Aula 18

Revisão

Argumentos Posicionais
Argumentos Palavra-chave
Argumentos Opcionais

- Argumentos posicionais são aqueles que são passados para a função na ordem em que eles aparecem na definição da função.
- Argumentos de palavra-chave são aqueles que são passados para a função usando o nome do argumento, seguido do sinal de igual e o valor. Por exemplo, na função abaixo, "base" e "altura" são argumentos de palavra-chave :
- Argumentos opcionais são especificados na definição da função com o sinal de igual "=" seguido de um valor padrão. Isso significa que se esses argumentos não forem fornecidos quando a função for chamada, eles usarão o valor padrão especificado.

```
>>> def soma(a, b=5):
...     """Realiza a soma de dois números e mostra."""
...     print(f'A soma de a={a} e b={b} é {a + b}.')
...
>>> # argumentos posicionais
>>> soma(1, 10)
A soma de a=1 e b=10 é 11.
>>> # argumentos palavra-chave
>>> soma(b=11, a=20)
A soma de a=20 e b=11 é 31.
>>> # argumento posicionais e palavra-chave
>>> soma(12, b=23)
A soma de a=12 e b=23 é 35.
>>> # argumento opcional
>>> soma(10)
A soma de a=10 e b=5 é 15.
>>> help(soma)
Help on function soma in module __main__:

soma(a, b=5)
    Realiza a soma de dois números e mostra.

>>> soma(a=11, 15)
  File "<stdin>", line 1
    soma(a=11, 15)
               ^
SyntaxError: positional argument follows keyword argument
>>>
```

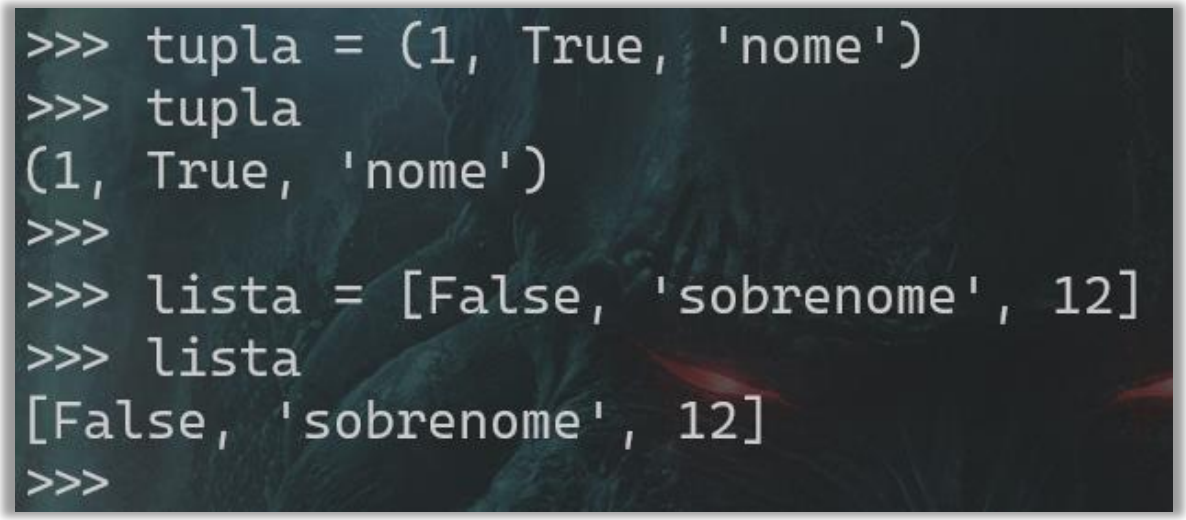
IMPORTANTE : os argumentos opcionais devem vir ao final da declaração dos argumentos na função. Se o primeiro argumento tiver um valor padrão, **todos** os subsequentes deverão ter valores padrão.
Abaixo, um exemplo de erro de sintaxe será gerado :

```
>>> def saudacao(nome=1, saudacao):
  File "<stdin>", line 1
    def saudacao(nome=1, saudacao):
                         ^^^^^^^^
SyntaxError: non-default argument follows default argument
>>>
```

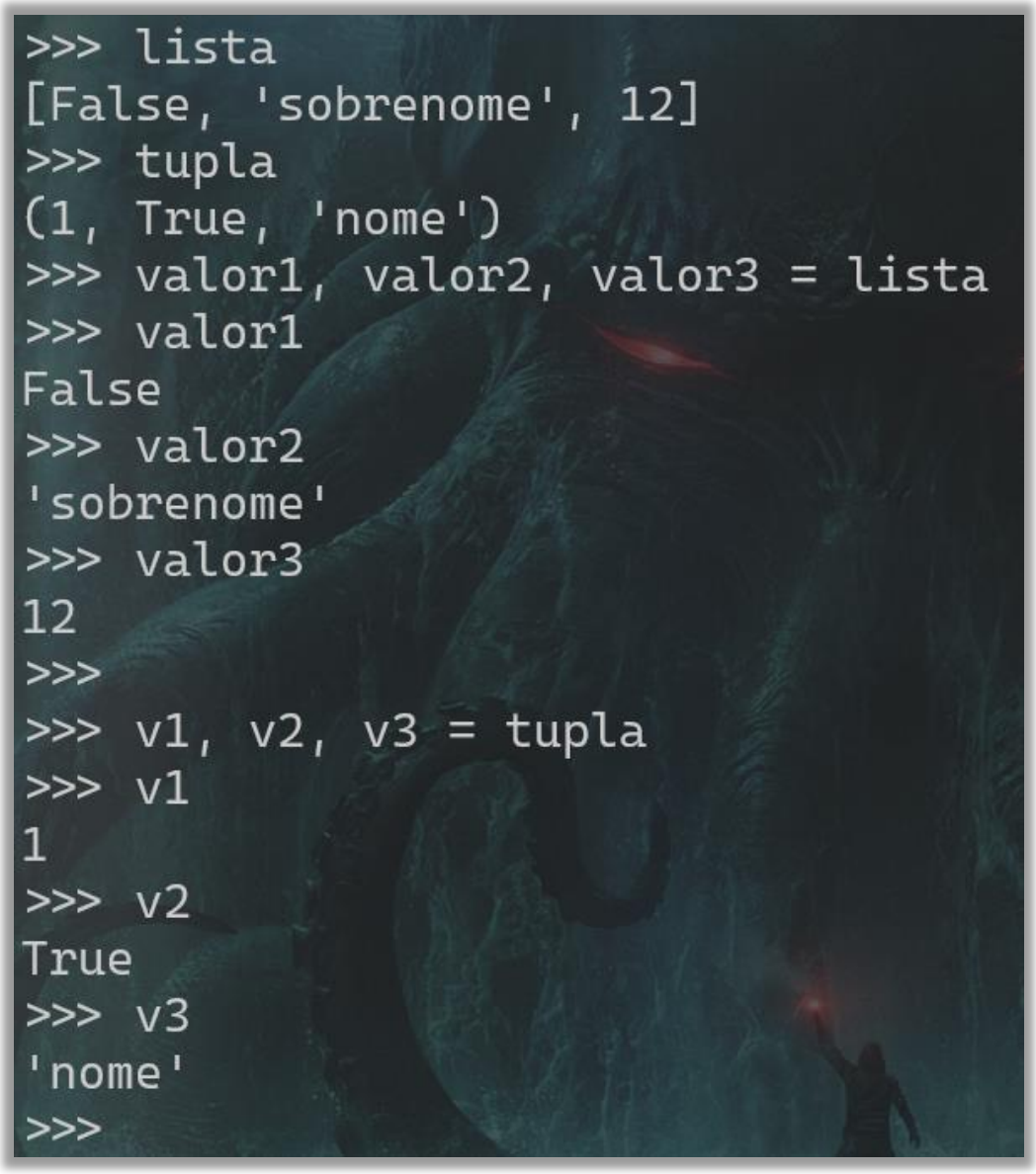
Empacotamento e Desempacotamento

O empacotamento e o desempacotamento são conceitos importantes em Python que permitem que você manipule vários valores em uma única variável.

O empacotamento se refere à criação de uma única variável que contém vários valores. Isso é feito por meio de tuplas ou listas em Python.



No código acima, tupla e lista são variáveis que contêm vários valores.
Já o desempacotamento se refere à extração de valores individuais de uma variável empacotada.



Quando realizamos esse processo de desempacotar, sempre temos que cuidar para passarmos o número de variáveis igual ao tamanho da tupla / lista. Mas tem uma forma de realizarmos isso sem gerar erros. Basta usarmos o asterisco antes do nome da variável. Ele simboliza que a variável destino vai receber parte da lista como lista.

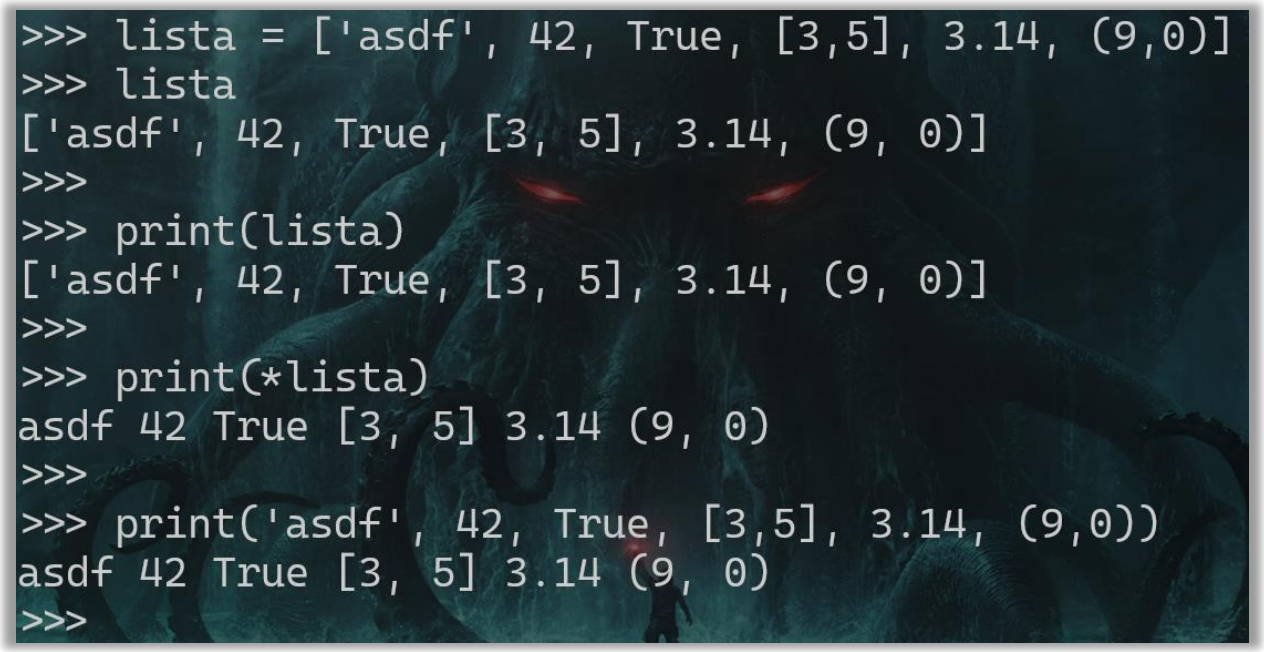
```
>>> tupla = (1, 'nome', False, 3.12, True)
>>> v1, v2 = tupla
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
>>>
>>> v1, v2, *v3 = tupla
>>> v1
1
>>> v2
'nome'
>>> v3
[False, 3.12, True]
>>>
```

```
>>> tupla = (1, 'nome', False)
>>> v1, v2, *v3 = tupla
>>> v1
1
>>> v2
'nome'
>>> v3
[False]
>>>
```

Também podemos usar o empacotamento e o desempacotamento em conjunto com funções em Python.

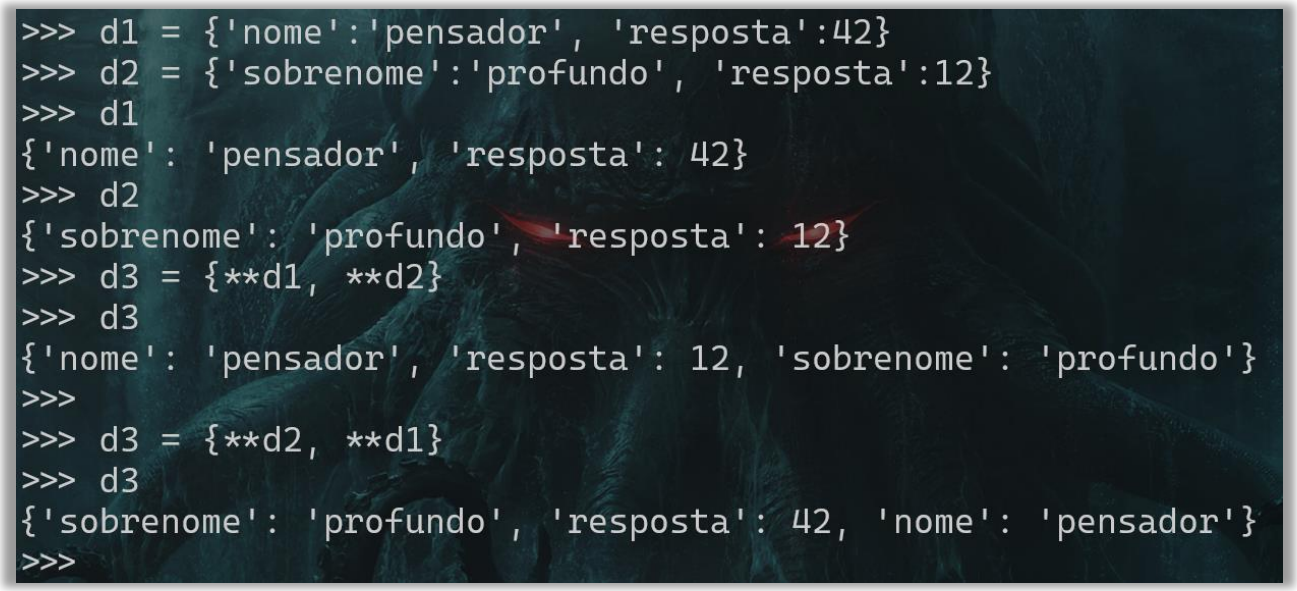
```
>>> def soma_quadrados(x, y):
...     return x**2 + y**2
...
>>> # criando a tupla com dois valores
>>> tupla = (3, 4)
>>>
>>> # chamando a função com os argumentos desempacotados
>>> resultado = soma_quadrados(*tupla)
>>>
>>> print(resultado)
25
>>>
```

O desempacotamento de listas fica mais claro de ver quando usamos o print para exibir com e sem a funcionalidade.



Como podemos ver acima, se colocarmos a lista dentro do print, ele vai exibir a lista como ela realmente é, uma lista de valores. Mas se desempacotarmos no print, teremos uma saída diferente, como se tivéssemos enviado os valores manualmente dentro do print, como visto no último print.

Assim como as listas, também podemos desempacotar os dicionários da mesma forma, mas apenas estruturas que aceitem esse tipo de valores, como outros dicionários e **kwargs nas funções.



No exemplo acima, temos dois dicionários d1 e d2. Quando juntamos eles em um novo dicionário, as chaves iguais ficam com os valores do último dicionário, conforme podemos ver acima ao trocarmos a ordem dos dicionários. As chaves diferentes são agrupadas.

```
>>> d1 = {'nome':'pensador', 'resposta':42}
>>>
>>> def teste(**kwargs):
...     print(kwargs)
...
>>> teste(**d1)
{'nome': 'pensador', 'resposta': 42}
>>>
>>> teste(d1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: teste() takes 0 positional arguments but 1 was given
>>>
```

No exemplo acima, temos o desempacotamento de um dicionário na chamada de uma função que espera receber diversos pares de chave:valor. Se simplesmente tentarmos passar um dicionário como variável, iremos levantar o TypeError. Veremos mais sobre **kwargs abaixo.

*args e **kwargs

É possível usar "empacotamento" e "desempacotamento" de variáveis com funções também. Isso pode ser feito passando uma tupla ou lista como argumento para uma função e, dentro da função, "desempacotar" esses argumentos para serem usados individualmente.

Os parâmetros *args e **kwargs são usados para lidar com um número variável de argumentos em funções. Eles permitem que você passe um número indefinido de argumentos para uma função, tornando seu código mais flexível e reutilizável.

*args

O parâmetro *args é usado para passar um número variável de argumentos posicionais para uma função. Quando um parâmetro é precedido por um asterisco (*), isso significa que ele receberá todos os argumentos posicionais restantes como uma tupla.

```
>>> def mostra(*args):
...     for arg in args:
...         print(arg)
...
>>> mostra(1, 'nome', True, "olá", 3.14)
1
nome
True
olá
3.14
>>>
```

A função acima vai receber uma quantidade indeterminada de valores e então mostrar todos eles.

Isso lembra uma certa função que já vimos. Esse comportamento não é parecido com o comportamento da função "print()"?
Execute "help(print)" no terminal e veja o que aparece como primeiro parâmetro.

Agora a mesma função de antes, mas exibindo os valores e tipos de cada item do *args.

```
>>> def mostra(*args):
...     print(f'O valor em *args é {args}.')
...     print(f'O tipo é {type(args)}.')
...     for arg in args:
...             print(f'O valor em arg é {arg}.')
...             print(f'O tipo do arg é {type(arg)}.\n')
...
>>> mostra(1, 'nome', True, "olá", 3.14)
O valor em *args é (1, 'nome', True, 'olá', 3.14).
O tipo é <class 'tuple'>.
O valor em arg é 1.
O tipo do arg é <class 'int'>.

O valor em arg é nome.
O tipo do arg é <class 'str'>.

O valor em arg é True.
O tipo do arg é <class 'bool'>.

O valor em arg é olá.
O tipo do arg é <class 'str'>.

O valor em arg é 3.14.
O tipo do arg é <class 'float'>.

>>>
```

A função abaixo vai receber diversos números, somar eles e retornar o total. Todos os valores passados para a função são empacotados para dentro da variável números, que depois vai passar pelo for e ter cada um de sus valores somados.

```
>>> def somador(*numeros):
...     """soma vários números"""
...     print(f'Lista : {numeros}')
...     soma = 0
...     for numero in numeros:
...             soma += numero
...     return soma
...
>>> print(somador(1,2,3.14,5,42,int('100')))
Lista : (1, 2, 3.14, 5, 42, 100)
153.14
>>>
```

**kwargs

O parâmetro **kwargs é usado para passar um número variável de argumentos nomeados para uma função. Quando um parâmetro é precedido por dois asteriscos (**), isso significa que ele receberá todos os argumentos nomeados restantes como um dicionário.

Por exemplo:

```
>>> def contato(**kwargs):
...     """vai receber argumentos nomeados"""
...     for chave, valor in kwargs.items():
...             print(f'{chave} : {valor}')
...
>>> contato(nome='Fulano', telefone="55555", idade=20)
nome : Fulano
telefone : 55555
idade : 20
>>>
```

Se tentarmos passar valores posicionais para a função acima, levantaremos um TypeError.

```
>>> def contato(**kwargs):
...     """vai receber argumentos nomeados"""
...     for chave, valor in kwargs.items():
...             print(f'{chave} : {valor}')
...
>>> contato(1,2,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: contato() takes 0 positional arguments but 3 were given
>>>
```

Combinando *args e **kwargs

Podemos combinar *args e **kwargs em uma única função, permitindo que lidemos com um número variável de argumentos posicionais e nomeados. Por exemplo:

```
>>> def combinado(*args, **kwargs):
...     """Função combinados."""
...     print(f'Valor args = {args}.')
...     print(f'Valor kwargs = {kwargs}.\n')
...     for arg in args:
...             print(f'Valor arg {arg}.')
...     for c, v in kwargs.items():
...             print(f'{c} : {v}.')
...
>>> combinado(1,2,3,42,nome='fulano',idade=42)
Valor args = (1, 2, 3, 42).
Valor kwargs = {'nome': 'fulano', 'idade': 42}.

Valor arg 1.
Valor arg 2.
Valor arg 3.
Valor arg 42.
nome : fulano.
idade : 42.
>>>
```

Nesse exemplo, a função "combinado" recebe tanto argumentos posicionais quanto nomeados. Os argumentos posicionais são armazenados em uma tupla chamada args, enquanto os argumentos nomeados são armazenados em um dicionário chamado kwargs. A função itera sobre a tupla e o dicionário e imprime cada valor.

Combinando Tudo

Agora, um exemplo da combinação do que vimos até agora sobre os parâmetros e argumentos das funções.

```
>>> def tudo(a, b=42, *args, **kwargs):
...     """Função demonstrando o uso de todos os parâmetros possíveis"""
...     print(f"a={a}")
...     print(f"b={b}")
...     print(f"args={args}")
...     print(f"kwargs={kwargs}")
...
>>>
```

```
>>> tudo(1)
a=1
b=42
args=()
kwargs={}
>>>
```

```
>>> tudo(1, 2)
a=1
b=2
args=()
kwargs={}
>>>
```

```
>>> tudo(1, 2, 3, 4, 5)
a=1
b=2
args=(3, 4, 5)
kwargs={}
>>>
```

```
>>> tudo(1, 2, 3, 4, 5, nome="Fulano", idade=42)
a=1
b=2
args=(3, 4, 5)
kwargs={'nome': 'Fulano', 'idade': 42}
>>>
```

No exemplo acima, temos todos os conceitos aplicados em apenas uma função.

Exercícios para Praticar

1. Crie uma lista de números e desempacote-os em variáveis separadas.
2. Empacote duas variáveis em uma tupla.
3. Desempacote uma tupla em duas variáveis separadas.
4. Empacote três variáveis em uma lista.
5. Desempacote uma lista em três variáveis separadas.
6. Crie uma lista de strings e use o operador * para desempacotá-la como argumentos de uma função print().
7. Empacote três strings em uma lista e use o operador * para desempacotá-la como argumentos de uma função print().
8. Crie uma tupla com dois números e desempacote-a como argumentos de uma função que retorna a soma dos dois números.
9. Empacote três variáveis em uma tupla e passe-a como argumento de uma função que retorna a média dos três números.
10. Desempacote uma tupla de tamanho variável em uma função e retorne a soma dos números.
11. Desempacote um dicionário em variáveis separadas.
12. Empacote duas variáveis em um dicionário com chaves personalizadas.
13. Desempacote um dicionário com chaves personalizadas em variáveis separadas.
14. Crie uma lista de tuplas e use o operador * para desempacotá-la como argumentos de uma função que retorna a soma de cada tupla.
15. Empacote duas variáveis em uma lista e adicione uma terceira variável a ela usando o método append().
16. Desempacote uma lista em duas variáveis separadas e remova o primeiro elemento da lista usando o método pop().
17. Empacote três variáveis em um dicionário e adicione uma quarta variável a ele com uma chave personalizada.
18. Desempacote um dicionário em três variáveis separadas e remova um elemento do dicionário usando o método pop().
19. Crie uma lista de tuplas e use o operador * para desempacotá-la como argumentos de uma função que retorna o menor valor de cada tupla.
20. Crie uma tupla de números e use o operador * para desempacotá-la como argumentos de uma função que retorna a média dos números.
21. Escreva uma função que receba dois números e retorne uma tupla com os dois números.
22. Escreva uma função que receba uma tupla com dois números e retorne a soma dos números.
23. Escreva uma função que receba uma lista de números e retorne a soma de todos os elementos.
24. Escreva uma função que receba uma lista de strings e retorne uma única string com todos os elementos concatenados.
25. Escreva uma função que receba uma lista de tuplas e retorne uma lista com as somas de cada tupla.
26. Escreva uma função que receba um dicionário e retorne uma lista com as chaves e valores em tuplas.
27. Escreva uma função que receba um dicionário e retorne uma lista com as chaves em uma lista e os valores em outra lista.
28. Escreva uma função que receba uma lista de números e um número k, e retorne uma lista com os k primeiros elementos da lista.
29. Escreva uma função que receba uma lista de números e um número k, e retorne uma lista com os k últimos elementos da lista.
30. Escreva uma função que receba uma lista de números e uma lista de booleanos, e retorne uma lista com apenas os números nas posições onde o valor booleano é True.
31. Escreva uma função que receba uma lista de strings e um número n, e retorne uma lista com as strings que possuem tamanho maior ou igual a n.
32. Escreva uma função que receba uma lista de tuplas e um número n, e retorne uma lista com as tuplas que possuem pelo menos um elemento com valor maior que n.
33. Escreva uma função que receba uma lista de dicionários e um valor k, e retorne uma lista com os dicionários que possuem a chave k.
34. Escreva uma função que receba uma lista de dicionários e um valor k, e retorne uma lista com os valores das chaves k nos dicionários.

35. Escreva uma função que receba um número arbitrário de argumentos e retorne uma tupla com todos os argumentos.
36. Escreva uma função que receba um número arbitrário de argumentos e retorne a soma de todos os números.
37. Escreva uma função que receba um número arbitrário de argumentos e retorne uma lista com os argumentos ordenados.
38. Escreva uma função que receba um número arbitrário de argumentos e retorne uma lista com os argumentos que são strings.
39. Escreva uma função que receba um número arbitrário de argumentos e um argumento k, e retorne uma lista com os argumentos que possuem tamanho maior ou igual a k.
40. Escreva uma função que receba um número arbitrário de argumentos e um dicionário com chaves personalizadas, e retorne um dicionário com as chaves personalizadas e os respectivos valores.
41. Escreva uma função que receba um número arbitrário de argumentos e retorne a soma de todos os números, utilizando *args.
42. Escreva uma função que receba um número arbitrário de argumentos e retorne uma lista com os argumentos ordenados, utilizando *args.
43. Escreva uma função que receba um número arbitrário de argumentos e retorne uma lista com os argumentos que são strings, utilizando *args.
44. Escreva uma função que receba um número arbitrário de argumentos e um argumento k, e retorne uma lista com os argumentos que possuem tamanho maior ou igual a k, utilizando *args.
45. Escreva uma função que receba um dicionário com informações de uma pessoa (nome, idade, endereço, etc.) e exiba essas informações de forma formatada, utilizando **kwargs.
46. Escreva uma função que receba um número arbitrário de argumentos e um dicionário com informações de uma pessoa, e retorne uma lista com todos os argumentos e informações da pessoa, utilizando *args e **kwargs.
47. Escreva uma função que receba um número arbitrário de argumentos e um dicionário com informações de uma pessoa, e atualize as informações da pessoa com os argumentos, utilizando *args e **kwargs.
48. Escreva uma função que receba um número arbitrário de argumentos e um dicionário com valores padrão, e retorne um dicionário com os valores atualizados pelos argumentos, utilizando **kwargs.
49. Escreva uma função que receba um número arbitrário de argumentos e um dicionário com valores padrão, e retorne uma lista com todos os argumentos e valores atualizados pelos valores padrão, utilizando *args e **kwargs.
50. Escreva uma função que receba um número arbitrário de listas e retorne uma lista com todos os elementos das listas concatenados, utilizando *args.
51. Escreva uma função que receba uma lista e um número arbitrário de índices, e retorne uma tupla com os elementos da lista nos índices especificados, utilizando *args.
52. Escreva uma função que receba uma lista de dicionários e um número arbitrário de chaves, e retorne uma lista com os valores das chaves especificadas em cada dicionário, utilizando *args.
53. Escreva uma função que receba uma lista de dicionários e um número arbitrário de chaves, e retorne uma lista com os dicionários que possuem todas as chaves especificadas, utilizando *args e **kwargs.
54. Escreva uma função que receba uma lista de dicionários e um número arbitrário de chaves, e retorne uma lista com os valores das chaves especificadas nos dicionários que possuem todas as chaves especificadas, utilizando *args e **kwargs.
55. Escreva uma função que receba um número arbitrário de tuplas, cada uma contendo um nome e um valor, e retorne um dicionário com os nomes como chaves e os valores como valores, utilizando *args.
56. Escreva uma função que receba um número arbitrário de dicionários e retorne um novo dicionário com a união de todos os dicionários, utilizando **kwargs.
57. Escreva uma função que receba um número arbitrário de dicionários e um número arbitrário de chaves, e retorne um novo dicionário com as chaves e valores especificados, utilizando **kwargs.
58. Escreva uma função que receba um número arbitrário de dicionários e um número arbitrário de chaves, e retorne uma lista com os valores das chaves especificadas em cada dicionário, utilizando *args e **kwargs.

59. Escreva uma função que receba uma lista de tuplas, cada uma contendo um nome e um valor, e retorne um dicionário com os nomes como chaves e os valores como valores, utilizando **kwargs.
60. Escreva uma função que receba um número arbitrário de dicionários e retorne uma lista com as chaves que existem em todos os dicionários, utilizando *args e **kwargs.
61. Escreva uma função que receba dois argumentos posicionais obrigatórios e retorne a soma entre eles.
62. Escreva uma função que receba um argumento posicional obrigatório e um argumento posicional com valor padrão igual a 10, e retorne o resultado da multiplicação entre eles.
63. Escreva uma função que receba um número arbitrário de argumentos posicionais e retorne a soma de todos eles.
64. Escreva uma função que receba um número arbitrário de argumentos posicionais e um argumento posicional com valor padrão igual a 1, e retorne o resultado da multiplicação entre todos eles.
65. Escreva uma função que receba um número arbitrário de argumentos posicionais e um argumento posicional com valor padrão igual a "soma", "multiplicação" ou "concatenação", e retorne a operação escolhida aplicada aos argumentos.
66. Escreva uma função que receba um dicionário e um número arbitrário de chaves, e retorne um novo dicionário contendo apenas as chaves especificadas como argumentos posicionais.
67. Escreva uma função que receba um número arbitrário de argumentos posicionais e um número arbitrário de pares chave:valor como **kwargs, e retorne um dicionário contendo todos eles.
68. Escreva uma função que receba uma lista de números e um número arbitrário de argumentos posicionais, e retorne uma lista contendo todos os números da lista original multiplicados pelos argumentos posicionais.
69. Escreva uma função que receba uma string e um número arbitrário de argumentos posicionais, e retorne a string com os argumentos posicionais substituindo as ocorrências de "{}" na ordem em que aparecem.
70. Escreva uma função que receba uma lista de strings e um número arbitrário de argumentos posicionais, e retorne uma lista com todas as strings da lista original concatenadas com os argumentos posicionais.
71. Escreva uma função que receba um número arbitrário de dicionários e um número arbitrário de chaves, e retorne uma lista com os valores das chaves especificadas em cada dicionário.
72. Escreva uma função que receba um número arbitrário de dicionários e retorne um novo dicionário contendo todas as chaves e valores dos dicionários originais.
73. Escreva uma função que receba um número arbitrário de argumentos posicionais e retorne uma lista contendo apenas os números pares.
74. Escreva uma função que receba uma lista de números e um número arbitrário de argumentos posicionais, e retorne uma lista contendo apenas os números da lista original que são divisíveis por pelo menos um dos argumentos posicionais.
75. Escreva uma função que receba uma lista de strings e um número arbitrário de argumentos posicionais, e retorne uma lista contendo apenas as strings da lista original que contêm pelo menos uma das palavras passadas como argumentos posicionais.
76. Escreva uma função que receba um número arbitrário de argumentos posicionais e retorne um dicionário contendo o número de ocorrências de cada valor único.
77. Escreva uma função que receba um número arbitrário de listas e retorne uma lista contendo os elementos que aparecem em todas as listas.
78. Escreva uma função que receba um número arbitrário de argumentos posicionais e um número arbitrário de pares chave:valor como kwargs, e retorne uma lista contendo os argumentos posicionais e os valores dos kwargs que têm chaves iguais.
79. Escreva uma função que receba um número arbitrário de dicionários e um número arbitrário de chaves, e retorne uma lista de tuplas contendo os valores das chaves especificadas em cada dicionário.
80. Escreva uma função que receba uma lista de strings e um número arbitrário de argumentos posicionais, e retorne uma lista de tuplas contendo cada string da lista original e o número de ocorrências de cada palavra passada como argumento posicional.