Parameters and returning

Table of contents

This lesson is all about parameters (inputs to a function) and `return`ing! Happy coding :^)

Parameters

Parameters are pieces of information passed between functions. `example_function(param)` has one parameter, `param`, that can be used the same way you'd use a variable inside of the `example_function(param)` code block.

Ones digit

Fill out the function `print_ones_digit(num)` , which takes as input an integer `num` and prints its ones digit. The modulo (remainder) operator, `%`, should be helpful to you here. We've written a `main()` function which asks for user input and then calls `print_ones_digit(num)` .

Here's a sample run (user input in bold italics):

```
$ python ones_digit.py
Enter a number: 42
The ones digit is 2
```

Print divisors

Fill out the function `print_divisors(num)`, which takes in a number and prints all of its divisors (all the numbers from 1 to `num` inclusive that `num` can be cleanly divided by (there is no remainder to the division). We've given you a `main()` which prompts the user to input a number and then calls your code for `print_divisors(num)`.

Here's a sample run (user input in bold italics):

```
$ python divisors.py
Enter a number: 12
Here are the divisors of 12
1
2
3
4
6
12
```

Print multiple

Fill out `print_multiple(message, repeats)`, which takes as parameters a string `message` to print, and an integer `repeats` number of times to print `message`. We've written the `main()` function for you, which prompts the user for a message and a number of repeats.

Here's a sample run of the program (user input in bold italics):

```
$ python print_multiple.py
Please type a message: Hello!
Enter a number of times to repeat your message: 6
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
```

Sentence generator

Implement the helper function `make_sentence(word, part_of_speech)` which will take a string `word` and an integer `part_of_speech` as parameters and, depending on the part of speech, place the word into one of three sentence templates (or one from your imagination!):

- If `part_of_speech` is **0**, we will assume the word is a **noun** and use the template: "I am excited to add this ____ to my vast collection of them!"
- If `part_of_speech` is **1**, we will assume the word is a **verb** use the template: "It's so nice outside today it makes me want to ____!"
- If `part_of_speech` is **2**, we will assume the word is an **adjective** and use the template: "Looking out my window, the sky is big and ____!" `make_sentence(word, part_of_speech)` should not return anything, just print the correct sentence with the word filled in the blank.

Here's a sample run of the program (user input in bold italics):

```
$ python sentence_generator.py
Please type a noun, verb, or adjective: groovy
Is this a noun, verb, or adjective?
Type 0 for noun, 1 for verb, 2 for adjective: 2
Looking out my window, the sky is big and groovy!
```

Return

Returning is an important idea -- when you `return`, you end the execution of whatever function you're in (you can't return if you're not inside of a function) and return to wherever in the code that function was called. You can also return different data types in order to pass information from one function back up to where that function was called.

Get name

Fill out the `get_name()` function to return your name as a string! We've written a `main()` function for you which calls your function to retrieve your name and then prints it in a greeting.

Here's a sample run of the program where the name we've decided to return is Karel:

```
$ python get_name.py
Howdy Karel ! 
```

Double

Fill out the `double(num)` function to return the result of multiplying `num` by 2. We've written a `main()` function for you which asks the user for a number, calls your code for `double(num)` , and prints the result.

Here's a sample run of the program (user input in bold italics):

```
$ python double.py
Enter a number: 2
Double that is 4
```

Is even

Fill out the `is_even(num)` function which returns whether or not the inputted integer `num` is even. Your function should return a boolean. We've written a `main()` function which asks a user for input and then prints whether or not the number is by calling your `is_even(num)` function to verify.

Here's a sample run of the program (user input in bold italics):

```
$ python is_even.py
Enter a number: 20
That number is even!
```

Chaotic counting

Fill out the `chaotic_counting()` function, which prints the numbers from 1 to 10, but with a catch. We've written a `done()` function which returns `True` with likelihood `DONE_LIKELIHOOD` -- at each number, before printing the number, you should call `done()` and check if it returns True or not. If `done()` returns True, we're done counting, and you should use a `return` statement to end the `chaotic_counting()` function execution and resume execution of `main()`, which will print "I'm done.". We've written `main()` for you -- check it out! Notice that we'll only print "I'm done" from `main()` once `chaotic_counting()` is done with its execution.

Here's a sample run of this sassy program:

```
$ python chaotic_counting.py
I'm going to count until 10 or until I feel like stopping, whichever comes first.
1
2
3
I'm done.
```