Theory: Arguments

20 minutes 0 / 0 problems solved

By now, you are on good terms with functions, since you know how to invoke and declare them. Let's deepen your knowledge a bit and discover some new features of functions.

First, a line should be drawn between the terms **"argument"** and **"parameter"**. Parameters represent what a function accepts, it's those names that appear in the function definition. Meanwhile, arguments are the values we pass to a function when calling it. We'll cover both arguments and parameters further.

§1. Positional arguments

There are different ways to assign arguments to a function. First of all, you can do it simply by **position**. In this case, values will associate with parameters in the order in which you passed them into your function **from left to right**. Such arguments are called **positional**, or **non-keyword**.

```
def subtract(x, y):
    return x - y


subtract(11, 4)  # 7
subtract(4, 11)  # -7
```

When we swapped the numbers in the second function call, we got a different result. Thus, you can see that the order determines how arguments are assigned.

§2. Named arguments

Another way to assign arguments is by **name**. Sometimes you might want to control the order of passed values. That's where **named**, or **keyword**, arguments come into play.

```
def greet(name, surname):
    print("Hello,", name, surname)


# Non-keyword arguments
greet("Willy", "Wonka")               # Hello, Willy Wonka

# Keyword arguments
greet(surname="Wonka", name="Willy")  # Hello, Willy Wonka
```

The order doesn't matter here since parameters are matched by name. However, keyword arguments are always written **after** non-keyword arguments when you call a function:

```
greet("Frodo", surname="Baggins")  # Hello, Frodo Baggins

greet(name="Frodo", "Baggins")     # SyntaxError: positional argument follows keyword argument
```

Make sure to mention each parameter **once**. To understand why this is important, let's think about what happens every time we call a function. In fact, arguments are initialized so that all operations with the values in this function start from scratch. You cannot initialize an argument twice, so if a value has already been passed and associated with some parameter, attempts to assign another value to this name will fail.

Current topic:
Arguments Stage 1

Topic depends on:
✓ Declaring a function 5★ Stage 1

Topic is required for:
Dictionary methods Stage 1
Operations with list
Sep and end arguments of print
Sorting a list

Table of contents:
↑ Arguments
§1. Positional arguments
§2. Named arguments
§3. Names are important
§4. PEP time
§5. Conclusions
Discussion

```
def greet(name, surname):
    print("Hello,", name, surname)


greet("Clementine", name="Rose")
# TypeError: greet() got multiple values for argument 'name'
```

As shown in the example, multiple values for the same name cause an error.

§3. Names are important

We have covered the main errors that you can face. Of course, there can be more parameters in a function:

```
def responsibility(developer, tester, project_manager, designer):
    print(developer, "writes code")
    print(tester, "tests the system")
    print(project_manager, "manages the product")
    print(designer, "develops design")
```

Note that when we use keyword arguments, names are important, not positions. Thus, the following example will work correctly:

```

responsibility(project_manager="Sara", developer="Abdul", tester="Yana", designer=
"Mark")
    # Abdul writes code
    # Yana tests the system
    # Sara manages the product
    # Mark develops design
```

However, if we call the function with the same order of names, but without named arguments, then the output will be wrong, with mixed responsibilities:

```
responsibility("Sara", "Abdul", "Yana", "Mark")
# Sara writes code
# Abdul tests the system
# Yana manages the product
# Mark develops design
```

This way, Python knows the names of the arguments that our function takes. We can ask to remind us of them using the built-in `help()` function.

```
help(responsibility)
# Help on function responsibility in module __main__:
# responsibility(developer, tester, project_manager, designer)
```

§4. PEP time

Look at the declared function and function calls shown in this topic one more time: `greet(name="Willy", surname="Wonka")`. Have you noticed missing spaces around the equality sign? Their absence is not accidental. By PEP 8 convention, you should not put spaces around `=` when indicating a keyword argument.

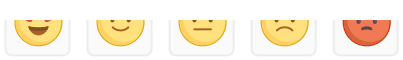
§5. Conclusions

Now that we've discussed some advanced features of functions, let's sum it up:

- There's a distinction between **parameters** and **arguments**.
- You can pass arguments to a function by **position** and by **name**.
- The **order** of declared **parameters** is important, as well as the **order of arguments** passed into a function.
- The `help()` function can tell you the **function arguments** by name.

Report a typo

706 users liked this piece of theory.. **11** didn't like it. **What about you?**



Start practicing

Comments (21) Hints (0) Useful links (1) Show discussion