Working with Functions in Python

Introduction to Programming - Python
Notes on Exams
Converting formulas to code:

$$xy^2 \neq (xy)^2$$

```python
>>> 10*5**2
250
>>> (10*5)**2
2500
```

$$xy^2 \rightarrow x*y**2 \text{ or } x*(y**2)$$
products += (prod, end = ' ')  

This is a syntax error. The "end" operand only works for the print statement, it's not a general purpose string thing. Instead, do this:

products += prod + " "
Loop Conditions

- There was a lot of redundancy in loop code. E.g.:
  ```python
top_loop = False
while not stop_loop:
    name = input("Enter product")
    if (name == 'end'):
        stop_loop = True
        break
```

- Don't use `continue` or `break` if you don't need to.

- If using boolean try to use the simplest form, e.g. "keep_going" makes more sense than "stop_loop"

- Don't use a boolean flag and break -- either one will work.

- Better to use neither, and instead use loop condition to see if the input value was the sentinel ("end").
For Loop Review
The "for" loop

"for" keyword starts the loop
"in" keyword
list of items to iterate over

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

indentation
statements to be executed

target variable
The “for” loop

- 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.
The “for” loop

```python
for c in [1, 2, 3, 4]:
    print(c)
```

1
2
3
4
## The `range()` function

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

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

Examples:

```python
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]
```
Programming Challenge: Multiplication Table

- Write a program that calculates the multiplication tables listed to the right

- 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>
Functions
Mathematical Functions

- In math you may have seen functions like this:

\[ f(x) = x^2 \]

\[ f(x, y) = \sqrt{x^2 + y^2} \]

\[ f : X \rightarrow Y \]

- This allows us to talk about properties of functions without having to be concerned with the specific formula.

- And to use functions without having to write the formula out in full each time.
Programming Functions

- A function is a group of statements that perform a specific task

- We can use a function without needing to know the details of how it does what it does

- 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
Functions

Functions allow us to break the program down into subtasks, allowing us to “divide and conquer” a programming problem.

They help make code easier to read and understand:
- Code that is reused frequently should be put in a function, to avoid redundancy and to simplify maintenance.
- Break complex code up into functional blocks to make code cleaner.
Defining Functions

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

# call the function
myfunction()
Some notes on 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, the “control” of the program has been transferred to the function. When it completes, it "returns" control to the code from where it was called.

- 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
Flow of Execution

Code

```python
print("Welcome!")
for x in range(3):
    print(x)
print("Goodbye!")
```

Output
### Flow of Execution

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
</table>
| ```python
def main()
    print("Welcome!")
    for x in range(3):
        print(x)
    print("Goodbye!")
def main()
``` | Welcome! |
Flow of Execution

**Code**

```python
print("Welcome!")

for x in range(3):
    print(x)

print("Goodbye!")
```

**Output**

Welcome!

```python

```
### Flow of Execution

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

**Code**

```python
print("Welcome!")
for x in range(3):
    print(x)
print("Goodbye!")
```

**Output**

Welcome!
0
1
Flow of Execution

**Code**

```python
print("Welcome!")
for x in range(3):
    print(x)
print("Goodbye!")
```

**Output**
```
Welcome!
0
1
```
Flow of Execution

Code

```python
print ("Welcome!")
for x in range(3):
    print (x)
print ("Goodbye!")
```

Output

Welcome!
0
1
2
<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><code>print(&quot;Goodbye!&quot;)</code></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Goodbye!</td>
</tr>
</tbody>
</table>
Flow of Execution – With Functions
Flow of Execution 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
Flow of Execution 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**
Flow of Execution with Functions

Code

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
Flow of Execution with Functions

Code

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 Execution 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

And now we're done.
Flow of Execution 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 Execution with Functions

Code

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!
Flow of Execution 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!
```
Flow of Execution 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 Execution with Functions

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

- Functions are like “mini programs”

- You can create variables inside functions just as you would in your main program

- 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

- Different functions can have their own local variables that use the same variable name, and they will not conflict with each other.
Variable Scope

- **Scope** (n): extent or range of view, outlook, application, operation, effectiveness

- All variables are defined within a scope, and only exist within that scope.

- Up until now, all of our variables have been *global*: their scope was the entire program.

- A *local* variable exists only within a given block. In most cases, we will be talking about local variables in functions.
def bugs():
    numbugs = int(input('How many bugs? '))
    print (numbugs)

bugs()
Local Variables

def bugs():
    numbugs = int(input('How many bugs? '))
    print(numbugs)

bugs()

print(numbugs)  # error! Variable numbugs
    # doesn't exist in this scope!
def newjersey():
    numbugs = 1000
    print ("NJ has", numbugs, "bugs")

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

newjersey()
newyork()
Global Variables
Global Variables

- 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

name = 'Craig'

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 declare it as “global” inside your function.

```python
name = 'Craig'

def showname():
    global name
    print("Function 1:", name)
    name = 'John'
    print("Function 2:", name)

print("Main program 1:", name)
showname()
print("Main program 2:", name)
```
Global variables can make code harder to understand and debug.

Functions that use global variables are generally dependent on those variables, making your code less portable and harder to maintain.

There are some situations where using global variables makes a lot of sense, but in general they should be avoided.
Passing Arguments to a Function
Passing Arguments to a Function

- Often you want to be able to send data to a function when you call it.

- This process is identical to what you’ve been doing with the built-in functions we have studied so far
  - \( x = \text{random.randint}(1,5) \)  # send 2 integers
  - \( y = \text{len}(\text{‘Craig’}) \)  # send 1 string
def square(num):
    print (num**2)  # num assumes the value of the # argument that is passed to # the function (5)

square(5)
Passing Arguments to a Function

- When defining your function, you must specify that it will expect to receive data.

- You do this by declaring 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. It will be local to this function.

- We call these variables "parameters".

- A function can have multiple parameters. Arguments are assigned to parameters based on their position in the function call.
def average(num1, num2, num3):
    sum = num1+num2+num3
    avg = sum / 3
    print (avg)

average(100,90,92)
Keyword Arguments

- Usually, arguments are assigned to parameters based on their position in the function call.

- Arguments can also be passed to specific parameters by specifying that parameter in the function call.

```python
def myfunc(x, y, z):
    if z>x:
        return y
    else:
        return x

myfunc(5,3,z=2)
```

- You've seen this already with the print function:

```python
print("some","string","data",sep=":",end=" ")
```
Programming Challenge: Distance Formula

- Write a program that asks the user to enter two points on a 2D plane (i.e. enter X1 & Y1, enter X2 & Y2)

- Compute the distance between those points using a function.

- Continually ask the user for numbers until they wish to quit

\[ d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]
Argument Mechanics
Argument Mechanics

- Function parameters are local to that function.

- When we pass an argument to a function in Python, the value of that argument gets copied into the parameter variable.

- Python functions are "call by value", which means we are actually sending the value to the function, and not an actual variable.

- 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
Argument Mechanics

```python
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)
```
Returning Data From Functions
Functions: Returning Results

- Values can be returned by functions

- We have been using these all semester:
  - somestring = input("Tell me your name")
  - somenumber = random.randint(1,5)

- Return data using a “return” statement.

- 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)
Programming Challenge: Combined Age

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

- 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
Programming Challenge: Distance Formula

- Write a program that asks the user to enter a point on a 2D plane (i.e. enter X & Y)
- Compute the distance between that point and the origin of the plane (0,0)
- If the distance is < 10, tell them they hit a bullseye. If the distance is >= 10, they missed!

\[
d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
\]
IPO Notation

- You should document your functions by describing their Input, Processing and Output (IPO)

- This allows other people to understand what your function does and how to use it, without having to understand the code.

- 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
```
Returning Boolean Values

- Boolean values can drastically simplify decision and repetition structures

- Example:
  - Write a program that asks the user for a part number
  - Only accept part #’s that are on the following list:
    - 100
    - 200
    - 300
    - 400
    - 500
  - Continually prompt the user for a part # until they enter a correct value
Functions can also return multiple values using the following syntax:

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

```python
p, q = testfunction()
```
Programming Challenge: Two Dice

- 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
Programming Challenge: Feet to Inches

- Write a function that converts feet to inches. It should accept a number of feet as an argument and return the equivalent number of inches.
Programming Challenge: Maximum of two values

- Write a function named “maximum” that accepts two integer values and returns the one that is the greater of the two to the calling program.