INTERMEDIATE PYTHON

LESSON 3 |*args & **kwargs| 17-12-18

***Video tutorial*:**

https://www.youtube.com/watch?v=x-dH73VMF_w&index=2&list=PLqEbL1vopgvsv6jz8xB3Aa6w1njhf3egY

Python *args and **kwargs

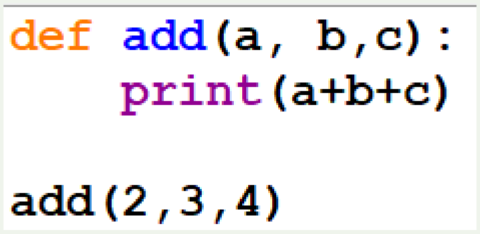
Python has a special syntax, * (single asterisk) and ** (double asterisks), that lets you pass a variable number of arguments to a function. By convention, these are written as *args and **kwargs, but only the asterisks are important; you could equally write *vars and **vars to achieve the same result.

*args is used to pass a non-keyworded variable-length argument list to your function. **kwargs lets you pass a keyworded variable-length of arguments to your function.

Using Args and Kwargs in Functions

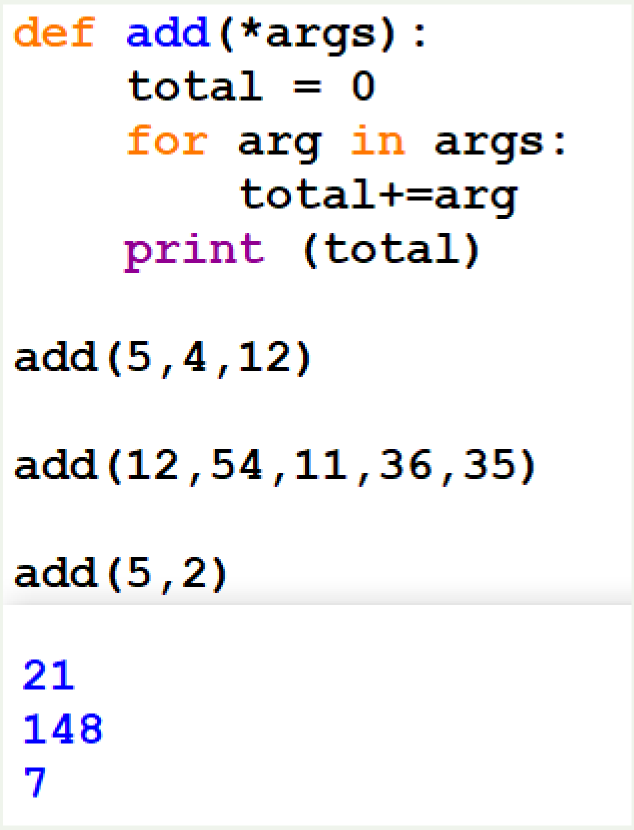
Args

The function below takes in three arguments. The three arguments have been explicitly defined, so any more or less will cause an error in the program.



```
def add(a, b,c):
    print(a+b+c)

add(2,3,4)
```

Notice how the use of *args makes it easy to use any number of arguments without having to change your code. *args provide more flexibility to your code since you can have as many arguments as you wish in the future.



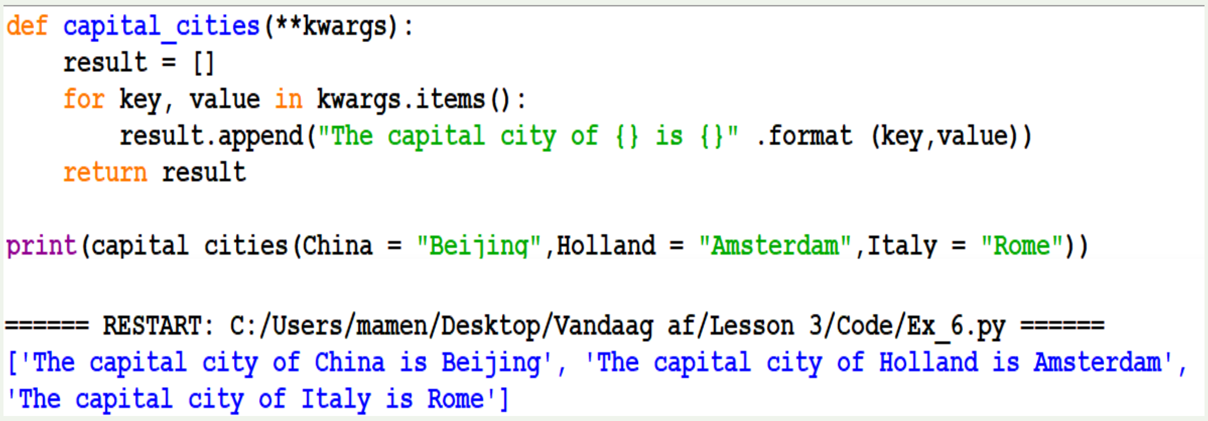
```python
def add(*args):
    total = 0
    for arg in args:
        total+=arg
    print (total)

add(5,4,12)

add(12,54,11,36,35)

add(5,2)
```

```
21
148
7
```

Kwargs

Kwargs allow you to pass keyword arguments to a function. They are used when you are not sure of the number of keyword arguments that will be passed in the function.

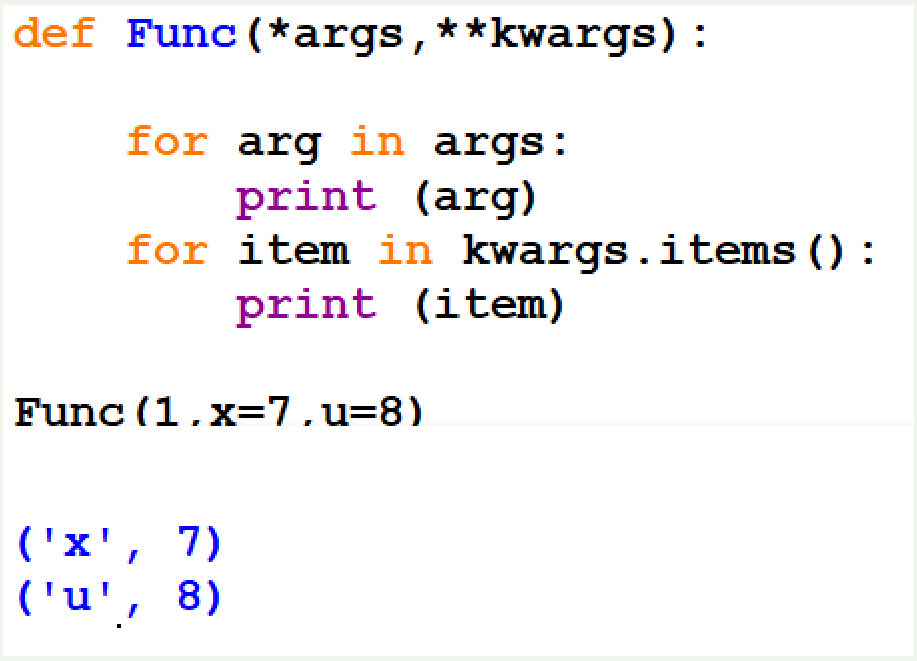
Here's a typical example of how it's done. The function below takes countries as keys and their capital cities as the values. It then prints out a statement which iterates over the kwargs and maps each keyword to the value assigned to it.



```
def capital_cities(**kwargs):
    result = []
    for key, value in kwargs.items():
        result.append("The capital city of {} is {}" .format (key,value))
    return result

print(capital_cities(China = "Beijing",Holland = "Amsterdam",Italy = "Rome"))

====== RESTART: C:/Users/mamen/Desktop/Vandaag af/Lesson 3/Code/Ex_6.py ======
['The capital city of China is Beijing', 'The capital city of Holland is Amsterdam',
'The capital city of Italy is Rome']
```

Using Both Args and Kwargs in a Function

When using both args and kwargs in the same function definition, *args must occur before **kwargs



```
def Func(*args,**kwargs):

    for arg in args:
        print (arg)
    for item in kwargs.items():
        print (item)

Func(1,x=7,u=8)
```

```
('x', 7)
('u', 8)
```

Conclusion

Below are some pointers to remember when using args and kwargs:

*args and **kwargs are special syntax that are used in functions to pass a variable number of arguments to a function.

*args occur before **kwargs in a function definition.

*args and **kwargs are best used in situations where the number of inputs will remain relatively small.

•You can use any name you want; args and kwargs are only by convention and not a requirement. For example, you can use *foo nstead of *args or **foo instead of **kwargs .

Test Your Knowledge: Quiz

1. What is the output of the following code, and why?

```
>>> def func(a, b=4, c=5):
print(a, b, c)
>>> func(1, 2)
```

2. What is the output of this code, and why?

```
>>> def func(a, b, c=5):
print(a, b, c)
>>> func(1, c=3, b=2)
```

3. How about this code: what is its output, and why?

```
>>> def func(a, *pargs):
print(a, pargs)
>>> func(1, 2, 3)
```

4. What does this code print, and why?

```
>>> def func(a, **kargs):
print(a, kargs)
>>> func(a=1, c=3, b=2)
```

5. What gets printed by this, and why?

```
>>> def func(a, b, c=3, d=4): print(a, b, c, d)
>>> func(1, *(5, 6))
```

6. One last time: what is the output of this code, and why?

```
>>> def func(a, b, c): a = 2; b[0] = 'x'; c['a'] = 'y'
>>> l=1; m=[1]; n={'a':0}
>>> func(l, m, n)
>>> l, m, n
```

Answers

1. The output here is 1 2 5, because 1 and 2 are passed to a and b by position, and

c is omitted in the call and defaults to 5.

2. The output this time is 1 2 3: 1 is passed to a by position, and b and c are passed

2 and 3 by name (the left-to-right order doesn't matter when keyword arguments

are used like this).

3. This code prints 1 (2, 3), because 1 is passed to a and the *pargs collects the

remaining positional arguments into a new tuple object. We can step through the

extra positional arguments tuple with any iteration tool (e.g., for arg in

pargs: ...).

4. This time the code prints 1 {'b': 2, 'c': 3}, because 1 is passed to a by name

and the **kargs collects the remaining keyword arguments into a dictionary. We

could step through the extra keyword arguments dictionary by key with any iteration

tool (e.g., for key in kargs: ...). Note that the order of the dictionary's

keys may vary per Python and other variables.

5. The output here is 1 5 6 4: the 1 matches a by position, 5 and 6 match b and c by

*name positionals (6 overrides c's default), and d defaults to 4 because it was not

passed a value.

6. This displays (1, ['x'], {'a': 'y'})—the first assignment in the function doesn't

impact the caller, but the second two do because they change passed-in mutable

objects in place.