KEY_Lesson19_Functions

February 4, 2020

1 Writing Functions

Remember: - You've already used lots of built-in functions in python. Examples are `print()` and `sum()`. Can you think of others? - Functions take an input (*arguments*) and *return* an output.

Guess what? You can also make your own functions! Why would you want to do this? - Have you ever copied and pasted code because you want to reuse it but with different data or in a slightly different way? If so, you might want to make that code into a function! - Using functions also makes your code much easier to read.

1.1 Anatomy of a Function

Functions have an **input** and an **output**. We provide the input, and then the function does things to generate the output. Another way to put this is functions *take* arguments (i.e. input) and *return* an output.

```
# Anatomy of a function
def my_function(input):
    # do things to input
    return(output)
```

Let's take a look at a few different analogies to get a better idea of what functions are.

1.2 Function Analogies

"Function" name	Input: what "function" takes	Under the hood: what "function" does	Output: what "function" returns
Vending machine	Money & snack choice	Some computational/mechanical process	Snack
Google maps	Start & end location	Finds fastest route	Directions for fastest route

Now let's try a real example.

1.3 Writing and using a simple function

Let's write a function called `power` that calculates the power of two numbers (a base and an exponent). It **takes** two numbers - a base and an exponent - and **returns** the base raised to the exponent. It's important to document what your function does so other people can use it.

[1]:
```
# write function to find power of two numbers
def power(base, exponent):
    # find power of base raised to exponent
    # example: power(3,2)
    p = base ** exponent
    return p
```

Now let's test our function out! You can use any two numbers as the input to the function.

Remember that each input for a function is called an **argument**. Each argument is given a variable name which allows us to use that input in our function. If you include the argument names, then you can include the numbers in any order you want:

[2]:
```
# using numbers as input
# explicitly name arguments (order doesn't matter)
print(power(exponent = 2, base = 3))
print(power(base = 3, exponent = 2))
```

```
9
9
```

Here, you should get the same answer for both.

If you decide to just include the numbers and not the names, then you have to make sure the numbers are in the correct order (i.e. the order in which the arguments are defined in the function - base first and exponent second):

[3]:
```
# using numbers as input
# using the order of the arguments (order matters)
print(power(3,2))
print(power(2,3))
```

```
9
8
```

Here, you should get a different answer for each. What is the base and what is the exponent for each of these examples?

For the first example, `base` is 3 and `exponent` is 2. For the second example, `base` is 2 and `exponent` is 3.

You can also use variables as input:

[4]:
```
# using variables as input
b = 3
e = 2
power(b,e)
```

[4]:
```
9
```

Just like built-in functions, you can also save the output of the function to a variable:

[5]:
```
# saving it to a variable
p = power(b,e)
print(p)
```

```
9
```

Let's try other inputs, because that's the real power of using functions (no pun intended).

[6]:
```
power(10,3)
```

[6]:
```
1000
```

Feel free to try out other inputs as well!

1.4 Scope of argument variables

The input arguments in the power function are `base` and `exponent`. These *variables* are defined only within the context of the function, not in the global environment. So we can print out `base` and `exponent` within the function, but **if we try to print out either of these variables outside**

of the function, we will get an error (unless it's defined in your global environment). Let's try it out. **What do you think happens if we try to print out `base` outside of the function?**

[7]:
```
# print base outside of function (uncomment this to run)
#print(base)
```

It doesn't exist! This is called the *scope* of the variables - they can only be seen in the function, but not in the global environment.

1.5 Optional arguments with default values

If you want something to normally happen, but have the option for it to not happen, you can use *optional arguments*. For instance, you may want the `power` function to, by default, square the `base` variable (i.e. have `exponent=2`), but give the user the option to change it if they want. In this case, if the user doesn't specify a value for exponent, the function uses the *default* option, which is defined where you define the argument in the function:

[8]:
```
# write function to find power of two numbers
def power(base, exponent=2):
    # find power of base raised to exponent
    # example: power(3,2)
    p = base ** exponent
    return p

# default
print(power(2))
# change exponent
print(power(2, 3))
```

```
4
8
```

Note: You have to include arguments that don't have default values. If not, then you get an error because there is nothing stored in that variable in the function, so the code inside can't be executed:

[9]:
```
# what happens if you run this line? (uncomment it)
#power(exponent=1)
```

What argument are we missing here?

We're missing the `base` argument.

1.6 Argument variable names

Another important note is that **it doesn't matter what we call the input arguments.** Right now, the input arguments are `base` and `exponent`. Let's try changing them to something totally random, maybe `pizza` and `pie`. Pizza and pie probably doesn't have anything to do with the input (two numbers), but the computer doesn't know that!

[10]:
```
# use pizza as variable name
#  function to find power of two numbers
def power(pizza, pie):
    # find power of base raised to exponent
    # example: power(3,2)
    return pizza ** pie

# test it out
power(3,2)
```

[10]: 9

Although you can name your input arguments anything since the computer doesn't care, **you actually want to name them something useful so that people reading the code (including your future self!) can more easily understand what's going on.** Thinking of good variable names can be hard, but it's important!

Nice job! You've learned so much about writing functions.

1.7 Summary of functions

- Functions are used to make your code easier to read and reuse.
- Functions take an input (*arguments*) and *return* an output.
- Arguments are variables that only exist inside the function.
- You can have *default* arguments for your function.
- Document your function well so other people know how to use it!