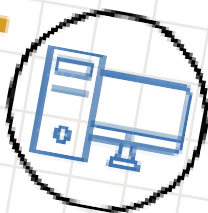


CODE
LAB TEEN



PYTHON AULA 20





BEM - VINDOS!



AGENDA

Hoje vamos revisar os conteúdos vistos até hoje, com maior foco em funções e laços de repetição

VARIÁVEIS-REVISÃO

Em Python, para criar uma variável, basta usar uma palavra qualquer a atribuir um valor utilizando o sinal “=”.

No exemplo abaixo temos variáveis guardando nome, nota e idade de um aluno:

```
nome = "João"  
nota = 7.8  
idade = 16
```

VARIÁVEIS-REVISÃO

Para receber um valor do usuário e atribuir à variável basta atribuir a função “input()” à variável:

```
nome = input()
```

VARIÁVEIS-REVISÃO

Uma variável pode ser uma lista contendo vários dados diferentes:

É importante lembrar que a lista enumera os itens a partir do 0, ou seja, o primeiro item é o 0, o segundo é o 1...

```
nomes = ["João", "Maria", "Enzo"]  
print(nomes[2]) # Saída: Enzo
```

VARIÁVEIS-REVISÃO

Uma variável pode ser usada para montar frases em outra variável:

```
nome = "Matheus"  
mensagem = "Bem-vindo, " + nome + "!"  
print(mensagem) # Saída: Bem-vindo, Matheus!"
```

VARIÁVEIS-REVISÃO

Atenção!

Deve-se evitar usar palavras reservadas como o “print” para criar variáveis, pois isso faz com que ela perca sua função original. No exemplo, o “input” perde sua função de receber valores do usuário:

```
input = "Caio"  
nome = input()  
print(nome) # Saída: TypeError: 'str' object is not callable
```

BIBLIOTECAS-REVISÃO

Bibliotecas são um conjunto de funções que podem ser importadas ao código. Para adicionar uma basta usar o comando “import” e o nome da biblioteca em seguida.

```
import numpy as np
```


BIBLIOTECAS-REVISÃO

Podemos utilizar o numpy para criar vetores, utilizando o `np.array()`. Com o vetor criado, podemos percorrer com o FOR

```
import numpy as np

vetor = np.array([1,2,3,4])

for i in range(4):
    print(f"indice: {i} valor: {vetor[i]}")
```

```
indice: 0 valor: 1
indice: 1 valor: 2
indice: 2 valor: 3
indice: 3 valor: 4
```

EXPRESSÕES ARITMÉTICAS-REVISÃO

As operações básicas em Python são:

Adição: `“+”`

Subtração: `“-”`

Multiplicação: `“*”`

Divisão: `“/”`

Divisão por inteiro: `“//”`

Resto da divisão: `“%”`

Potência: `“**”`

EXPRESSÕES ARITMÉTICAS-REVISÃO

Exemplos:

`a = 5`

`b = 2`

`soma = a + b # soma = 7`

`subtracao = a - b # subtracao = 3`

`multiplicacao = a * b # multiplicacao = 10`

`divisao = a / b # divisao = 2.5`

`divisaoInteira = a // b # divisaoInteira = 2`

`divisaoResto = a % b # divisaoResto = 1`

`potencia = a ** b # potencia = 2^5 = 32`

CONDICIONAIS-REVISÃO

Laços de condição são funções que, se a condição estabelecida for atendida, realiza certa ação. Em Python existem dois tipos de condicionais:

simples: `if`

composta: `if else`

CONDICIONAIS-REVISÃO

Condicionais simples usam apenas o `if`. Para usar o `if` deve-se escrever `“if():”` e dentro do parênteses escrever a condição desejada. Importante lembrar que é necessário indentar corretamente a parte do código que estará dentro do `if`.

CONDICIONAIS-REVISÃO

Exemplo:

```
a = 2
if(a > 0):
    a = a + 1;
print(a) # Saída: 3
```

CONDICIONAIS-REVISÃO

Condicionais compostas usam tanto o `if` quanto o `else`. Para usar o `else`, basta escrever “`else:`” logo após o fim do bloco de códigos dentro do `if`. Também é necessário indentar o código que ficará dentro do `else`.

CONDICIONAIS-REVISÃO

Exemplo:

```
a = -2
if(a > 0):
    a = a + 1;
else:
    a = a - 1
print(a) # Saída: -3
```


ELIF-REVISÃO

“Elif():” é uma forma reduzida de se escrever “else if():”. Essa estrutura permite adicionar várias condições que tem relação entre si de forma sequencial. É interessante quando se tem mais de duas condições possíveis.

ELIF-REVISÃO

Exemplo:

```
a = 5
if(a > 0):
    print("Número positivo")
elif(a < 0):
    print("Número negativo")
else:
    print("Número igual à zero")
```

SWITCH CASE-REVISÃO

Switch case é uma estrutura de decisão comum nas linguagens de programação. Entretanto o python não possui uma função nativa para switch case. Porém é possível utilizar um dicionário para simular um switch.

SWITCH CASE-REVISÃO

Exemplo:

```
veiculo = "carro"

match veiculo:
    case "moto":
        print("veículo é uma moto")
    case "carro":
        print("veículo é um carro")
    case "caminhão":
        print("veículo é um caminhão")
    case _:
        print("não é nenhum desses")
```

```
veículo é um carro
```

SWITCH CASE E ELIF-REVISÃO

Tanto o switch case quanto o elif possuem o mesmo propósito: criar uma estrutura que seja capaz de analisar vários casos e responder de forma diferente para cada um. Sempre é possível reescrever um deles utilizando o outro, porém é preciso analisar a situação e escolher o que melhore se encaixa para as necessidades do código.

FOR-REVISÃO

Utilizando o For, podemos repetir um bloco de código uma quantidade fixa de vezes.

Repetindo n
vezes

Código

Bloco de
Código

FOR-REVISÃO

Relembrando como o **FOR** funciona:

Variável de
Iteração

```
for i in range(1, 4, 1):  
    print(i)
```

1
2
3

Fim
(não incluso)

Passo

Início

FOR-REVISÃO

Relembrando como o **FOR** funciona:

Nesse caso, o for
vai acontecer de 1
a 3, com passo 1

```
for i in range(1, 4, 1):  
    print(i)
```

1
2
3

A saída será
imprimir i em cada
iteração

FOR-REVISÃO

Desafio: Qual a saída do código a seguir?


```
for i in range(11):  
    print(2*i + 2)
```

FOR-REVISÃO

Desafio: Qual a saída do código a seguir?

```
for i in range(11):  
    print(2*i + 2)
```

2
4
6
8
10
12
14
16
18
20
22



**Atenção ao valor
de i e ao +2 dentro
do print**

FOR-REVISÃO

Agora vamos analisar as 4 primeiras iterações do laço

Iteração	1
i	0
2i	0
2i+2	2

FOR-REVISÃO

Agora vamos analisar as 4 primeiras iterações do laço

Iteração	1	2
i	0	1
2i	0	2
2i+2	2	4

FOR-REVISÃO

Agora vamos analisar as 4 primeiras iterações do laço

Iteração	1	2	3
i	0	1	2
2i	0	2	4
2i+2	2	4	6

FOR-REVISÃO

Agora vamos analisar as 4 primeiras iterações do laço

Iteração	1	2	3	4
i	0	1	2	3
2i	0	2	4	6
2i+2	2	4	6	8

Lembrando que o i vai até o final - 1

FOR-REVISÃO

Agora vamos ver como o FOR funciona com strings e listas. Podemos usar o FOR para percorrer listas e strings e fazer operações.



FOR-REVISÃO

Neste exemplo, o código imprime cada elemento da lista. Em cada iteração, `i` tem o valor de um dos elementos da lista.

```
lista = ["abacaxi", "tenis", "gato"]  
  
for i in lista:  
    print(i)
```

```
abacaxi  
tenis  
gato
```

Percebam que não usamos mais o `range()` e sim a própria lista

FUNÇÕES - REVISÃO

Uma função é um tipo de recurso que pega informações e devolve para nós uma outra informação.

Informações
de Entrada

“Bom dia!”

Função

Informações
de Saída

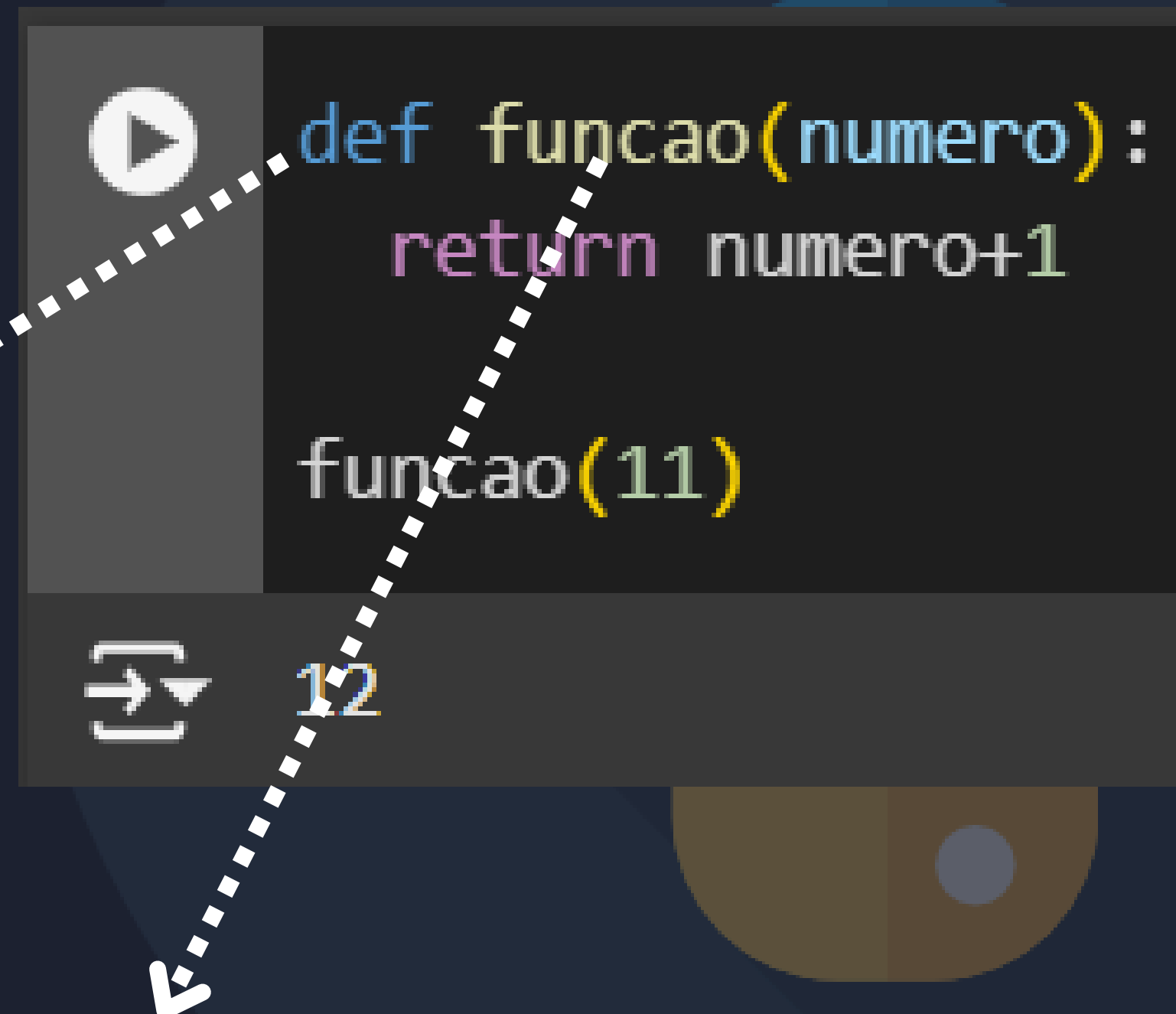
“Bom dia! Alunos!”

FUNÇÕES - REVISÃO

Na linguagem de programação Python, as funções tem uma estrutura especial.

`def:` diz para o computador que estamos iniciando uma função.

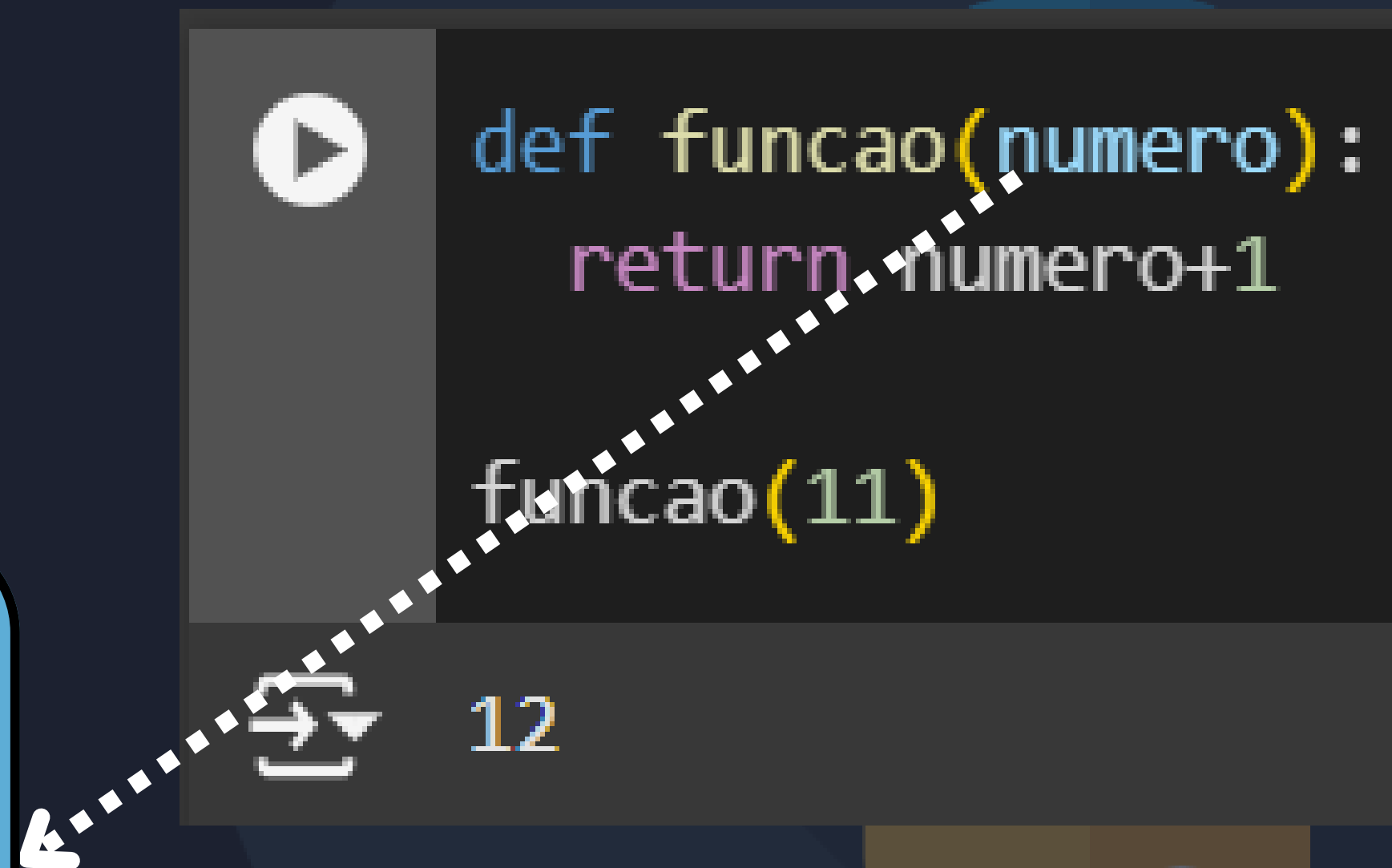
nome da função: pode ser qualquer nome, assim como a variavel.



FUNÇÕES-REVISÃO

Na linguagem de programação Python, as funções tem uma estrutura especial.

parâmetros: são as informações de entrada da função que serão guardados em variáveis, neste caso chamada numero.



```
def funcao(numero):  
    return numero+1  
  
funcao(11)
```

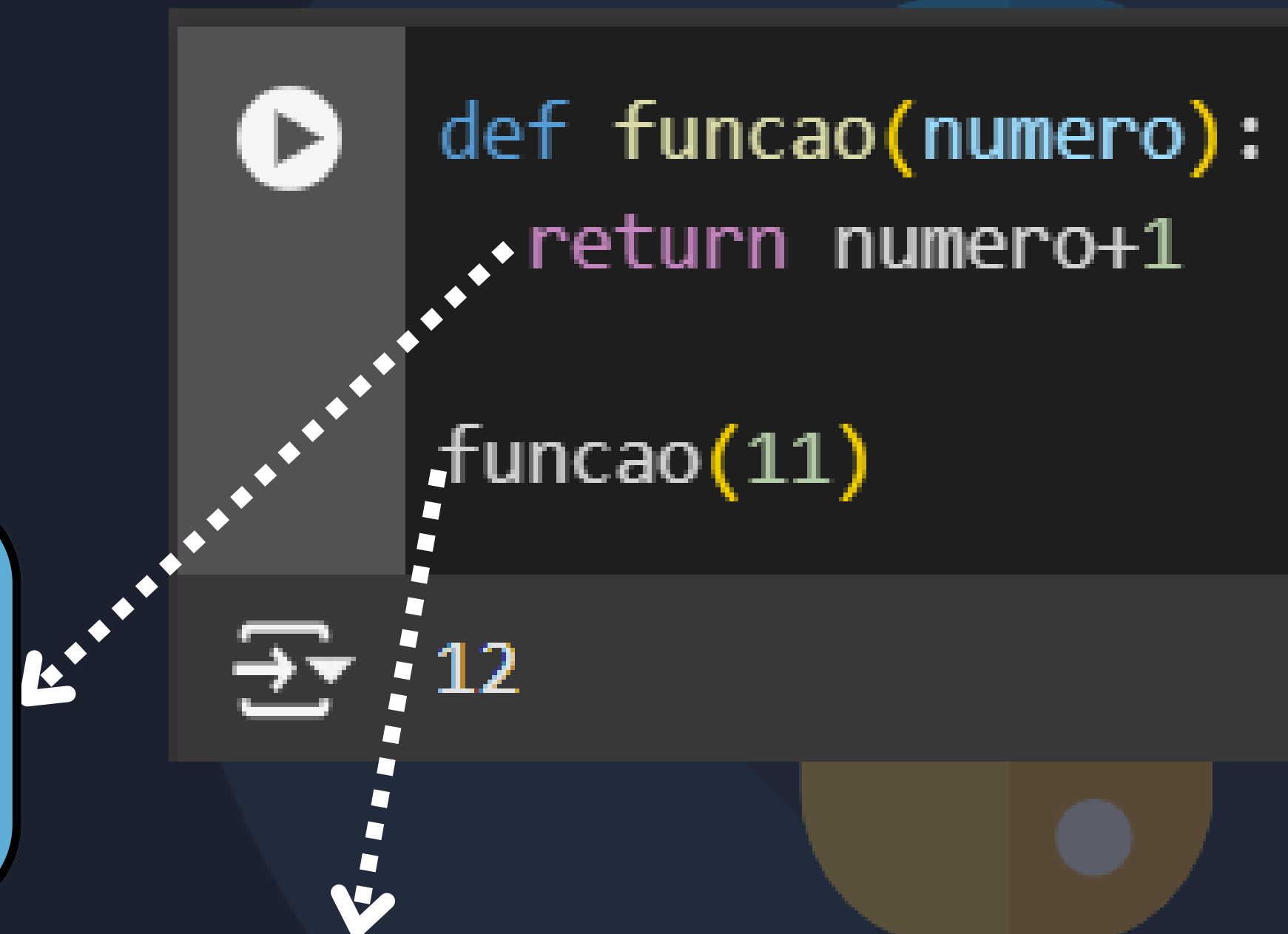
12

IMPORTANTE: Note a indentação que forma o “bloco” da função.

FUNÇÕES - REVISÃO

Na linguagem de programação Python, as funções tem uma estrutura especial.

return: diz para o código que ele vai devolver uma informação de saída.



OBS: Note que chamamos a função e pedimos para que ela some 11 com 1. Logo, a função retorna 12.

FUNÇÕES-REVISÃO

Vamos alterar o código e ver o que acontece, para fixar a sintaxe.

Note a mudança no nome da função e o fato de que podemos reutilizá-la no código para somar 1 a QUALQUER valor que quisermos.



```
def teste(numero):  
    return numero+1
```

```
print(teste(11))  
print(teste(111))
```



```
12  
112
```

FUNÇÕES-REVISÃO

Vamos alterar o código e ver o que acontece, para fixar a sintaxe.

Modificamos a função teste para que ela receba mais de um parâmetro. Veja que, os parâmetros são separados por vírgulas.



```
def teste(numero_um, numero_dois):  
    return numero_um + numero_dois  
  
print(teste(11, 45))
```



56

Quando chamamos a função com mais de um parâmetro também colocamos uma vírgula. A função está somando o numero_um (11) e o numero_dois (45).

LISTA-REVISÃO

Estruturas de dados mutáveis que armazenam coleções ordenadas de elementos.

- Adição: Use `append()` ou `insert()` para colocar mais itens.
- Subtração: Use `remove()` ou `pop()` retirar.

```
frutas = ["maçã", "banana", "laranja"]  
print(frutas[0]) # "maçã"
```

```
frutas.append("uva") # Adiciona "uva" à lista  
print(frutas) # ["maçã", "banana", "laranja", "uva"]  
  
frutas.remove("banana") # Remove "banana" da lista  
print(frutas) # ["maçã", "laranja", "uva"]
```

VETORES-REVISÃO

Estruturas para cálculos matemáticos eficientes, usando arrays.

- Operções Aritiméticas:
+ , - , * , / , ^ ...
- Estatísticas: mean(), sum(), max(), min()

```
import numpy as np
```

```
vetor = np.array([1, 2, 3, 4, 5])  
print(vetor) # [1 2 3 4 5]
```

```
print(vetor + 2) # [3 4 5 6 7]  
print(vetor - 1) # [0 1 2 3 4]
```

```
print(np.mean(vetor)) # Média: 3.0  
print(np.sum(vetor)) # Soma: 15  
print(np.max(vetor)) # Máximo: 5  
print(np.min(vetor)) # Mínimo: 1
```


LISTAS VS VETORES

Aspecto	Listas	Vetores (NumPy)
Definição	Estruturas padrão do Python que podem guardar diferentes tipos de dados.	Estruturas específicas para números, otimizadas para cálculos matemáticos.
Tipos de Dados	Pode misturar tipos (ex.: números, strings).	Todos os elementos devem ter o mesmo tipo.
Performance	Mais lentas para cálculos.	Muito mais rápidas para operações numéricas.
Operações	Não realizam cálculos diretamente.	Suporte direto a soma, multiplicação, etc.

MANIPULAÇÃO DE STRINGS-REVISÃO

- `upper()` / `lower()`: Transforma todos os caracteres da string em maiúsculas ou minúsculas.
- `replace("antigo", "novo")`: Substitui uma parte específica da string por outra.
- `split(" ")` / `join(lista)`: Divide uma string em partes (lista) ou junta uma lista em uma string.
- `len(string)`: Retorna o número total de caracteres na string.

MANIPULAÇÃO DE STRINGS-REVISÃO

```
# 1. upper() / lower()
texto = "Python é incrível!"
print(texto.upper()) # "PYTHON É INCRÍVEL!"
print(texto.lower()) # "python é incrível!"

# 2. replace("antigo", "novo")
frase = "Eu gosto de Python."
print(frase.replace("Python", "programar")) # "Eu gosto de programar."
```

MANIPULAÇÃO DE STRINGS-REVISÃO

```
# 3. split(" ") / join(lista)
texto_dividir = "Aprender Python é divertido"
palavras = texto_dividir.split(" ") # Divide em palavras
print(palavras) # ['Aprender', 'Python', 'é', 'divertido']

frase_juntar = " ".join(palavras) # Junta de volta
print(frase_juntar) # "Aprender Python é divertido"

# 4. len(string)
texto_tamanho = "Python"
print(len(texto_tamanho)) # 6 (quantidade de caracteres)
```



***REVISAMOS BASTANTE
COISA HOJE...***

FICOU ALGUMA DUVIDA?

OBRIGADO!

Contem para gente o que você achou da aula de hoje:



<https://forms.gle/UgQfD8MfGhmP6NXS7>