



The Matter of Arguments





Table of Contents

Arguments vs Parameters

Correct Use of Arguments

Arbitrary Number of Arguments



1

Arguments vs Parameters

Arguments vs Parameters

- ▶ In fact, **arguments** are some kind of variable. So you can think of them as aliases for variables. This is exactly what we call the **parameters**. That is, the values you assign to the parameters defined in a function are arguments.
- ▶ If you look at the Python documentation on this topic, you will notice that the terms parameters and arguments are used almost the same way. But they are slightly different things.

Arguments vs Parameters (review)

- Let's take a look at this pre-class example :

```
1 def who(first, last) : # 'first' and 'last' are the parameters(or variables)
  )
2   print('Your first name is :', first)
3   print('Your last name is :', last)
4
5 who('Guido', 'van Rossum') # 'Guido' and 'van Rossum' are the arguments
6 print()
7 who('Marry', 'Bold') # 'Marry' and 'Bold' are also the arguments
```

Arguments vs Parameters (review)

- You will better understand this topic through an example

```
1 def who(first, last) : # 'first' and 'last' are the parameters(or variables)
  2     print('Your first name is :', first)
  3     print('Your last name is :', last)
  4
  5 who('Guido', 'van Rossum') # 'Guido' and 'van Rossum' are the arguments
  6 print()
  7 who('Marry', 'Bold') # 'Marry' and 'Bold' are also the arguments
```

first and last are the parameters

```
1 Your first name is : Guido
2 Your last name is : van Rossum
3
4 Your first name is : Marry
5 Your last name is : Bold
```

the names passed into the function are the arguments

Arguments vs Parameters (review)

- If you passed only one argument into the parameter of the `who()` function, it gives an error. Consider the following example :

```
1 who('Joseph') # we passed only one argument into the function
```

Arguments vs Parameters (review)

- If you passed only one argument into the parameter of the `who()` function, it gives an error. Consider the following example :

```
1 who('Joseph') # we passed only one argument into the function
```

```
1 Traceback (most recent call last):
2   File "code.py", line 5, in <module>
3     who('Joseph')
4 TypeError: who() missing 1 required positional argument: 'last'
```



Correct Use of Arguments

2

- ▶ Positional Arguments
- ▶ Keyword Arguments

Positional Arguments (review)

- When calling a function with **positional arguments**, they **must** be passed **in order from left to right**.

```
1 def pos_args(a, b):  
2     print(a, 'is the first argument')  
3     print(b, 'is the second argument')  
4  
5 pos_args(3,4)  
6 print()  
7 pos_args(4,3)
```

What is the output? Try to figure out in your mind...

Positional Arguments (review)

- When calling a function with **positional arguments**, they must be passed **in order from left to right**.

```
1 def pos_args(a, b):  
2     print(a, 'is the first argument')  
3     print(b, 'is the second argument')  
4  
5 pos_args(3,4)  
6 print()  
7 pos_args(4,3)
```

```
1 3 is the first argument  
2 4 is the second argument  
3  
4 4 is the first argument  
5 3 is the second argument  
6
```

Positional Arguments (review)

- When calling a function with **positional arguments**, they must be passed **in order from left to right**.

```
1 pos_args('first','second')
2 print()
3 pos_args('second', 'first')
```

Positional Arguments (review)

- When calling a function with **positional arguments**, they must be passed **in order from left to right**.

```
1 pos_args('first','second')
2 print()
3 pos_args('second', 'first')
```

```
1 first is the first argument
2 second is the second argument
3
4 second is the first argument
5 first is the second argument
6
```

Positional Arguments



▶ Task :

- ▶ Define a function named **texter** to print the following output in accordance with input .

```
a = "i"  
b = "love"  
c = "you"
```

```
texter(c, a, b)
```

Output

```
i love you
```

Positional Arguments



- ▶ The function definition can be like :

```
1 def texter(text1, text2, text3):  
2     print(f"{text2} {text3} {text1}")  
3  
4  
5
```

Keyword Arguments (review)

- ▶ If you do not want to allow the sequences/positions of the arguments to restrict you when you call a function, you can also call these arguments by keywords.
- ▶ Commonly and traditionally, **kwargs** is used as an abbreviation of **keyword arguments**.

The formula syntax is : **kwargs=values**.

Keyword Arguments (review)

- ▶ Consider the following example :

```
1 def who(first, last) : # same structure as the previous one
2     print('Your first name is :', first)
3     print('Your last name is :', last)
4
5 who(first='Guido', last='van Rossum') # calling the function is different
6 # we used kwargs to pass the values into the function
```

Keyword Arguments (review)

- ▶ Consider the following example :

```
1 def who(first, last) : # same structure as the previous one
2     print('Your first name is :', first)
3     print('Your last name is :', last)
4
5 who(first='Guido', last='van Rossum') # calling the function is different
6 # we used kwargs to pass the values into the function
```

```
1 Your first name is : Guido
2 Your last name is : van Rossum
3
```

Keyword Arguments

▶ Task :

- ▶ Call the `texter` function using keyword arguments in order to get the same output as the previous one.

```
1 def texter(text1, text2, text3):  
2     print(f"{text2} {text3} {text1}")  
3  
4  
5
```

Keyword Arguments

- ▶ The calling syntax can be like :

```
1 def texter(text1, text2, text3):  
2     print(f"{text2} {text3} {text1}")  
3  
4 texter(text1="you", text2="i", text3="love")  
5 texter(text2="i", text3="love", text1="you")  
6 |
```

kwargs saves us from positional restrictions.

Output

```
i love you  
i love you
```

Keyword Arguments (review)

- Let's call the **parrot** function :

```
1 def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):  
2     print("-- This parrot wouldn't", action, end=' ')  
3     print("if you put", voltage, "volts through it.")  
4     print("-- Lovely plumage, the", type)  
5     print("-- It's", state, "!")  
6  
7 parrot('a thousand', state='pushing up the daisies') # 1 positional, 1 keyword  
8
```

Discuss in class! Try to figure out the output in your mind...

Keyword Arguments (review)

- Let's call the `parrot` function in several ways below one by one:

```
1 def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):  
2     print("-- This parrot wouldn't", action, end=' ')  
3     print("if you put", voltage, "volts through it.")  
4     print("-- Lovely plumage, the", type)  
5     print("-- It's", state, "!")  
6  
7 parrot('a thousand', state='pushing up the daisies') # 1 positional, 1 keyword  
8
```

Output

```
-- This parrot wouldn't voom if you put a thousand volts through it.  
-- Lovely plumage, the Norwegian Blue  
-- It's pushing up the daisies !
```

Keyword Arguments (review)

- Considering the defined functions, all the following calls would be invalid:

```
1 parrot()                      # required argument missing
2 parrot(voltage=5.0, 'dead')    # non-keyword argument after a keyword argument
3 parrot(110, voltage=220)       # duplicate value for the same argument
4 parrot(actor='John Cleese')    # unknown keyword argument
5
```

- In a function call, **keyword arguments must** follow **positional arguments**.
- All the keyword arguments passed **must** match one of the arguments accepted by the function (e.g. **actor** is not a valid argument for the **parrot** function), and their **order is not important**.



3

Arbitrary Number of Arguments

Default Arguments

- When calling a function defined by parameters with default values, there is no obligation to pass any arguments into the function.
- Let's see how it works in an example :

```
1 def city(capital, continent='Europe'):  
2     print(capital, 'in', continent)  
3  
4 city('Athens') # we don't have to pass any arguments into 'continent'  
5 city('Ulaanbaatar', continent='Asia') # we can change the default value by kwargs  
6 city('Cape Town', 'Africa') # we can change the default value by positional args.
```

What is the output? Try to figure out in your mind...

Default Arguments

- When calling a function defined by parameters with default values, there is no obligation to pass any arguments into the function.
- Let's see how it works in an example :

```
1 def city(capital, continent='Europe'):  
2     print(capital, 'in', continent)  
3  
4 city('Athens') # we don't have to pass any arguments into 'continent'  
5 city('Ulaanbaatar', continent='Asia') # we can change the default value by kwargs  
6 city('Cape Town', 'Africa') # we can change the default value by positional args.
```

```
1 Athens in Europe  
2 Ulaanbaatar in Asia  
3 Cape Town in Africa  
4
```

*args and **kwargs (review)

- ▶ The arguments we have used in the functions so far limit us to a certain extent. If we don't define the default values for the arguments, we will always need to pass the exact numbers of the arguments to match the number of the parameters defined in the function. But, there will be some situations when you might want to pass an arbitrary number of arguments.

*args and **kwargs (review)

- ▶ *Finally*, the least frequently used option is to specify that a function can be called with an arbitrary number of arguments. These arguments will be wrapped up in a tuple. Before the variable number of arguments, zero or more normal arguments may occur.

*args and **kwargs

- The logic of *args and **kwargs

```
def name(*parameter)
```

name(multiple arguments)

```
def name(**parameter)
```

name(multiple arguments)

*args and **kwargs (review)

- For example, let's define a function which takes two kinds of fruit and prints them.

```
1 def fruiterer(fruit1, fruit2) :  
2     print('I want to get', fruit1, 'and', fruit2)  
3  
4 fruiterer('orange', 'banana')  
5
```

*args and **kwargs (review)

- For example, let's define a function which takes two kinds of fruit and prints them.

```
1 def fruiterer(fruit1, fruit2) :  
2     print('I want to get', fruit1, 'and', fruit2)  
3  
4 fruiterer('orange', 'banana')  
5
```

```
1 I want to get orange and banana
```

*args and **kwargs (review)

- What if the user wants to get more than two kinds of fruit? Since we don't know how many kinds of fruit each user will enter, using '**arbitrary numbers of arguments**' is the most intelligent method. Consider the following example :

```
1 def fruiterer(*fruit) :  
2     print('I want to get :')  
3     for i in fruit :  
4         print('-', i)  
5  
6 fruiterer('orange', 'banana', 'melon', 'ananas')  
7
```

What is the output? Try to figure out in your mind...

*args and **kwargs (review)

- What if the user wants to get more than two kinds of fruit? Since we don't know how many kinds of fruit each user will enter, using '**arbitrary numbers of arguments**' is the most intelligent method. Consider the following example :

```
1 def fruiterer(*fruit) :  
2     print('I want to get :')  
3     for i in fruit :  
4         print('-', i)  
5  
6 fruiterer('orange', 'banana', 'melon', 'ananas')
```

```
1 I want to get :  
2 - orange  
3 - banana  
4 - melon  
5 - ananas
```

*args and **kwargs

▶ Task :

- ▶ Define a function named `slicer` to collect even numbers into the list `evens`, odd numbers into the list `odds` from the given numbers by using `*args`.

```
1 | 
2 | slicer(1, 2, 3, 4, 5, 6, 7, 8, 9)
3 | 
```

Output

```
even list : [2, 4, 6, 8]
odd list : [1, 3, 5, 7, 9]
```

*args and **kwargs

- ▶ The defining of that function can be like :

```
1 def slicer(*num) :  
2     evens = []  
3     odds = []  
4     for i in num :  
5         if i%2 == 0 :  
6             evens.append(i)  
7         else :  
8             odds.append(i)  
9     print("even list : ", evens)  
10    print("odd list : ", odds)
```

*args and **kwargs (review)

- You can examine the following example :

```
1 def animals(**kwargs):  
2     for key, value in kwargs.items():  
3         print(value, "are", key)  
4  
5 animals(Carnivores="Lions", Omnivores="Bears", Herbivores  
       ="Deers", Nomnivores="Human")  
6
```

-Pay attention to
the **key-value** pairs

What is the output? Try to
figure out in your mind...



*args and **kwargs (review)

- ▶ You can examine the following example :

```
1 def animals(**kwargs):  
2     for key, value in kwargs.items():  
3         print(value, "are", key)  
4  
5 animals(Carnivores="Lions", Omnivores="Bears", Herbivores  
       ="Deers", Nomnivores="Human")  
6
```

```
1 Lions are Carnivores  
2 Bears are Omnivores  
3 Deers are Herbivores  
4 Human are Nomnivores  
5
```

*args and **kwargs



▶ Task :

- ▶ Define a function named `organizer` to collect the given names into the `list names`, ages into the `list ages` by using `**kwargs`.

```
1
2 organizer(Beth=26, Oscar=42, Justin=18, Frank=33)
3 |
```

Output

```
['Beth', 'Oscar', 'Justin', 'Frank']
[26, 42, 18, 33]
```

*args and **kwargs

- ▶ The defining of that function can be like :

```
1 def organizer(**people):  
2     names = []  
3     ages = []  
4     for key, value in people.items():  
5         names.append(key)  
6         ages.append(value)  
7     print(names, ages, sep="\n")  
8
```

*args and **kwargs (review)

- Carefully examine the following examples :

```
1 def brothers(bro1, bro2, bro3):  
2     print('Here are the names of brothers :')  
3     print(bro1, bro2, bro3, sep='\n')  
4  
5 family = ['tom', 'sue', 'tim']  
6 brothers(*family)  
7
```

Calling a function
using *args

What is the output? Try to
figure out in your mind...

*args and **kwargs (review)

- Carefully examine the following examples :

```
1 def brothers(bro1, bro2, bro3):  
2     print('Here are the names of brothers :')  
3     print(bro1, bro2, bro3, sep='\n')  
4  
5 family = ['tom', 'sue', 'tim']  
6 brothers(*family)  
7
```

```
1 Here are the names of brothers :  
2 tom  
3 sue  
4 tim  
5
```

*args and **kwargs

▶ Task :

- ▶ Define a function named `merger` to prints the following output from the given tuple `genius`.
- ▶ Call the function using variable `genius` as `*args`.

```
genius = ('Bill', 'Rossum', 'Guido van', "Gates")
```

Output

```
For me, Bill Gates and Guido van Rossum are geniuses
```

*args and **kwargs



- ▶ The defined function and calling it can look like :

```
1 def merger(arg1, arg2, arg3, arg4):  
2     print(f"For me, {arg1} {arg4} and {arg3} {arg2} are geniuses.")  
3  
4 genius = ('Bill', 'Rossum', 'Guido van', "Gates")  
5  
6 merger(*genius)  
7
```

*args and **kwargs (review)



- Carefully examine the following examples :

```
1 def gene(x, y): # defined by positional args
2     print(x, "belongs to Generation X")
3     print(y, "belongs to Generation Y")
4
5 dict_gene = {'y' : "Marry", 'x' : "Fred"}
6 gene(**dict_gene) # we call the function by a single
7 argument(variable)
```

-Calling a function using **** kwargs**
-Pay attention to the type of **kwarg**. Since it works with **keys**, the type is a **dict**.

What is the output? Try to figure out in your mind...

*args and **kwargs (review)



- Carefully examine the following examples :

```
1 def gene(x, y): # defined by positional args
2     print(x, "belongs to Generation X")
3     print(y, "belongs to Generation Y")
4
5 dict_gene = {'y' : "Marry", 'x' : "Fred"}
6 gene(**dict_gene) # we call the function by a single
7 argument(variable)
```

```
1 Fred belongs to Generation X
2 Marry belongs to Generation Y
3
```



THANKS!

Any questions?

You can find me at:

- ▶ @Henry
- ▶ henry@clarusway.com

