Introduction to Computer Programming

Working with Functions in Python

CSCI-UA.0002-005
For Loops

```
for variable in [value1, value2, etc]:
    statement
    statement
    statement
```

- The "for" keyword begins a loop.
- "in" keyword.
- Target variable.
- List of items to be iterated over.
- Statements to be executed.
- Indentation indicates that the statements under the while should be repeated.
But first... “for” loop review

The “for” loop will iterate once for each item defined in the list passed to it when the loop begins.

Lists in Python are defined by the square bracket characters “[“ and “]”. Items in a list are separated by a comma.

The first time a “for” loop iterates the target variable will assume the value of the first item in the list.

The second time a “for” loop iterates the target variable will assume the value of the second item in the list.

This continues until you reach the end of the list.
For Loops

for c in [1, 2, 3, 4]:
    print (c)
# range () function

<table>
<thead>
<tr>
<th>range () call</th>
<th>iterable</th>
</tr>
</thead>
<tbody>
<tr>
<td>range(5)</td>
<td>[0, 1, 2, 3, 4]</td>
</tr>
<tr>
<td>range(10)</td>
<td>[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]</td>
</tr>
</tbody>
</table>
range () function

You can pass additional parameters to the range() function to cause it to behave differently.

For Examples:

```
range(1,5)      # set a start and end value for the range
# [1,2,3,4]

range(5,10)    # [5,6,7,8,9]

range(0,10,2)  # set a start, end and step value
# [0,2,4,6,8]

range(1,10,2)  # [1,3,5,7,9]
```
Challenge

Write a program that calculates the multiplication tables listed to the bottom

Match the formatting in the diagram

Extension: Allow the user to enter the starting number and the ending number

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
</tr>
</tbody>
</table>
A function is a group of statements that exist within a program for the purpose of performing a specific task.

Since the beginning of the semester we have been using a number of Python’s built-in functions, including:

- `print()`
- `range()`
- `len()`
- `random.randint()`
- … etc
Most programs perform tasks that are large enough to be broken down into subtasks

Because of this, programmers often organize their programs into smaller, more manageable chunks by writing their own functions

Instead of writing one large set of statements we can break down a program into several small functions, allowing us to "divide and conquer" a programming problem
Functions, like variables must be named and created before you can use them.

The same naming rules apply for both variables and functions.

You can’t use any of Python’s keywords.
No spaces.

The first character must be A-Z or a-z or the “_” character.

After the first character you can use A-Z, a-z, “_” or 0-9.

Uppercase and lowercase characters are distinct.
Functions

def myfunction():
    print ("Printed from inside a function")

# call the function
myfunction()
Functions

When you run a function you say that you “call” it

Once a function has completed, Python will return back to the line directly after the initial function call

When a function is called programmers commonly say that the “control” of the program has been transferred to the function. The function is responsible for the program’s execution.

Functions must be defined before they can be used. In Python we generally place all of our functions at the beginning of our programs.
# Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(&quot;Welcome!&quot;)</code></td>
<td></td>
</tr>
<tr>
<td><code>for x in range(3):</code>&lt;br&gt;<code>    print(x)</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(&quot;Welcome!&quot;)</code></td>
<td>Welcome!</td>
</tr>
<tr>
<td><code>for x in range(3):</code></td>
<td></td>
</tr>
<tr>
<td><code>print(x)</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
# Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print (&quot;Welcome!&quot;)</code></td>
<td><code>Welcome!</code></td>
</tr>
<tr>
<td><code>for x in range(3):</code>&lt;br&gt;<code>print (x)</code></td>
<td></td>
</tr>
<tr>
<td><code>print (&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print (&quot;Welcome!&quot;)</code></td>
<td>Welcome!</td>
</tr>
<tr>
<td><code>for x in range(3):</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>print (x)</code></td>
</tr>
<tr>
<td><code>print (&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(&quot;Welcome!&quot;)</code></td>
<td>Welcome!</td>
</tr>
<tr>
<td><code>for x in range(3): print(x)</code></td>
<td>0</td>
</tr>
<tr>
<td><code>print(&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Output</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td><code>print (&quot;Welcome!&quot;)</code></td>
<td>Welcome!</td>
</tr>
<tr>
<td><code>for x in range(3):</code></td>
<td>0</td>
</tr>
<tr>
<td><code>print (x)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>print (&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print (&quot;Welcome!&quot;)</code></td>
<td>Welcome!</td>
</tr>
<tr>
<td><code>for x in range(3):</code></td>
<td>0</td>
</tr>
<tr>
<td><code>print (x)</code></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>print (&quot;Goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(&quot;Welcome!&quot;)</code></td>
<td><code>Welcome!</code></td>
</tr>
<tr>
<td><code>for x in range(3):</code></td>
<td></td>
</tr>
<tr>
<td><code>print(x)</code></td>
<td>0</td>
</tr>
<tr>
<td><code>print(&quot;Goodbye!&quot;)</code></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Code</td>
<td>Output</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><code>print (&quot;Welcome!&quot;)</code></td>
<td>Welcome!</td>
</tr>
<tr>
<td><code>for x in range(3):</code></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td><code>print (x)</code></td>
<td></td>
</tr>
<tr>
<td><code>print (&quot;Goodbye!&quot;)</code></td>
<td>Goodbye!</td>
</tr>
</tbody>
</table>
Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>def hello():</td>
<td></td>
</tr>
<tr>
<td>print(&quot;Hi there!&quot;)</td>
<td></td>
</tr>
<tr>
<td>print(&quot;I'm a function!&quot;)</td>
<td></td>
</tr>
<tr>
<td>print(&quot;Good morning&quot;)</td>
<td></td>
</tr>
<tr>
<td>print(&quot;Welcome to class&quot;)</td>
<td></td>
</tr>
<tr>
<td>hello()</td>
<td></td>
</tr>
<tr>
<td>print(&quot;And now we're done.&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
# Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def hello():</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;Hi there!&quot;)</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;I'm a function!&quot;)</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;Good morning&quot;)</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;Welcome to class&quot;)</code></td>
<td></td>
</tr>
<tr>
<td><code>hello()</code></td>
<td></td>
</tr>
<tr>
<td><code>hello()</code></td>
<td></td>
</tr>
<tr>
<td><code>print(&quot;And now we're done.&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
## Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>def hello():</td>
<td>Good morning</td>
</tr>
<tr>
<td>print (&quot;Hi there!&quot;)</td>
<td></td>
</tr>
<tr>
<td>print (&quot;I'm a function!&quot;)</td>
<td></td>
</tr>
<tr>
<td>print (&quot;Good morning&quot;)</td>
<td></td>
</tr>
<tr>
<td>print (&quot;Welcome to class&quot;)</td>
<td></td>
</tr>
<tr>
<td>hello()</td>
<td></td>
</tr>
<tr>
<td>print (&quot;And now we're done.&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
Flow of Functions With Functions

Code

```python
def hello():
    print("Hi there!")
    print("I'm a function!")

print("Good morning")
print("Welcome to class")

hello()

print("And now we're done.")
```

Output

Good morning
Welcome to class
# Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>def hello():</td>
<td></td>
</tr>
<tr>
<td>print(&quot;Hi there!&quot;)</td>
<td>Good morning</td>
</tr>
<tr>
<td>print(&quot;I'm a function!&quot;)</td>
<td>Welcome to class</td>
</tr>
<tr>
<td>print(&quot;Good morning&quot;)</td>
<td></td>
</tr>
<tr>
<td>print(&quot;Welcome to class&quot;)</td>
<td></td>
</tr>
<tr>
<td>hello()</td>
<td></td>
</tr>
<tr>
<td>print(&quot;And now we're done.&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
</table>
| `def hello():
    print("Hi there!")
    print("I'm a function!")

    print("Good morning")
    print("Welcome to class")

    hello()

    print("And now we're done.")` | Good morning
Welcome to class |
Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
</table>
| ```python
def hello():
    print("Hi there!")
    print("I'm a function!")

    print("Good morning")
    print("Welcome to class")

    hello()

    print("And now we're done.")``` | Good morning  
Welcome to class  
Hi there!  
And now we're done. |


Flow of Functions With Functions

**Code**

```python
def hello():
    print("Hi there!")
    print("I'm a function!")

print("Good morning")
print("Welcome to class")

hello()

print("And now we're done.")
```

**Output**

Good morning
Welcome to class
Hi there!
I'm a function!

And now we're done.
# Flow of Functions With Functions

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
</table>
| def hello():
  print ("Hi there!")
  print ("I'm a function!")
print ("Good morning")
print ("Welcome to class")
hello()
print ("And now we're done.") | Good morning
Welcome to class
Hi there!
I'm a function! |
# Flow of Functions With Functions

## Code

```python
def hello:
    print("Hi there!")
    print("I'm a function!")

print("Good morning")
print("Welcome to class")

hello()

print("And now we're done.")
```

## Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
And now we're done.
```
Multiple Functions

def hello():
    print ("Hello there!")

def goodbye():
    print ("See ya!")

hello()
goodbye()
Multiple Functions

def main():
    print (“I have a message for you.”)
    message()
    print (“Goodbye!”)

def message():
    print (“The password is ‘foo’”)

main()
Write a program that prints the pattern to below using functions

```
*               *
*               *
*               *
*               *
*               *
*               *
*               *
*               *
*               *
*               *
```

```
Multiple Functions

Ask the user for the height for a rectangle
Then draw a rectangle of that height

```
***************
*             *
*             *
*             *
*             *
***************
```
Local Variables

Functions are like “mini programs”

You can create variables inside functions just as you would in your main program
def bugs():
    numbugs = int(input('How many bugs? '))
    print (numbugs)

bugs()
Local Variables

However, variables that are defined inside of a function are considered “local” to that function.

This means that they only exist within that function. Objects outside the “scope” of the function will not be able to access that variable.
def bugs():
    numbugs = int(input('How many bugs? '))
    print (numbugs)

bugs()
print (numbugs) # error! Variable numbugs
    # doesn’t exist in this scope!
Local Variables

Different functions can have their own local variables that use the same variable name.

These local variables will not overwrite one another since they exist in different “scopes.”
Local Variables

def newjersey():
    numbugs = 1000
    print ("NJ has", numbugs, "bugs")
def newyork():
    numbugs = 2000
    print ("NY has", numbugs, "bugs")

newjersey()
newyork()
Passing Arguments to a Function

Sometimes it’s useful to not only call a function but also send it one or more pieces of data as an argument. This process is identical to what you’ve been doing with the built-in functions we have studied so far.

```python
x = random.randint(1,5)  # send 2 integers
y = len('Craig')        # send 1 string
```
Passing Arguments to a Function

```python
def square(num):
    print (num**2)  # num assumes the value of the argument that is passed to the function (5)
square(5)
```
When passing arguments, you need to let your function know what kind of data it should expect in your function definition. You can do this by establishing a variable name in the function definition. This variable will be auto declared every time you call your function, and will assume the value of the argument passed to the function.
You can actually pass any number of arguments to a function. One way to do this is to pass in arguments “by position”.
Passing Multiple Arguments to a Function

def average(num1, num2, num3):
    sum = num1+num2+num3
    avg = sum / 3
    print (avg)

average(100,90,92)
Challenge

Write a “joke” generator that prints out a random knock knock joke.

Extension: Write a “drum roll” function that pauses the program for dramatic effect! Have the drum roll function accept a parameter that controls how long it should pause.
Argument Mechanics

When we pass an argument to a function in Python we are actually passing it’s “value” into the function, and not an actual variable.
def change_me(v):
    print ("function got:", v)
    v = 10
    print ("argument is now:", v)

myvar = 5
print ("starting with:", myvar)
change_me(myvar)
print ("ending with:", myvar)
Argument Mechanics

We call this behavior “passing by value”

We are essentially creating two copies of the data that is being passed – one that stays in the main program and one that is passed as an argument into our function.

This behavior allows us to set up a “one way” communication mechanism – we can send data into a function as an argument, but the function cannot communicate back by updating or changing the argument in any way (we will talk about how to communicate back to the caller in just a second!)
Global Variables

When you create a variable inside a function we say that the variable is “local” to that function.

This means that it can only be accessed by statements inside the function that created it.

When a variable is created outside all of your functions it is considered a “global variable”.

Global variables can be accessed by any statement in your program file, including by statements in any function.

All of the variables we have been creating so far in class have been global variables.
Global Variables

```python
name = 'Ada'

def showname():
    print ("Function:", name)

print ("Main program:", name)
showname()
```
Global Variables

If you want to be able to change a global variable inside of a function you must first tell Python that you wish to do this using the “global” keyword inside your function.
Global Variables

Global variables can make debugging difficult

Functions that use global variables are generally dependent on those variables, making your code less portable

With that said, there are many situations where using global variables makes a lot of sense.
Challenge

Write a very brief “choose your own adventure” style game using functions

Reference:  http://thcnet.net/zork/
Value Returning Functions

Value returning functions are functions that return a value to the part of the program that initiated the function call.

They are almost identical to the type of functions we have been writing so far, but they have the added ability to send back information at the end of the function call.

We have secretly been using these all semester!

```python
somestring = input("Tell me your name")
somenumber = random.randint(1,5)
```
You use almost the same syntax for writing a value returning function as you would for writing a normal function.

The only difference is that you need to include a “return” statement in your function to tell Python that you intend to return a value to the calling program.

The return statement causes a function to end immediately. It’s like the break statement for a loop.

A function will not proceed past its return statement once encountered. Control of the program is returned back to the caller.
Value Returning Functions

def myfunction(arg1, arg2):
    statement
    statement
    ...
    statement
    return expression

# call the function
returnvalue = myfunction(10, 50)
Challenge

Write a function that takes two age values as integers, adds them up and returns the result as an integer.
Challenge

Prompt the use for an item price (using a function)
Apply a 20% discount to the price (using a function)
Print the starting price and the discounted price

Extension:

Don’t accept price values less than $0.05 – repeatedly ask the user to enter new data if this happens

Repeat the discounting process until the user elects to stop entering data
IPO Notation

As you start writing more advanced functions you should think about documenting them based on their Input, Processing and Output (IPO)

Example:

```python
# function: add_ages
# input: age1 (integer), age2 (integer)
# processing: combines the two integers
# output: returns the combined value
def add_ages(age1, age2):
    sum = age1+age2
    return sum
```
Functions can also return multiple values using the following syntax:

```python
def testfunction():
    x = 5
    y = 10
    return x, y

p, q = testfunction()
```
Write a function that simulates the rolling of two dice

The function should accept a size parameter (i.e. how many sides does each die have)

The function should return two values which represent the result of each roll

Extension:

Make sure both numbers that you return are different (i.e. you can’t roll doubles or snake eyes)

Build in an argument that lets you specify whether you want to enforce the no doubles policy
All programming languages come pre-packaged with a standard library of functions that are designed to make your job as a programmer easier.

Some of these functions are built right into the “core” of Python (print, input, range, etc).

Other more specialized functions are stored in a series of files called “modules” that Python can access upon request by using the “import” statement.

```python
import random
import time
```
On a Mac you can actually see these files here:

/Library/Frameworks/Python.framework/
Version/3.2/lib/python3.2/

To see information about a module, you can do the following in IDLE:

```python
import modulename
help(modulename)
```
Modules

The import statement tells Python to load the functions that exist within a specific module into memory and make them available in your code.

Because you don’t see the inner workings of a function inside a module we sometimes call them “black boxes.”

A “black box” describes a mechanism that accepts input, performs an operation that can’t be seen using that input, and produces some kind of output.
Modules
We call functions that exist within a module by using “dot notation” to tell Python to run a function that exists in that module.

Example:

```python
num = random.randint(1,5)
```
You can list the functions that exist in a particular module by using the `help()` function.

The `help()` function takes one argument (a string that represents the name of the module) and returns the user manual for that module.
You can easily create your own modules that you can populate with your own functions. Here’s how:

Create a new python script (i.e. “myfunctions.py”)

Place your function definitions in this script

Create a second python script (i.e. “myprogram.py”)

Modules
Import your function module using the import statement:

    import myfunctions

Call your functions using dot notation

    myfunctions.function1()
    myfunctions.dosomethingelse()
Challenge

Create a module called “geometry_helper”

Write two functions in this module:

  Area of circle
  Perimeter of circle

Each of these functions will accept one argument (a radius) and will print out the result to the user.
Additional Functions inside the Random()

Floating point random #'s

```python
num = random.random()        # generates a float
# between 0 and 1
num = random.uniform(1,10)  # generates a float
# between 1 and 10
```
As we mentioned in an earlier class, the computer does not have the ability to generate a truly random #

It uses a series of complicated mathematical formulas that are based on a known value (usually the system clock)

The seed function in the random module allows you to specify the “seed” value for the random #’s that the various random number functions generate

You can seed the random # generator with whatever value you want – and you can even reproduce a random range this way!
next steps:

begin “Self Paced Learning Module # 7”