


Hi! I just released the alpha version of my new book; Practical Python Projects. Learn more about it [on my blog](#). In 325+ pages, I will teach you how to implement 12 end-to-end projects. You can buy it from [Feldroy.com](#).

[Docs](#) » 1. \*args and \*\*kwargs

# 1. \*args and \*\*kwargs

I have come to see that most new python programmers have a hard time figuring out the \*args and \*\*kwargs magic variables. So what are they ? First of all, let me tell you that it is not necessary to write \*args or \*\*kwargs. Only the  (asterisk) is necessary. You could have also written \*var and \*\*vars. Writing \*args and \*\*kwargs is just a convention. So now let's take a look at \*args first.

## 1.1. Usage of \*args

\*args and \*\*kwargs are mostly used in function definitions. \*args and \*\*kwargs allow you to pass an unspecified number of arguments to a function, so when writing the function definition, you do not need to know how many arguments will be passed to your function. \*args is used to send a **non-keyworded** variable length argument list to the function. Here's an example to help you get a clear idea:

```
def test_var_args(f_arg, *argv):
    print("first normal arg:", f_arg)
    for arg in argv:
        print("another arg through *argv:", arg)

test_var_args('yasooob', 'python', 'eggs', 'test')
```

This produces the following result:

```
first normal arg: yasooob
another arg through *argv: python
another arg through *argv: eggs
another arg through *argv: test
```

I hope this cleared away any confusion that you had. So now let's talk about \*\*kwargs

## 1.2. Usage of **\*\*kwargs**

**\*\*kwargs** allows you to pass **keyworded** variable length of arguments to a function. You should use **\*\*kwargs** if you want to handle **named arguments** in a function. Here is an example to get you going with it:

```
def greet_me(**kwargs):
    for key, value in kwargs.items():
        print("{0} = {1}".format(key, value))

>>> greet_me(name="yasoob")
name = yasoob
```

So you can see how we handled a keyworded argument list in our function. This is just the basics of **\*\*kwargs** and you can see how useful it is. Now let's talk about how you can use **\*args** and **\*\*kwargs** to call a function with a list or dictionary of arguments.

## 1.3. Using **\*args** and **\*\*kwargs** to call a function

So here we will see how to call a function using **\*args** and **\*\*kwargs**. Just consider that you have this little function:

```
def test_args_kwargs(arg1, arg2, arg3):
    print("arg1:", arg1)
    print("arg2:", arg2)
    print("arg3:", arg3)
```

Now you can use **\*args** or **\*\*kwargs** to pass arguments to this little function. Here's how to do it:

```
# first with *args
>>> args = ("two", 3, 5)
>>> test_args_kwargs(*args)
arg1: two
arg2: 3
arg3: 5

# now with **kwargs:
>>> kwargs = {"arg3": 3, "arg2": "two", "arg1": 5}
>>> test_args_kwargs(**kwargs)
arg1: 5
arg2: two
arg3: 3
```

Order of using **\*args** **\*\*kwargs** and formal args

So if you want to use all three of these in functions then the order is

```
some_func(fargs, *args, **kwargs)
```

## 1.4. When to use them?

It really depends on what your requirements are. The most common use case is when making function decorators (discussed in another chapter). Moreover it can be used in monkey patching as well. Monkey patching means modifying some code at runtime. Consider that you have a class with a function called `get_info` which calls an API and returns the response data. If we want to test it we can replace the API call with some test data. For instance:

```
import someclass

def get_info(self, *args):
    return "Test data"

someclass.get_info = get_info
```

I am sure that you can think of some other use cases as well.

**AI-powered ad network for devs.** Get your message in front of the right developers with EthicalAds.

*Ads by EthicalAds*