Functions, Part 1

October 12, 2020

Administrative Notes

We’ll start by going over Exam 1

What is a function?

In math, it's a unique mapping between each input and one output

$f(x) = x^2 + 2x + 1$

Put in a value for x, you'll get back one output.

In python, it's sort of the same idea, but not exactly

A block of code, called by its name, that optionally takes input and returns a specific output

An example of a function in Python

```
def calculate_days (years, months, days):
    num_days = 365 * years + 30 * months + days
    if years > 4:
        num_days += 1
    return num_days
```

This is the name of the function. It must be a legal Python variable name

These are the parameters.

The word “def” means you’re defining a function

This code is the body of the function

The “return” statement is a way of passing a value back to where the function was called from

Calling a function

To call the function the previous slide, calculate_days:

```
yrs = 23

mos = 3

days = 22

length_of_time = calculate_days(yrs, mos, days)
```

These are arguments. They will be matched up with parameters, in order

If the function returns a value, you have to store it somewhere if you want to use it

Call the function by using its name, with appropriate arguments

Why use functions?

Use a function whenever your program is going to do the same thing multiple times

- No, not like in a loop. In different *parts* of the program.
- You ***could*** rewrite the code each time, but you're likely to do it differently somewhere
- You could just copy and paste the same code each time, but what if you're just copying and pasting a bug?

Using functions simplifies the program and makes it easier to get the program correct

Functions vs. methods

calculate_days() is a *function*; str.split() is a *method*. What's the difference?

A ***method*** is explicitly tied to ***one object***; the object on which it is invoked.

A ***function*** is not tied to anything; it operates on the values passed to its parameters (if there are any).

Hopefully this becomes clear over the next couple of days!

Built-in functions and user-defined functions

Python has a number of built-in functions that you've already used:

```
print("Hello, world")   # print is a function; "Hello, world" is its argument.

len(num_list)           # len is a function; num_list is its argument. It returns the number of elements in num_list
```

You can add to the built_in functions with functions that you will define yourself. Like calculate_days, a few slides ago.

Where do you define functions?

Usually, above the main program - but that's *NOT* required.

Start with your header comment

```
LABELS = ['A', 'B', 'Others']
```

Now define your constants

```
def calculate_days(days, months, years)
```

Then define your functions

```
If __name__ == "__main__":
```

And finally your main program

Where do you call functions?

Anywhere other than on the left hand side of an equals sign

The call to a function can be anywhere in the program where the function is needed

Can a function call another function?

- YES

Can a function call itself?

- YES; that's called 'recursion' and we'll get to that

Can a function call the main program?

- NO; it can return values to the main program but that's all

Matching arguments and parameters

I said that parameters and arguments are matched in order. What does that mean?

```
def subtract(x, y):
   print(x,”-”,y,”=”,str(x-y))
if __name__ == “__main__”:
   x = 3
   y = 4
   subtract(y, x)
```

What happens if the arguments and parameters don’t match?

```
def subtract(x, y):
   print(x,”-”,y,”=”,str(x-y))
if __name__ == “__main__”:
   x = 3
   y = 4
   subtract(y)
```

Now some examples

The rest of the class period will be taken up with some examples of programming using functions. More explanations on Wednesday.