100 XP

Noções básicas das funções do Python

3 minutos

As funções são a próxima etapa após aprender as noções básicas de programação do Python. Em seu modo mais simples, uma função contém um códigos que sempre retornam um ou mais valores. Em alguns casos, a função também pode ter entradas opcionais ou obrigatórias.

Quando você começa a escrever códigos que duplicam outras partes do programa, você está diante de uma excelente oportunidade de extrair esse código e criar uma função. Embora o compartilhamento de código comum por meio de funções seja útil, também é possível limitar o tamanho do código extraindo partes dele para criar funções menores (e mais legíveis).

Programas que evitam a duplicação de código e dão preferência para funções menores são mais legíveis e mais fáceis de manter. Eles também são mais fáceis de depurar quando algo não está funcionando conforme o esperado.

Várias regras sobre as entradas de função são essenciais para que você aproveite totalmente tudo o que as funções têm a oferecer.

> **Importante**
>
> Embora estejamos usando o termo *entrada* para descrever o que as funções recebem, esses elementos geralmente são chamados de *argumentos* ou *parâmetros*. Para manter a consistência, neste módulo vamos nos referir às entradas como *argumentos*.

Funções sem argumentos

Para criar uma função, use a palavra-chave `def` seguida por um nome, parênteses e o corpo da função com seu respectivo código:

Python

```
def rocket_parts():
    print("payload, propellant, structure")
```

Nesse caso, `rocket_parts` é o nome da função. Esse nome é seguido por parênteses vazios, o que indica que essa função não requer nenhum argumento. Por último vem o código, recuado com quatro espaços. Para usar a função, você deve chamá-la pelo nome e usar parênteses:

Python

```
rocket_parts()
```

Output

```
payload, propellant, structure
```

A função `rocket_parts()` não recebe nenhum argumento e imprime uma instrução. Se você precisar usar um valor que uma função está retornando, você pode atribuir a saída da função a uma variável:

Python

```
output = rocket_parts()
```

Output

```
payload, propellant, structure
```

Python

```
output is None
```

Output

```
True
```

Pode parecer surpreendente que o valor da variável `output` seja `None`. Isso ocorre porque a função `rocket_parts()` não retornou um valor explicitamente. No Python, se uma função não retornar um valor explicitamente, ela retornará *implicitamente* `None`. Atualizar a função para retornar a cadeia de caracteres em vez de imprimi-la faz com que a variável `output` tenha um valor diferente:

Python

```
def rocket_parts():
    return "payload, propellant, structure"
output = rocket_parts()
output
```

Output

```
payload, propellant, structure
```

Se você precisar usar o valor de uma função, essa função *deverá* retornar esse valor explicitamente. Caso contrário, `None` será retornado.

> **Observação**
>
> Você não precisa sempre atribuir o retorno de uma função. Na maioria dos casos em que a função não retorna nenhum valor explicitamente, isso significa que você não precisa atribuir ou usar o `None` implícito retornado.

Argumentos necessários e opcionais

No Python, várias funções internas exigem argumentos. Algumas funções internas tornam os argumentos opcionais. As funções internas estão imediatamente disponíveis, portanto, você não precisa importá-las explicitamente.

Um exemplo de função interna que requer argumento é `any()`. Essa função usa um objeto iterável (por exemplo, uma lista) e retorna `True` se algum item do objeto iterável for igual a `True`. Caso contrário, ele retornará `False`.

Python

```
any([True, False, False])
```

Output

```
True
```

Python

```
any([False, False, False])
```

Output

```
False
```

Se você chamar `any()` sem nenhum argumento, uma exceção útil será gerada. A mensagem de erro explica que você precisa de pelo menos um argumento:

Python

```
any()
```

Output

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: any() takes exactly one argument (0 given)
```

Você pode verificar que algumas funções permitem o uso de argumentos opcionais usando outra função interna chamada `str()`. Essa função cria uma cadeia de caracteres por meio de um argumento. Se nenhum argumento for passado, ela retornará uma cadeia de caracteres vazia:

Python

```
str()
```

Output

```
''
```

Python

```
str(15)
```

Output

```
'15'
```

Unidade seguinte: Usar argumentos de função no Python

Continuar >