# Parameter vs Argument Python

Asked 4 years, 8 months ago Modified 5 months ago Viewed 30k times

32

So I'm still pretty new to Python and I am still confused about using a parameter vs an argument. For example, how would I write a function that accepts a string as an argument?

python string function parameters arguments

12

Share Follow

asked Nov 7, 2017 at 23:21

svlk

461 1 4 7

um, what do you mean "using a parameter vs an argument"? Are you asking what those words mean? – juanpa.arrivillaga Nov 7, 2017 at 23:25

4 Answers

Sorted by:

Trending sort available

Highest score (default)

45

Generally when people say parameter/argument they mean the same thing, but the main difference between them is that the parameter is what is declared in the function, while an argument is what is passed through when calling the function.

```
def add(a, b):
    return a+b

add(5, 4)
```

Here, the parameters are `a` and `b`, and the arguments being passed through are `5` and `4`.

Since Python is a dynamically typed language, we do not need to declare the types of the parameters when declaring a function (unlike in other languages such as C). Thus, we can not control what exact type is passed through as an argument to the function. For example, in the above function, we could do `add("hello", "hi")`.

This is where functions such as `isinstance()` are helpful because they can determine the type of an object. For example, if you do `isinstance("hello", int)`, it will return False since `"hello"` is a string.

Share Follow

answered Nov 7, 2017 at 23:24

TerryA

parameter – Nameless Apr 2, 2021 at 21:11

This concept is also nicely explained here - codingeek.com/tutorials/python/function-argument – Hitesh Garg Apr 12, 2021 at 7:51

10

See the FAQ:

### What is the difference between arguments and parameters?

Parameters are defined by the names that appear in a function definition, whereas arguments are the values actually passed to a function when calling it. Parameters define what types of arguments a function can accept. For example, given the function definition:

```
def func(foo, bar=None, **kwargs):
    pass
```

*foo*, *bar* and *kwargs* are parameters of `func`. However, when calling `func`, for example:

```
func(42, bar=314, extra=somevar)
```

the values `42`, `314`, and `somevar` are arguments.

See also:

- language-agnostic What's the difference between an argument and a parameter?
  - This answer is my favourite.

For defining a function that accepts a string, see TerryA's answer. I just want to mention that you can add type hints to help people using your function to tell what types it accepts, as well as what type it returns.

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

Share Follow

edited May 30, 2021 at 18:54

answered May 29, 2021 at 19:07

wjandrea
**23.6k** 7 50 70

In Programming lingo, arguments refers to the data you are passing to the function that is being called whereas the parameter is the name of the data and we use the parameter inside

for example:

```
def functionname(something):
     do some stuff with {something}

functionname(abc)
```

in this case,

- `abc` --> argument
- `something` --> parameter

Share Follow

edited Sep 21, 2021 at 18:03

Chris
**114k** 83 254 234

answered Sep 21, 2021 at 17:33

M

Manikantan U V
**71** 1 2

---

0

A parameter is the placeholder; an argument is what holds the place.

Parameters are conceptual; arguments are actual.

Parameters are the function-call signatures defined at compile-time; Arguments are the values passed at run-time.

Mnemonic: "Pee" for Placeholder Parameters, "Aeigh" for Actual Arguments.

Share Follow

answered Jan 31 at 4:27

Bilbo
**300** 1 9

MARCH 23, 2022 / #PYTHON

# How to Use *args and **kwargs in Python

Ashutosh Krishna

In this article, we'll discuss `*args` and `**kwargs` in Python along with their uses and some examples.

When writing a function, we often need to pass values to the function. These values are called **function arguments**.

## Problem with Function Arguments

Let's define a function to add two numbers in Python. We'll write it like this:

```
def add(x, y):
    return x+y

print(add(2,3))
```

Output:

What if you need to add three numbers? Simple, we can modify the function to accept three arguments and return their sum as:

```
def add(x, y, z):
    return x+y+z

print(add(2, 3, 5))
```

Output:

```
10
```

Wasn't that quite simple? Yes, it was!

But what if we're again required to add two numbers only? Will our modified function help us get the sum? Let's see:

```
def add(x, y, z):
    return x+y+z


print(add(2, 3))
```

Output:

You see the problem?

The problem arises when we have a variable number of arguments. Should we keep modifying the function to accept the exact number of arguments? Of course not, we won't be doing this.

So there must be some other way to do it. This is where `*args` and `**kwargs` jump in.

You can use `*args` and `**kwargs` as arguments of a function when you are unsure about the number of arguments to pass in the functions.

# How to Use *args in Python

`*args` allows us to pass a variable number of non-keyword arguments to a Python function. In the function, we should use an asterisk ( `*` ) before the parameter name to pass a variable number of arguments.

```
def add(*args):
    print(args, type(args))

add(2, 3)
```

Output:

Thus, we're sure that these passed arguments make a tuple inside the function with the same name as the parameter excluding `*`.

Now let's rewrite our `add()` function with a variable number of arguments.

```
def add(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total


print(add(2, 3))
print(add(2, 3, 5))
print(add(2, 3, 5, 7))
print(add(2, 3, 5, 7, 9))
```

Output:

```
5
10
17
26
```

Note that the name of the argument need not necessarily be `args` – it can be anything. In this case it's `numbers`. But it's generally a standard way to use `*args` as the name.

# How to Use **kwargs in Python

argument.

```
def total_fruits(**kwargs):
    print(kwargs, type(kwargs))


total_fruits(banana=5, mango=7, apple=8)
```

Output:

```
{'banana': 5, 'mango': 7, 'apple': 8} <class 'dict'>
```

Thus we see that the arguments, in this case, are passed as a dictionary and these arguments make a dictionary inside the function with name same as the parameter excluding `**`.

Now, let's complete the `total_fruits()` function to return the total number of fruit.

```
def total_fruits(**fruits):
    total = 0
    for amount in fruits.values():
        total += amount
    return total


print(total_fruits(banana=5, mango=7, apple=8))
print(total_fruits(banana=5, mango=7, apple=8, oranges=10))
print(total_fruits(banana=5, mango=7))
```

```
20
30
12
```

Note that the name of the argument need not necessarily be `kwargs` – again, it can be anything. In this case, it's `fruits`. But it's generally a standard way to use `**kwargs` as the name.

# Conclusion

In this article, we learned about two special keywords in Python – `*args` and `**kwargs`. These make a Python function flexible so it can accept a variable number of arguments and keyword arguments, respectively.

Thanks for reading!

You can find the code for this blog here.

**Ashutosh Krishna**

Application Developer Intern at Thoughtworks India

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

**You can make a tax-deductible donation here.**

**Trending Guides**

| | |
|---|---|
| Discord Won't Open | C vs C++ |
| Learn Java Courses | Python JSON |
| SQL Like Statement | What is a PC? |
| File Explorer Error | What is Coding? |
| Python Find in List | Text Box in HTML |
| Functions in Python | Meta Tag in HTML |
| Python Reverse List | Append in Python |
| Create a Table in SQL | Python Not Equal |