Theory: Kwargs

42 minutes 7 / 8 problems solved

Start practicing

2654 users solved this topic. Latest completion was about 2 hours ago.

With `*args` you can create more flexible functions that accept a varying number of *positional* arguments. You may now wonder how to do the same with *named* arguments. Fortunately, in Python, you can work with ***keyword*** arguments in a similar way.

§1. Multiple keyword arguments

Let's get acquainted with the `**` operator used to pass a varying number of keyword arguments into a function. `**kwargs` collects all possible extra values in a dictionary with keywords as keys.

By convention, people use special names for this kind of arguments: `*args` for positional arguments and `**kwargs` for keyword arguments, but you can call them whatever you want. The main thing is that a single asterisk `*` matches a value by position and a double asterisk `**` associates a value with a name, or keyword. So, `**kwargs` differs from `*args` in that you will need to assign keywords.

Here is an example:

```
def capital(**kwargs):
    for key, value in kwargs.items():
        print(value, "is the capital city of", key)



capital(Canada="Ottawa", Estonia="Tallinn", Venezuela="Caracas", Finland="Helsinki
")
```

Once the function has been invoked, these 4 lines will be printed:

```
Ottawa is the capital city of Canada
Tallinn is the capital city of Estonia
Caracas is the capital city of Venezuela
Helsinki is the capital city of Finland
```

So, everything works just fine! And again, the number of arguments we pass may differ in the next call.

Note that the names in a call are without quotes. That is not a mistake. Moreover, the names should be valid, for example, you cannot start a keyword with a number. Follow the same naming rules as for variables.

It is also possible to combine `*args` and `**kwargs` in one function definition:

```
def func(positional_args, defaults, *args, **kwargs):
    pass
```

The order is crucial here. Just as non-keyword arguments precede keyword arguments, `*args` must come before `**kwargs` in this case. Otherwise, both when creating and calling a function with `*args` and `*kwargs` in the wrong order, a `SyntaxError` will appear:

```
def func(positional_args, defaults, **kwargs, *args):
# SyntaxError: invalid syntax

func(positional_args, defaults, **kwargs, *args)

# SyntaxError: iterable argument unpacking follows keyword argument unpacking
```

§2. Unpacking in function calls

Current topic:

✓ Kwargs

Topic depends on:

✗ Operations with dictionary Stage 1

✓ Args

Topic is required for:

Rendering templates

Table of contents:

There are two unpacking operators in Python: a single asterisk `*` unpacks elements of an iterable object and a double asterisk `**` works with dictionaries. Let's try to get key-value pairs from a dictionary and pass them as keyword arguments using a double asterisk `**`:

```
def say_bye(**names):
    for name in names:
        print("Au revoir,", name)
        print("See you on", names[name]["next appointment"])
        print()


humans = {"Laura": {"next appointment": "Tuesday"},
          "Robin": {"next appointment": "Friday"}}

say_bye(**humans)

# Au revoir, Laura
# See you on Tuesday
#
# Au revoir, Robin
# See you on Friday
```

By default, you iterate over keys in a dictionary, so be careful with this. You might need this type of unpacking when setting specific parameters of a function. Saving values in a dictionary and then unpacking them in this way might be much easier than listing them in each call manually. Also, it will save time when you choose to fine-tune these parameters.

§3. Recap

Let's go over the main points discussed in the topic:

- If you want to work with a varying number of **keyword** arguments, make use of `**kwargs`.
- The variable name `kwargs` is conventional, you can always choose another one.
- Notice the difference: `*args` provides access to a **tuple** of remaining values, while `**kwargs` collects remaining key-value pairs in a **dictionary**.
- The order of parameters in the function definition is important, as well as the order of passed arguments.
- In function calls, now you can use both **unpacking operators**: a single asterisk `*` for iterable objects and a double asterisk `**` for dictionaries.

Report a typo

Thanks for your feedback!

Write here how we could improve this theory

Start practicing

Comments (5)   Hints (0)   Useful links (2)   Show discussion