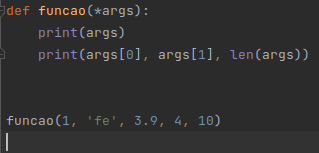
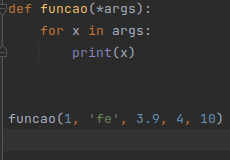
**56 – Funções parte 2**

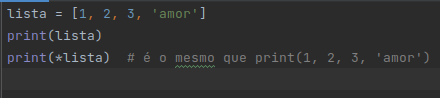
**\*args**

Em python, podemos usar um número indefinido de argumentos à uma função, para isso devemos utilizar uma tupla especial como parâmetro, que na lista de argumentos deve ser precedida de um asterisco \*, por conveção a nomeamos de args: \***args**:

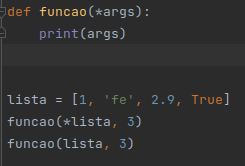
 

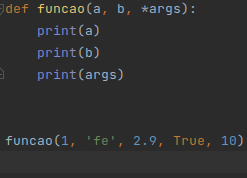
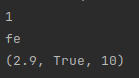
Também podemos desempacotar objetos iteráveis nos parâmetros de uma função, para isso basta acrescentar o asterisco \* antes desse objeto:

Com isso:

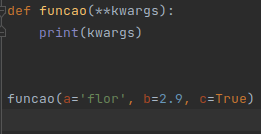
 

A tupla \*args não precisa ser o único parâmetro da função, mas ela deve estar no final da lista de parâmetros da função:

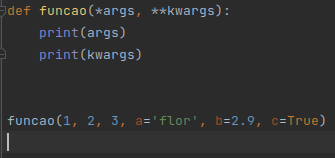
 

**\*\*kwargs**

Ao utilizar o argumento precedido de dois asteriscos \*\*, cujo o nome é convencionado como kwargs (key words args) **\*\*kwargs**, estamos criando um dicionário (<https://pythonacademy.com.br/blog/dicts-ou-dicionarios-no-python> <https://docs.python.org/3/library/stdtypes.html?highlight=dict#dict> ), cujas chaves são os nomes dos argumentos informados ao chamar a função, e os valores são as respectivas atribuições ao chamar a função. (Essas atribuições são feitas com a mesma sintaxe da chamada de função passando os argumentos de forma nomeada)

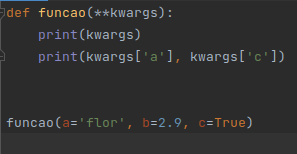
 

No caso se existir o parâmetro args, o kwargs deve vir depois dele na lista de parâmetros.

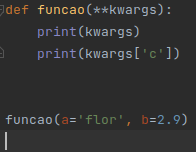
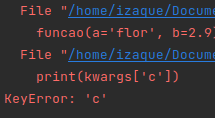
 

Podemos acessar os valores das chaves da seguinte forma:

kwargs[<nome chave>]

Mas dessa forma, caso buscarmos por o valor de uma chave que não foi passada como argumento, teremos um erro:

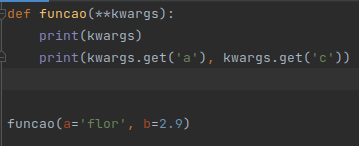
 

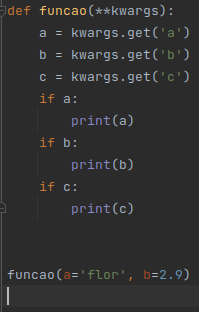
Para contornar isso podemos fazer o seguinte método .get:

<https://docs.python.org/3/library/stdtypes.html?highlight=dic%20get#mapping-types-dict>

**get**(*key*[, *default*])

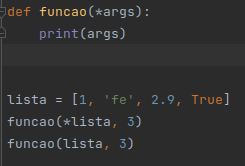
Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to None, so that this method never raises a [KeyError](https://docs.python.org/3/library/exceptions.html#KeyError).

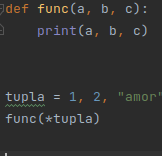
 

**Desempacotamento de (listas ou tuplas) e de dicionários em parâmetro de funções:**

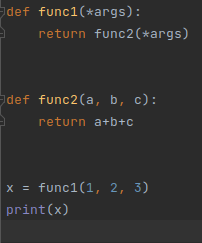
Já falamos, sobre desempacotamento de Listas como argumento de função:

É análogo para tuplas:

Com isso, podemos desempacotar a tupla **args** de uma função, para colocar em outra função:

Veja que a função func1 é chamada e é passado os argumentos 1, 2, 3 para a tupla args:

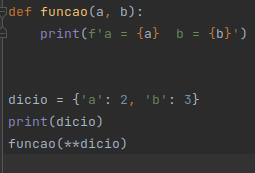
args = (1, 2, 3*)*

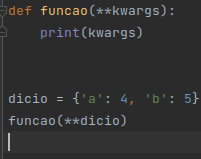
E dentro da função func1 é chamada a função func2, sendo que os argumentos da func2 é o desempacotamento da tupla args, ou seja, o mesmo que:

*return func2(1, 2, 3)*

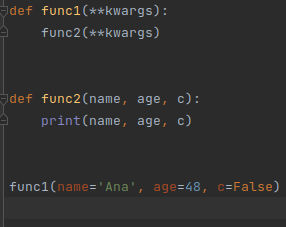
Com isso na função, func2 temos que os valores dos parâmetros são: a=1, b=2, c=3.

Já vimos brevemente sobre dicionários, podemos também desempacotá-los em funções, de tal forma que os parâmetros que possuem o mesmo nome que as chaves recebem os respectivos valores delas. Para desempacotar um dicionário usamos dois asteriscos \*\*.

Com isso, podemos desempacotar o dicionário **kwargs** de uma função para colocar em outra função:

Passamos os argumentor de forma nomeada na func1:

kwargs = {name: ‘Ana’, age:48, c:False}

E então desempacotamos o dicionário kwargs na func2, obtendo:

name=’Ana’, age=48, c=False.

**Escopo de variáveis**

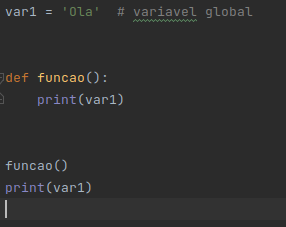
Quando criamos uma variável fora de uma função, ela possui o escopo global, quando criamos uma variável dentro de uma função ela possui o escopo local daquela função.

Isto significa que:

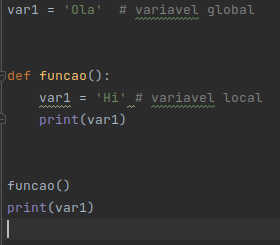
No caso de uma variável global podemos acessar o valor dela em todo nosso código, inclusive dentro de funções, porém não podemos alterar o seu valor dentro de uma função.

Em funções, a partir do momento que ela é executada todos os parâmetros dela e as variáveis que são criadas nela são criadas dentro dessa função (escopo local), podemos acessar e alterar o valor dessas variáveis dentro dessa função, quando essa função termina todas essas variáveis locais são apagadas. Logo, não podemos ter o acesso a essas variaveis locais fora dessa função.

Vamos ver um pouco na prática:

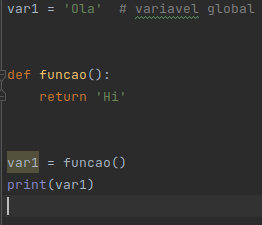
 

Criamos uma variavel global, e podemos acessá-la tanto dentro quanto fora de outras funções.

Nesse caso criamos uma variável global com nome var1, e dentro da função funcao() criamos uma variávei local também chamada var1, apesar de não ser uma boa prática criar variáveis de mesmo nome em escopos (global, local) é possível. Então foi isso que fazemos, perceba que quando adicionamos o valor ‘Hi’ adicionamos à variavel local var1, e não modificamos a variável global var1.

Mas claro que podemos modificar o valor de uma variável global se ela receber o valor retornado pela função:

Há também uma observação:

Quando criamos uma variável local de mesmo nome de uma global, não podemos, dentro da função, acessar o valor da variável, sem antes ter a criado dentro dessa função. Ou seja não podemos acessar o valor da variavel global. Isso porque o interpretador irá entender que aquele nome se refere apenas a uma variável local, e não da pra você acessar o valor de uma variável antes de criá-la.

