CST0006 - Computer Programming Foundations

INTRODUCTION to WEEK08

In this lecture and lab section we are going to talk about modular programming and code reuse. Many times in programming you will reuse code that you have written or code that someone else has written.

Most programs you write will include code that someone else has written or functions that have been included in the language you are programming in. Code reuse, is one of the most valuable/useful ways to build a program.

After this lecture and lab section, you will understand what a function is, how to create, import and use functions and modules that will add extra functionality to your program.

Modular Programming

Modular programming is when you break down your program into independent sections of functionality or sub-routines as their own separate logical components. These individual modules and can be organized and saved in their own separate file which will allow them to be debugged, tested and maintained as logical units that do one specific task or deal with a set of similar tasks. Another benefit of modular programming is it allows you the opportunity to split your program among many developers and modules can be swapped out and replaced with newer better versions of themselves.

Functions

Functions are a block of code that you can "call" in your code. Instead of repeating yourself you can create a function and call that function multiple times. Some functions are predefined by the language, some can be "user-defined functions" created by you or a colleague. An example of a predefined function is the print statement. Print statements are available to you in every language, someone has gone through the trouble of creating and debugging a print function that you can can call, over and over again as you see fit within your algorithm to solve your problem. The exact syntax to call the print function is different language by language. In some languages, the print function is called print(), in other languages it is called printf(), even further, some languages it is called system.out.println().

Knowing how each print statement is invoked is one of the challenges of learning each programming language, but the benefits of using this predefined function far outweigh you creating and debugging your own.

Parameters and Arguments

Parameters are the variable names and data types associated to the input data that you declare in the function. Arguments are the actual data values you pass into the function.

If you crated a function with a parameter that you called "*num*" and a data type of integer, then the argument you pass in would be 2.

Said another way that makes a bit more sense, if you were creating a function that looped through the number of children a parent has, to get the names of their children, you first need to pass in an integer into that function. Within that function, you will be looping the question "What is your (*num*) child's name?" and you need that number to know how many times to ask that question.

The data type of that variable is important, because if the parent just said Hansel and Gretel, you can't use Hansel and Gretel to loop with, because they are string values not integers.

Pass-by-Reference vs Pass-by-Value

This is one of the most difficult concepts to figure out in any language you are learning.

Pass-by-value makes a copy of the variable and gives it to the function.

Pass-by-reference you are passing the memory address that the variable is pointing to.

Python isn't either of these. Python uses pass by object reference. Which means that some functions can change the variable depending on if the data type is mutable or not.

An example of mutable and immutable data types:

```
# example one

animals = ["dog", "cat", "pig", "fox"]

print("before the function:", animals)

def test(animals):
    new_animals = ["emu", "kangaroo"]
    animals.extend(new_animals)
    print("inside of the function:", animals)

test(animals)

print("after the function:", animals)
```

```
# example two

a = 2
b = 3

def change(a):
    a = a + 1

change(a)

print(a)
print(b)
```

Default Values

Default values are arguments that you can assign to a parameter, that you can set up so that if something calls the function without specifying an argument, then the default value will be chosen.

```
def deck(cards = 52):
    print("The number of cards in this deck:", cards)

deck(12)
deck()
```

Return values

Return values are considered to be the output value of a function. In other words, it is the value that is returned to the function caller after the function is finished running. An example of this would be:

```
num = 2

def double_num(num):
    return num * 2

print(double_num(num))
```

In some languages, if a function doesn't have a return value, it is called a procedure. In either case, it's a function that just goes and does something and doesn't require any kind of return value that needs to be acted on. This of the print function. You call it to print something and it should just go an print something.

```
print("There's a hole in the bucket, dear Liza, dear Liza")
```

If a function in Python doesn't specifically include a return statement, the function will have a default return value of "None".

Scope

Scope is when variables are active or alive in your code.

global names vs local names

Compare:

```
def bucket():
    print(s)

s = "There's a hole in the bucket, dear Liza, dear Liza"

bucket()
```

With:

```
def bucket():
    s = "Well FIX it, dear Henry!"
    print(s)

s = "There's a hole in the bucket, dear Liza, dear Liza"

bucket()
```

And with:

```
def bucket():
    global s
    s = "new"
    print(s)

s = "There's a hole in the bucket, dear Liza, dear Liza"

bucket()
print(s)
```

Import Statements

After you've added modularity to your program, you will want to add all those pieces back into one program. You do this by import statements. You can import your own code, or code that is written by Python as additional modules.

You can use an import statement to include a random number generator:

```
import random

random_num = random.randint(1, 100)
print(random_num)
```

If you only want to include one of the functions from a module or want to access each function directly:

```
from random import randint

num = randint(1,100)

print(num)
```

If you create your own module you will name it something like "awesome.py", then use the import statement to include your awesome.py into your code:

```
import awesome
```

Notice that we didn’t include the .py extension. All the same rules apply to your own modules as the modules that have been provided by the language.

If you want to import all functions in one using a from import statement:

```
from awesome import *
```