Python II
OOP

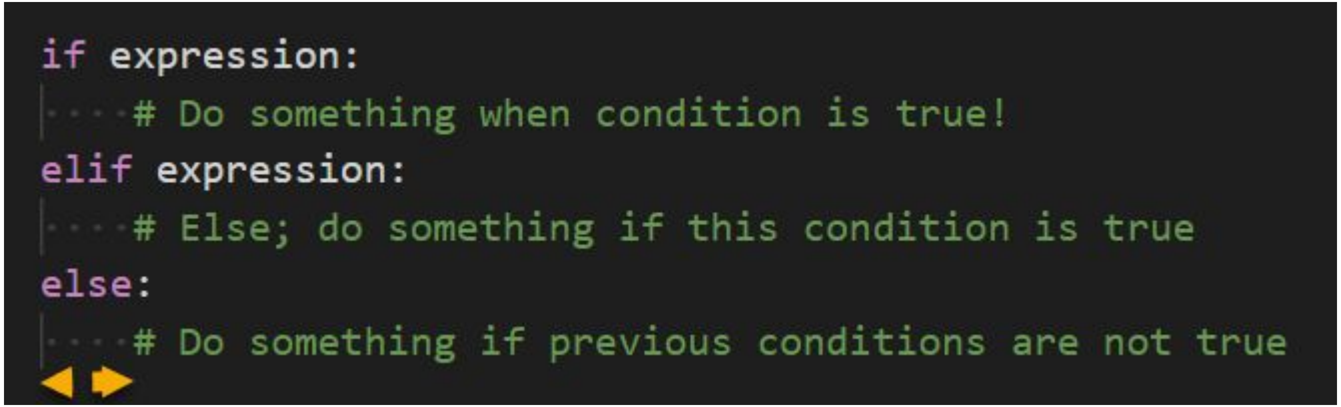
~/IFT383/mod-6

Agenda

- Control structures
  - if
  - loops
    - while
    - for
  - List comprehensions
- Functions

if statement

- Control statements in Python do not encapsulate code blocks in curly brackets like Awk and Perl
  - Python instead use indentation to distinguish code segments
  - Yse consistent indentation; tabs or spaces; avoid using both in the same file
- The if statement follows the general form that we have seen before
  - The keyword if, followed by a condition and a colon “:”

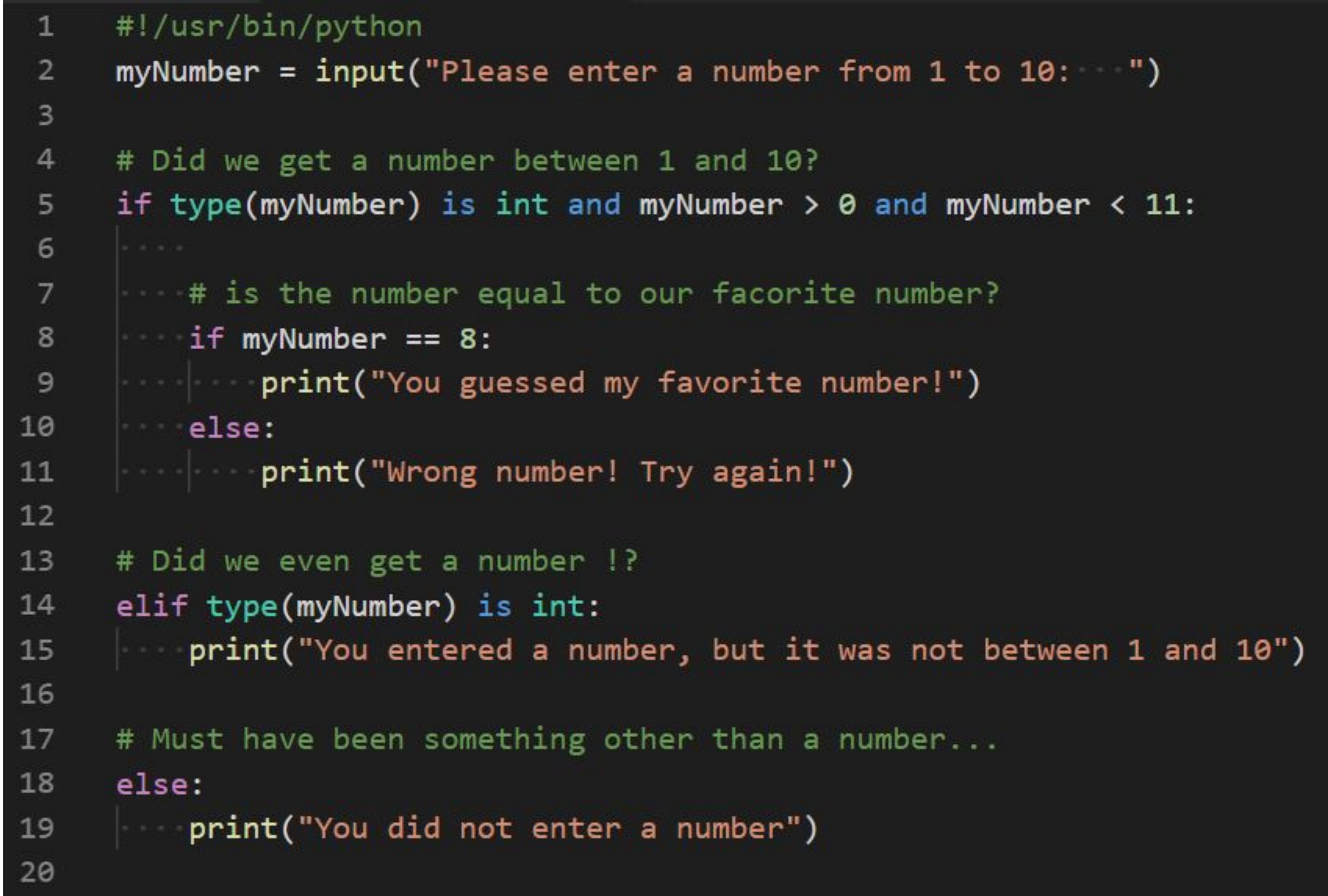


```
if expression:
    # Do something when condition is true!
elif expression:
    # Else; do something if this condition is true
else:
    # Do something if previous conditions are not true
```

Conditions

- Conditions are **boolean** expressions; either return a **true** or **false** value
  - Relational operators
    - ==, != <. >, <=, >=, in, not in
    - Can be used for strings, numbers and other data types
  - Logical operators
    - and, or, not
      - and replaces &&
      - or replaces ||
      - not replaces !

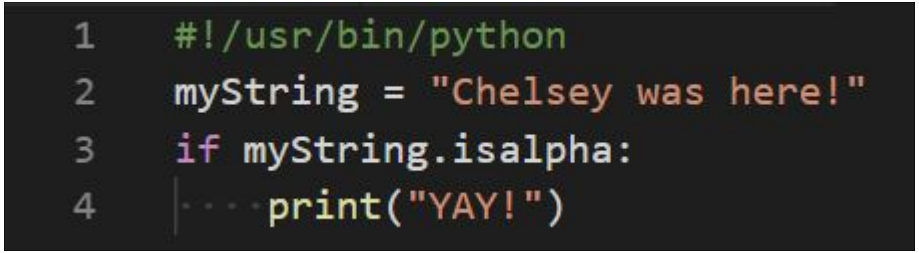
If example (guessNumber.py)



```python
#!/usr/bin/python
myNumber = input("Please enter a number from 1 to 10:   ")

# Did we get a number between 1 and 10?
if type(myNumber) is int and myNumber > 0 and myNumber < 11:
    
    # is the number equal to our facorite number?
    if myNumber == 8:
        print("You guessed my favorite number!")
    else:
        print("Wrong number! Try again!")

# Did we even get a number !?
elif type(myNumber) is int:
    print("You entered a number, but it was not between 1 and 10")

# Must have been something other than a number...
else:
    print("You did not enter a number")

```

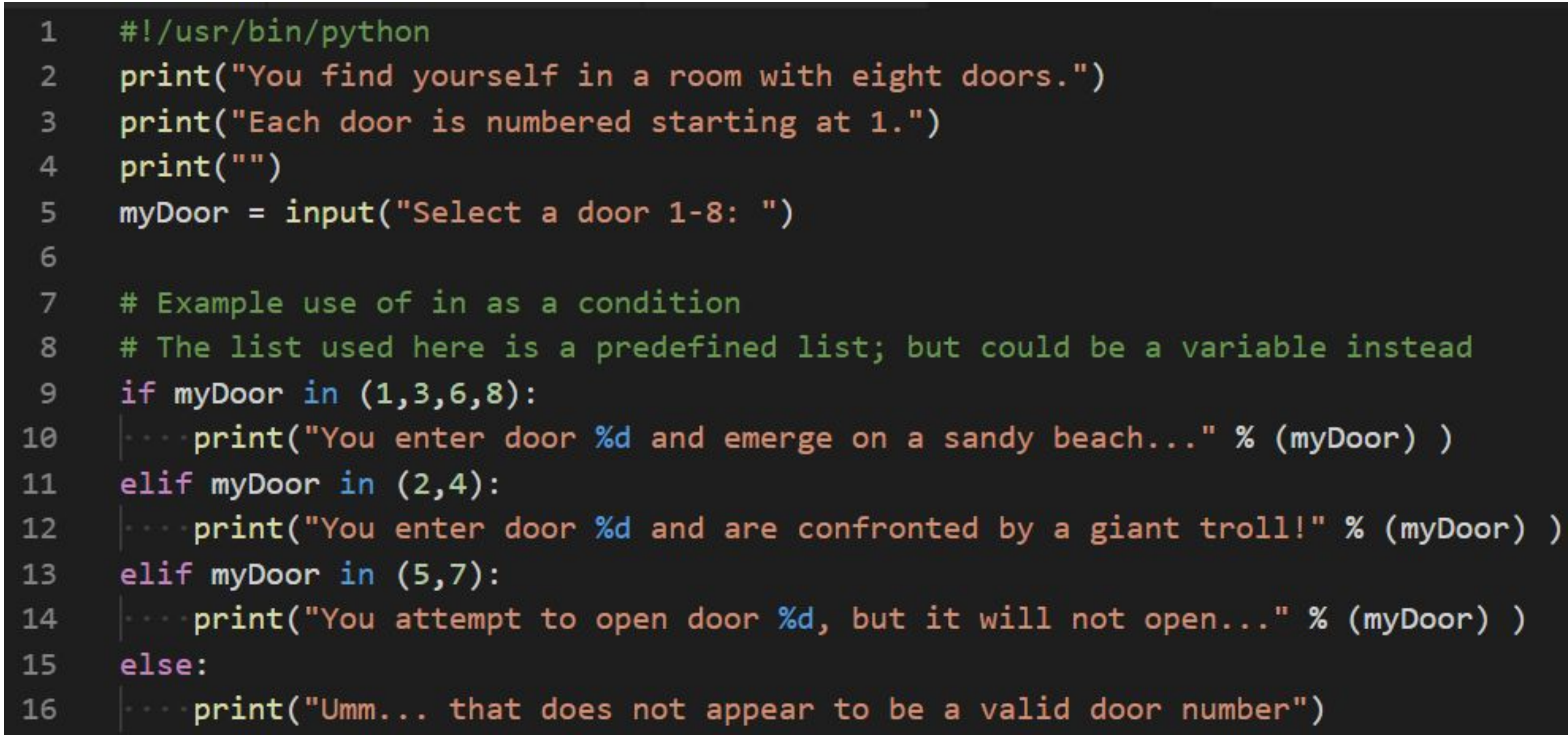
String conditions

- Strings have additional built-in functions that return **boolean** values
- We can use these as part of our conditions
  - myString.startswith()
  - myString.endswith()
  - myString.isDigit()
  - myString.isAlpha()
- Example;



```
#!/usr/bin/python
myString = "Chelsey was here!"
if myString.isalpha:
    print("YAY!")
```

Example (doors.py)
Conditions with **in**

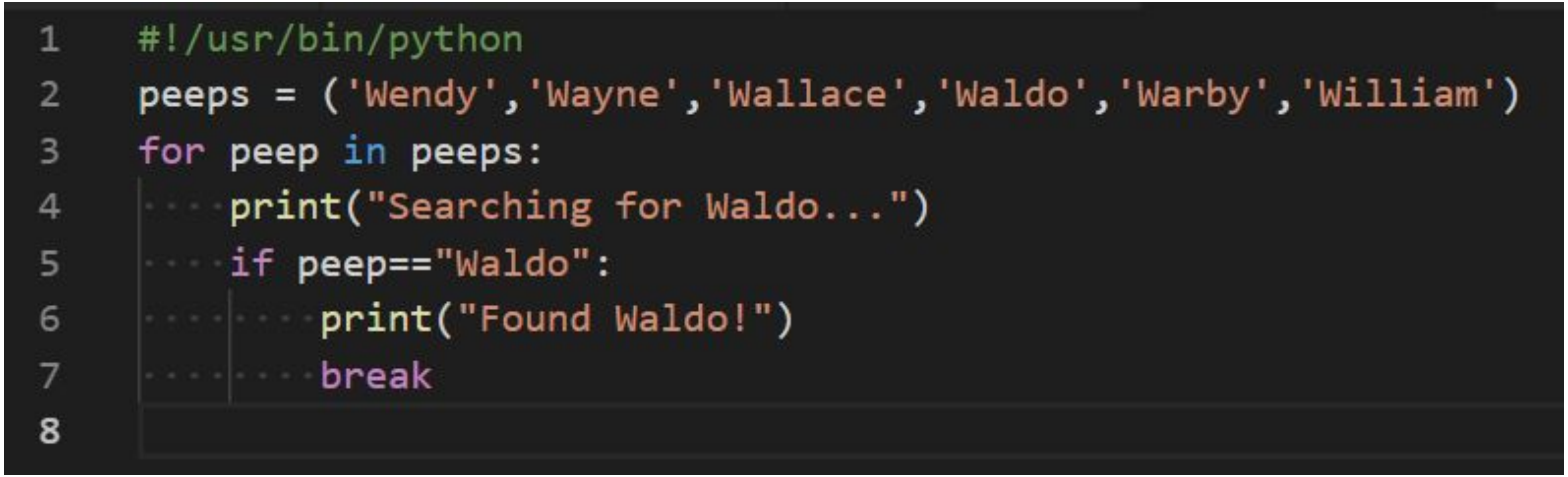


```python
#!/usr/bin/python
print("You find yourself in a room with eight doors.")
print("Each door is numbered starting at 1.")
print("")
myDoor = input("Select a door 1-8: ")

# Example use of in as a condition
# The list used here is a predefined list; but could be a variable instead
if myDoor in (1,3,6,8):
    print("You enter door %d and emerge on a sandy beach..." % (myDoor) )
elif myDoor in (2,4):
    print("You enter door %d and are confronted by a giant troll!" % (myDoor) )
elif myDoor in (5,7):
    print("You attempt to open door %d, but it will not open..." % (myDoor) )
else:
    print("Umm... that does not appear to be a valid door number")
```

Loops

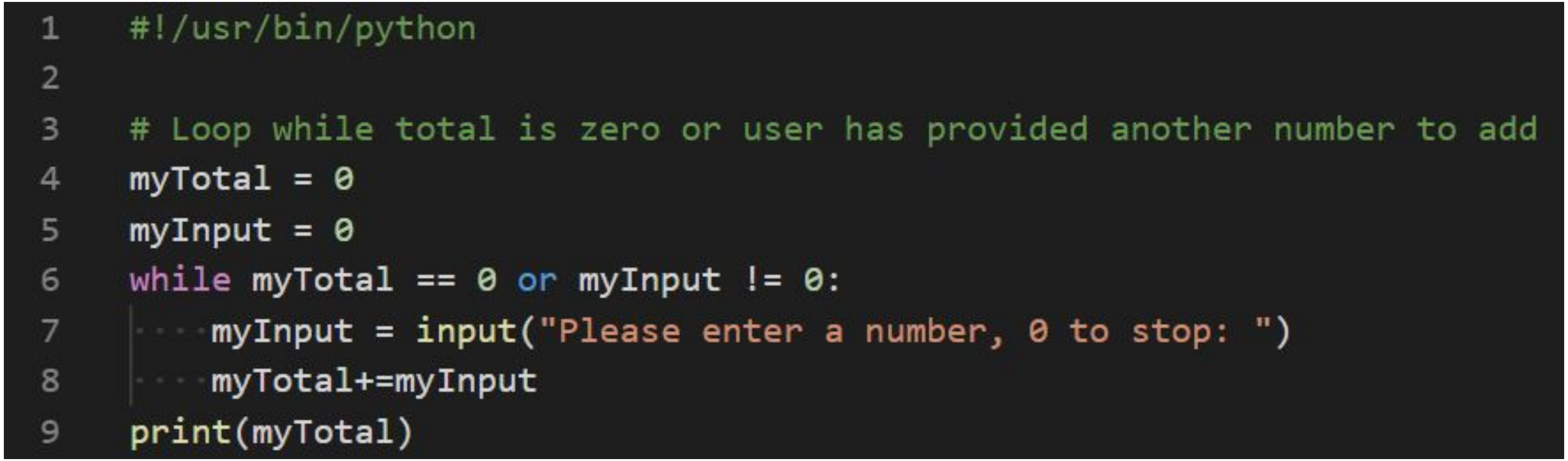
- **while** loop
    - Continues to loop **while** a condition evaluates to **true**
    - The **break** keyword can be used to prematurely exit a loop
- for loop
    - Iterates through any **iterable** object, such as a list, dictionary or tuple
    - The **break** keyword is also applicable to for loops
        - Example; if you are searching for something, you can exit the loop once you found what you are looking for

for loop example (waldo.py)



```python
#!/usr/bin/python
peeps = ('Wendy','Wayne','Wallace','Waldo','Warby','William')
for peep in peeps:
    print("Searching for Waldo...")
    if peep=="Waldo":
        print("Found Waldo!")
        break

```

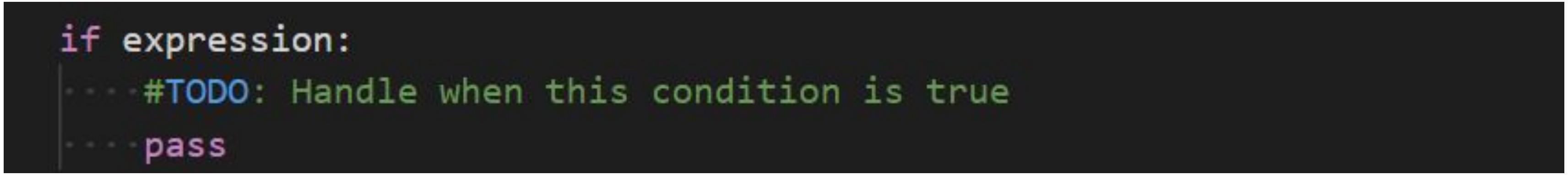
While loop example (addNumbers.py)



```python
#!/usr/bin/python

# Loop while total is zero or user has provided another number to add
myTotal = 0
myInput = 0
while myTotal == 0 or myInput != 0:
    myInput = input("Please enter a number, 0 to stop: ")
    myTotal+=myInput
print(myTotal)
```

pass statement

- Provides a placeholder where code will go in the future
- Prevents compilation errors when Python expects an indented statement
- Does absolutely nothing; siminal to **noop** or **yield** in other languages

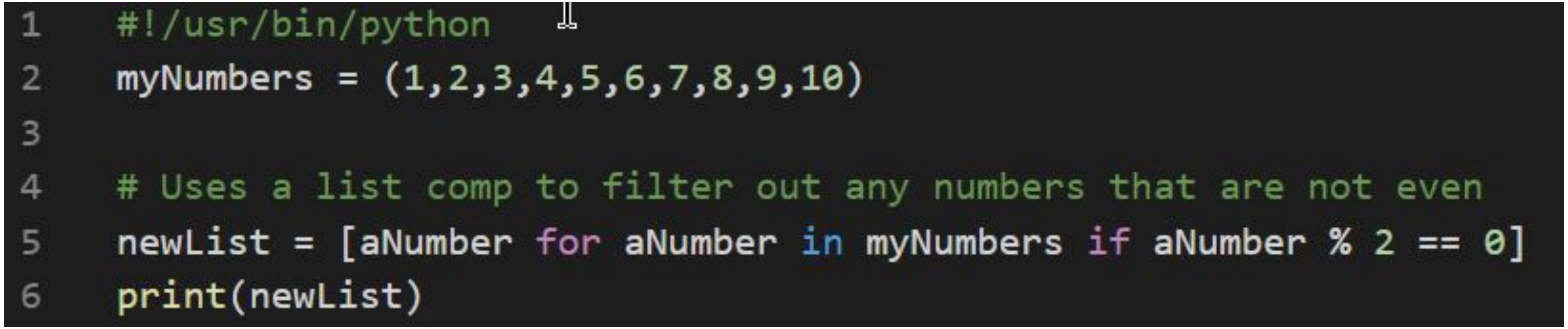


```
if expression:
    #TODO: Handle when this condition is true
    pass
```

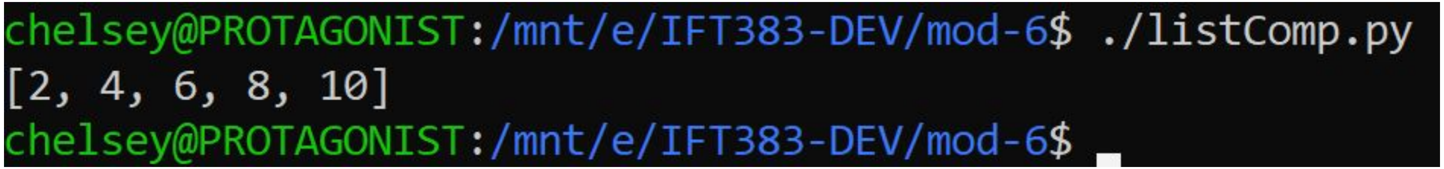
List Comprehensions

- A mechanism for filtering or modifying a list or dictionary object
- Syntax;
  - [myVar**2 for myVar in myList]
    - squares each element in myList
    - returns a new list containing the modified elements
  - [aNumber for aNumber in myNumbers if aNumber > 10]
    - Creates a new list based on myNumbers
    - The new list will only contain elements from myNumbers that are greater than 10

List Comp example (listComp.py)



```python
#!/usr/bin/python
myNumbers = (1,2,3,4,5,6,7,8,9,10)

# Uses a list comp to filter out any numbers that are not even
newList = [aNumber for aNumber in myNumbers if aNumber % 2 == 0]
print(newList)
```



```
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$ ./listComp.py
[2, 4, 6, 8, 10]
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$
```

Functions

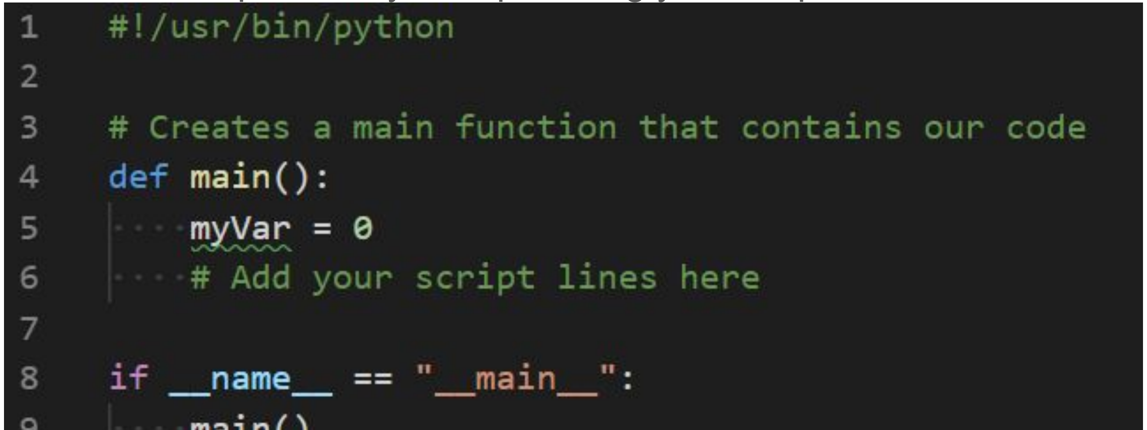
Introduction to Python functions

- As we have seen in other languages; functions (subroutines in Perl) provide a container for code that allows it to be reused throughout our script
- The **def** keyword is used to define a function
- The function will return the value provided to the **return** keyword
- Syntax;

```
def functionName (arg1, arg2, argN):
        return arg1 + arg2
```

function variable scope

- In the previous module; we briefly mentioned that Python is a scoped language
- This means that variables declared within a function are **scoped** to that function
- Minimizing the number of **global** variables used in your Python program is generally considered a best practice
    - This can be accomplished by encapsulating your script into one function

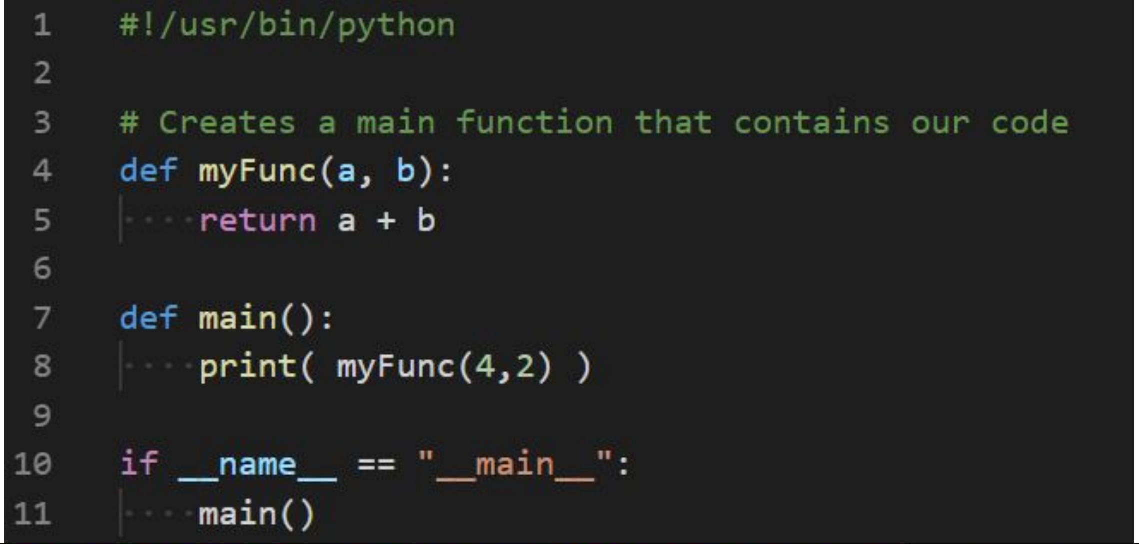


```
#!/usr/bin/python

# Creates a main function that contains our code
def main():
    myVar = 0
    # Add your script lines here

if __name__ == "__main__":
    main()
```

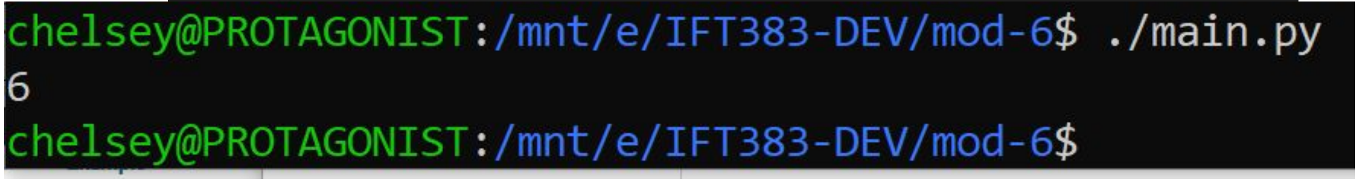
if __name__ == "__main__":

- When your script is executed from the command line; it is automatically given a value in the __name__ variable of __main__
- When your script is **imported** into another script, the __name__ variable will by default contain the name of your script file
- We can use __name__ to determine if our script is running as part of an import, or directly invoked
- This pattern causes Python to work similarly to other languages such as Java that call a main function when an program starts

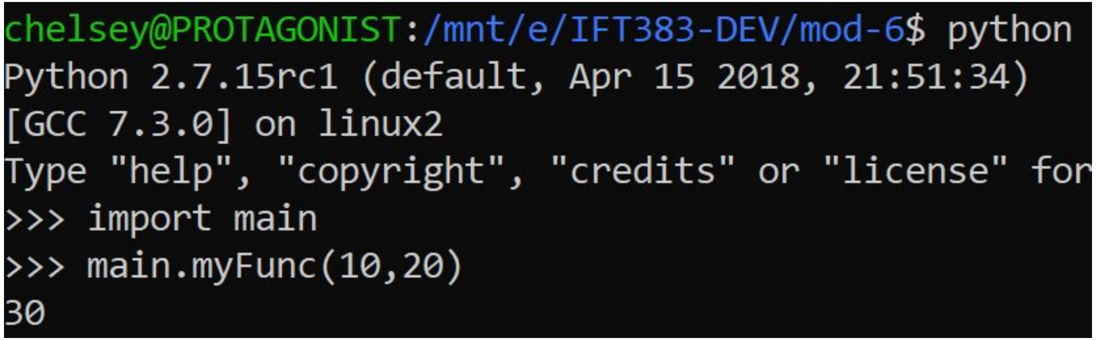
execution example (main.py)



```python
#!/usr/bin/python

# Creates a main function that contains our code
def myFunc(a, b):
    return a + b

def main():
    print( myFunc(4,2) )

if __name__ == "__main__":
    main()
```



```
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$ ./main.py
6
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$
```



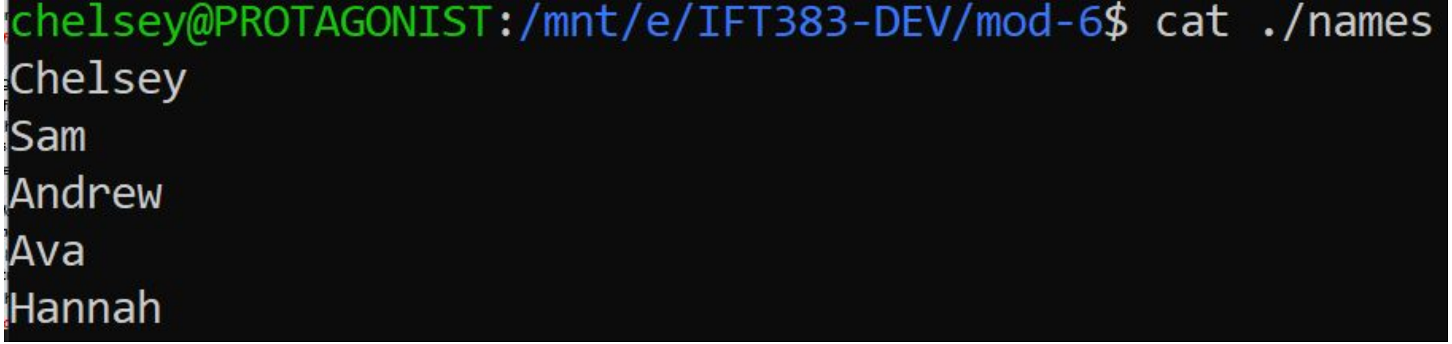
```
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$ python
Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for
>>> import main
>>> main.myFunc(10,20)
30
```

File I/O (Input/Output)

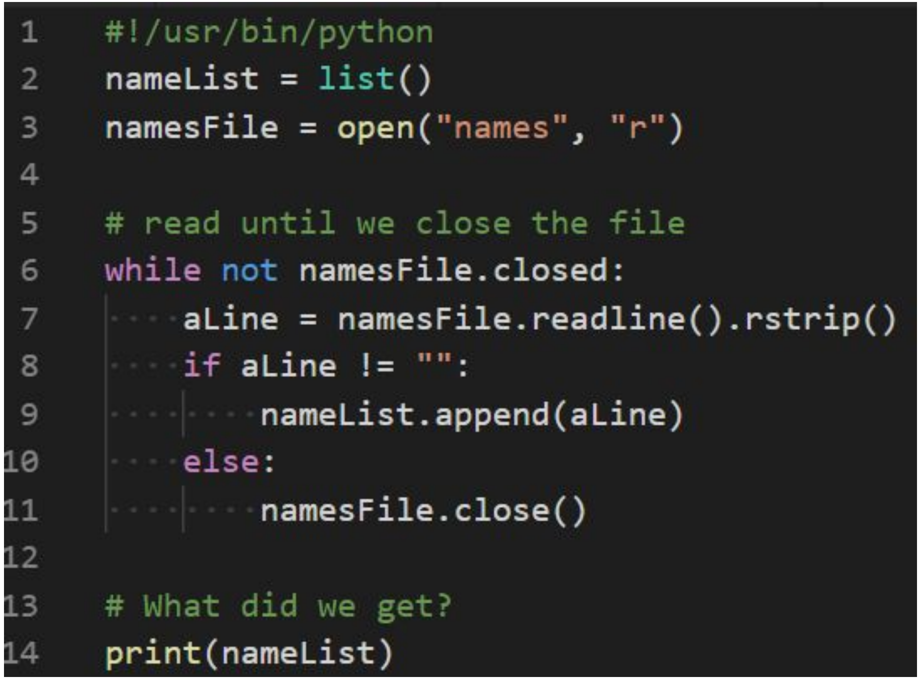
Reading from a file

- the **open** function will open a file and provide a reference to that file as an object
- That object has a number of functions, including; read(), readline() and readlines()
- Syntax;
  - fileObject = open(“file”, “mode”)
    - fileObject is where the reference to the file will be stored
    - “file” is the path or name of the file to be opened
    - “mode” tells open what we plan to do with the file
      - “r” read
      - “w” write
      - “a” append

readline() example (readline.py)

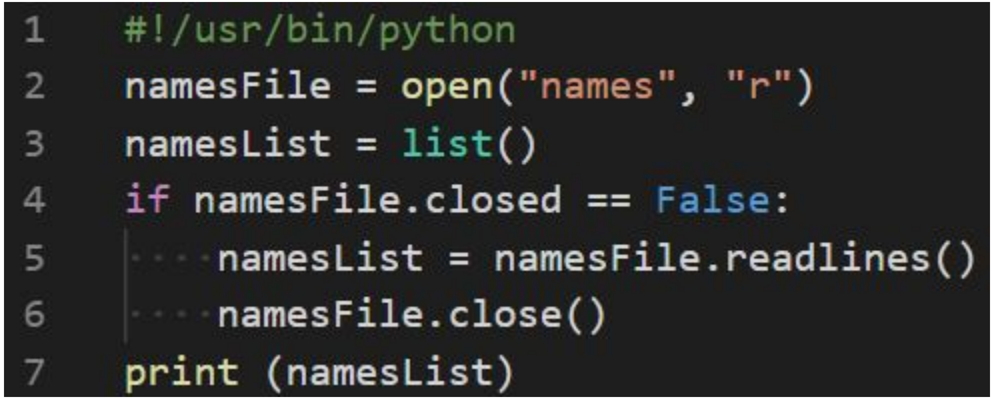


```
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$ cat ./names
Chelsey
Sam
Andrew
Ava
Hannah
```

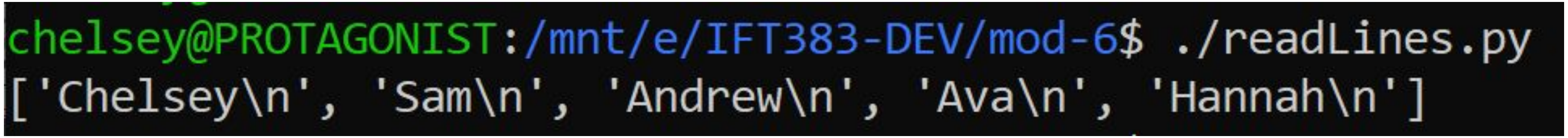


```python
#!/usr/bin/python
nameList = list()
namesFile = open("names", "r")

# read until we close the file
while not namesFile.closed:
    aLine = namesFile.readline().rstrip()
    if aLine != "":
        nameList.append(aLine)
    else:
        namesFile.close()

# What did we get?
print(nameList)
```

readlines() example (readl .py)



```python
#!/usr/bin/python
namesFile = open("names", "r")
namesList = list()
if namesFile.closed == False:
    namesList = namesFile.readlines()
    namesFile.close()
print (namesList)
```

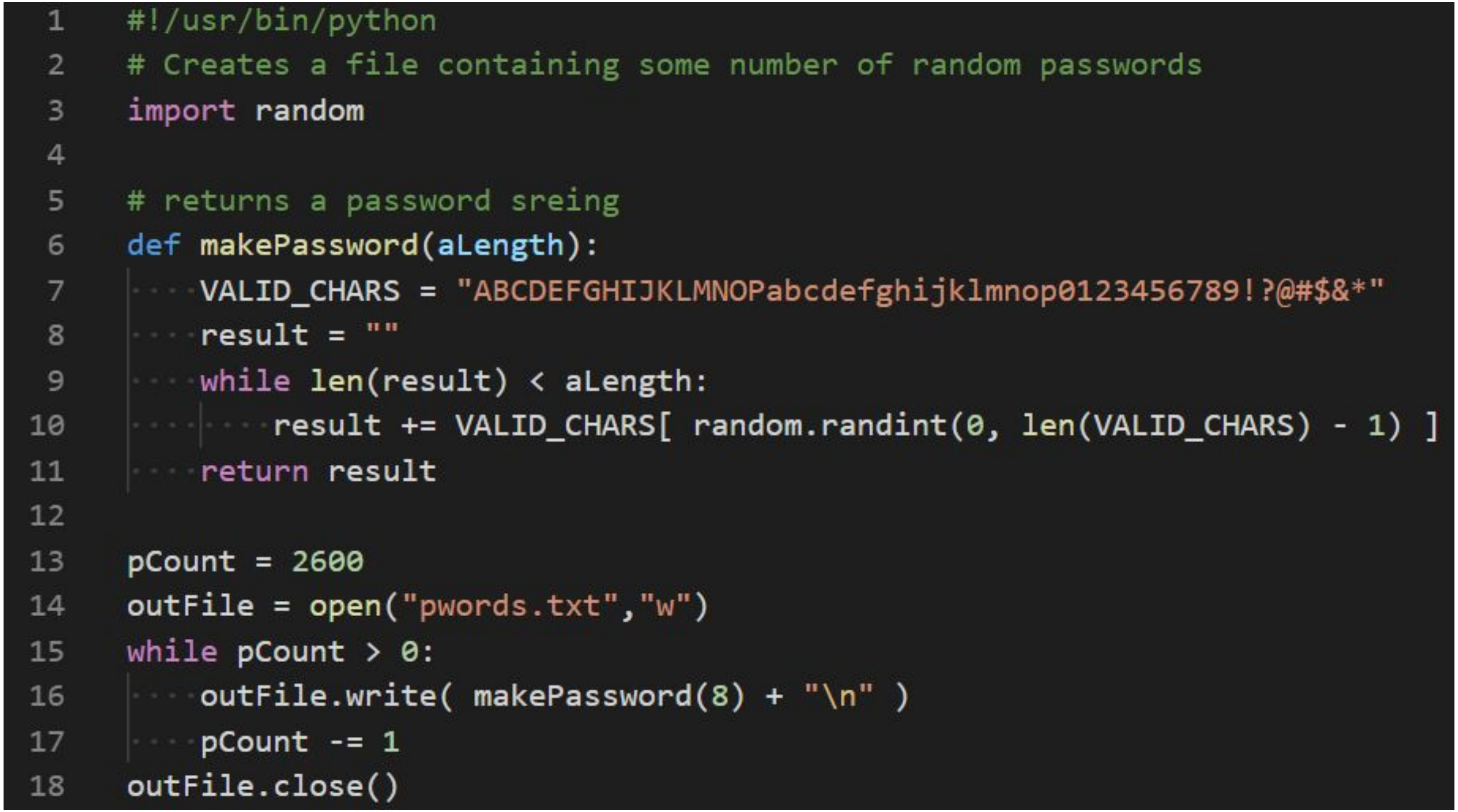


```
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$ ./readLines.py
['Chelsey\n', 'Sam\n', 'Andrew\n', 'Ava\n', 'Hannah\n']
```

Writing to a file

- Use the x, w or a modes when calling open()
  - Example: myFile = open(“results.csv”, “w”)
    - Opens a file for writing, fails if file does not exist
  - Example: myFile = open(“research.csv”, “a”)
    - Open file for writing; append data rather than overwriting
- Commonly used methods
  - write(“string”) - write string to the file (does not append newline)
  - writelines(list) - write a series of things to the file (no newlines either!)
  - close() - close the file so the OS knows you are done with it (IMPORTANT!)

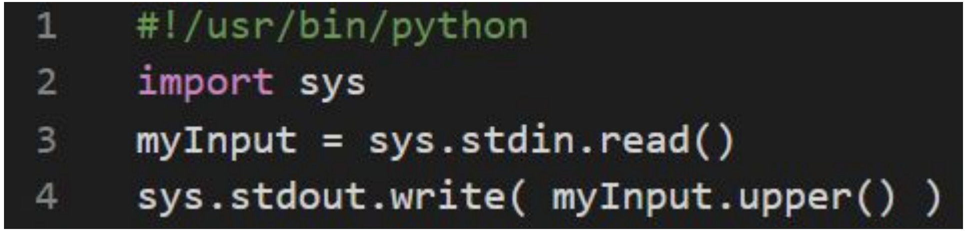
File output example (makePasswords.py)



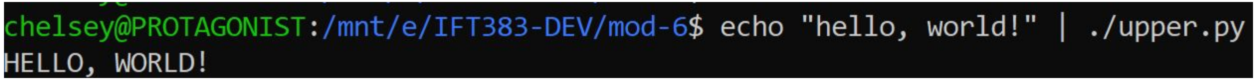
```python
#!/usr/bin/python
# Creates a file containing some number of random passwords
import random

# returns a password sreing
def makePassword(aLength):
    VALID_CHARS = "ABCDEFGHIJKLMNOPabcdefghijklmnop0123456789!?@#$&*"
    result = ""
    while len(result) < aLength:
        result += VALID_CHARS[ random.randint(0, len(VALID_CHARS) - 1) ]
    return result

pCount = 2600
outFile = open("pwords.txt","w")
while pCount > 0:
    outFile.write( makePassword(8) + "\n" )
    pCount -= 1
outFile.close()
```

Standard in, standard out and standard error

- Similar to accessing files; you can also read and write to UNIX streams
- The **sys** module contains objects for all of these streams
- Example; (reverse.py)



```python
#!/usr/bin/python
import sys
myInput = sys.stdin.read()
sys.stdout.write( myInput.upper() )
```



```
chelsey@PROTAGONIST:/mnt/e/IFT383-DEV/mod-6$ echo "hello, world!" | ./upper.py
HELLO, WORLD!
```