Python: Functions

Computer Science ICS20

What is a Function?

A function is a stand-alone program that usually performs a specific task. We have been using functions in almost all of our Python programs when we use the input() and print() functions. More recently, we have used other functions such as len(), max(), min(), and sum(). The reason that we use functions are the following:

1. functions allow use to reuse code.
2. functions aid in solve large problems by splitting them up into smaller problems.
3. functions allow teams of people to work together on a single programs by providing better modularity.
4. functions allow us to abstract a process, freeing us to think about the bigger picture.

Defining a Function

Functions in Python are defined using the following template:

```
def function_name(optional parameters, ...):
        <body of function goes here>
```

Let's create our first function. For this example we will create a function that calculates the area of a circle of a given radius. Following the template above we could write the following:

```
def areaOfCircle(r):
        area = 3.14159 * r * r
        return area
```

Notice that in this function the variable "area" is calculated and then returned. But returned to whom or what?

Calling a Function

It is important to realize that the above code is not executed unless we tell it to. Getting a function to execute its code is referred to as **calling** the function and it is as simple as typing the function name whenever you want to run it. The one thing that must be remembered is that the function must be defined **before** it can be called. Here is an example of a program that uses our areaOfCircle function:

```
radius = int(input("Enter the radius of the circle: "))
area = areaOfCircle(radius)
str = "The area of the circle with radius " + radius + " is: " + str(area)
print(str)
```

Notice that in the second line of the above program, the function is called simply by typing it's name and passing a radius value as a parameter. Once the function finishes, it will return the computed area which, in this case, is saved to another variable called "area".

The Parameters of a Function

In the example above, we required that the function be passed a parameter (or sometimes called argument) which was interpreted as a radius value. However, we can pass other parameters as well, or none at all. There are 3 ways to pass parameters to a function that we will look at:

- required arguments
- keyword arguments
- default arguments

Required Arguments

Required arguments are the parameters passed to a function in correct positional order. Here the number of arguments in the function call should match exactly with the function definition. For the areaOfCircle() function, exactly one argument must be passed to it, otherwise it gives a syntax error telling you how many arguments should be provided. For example if we called the areaOfCircle() function without any arguments, we would get the following result:

```
        areaOfCircle();
TypeError: areaOfCircle() takes exactly 1 argument (0 given)
```

Keyword Arguments

Keyword arguments allow you to identify the arguments by the parameter name. As an example:

```
print(areaOfCircle(radius = 5))
```

will print the area of a circle with radius = 5. In order to see this better let's change our function to accept two variables like this:

```
def areaOfCircle(r, prt):
        area = 3.14159 * r * r
        if prt == True:
                print(area)
        return area
```

The new function now has an additional Boolean variable "prt", which will print the area as well as return its value if it is true. We can now call it like this:

```
A = areaOfCircle(radius = 5, prt = True)
```

or like this:

```
A = areaOfCircle(prt = True, radius = 5)
```

Both would print the area as well as return its value and assign it the variable "A". The order of the parameters does not matter if you use keyword arguments. Of course, if we choose to use the required agruments we could also write the following:

```
A = areaOfCircle(5, True)
```

In this case we have to remember to pass the arguments in the correct order as indicated by the function definition.

Default Arguments

A default argument is an argument that takes on a default value if it is not provided in the function call. The function definition specifies the default value. So we could once again change our areaOfCircle() function to have a default value:

```
def areaOfCircle(r, prt = False):
        area = 3.14159 * r * r
        if prt == True:
                print(area)
        return area
```

Now that the second parameter has a default value we can optionally omit it in a function call:

```
A = areaOfCircle(5)
A = areaOfCircle(5, True)
```

In the first line the default value is “False” and so the result just returned and not printed. In the second line, the area is printed since the second argument in explicitly provided.

The *return* Statement

The return statement in a function immediately exits the function and resumes executing the line right after the line that called the function. A return statement with no arguments returns a “None” data type. In our example the return statement returns a number representing the area of the circle. Any line after the return statement will never be reached so you should avoid this situation.

Exercises

For the following functions that you will create, you will need to write your own code to test them.

1. Write a Python function that returns the largest of three given numbers.
2. Write a Python function that converts temperatures values between Celsius and Fahrenheit.
3. Write a Python function that returns the average value of a list of integers.
4. Write a Python function that returns the median value of a list of integers.
5. Write a Python function that returns the factorial of a given number.
6. Write a Python function that finds the roots of a linear equation.
7. Write a Python function that finds the roots of a quadratic equation.
8. Write a Python function that determines if a given string is a palindrome.
9. Two numbers are friendly if each one is the sum of the divisors of the other (include 1, but not the number itself in the divisors). Eg: 220 and 284 are friendly:

   1+2+4+5+10+11+20+22+44+55+110=284 1+2+4+71+142=220.

   Write a Python function that returns true if two given numbers are friendly.

10. A number n is fat if the sum of its divisors > 3n. A number n is lean if the sum of its divisors = n+1. A number n is a Jack Sprat number if n is lean and n+1 is fat.
    Write a Python function that returns true if a given number is a Jack Sprat number.