Funções

Funções são **blocos de código reutilizáveis** que executam uma tarefa específica.

```
def nome_da_função(parâmetros):
    # Bloco de código da função
    return valor_opcional
```

Sua sintaxe é definida usando a palavra-chave def, seguida pelo nome da função e parênteses que podem conter parâmetros. O bloco de código da função é **indentado** e pode conter um retorno opcional com return.

Funções

```
def ola_mundo(nome):
    return f"Olá, {nome}!"


ola_mundo("Laís")
```

Parâmetros são variáveis definidas na declaração da função, elas podem ser obrigatórias ou opcionais, sendo usadas para tornas as funções mais flexíveis, permitindo que elas processem diferentes valores sem precisar modificar o código.

Argumentos são os valores passados para a função quando ela é chamada, caso essa função tenha um parâmetro obrigatório.

Para chamar uma função, basta usar o nome da função seguido de parênteses.

alura

Funções

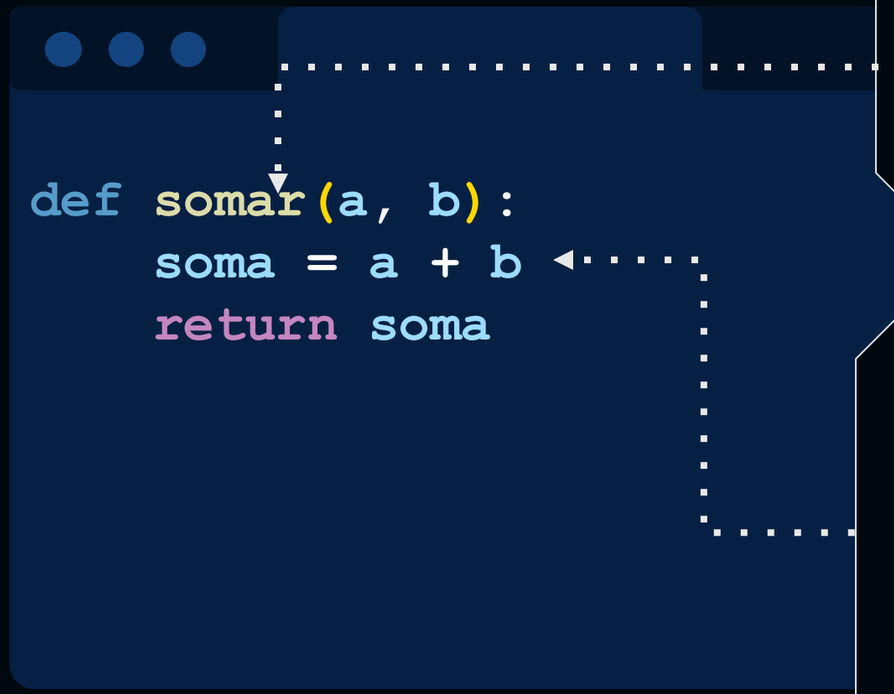
```python
def ola_mundo(nome):
    return f"Olá, {nome}!"


print(ola_mundo("Laís"))
```

A saída desse código é:
Olá, Laís!

Funções simples

Estas são as funções criadas e personalizadas pelo dev para realizar tarefas específicas.



A função somar recebe dois parâmetros (a e b) e retorna a soma desses valores.

O bloco de execução e as variáveis criadas dentro de uma função são locais, ou seja, **só existem dentro dela**. Isso significa que elas não podem ser acessadas fora do seu bloco de código.

Funções com parâmetros padrão

Estas são as funções permitem definir valores padrão para os parâmetros caso nenhum argumento seja passado.

```python
def cumprimentar(nome = "Visitante"):
    print(f"Olá, {nome}!")


cumprimentar() # Saída: Olá, Visitante!

cumprimentar("Jade") # Saída: Olá, Jade!
```

Se nenhum nome for passado, **o padrão será "Visitante"**. Se um nome for fornecido, ele será usado na saudação.

Funções built-in

São funções já incorporadas ao Python que podem ser usadas diretamente sem precisar importá-las ou defini-las. Referente à Interação, temos:

Método	Conceito	Exemplo	Saída
print()	Exibe valores no console.	print("Olá, mundo!")	Olá, mundo!
input()	Lê uma entrada do usuário como string.	nome = input("Digite: ")	Digite: Júnior

Funções built-in

Já para as funções de manipulação, temos:

Método	Conceito	Exemplo	Saída
type()	Retorna o tipo de um objeto.	type(10)	<class 'int'>
isinstance()	Verifica se um objeto pertence a um tipo específico ou a uma tupla de tipos. Retorna True se for do tipo indicado, caso contrário, retorna False.	isinstance(10.5, int) isinstance("Python", (int, str))	False True
len()	Retorna o tamanho de uma string, lista ou tupla.	len("Python")	6

Funções built-in

Já para as funções de conversão e criação de tipos, temos:

Método	Conceito	Exemplo	Saída
str()	Converte um valor para string.	str(123)	"123"
int()	Converte um valor para inteiro.	int("10")	10
float()	Converte um valor para decimal.	float("3.14")	3.14
bool()	Converte um valor para **True** ou **False**.	bool(1)	True
list()	Converte um iterável para lista.	list("abc")	["a", "b", "c"]
dict()	Cria um dicionário.	dict(nome = "Ana")	{"nome" : "Ana"}
set()	Cria um conjunto.	set([1, 2, 3])	{1, 2, 3, 4}

Funções built-in

Já para as funções de matemáticas, temos:

Método	Conceito	Exemplo	Saída
abs()	Retorna o valor absoluto de um número.	abs(-10)	10
round()	Arredonda um número.	round(3.1415, 2)	3.14
max()	Retorna o maior valor entre os fornecidos.	max(3, 5, 1)	5
min()	Retorna o menor valor entre os fornecidos.	min(3, 5, 1)	1
sum()	Retorna a soma de uma lista de números.	sum([1, 2, 3])	6

Funções built-in

Já para as funções que trabalham com elementos iteráveis:

Método	Conceito	Exemplo	Saída
filter(operação, iterável)	Filtra elementos de um iterável com base em uma condição.	list(filter(lambda x : x > 2, [1, 2, 3, 4]))	[3, 4]
map(função, iterável)	Aplica uma função a cada elemento de um iterável.	list(map(lambda x : x * 2, [1, 2, 3]))	[2, 4, 6]
zip(iterável, iterável2, ...)	Une dois ou mais iteráveis, criando pares de elementos correspondentes.	list(zip([1, 2, 3], ["a", "b", "c"]))	[(1,"a"), (2, "b"), (3, "c")]
sorted(iterável, key=função, reverse=bool)	Retorna uma nova lista ordenada a partir de um iterável. Pode receber um argumento key para personalizar a ordenação.	sorted([3, 1, 4, 2])	[1, 2, 3, 4]
reversed(iterável)	Retorna um iterador com os elementos de um iterável na ordem inversa.	list(reversed([1, 2, 3]))	[3, 2, 1]

Funções built-in

Para conhecer todas as funções built-in, consulte a documentação oficial do Python.

Funções recursivas

São aquelas funções que **chamam a si mesmas** para resolver problemas repetitivos até atingir uma condição de parada.

```python
def fatorial(n):
    if n == 0:
        return 1
    return n * fatorial(n - 1)

print(fatorial(5)) # Saída: 120
```

Condição de Parada

Chamada Recursiva

Elas são usadas para resolver problemas que podem ser divididos em subproblemas menores, como cálculos matemáticos e algoritmos de estrutura de dados.

Funções anônimas (lambdas)

Lambda é uma função pequena e sem nome que pode ter múltiplos argumentos, mas apenas uma expressão.

```
# Sintaxe
lambda argumentos: expressão


soma = lambda a, b: a + b
print(soma(3, 5)) # Saída: 8
```

Lambdas são úteis para operações simples ou quando precisamos de funções curtas dentro de outras funções.

Funções dentro de funções (closures)

Closure é uma **função interna** que lembra do ambiente onde foi criada, mesmo após a execução da função externa ter terminado. Isso permite que ela mantenha o estado de variáveis sem a necessidade de usá-las globalmente.

```python
def multiplicador(n): # Função externa
    def multiplica(x): # Closure
        return x * n
    return multiplica # Retorna a Função Interna


triplo = multiplicador(3)
valor = triplo(5)
print(valor)    # Saída: 15
```

Usa a variável 'n' da função externa.

As closures são úteis para encapsular lógicas e criar funções especializadas sem usar variáveis globais.

Funções dentro de funções (closures)

Outro exemplo do uso de Closures seria um gerador de saudação personalizado:

```
def criar_saudacao(saudacao):
    def saudar(nome):
        return f"{saudacao}, {nome}!"
    return saudar

bom_dia = criar_saudacao("Bom dia")
boa_noite = criar_saudacao("Boa noite")
print(bom_dia("Vini"))     # Saída: Bom dia,
Vini!
print(boa_noite("Ana")) # Saída: Boa noite, Ana!
```

alura

Compartilhe um resumo de seus novos conhecimentos em suas redes sociais.

#aprendizadoalura