CSC 110: Week 4, Lecture 1

Dr. Bodine

Last week:

- String data type
  - Sequence of characters
- String operations and methods
- Encoding/Decoding
- File handling
  - Opening
  - Reading
  - Writing
  - Closing

Quiz!

- Standard 20 minutes to answer questions

- Again this week: any questions you did not answer correctly you can get ½ credit for explaining what you missed.

  - Reminder: you can't just write the correct answer, you have to explain your reasoning and what you learned from going back to look at the problem

  - Why? What does this activity ask us to do?

Homework Assignment 3

- Remember, you can't use structures we haven't discussed yet.
  - For example, if statements.
  - Why not? Am I just being mean?

- Lookup table
  - You need to define your lookup table (see page 129—130, later in chapter)
  - Relate the score to an index, similar to median last week

- Input and output files
  - Check your output with Notepad++ (right-click on file, select edit)

This week…

- Functions!
  - Subprograms
  - Why do we need them?
- Defining functions
  - Informally and formally
  - Organization of code
- Parameters and scope
  - Inputs to our functions
- Results
  - Outputs
  - Returns; modifying parameters

Functions

- Functions are used to:
  - Avoid duplicating code
  - Make programs modular (reusable portions)
  - Make programs understandable/readable

- Functions are a shorthand way of referring to blocks of code

- Haven’t we already been using functions?
  - Any examples?

Defining and calling functions

- Define functions
  - Give a sequence of statements a name
  - Code is not executed at definition

```
>>> def testfunction():
        print("This is a test function!")
```

- Call (or invoke) functions
  - When we use the function
  - Actually executes the statements

```
>>> testfunction()
This is a test function!
>>>
```

Defining functions in python

- In python, use `def` to define functions
- Just like we have been doing with main!

```
def main():
        <insert what you want the function to do here>
```

- Now if we want to have another function we can add:

```
def otherfunction():
       <insert what you want the function to do here>
```

- Then we can call `otherfunction()` inside our program
  - Even inside the `main()` function

Think-pair-share

- Thinking about the homework assignments so far, are there examples of where we could have used functions? What would be some potential issues (things we haven't discussed that might be tricky) in defining those functions?

Simple function example

- Book uses Happy birthday song!

```
>>> def happy():
        print("Happy birthday to you!")


>>> def sing():
        happy()
        happy()
        print("Happy birthday, dear Buttercup.")
        happy()


>>> sing()
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Buttercup.
Happy birthday to you!
>>>
```

Simple function example

- Book uses Happy birthday song!

```
>>> def happy():
        print("Happy birthday to you!")


>>> def sing():
        happy()
        happy()
        print("Happy birthday, dear Buttercup.")
        happy()


>>> sing()
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Buttercup.
Happy birthday to you!
>>>
```

What if we want to sing to someone else????

The idea of parameters/arguments

- Inputs to the function
  - Values that the code has access to and will want to use
  - Only accessible from within the function
  - Cannot be used later

```
>>> def testfunction(testparameter):
        testparameter = testparameter**2
        print(testparameter)


>>> testfunction(3)
9
>>> testparameter
Traceback (most recent call last):
  File "<pyshell#122>", line 1, in <module>
    testparameter
NameError: name 'testparameter' is not defined
>>>
```

Think-pair-share

- Think-pair-share: Consider the following code:

```
#This is a sample function definition

def main():
    score = eval(input("Enter the score "))
    printscore(score)

def printscore(inscore):
    print("The score is ", inscore)


main()
```

- What is the difference between score and inscore? What happens to the parameters after the function is executed?

Scope: where variables are accessible

- *Local* variables
  - Can only be used inside the given function
  - Can have the same name as local variables in other functions!
  - Can be passed as parameter to allow access from another function

- *Global* variables
  - Can be accessed anywhere in the program
  - Not available in all languages
  - Controversial in some languages and with some programmers

Examining scope

```
>>>
>>> x = 5
>>> def testfunction():
        for i in range(4):
                print(x+i)


>>> testfunction()
5
6
7
8
>>>
```

Examining scope

```
>>> x = 5
>>> def testfunction():
        for i in range(4):
                x+=i
                print(x)


>>> testfunction()
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    testfunction()
  File "<pyshell#63>", line 3, in testfunction
    x+=i
UnboundLocalError: local variable 'x' referenced before assignment
```

Examining scope

```
>>> def testfunction():
        x = 5
        for i in range(4):
                x+=i
                print(x)


>>> testfunction()
5
6
8
11
```

Examining scope

```
>>> def testfunction():
        x = 5
        for i in range(4):
                x+=i
                print(x)


>>> testfunction()
5
6
8
11
>>> x
5
>>>
```

Parameters

- Values we want to access inside function
- How do we specify functions and parameters?

```
def <name>(<formal-parameters>):
        <body>
```

Parameter example

- Consider our first homework example. Can redefine the main function of our squaring/reciprocal finding program to take a value as input on the command line instead of prompting user for it.

Parameter example solution

```
>>> def testfunction(inputvalue):
        #Calculate square and reciprocal
        square = inputvalue**2
        reciprocal = 1/inputvalue

        #print square and reciprocal
        print("The square of",inputvalue, "is", square)
        print("The reciprocal of", inputvalue,"is",reciprocal)


>>> testfunction(3)
The square of 3 is 9
The reciprocal of 3 is 0.3333333333333333
>>>
```

But what if I wanted to break it down further?

- What if I wanted to put the squaring and finding the reciprocal in their own functions? Can I do that?
- What might be an issue with that?
- What functionality do I have to add to get that?

Our homework example

```
>>> def squarefunction(value):
        print("The square of",value,"is",value**2)


>>> def reciprocalfunction(value):
        print("The reciprocal of",value,"is",1/value)


>>> def testfunction(value):
        squarefunction(value)
        reciprocalfunction(value)


>>> testfunction(3)
The square of 3 is 9
The reciprocal of 3 is 0.3333333333333333
```

But now, what if we want to extend this to print the square of the reciprocal? Print statements don't mean we can't compose these functions are get our result - > Not modular coding…

Results from functions

- Previous examples suspiciously focused on printing results to screen
- Returns allow us to use what we compute in functions in other places in the program
  - Get a handle on the information so we can access it later

Our homework example

```
>>> def squarefunction(value):
        return value**2

>>> def reciprocalfunction(value):
        return 1/value
```

Our homework example

```
>>> def squarefunction(value):
        return value**2

>>> def reciprocalfunction(value):
        return 1/value


>>> def testfunction():
        print("This program finds the reciprocal and square of a value you enter.")
        invalue = eval(input("Enter a value: "))
        print("The square is ",squarefunction(invalue))
        print("The reciprocal is ",reciprocalfunction(invalue))

>>> testfunction()
This program finds the reciprocal and square of a value you enter.
Enter a value: 4
The square is  16
The reciprocal is  0.25
```