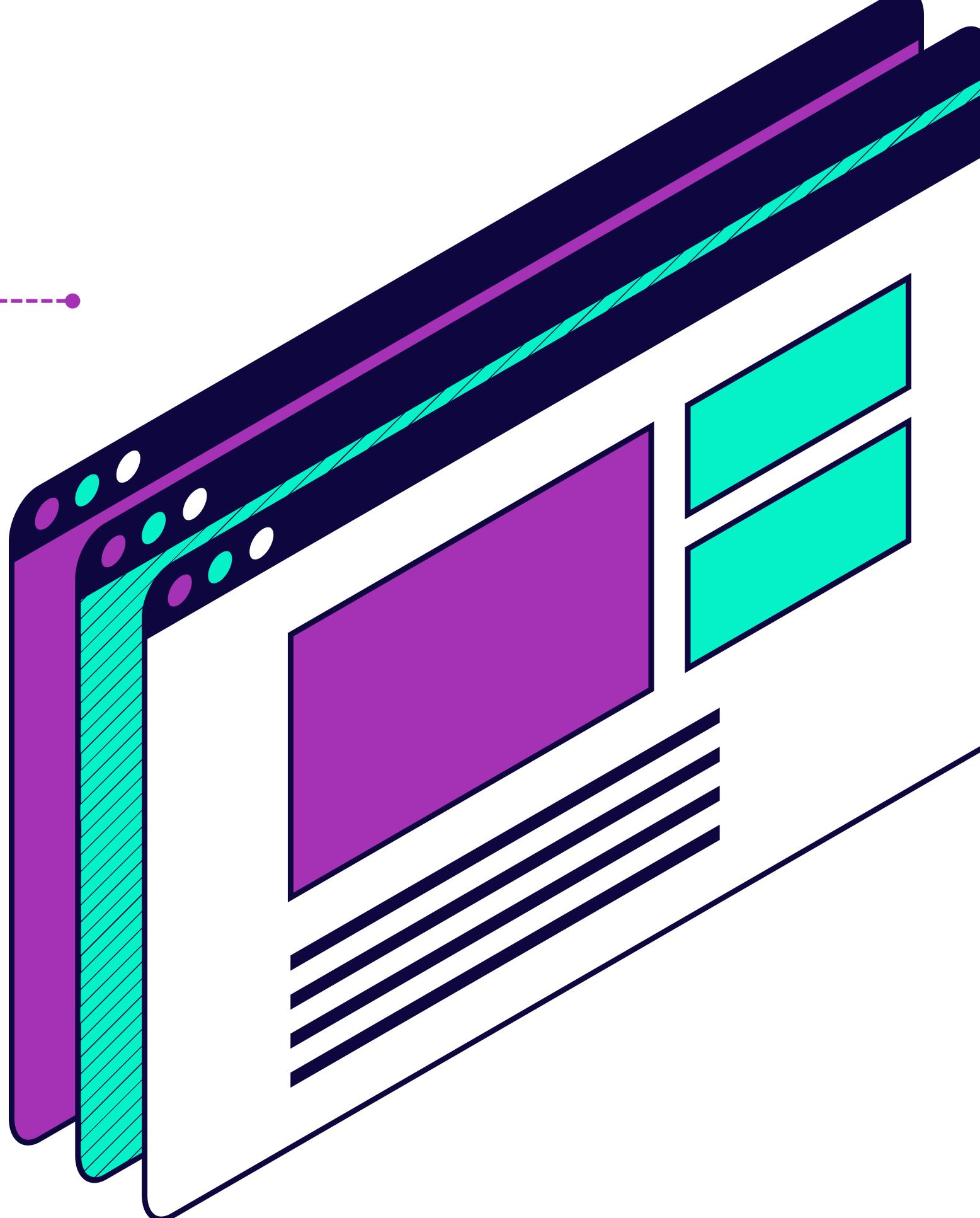
# Exercício 5

---

**Escreva um algoritmo que leia números até receber o número 0 e mostre a soma dos números digitados.**

Exemplo de Saída esperada:

```
>> Digite um número: 5  
>> Digite um número: 13  
>> Digite um número: 0  
>> A soma dos valores é 18
```

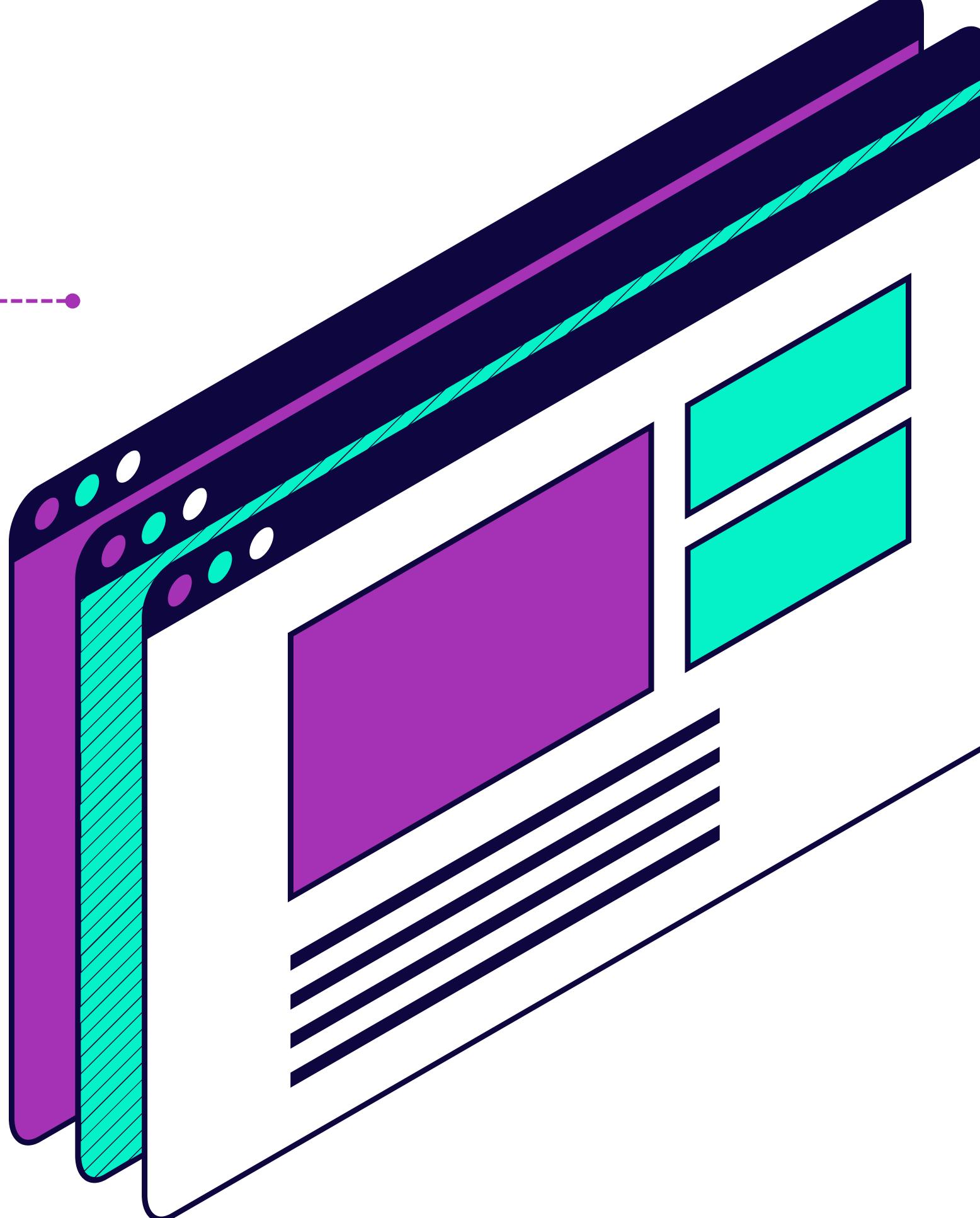


# Exercício 6

---

**Escreva um algoritmo  
encontre o índice do item  
“banana” na lista a seguir:  
[ “laranja”, “uva”, “banana”,  
“melancia”]**

Exemplo de Saída esperada:  
» O item “banana” foi encontrado no  
índice 2



# Funções

```
def soma(x, y):  
    resultado = x + y  
    return resultado
```

```
def duplicar(x):  
    resultado = 2 * x  
    return resultado
```

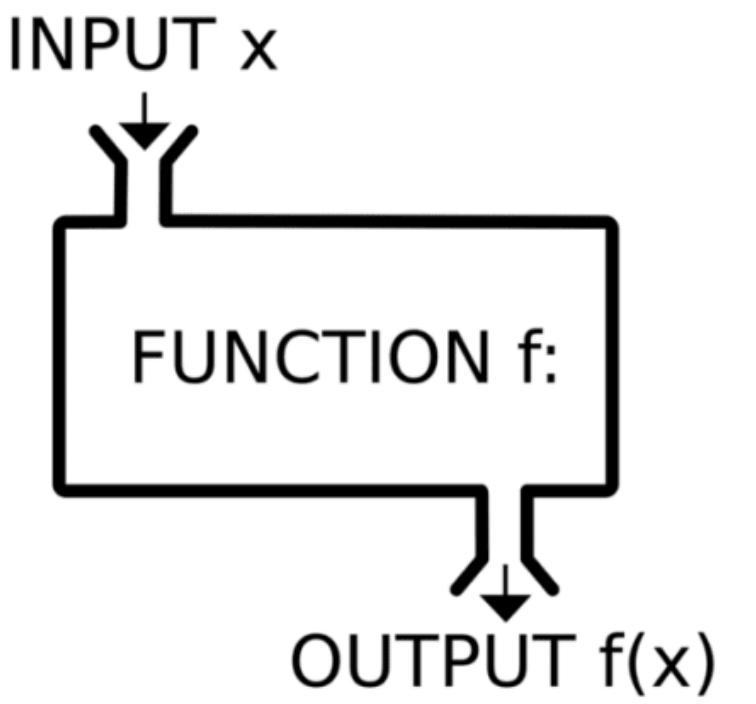
```
def fazer_nada(x):  
    return x
```

Funções são blocos de código que podem ser executados sempre que a função é chamada.

Algumas funções importantes já vem implementadas pelo próprio python, como por exemplo:

- `print(x)`: Imprime x no terminal;
- `input(x)`: Imprime x e captura entradas do usuário no terminal;
- `len(x)`: Calcula tamanho de x;
- `range(x)`: Cria uma lista com valores de 0 a x.

Mas também podemos criar nossas próprias funções



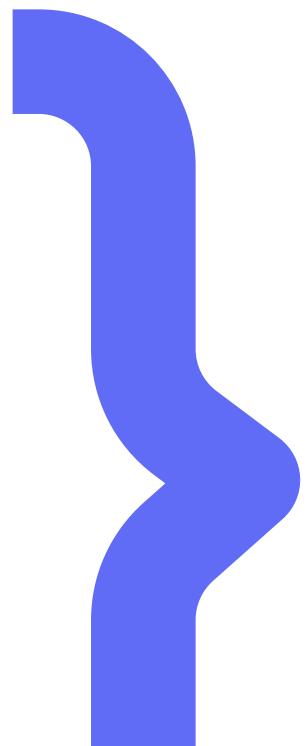
```
def nome(parâmetros):  
    #escopo
```

# Funções

Uma função em Python é dividida em 3 partes: nome, parâmetros e escopo da função.

A função, antes de ser utilizada, deve ser definida, de forma semelhante a uma variável.

As funções contribuem muito para a legibilidade e reutilização de código.



# Obrigado pela atenção!

