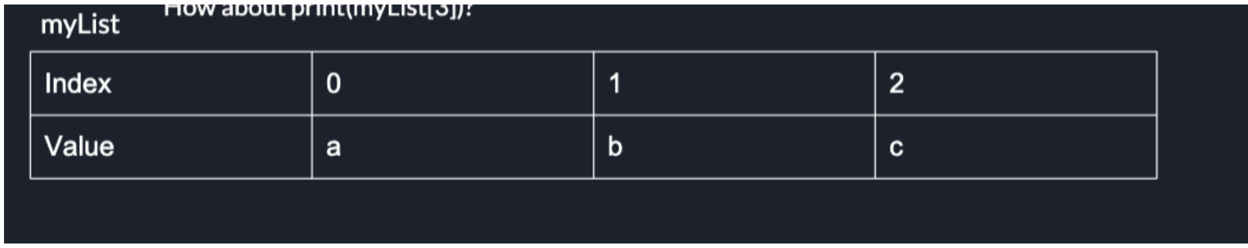
**9/15 Data Types**

- Data Type is the classification of data items
  - Integer
  - Float
  - String
  - List
  - Dictionary
  - Bool
  - Tuples
- print(): will print to the screen; is a function
- Integer (int)
  - A whole number (ex. 2)
- Float (float)
  - Numeric with multiple decimal places
- String (str)
  - Single character or collection of characters (ex. A = ‘ok’, “ok”, “‘ok”’)
- List (list)
  - An array or sequence of ordered data that is <u>mutable</u> (can change) and of different types (ex. myList = [‘a’,’b’,’c’] or [‘a’,’ok’,14]
  - Index: A value’s position within a sequence
  - Index values: ex. print(myList[0]) would return the string a; print(myList[0:2]) print all in between, [0:] would be all
  - 

    How about print(myList[3])?

    myList

    | Index | 0 | 1 | 2 |
    |---|---|---|---|
    | Value | a | b | c |
- Dictionary *{* (dict)
  - An array or sequence of data that is <u>immutable</u> (can’t be changed after set)and can be of different types
  - Heroes = {‘Peter’:’Spider-Man’} {‘Eddie’:’Venom’} {‘Miles’:’Spider-Man’} - Element
  - Key↑ Value↑
  - print(heroes[peter]) would print “spider-man”
- Tuples *[*
  - An ordered collection that is <u>immutable</u>
  - tupleFun = (“wow”, “whoa” “ok”)
  - print(tupleFun[number or number range ex 0 or 0:1])
- Boolean (bool)
  - True or false statement
  - Ex. a = true, print(a) will return true
- Operators
  - Arithmetic: =,-,*,/,%, etc.
  - Comparison (relational) operators: ==, !=, >>, <<, >=, <=

    - Assignment operators: =, +-, -=
    - Logical operators: and, or, not
    - Identity Operators: is, is not

**Variables**

- A variable is assigning a location to store a chosen value that may be invoked to call on it
    - Ex. number = 1, so print(number) -> 1
- Consider this:
    - firstName = ‘Jack’
    - lastName = ‘Fallon’
    - fullName = firstName + -- + lastName
- Or
    - userInput = input(“what is your salary?”)
    - #assume a 26 week pay period and 20% tax
    - weekly = (userInput/26)
    - monthly = (weekly *4)
    - print(‘So your gross pay is’ + userInput + ‘Each month you make + monthly’ + ‘,biweekly you make + (weekly *2) +’, and weekly, you make’ + weekly + ‘. But federal tax is 20% so you actually make’ + (userInput - (userInput*.2)yearly)
- Variable Syntax
    - camelCase
    - Do not use special characters (!@#$%^&*) because they can mean things
    - Assigned on the left side: variable name declaration = value
    - So: variable1 = variable 2 + 3

**Input**

- UIn = input(“user sees this”)
    - Waits for user response

**9/22 Numeric & Boolean Types**

- Built in numeric types
    - Int: 1000
    - Float 1000.0
    - Complex - Deals with imaginary numbers (not needed for the class)
- Integer
    - Whole number
    - Requires least memory storage (most efficient)
    - Change data type by int(variable)
- Float
    - Number with decimal point or scientific notation (1E3)
    - Requires more memory than int
    - Change data type to a float by float(variable
- Arithmetic
    - + addition

    - - subtraction
    - * multiplication
    - / & // division & floor division
    - % modulus
- DIvision vs Floor Division
    - For integers, we know these as whole numbers, so what happens here?
    - var1 = 2
    - var2 = 5
    - print(var2/var1) -> 2.5
    - print(var2//var1) -> 2
- Modulus
    - Ex. 5 % 2 returns 1
        - 5 -2 = 3, 3 - 2 = 1
    - 6 % 2 returns 0
        - 6 - 2 = 4, 4 - 2 = 2, 2 - 2 = 0
- Functions available to numeric types
    - abs(x) - absolute value of x
    - sqrt(x) - square root of x, x must be > 0
    - log(x) - Natural log of x, x must be > 0
    - max(x1,x2,...) - of the input variables return the largest one
    - min(x1,x2,...) - of the input variables return the largest one
    - ceil(x) - returns the smallest int greater than x
    - floor(x) - returns the largest int smaller than x
    - sin(x) - sin of x in radians (not degrees)
    - cos(x) - cosine of x in radians
    - tan(x) - tangent of x in radians
    - degrees(x) - converts an angle x from radians to degrees
    - radians(x) - reverse of degrees
    - *Constants of pi and e are known by python
- Other things you can do
    - number += x (add x to *number* and set that as new value)
        - num = 1
        - num += 2
        - new num = 3
- Boolean data
    - True or false
    - Data can be evaluated as true or false using bool(variable)
    - 0 is false, anything else is true
    - Data is false when
        - None
        - False
        - Zero
        - Empty sequences/mapping

    - Objects of classes which has__bool__() or __len()__ method which returns 0 or false

**9/29 Strings**

- Substrings
  - Subset of characters found within a string
    - ello would be a substring of Hello; so could H
  - ex:
    - var1 = “Spider Man does whatever a spider can!”
    - var1 [0:10]
      - prints “Spider Man”
  - ex 2:
    - var3 = “Friendly neighborhood”
    - var1 = “Spider Man does whatever a spider can!”
    - var3 + var1[0:10] + var1[38]
      - prints “Friendly neighborhood Spider Man!”
- Escape characters
  - **sep=**
  - Characters you can put in a string to introduce formatting
  - \n new line
    - adds line break
  - \r carriage return
    - drops to the next line, like \n
  - \s space
    - adds a space
  - \t tab
    - adds a tab
  - ex:
    - print(“Spider”, “Man”, sep=’\s’) prints Spider Man
- Method
  - A function that “belongs to” an object
  - Object
    - Collection of data (variables) and methods (functios) that act on those data
    - A string is actually an object
    - When we declare a string, the system is creating a copy of the string object and setting va

  equal to value
  - Objects have methods
  - str.capitalize()
    - return a copy of the string with its first character capitalized, reset lowercase
  - str.isUpper()
    - Returns True if ALL chars in a string are uppercase
  - str.isLower()
    - Returns True if ALL chars in a string are lowercase
  - str.upper()

        - returns the string in a uppercase form
    - str.lower()
        - Returns the string in a lowercase form
    - str.casefold()
        - Casefolded strings may be used for caseless matching
    - str.find(‘A string’)
        - Looks for the A string part within the str variable. Returns the position
    - ex:
        - str.index(strX, beg = 0, end = len(str))
            - searches str for a pattern of strX, and returns the position
    - ex 2:
        - x = jack
        - x.find (‘j’)
            - return 1
                - 1 because there is only 1 ‘j’
        - print(myL[var1: ]

**<u>10/6 Branching</u>**

- Branching boils down to decision making: If I have a value of x, do 1, otherwise do 2
- If else example:
    - If (x==1):
        - x = x + 1
    - else:
        - x = 0
- Won’t work without :
- Python uses indentation instead of curly brackets
- Else if (elif) example:
    - if(x==1):
        - x = x + 1
    - elif(x==2):
        - x = x - 1
    - else:
        - print(x)
- Repeat
    - After if/else statements, you must have : symbol
    - Tells python to expect a set of code to execute following the condition
- Can be <u>nested</u>
    - If I have value x, do 1
        - If value x is now y, do something

- Nesting example:
    - UIn = int(input('#'))
    - If(UIn == 1):
        - UIn = 2
            - if(UIn !=3):
                - print('test')
    - else:
        - print(UIn)
- Loops
    - Iteration
        - Repetition of a process (code nested within loop)
    - Counter Loops
        - Where a counter is set and dictates when the loop stops
        - x = 0
        - While x <= 4:
            - Print "Hello"
            - x += 1
            - //stops after printing 5 times
    - Conditional Loops
        - Where a condition is set and dictates when the loop stops
        - While true:
            - Do stuff
        - While (!a.isEmpty())
            - Print "Hello"
            - if(a==3):
                - print ("Halfway")
            - a+=1
    - Collection Loops
        - When we iterate over an object/collection
        - Includes strings, lists, and dictionaries
    - While
        - Indefinite
        - A loop that will run 'while' a condition is met (or is true)
        - Can be printed on only one line if needed
        - Can have else statement too
        - answer = 'yes'
        - While answer == 'yes':
            - Do stuff
    - For
        - Used when we know how many times we want to loop
        - To iterate over an object like a list or string
        - When we want to loop, at least once***
        - can have else statement too
        - Range function:

            - Returns a range from 0 to n-1
            - range(5) is actually 0,1,2,3,4
            - Can return the range of a list by doing the following
                - list(range(5))
                - //auto incrementing timer, iterates 5 times
            - for x in list(range(5))
            - // for is item, list(range()) is collection
                - print(x)
            - Range can be passed up to 3 arguments, but 1 is the minimum, so you could say there are 2 optional arguments
            - range(end)
                - end at this point
            - range(start,end)
                - start at this point, then end here
                - end is iterations, not a point
            - range(start,end,step)
                - same as above, but indicate the step
                - step is how many skips on number line
                - can be used for odds/evens with step as 1
        - Using length
            - borgcode = [‘you’,’will’,’be’,’assimilated’]
            - For word in range(length(borgCode))
                - print (word)
        - With else
            - hardestThemeEver = [‘na’,’na’,’na’,’na’,’na’,’na’,’na’]
            - for i in hardestThemeEver: print (“Na”)
            - Else:
                - print(“Batman!”)
    - Nested loops
        - Do outer loop logic once
        - Complete all iterations of inner loop
        - When finished, trigger the outer loop again, followed by the inner loop
        - ex:
        - While i<= 3:
            - print(‘Lift’)
            - i += 1
            - While j <= 3:
                - print (‘Eat)
                - j += 1
        - else: print (‘uh-oh’)
        - //1 lift 4 eat

temp:

```
statesVisited = 0
while statesVisited <= 10:
    userIn = input('Which state would you like to visit? \n'
            'New Jersey \n'
            'Pennsylvania \n'
            'West Virginia \n'
            'Ohio \n'
            'Indiana \n'
            'Illinois \n'
            'Missouri \n'
            'Kansas \n'
            'Colorado \n'
            'Utah \n')
    if (userIn == "New Jersey" or userIn == "new jersey" or userIn == "New jersey"):
        print("You are in New Jersey")
        statesVisited += 1
        print(statesVisited)
    if (userIn == 'Pennsylvania' or userIn == 'pennsylvania')
        print("You are in Pennsylvania")
        statesVisited += 1
        print(statesVisited)
```

## 10/20 Midterm Review

- for x in myObj:
- ## ^item(x) ^collection (myObj)
- When do we use while loop
    - indefinite
    - while x < 4
        - print(x)
- For loop
    - definite
    - for y in x
        - print y
- When using True, capitalize ‘T’
- 1 else per if
- Know lists

**Lists and Dictionaries**

Lists

- myList = [] ##empty list
- Use square brackets
- Can contain mixed data
- Can be reference by index value
- An ordered set of values
- It's mutable
    - myL = [x, y, z]
    - myL.insert(0, a)
    - 0th element changes from x to a, now the list is a, x, y, z

List Methods

- list.append(x)
    - Add an item to the end of the list. Equivalent to a[len(a):] = [x]
- list.insert(i, x)
    - Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x)
- list.remove(x)
    - Remove the first item from the list whose value is equal to x. It raises a ValueError if there is no such item
- list.pop([i])
    - Removes the item at the given position in the list, and returns it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. Seen frequently in the Python Library Reference
- list.clear()
    - Remove all items from the list. Equivalent to del a[:]
- list.index(x[, start[, end]])
    - Return zero-based index in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item.
- list.count(x)
    - Return the number of times x appears in the list
- list.sort(key=None, reverse = False)
    - Sort the items of the list in place (the arguments can be used for sort customization, see sorted() for their explanation) Sorts numerically and alphabetically
- list.reverse()
    - Reverse the elements of the list in place

Dictionaries

- myDictionary = {}
- Uses curly braces
- An unordered set of values
- Immutable set of keys to values

Key Value Pair

- You have a key that corresponds to a value
- ex: myDictionary = {‘Spider-Man’:’Peter Parker’,’Venom’:’Eddie Brock’
- Spider-Man and Venom are keys, tied to the values Peter Parker and Eddir Brock
- myDictionary[‘Venom’] would return “Eddie Brock”

Dictionary Techniques

- Dictionary[key] = newValue
- To add a key:
- myDictionary[‘Alignment’] = ‘Hero’

Removing Elements

- del dictionary[key]
    - deletes the value tied to the key
- dictionary.clear()
    - clears all value entries within your dictionary, where dictionary is the name of the dictionary you want to clear

More Manipulation

- len(dictionary)
    - returns the number of elements in the dictionary
- str(dictionary)
    - returns a printable version of the entire dictionary
- dict.items()
    - returns a list of (key, value) tuple pairs
- dictionary.keys()
    - returns a list of keys from the dictionary

More notes

- Keys must be unique, you can’t have keys in the same dictionary with the same name (newer one overrides the previous)
- Keys are immutable. They can be strings, numbers, or tuples. You cannot have a dictionary of keys where the keys are lists.

**<u>11/10 Functions</u>**

Function examples:

- print()
- len()
- abs()
- min()
- max()

How to make one:

- def functionname(argument)
    - Def stands for defining a function
    - All lowercase
    - Argument is what we want to pass
    - Doesn't need to return something

Example:

- def addtwo (myNum):
    - myNum += 2
    - return myNum
- def noreturnnum (myNum):
    - myNum += 2
    - print (myNum)
    - return

Scoping Rules

- def mystats (name, age):
    - title = 'Mr.'
    - print('My name is', title, name, 'in a year, i'll be', (int(age)+1))
    - return
- title only exists within the scope of the mystats function, it cannot be called outside of it. This is called <u>local scope</u>
- Using variable name 'title' outside of the function would be called <u>global scope</u>

Multiple arguments

- Functions can have multiple arguments passed in:
    - def mystats (name, age):
    - print('My name is', name, 'in a year, i'll be', (int(age)+1))
    - return

Default arguments

- An argument is defaultly set to this if nothing is passed
- def mystats (name, age = 29):
    - print('My name is', name, 'in a year, i'll be', (int(age)+1))
    - return
- In the above case, if an age is not passed when we call the function, it will default to 29

Variable Length Arguments

- Sometimes you may not know how many args to expect, in which case we do the following
- def funmulti (arg1, *myList)

- print(arg1)
- for i in myList:
    - print(i)
- return
- the * means this arg is of variable size, figures out how large it is when it's passed in

Keyword Arguments
- def mystats (name, age):
    - print('My name is', name 'in a year, i'll be', (int(age)+1))
    - return

**Using Modules, Generating Random Values, & Error Handling**

Modules
- An external script of library that a programmer may import into another script to gain access to functions, methods, and variables found within the external script or library
- ex:
- #Lab Numerics
- import Math
- math.plt() ##stating that I want to use the plt() function found inside the external math module. In the Numerics lab, I have not programmed a plt() but I'm making use of one. By typing Math, I am signifying that I want 'something' from inside the Math module
- Fraction of a module
    - If you only want a portion of the module
    - from Math import plt as plot
    - Now I have only grabbed the plt. Notice the as plot part?
    - Makes it so you can write plot() instead of plt()

Random
- To use randomization, it's easiest to import the Random module
    - import Random
    - from random import functionName
- To use
    - First we set a seed
    - random.seed(a=None, version = 2)
    - a refers to a specific start. If a is set as a specific number the seed will always generate the same values. Think of it as you passing a variable into a function, if the variable never changes, then the output should never change.
    - If random.seed() is empty, system time is used to generate the random value
- To generate
    - random()        random float between 0 and 1
    - uniform(start,end)       random float between start and end
    - randrange(#)   Random integer between 0 and #
    - choice([ele1, ele2, ele3])        One random element from the collection
    - shuffle([ele1, ele2, ele3])        Shuffle a list

  - sample([ele1, ele2, ele3], l=#)  Select # random samples from the collection
  - randint(start,end)  random int from start to end
- Bad ex
  - import random
  - random.seed(2)
  - print(random.randint(0,5))
  - Produces same random value every time (Pseudo Randomization just like C++)
  - This is because seed is set to 2
- Good ex
  - import random
  - print(random.randint(0,5))
  - Will produce unpredictable random value

Error Handling/ Exception Handling

- Try/except statement helps with handling bad input
- I try to do something, if that fails, apply an exception (kind of like if else)
- ex:
  - states = [‘VA’, ‘NJ’]
  - try:
    - print(states[1]) #works
    - print(states[2]) #will fail, and error
  - except
    - print(‘an error occurred’)
  - Prints:
  - NJ
  - an error occurred
  - Notice the first statement got through, but not the second. This is because it ‘tries’ to perform as much as it can and anything that fails is omitted, and the exception code replaces it.
- Can have multiple exceptions
  - try:
    - #statements
  - except IndexError:
    - #statements
  - except ValueError
    - #statements
  - #Note: Only one exception will be executed
- Multiple Exceptions
  - def fun(a):
    - if a < 4:
      - # throws ZeroDivisionError for a = 3
      - b = a/(a-3)
    - #throws NameError if a >= 4
    - print (“Value of b = “, b)

- try:
    - fun(3)
    - fun(5)
- #note that braces () are necessary here for
- #multiple exceptions
- except ZeroDivisionError:
    - print(“ZeroDivisionError Occurred and Handled”)
- except NameError:
    - print(“NameError Occurred and Handled”)
- Error because division by 0 would occur. The second except doesn’t execute, since 1 error was already found.

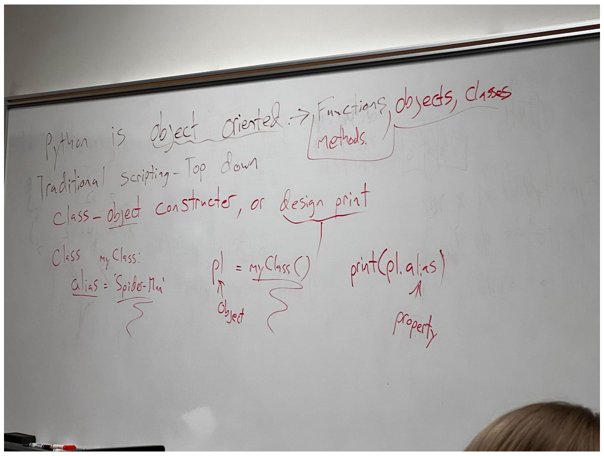
- Else for exception handling
    - def AbyB(a, b):
        - try:
            - c = ((a + b) / (a - b)
        - except ZeroDivisionError:
            - print(“a/b result in 0”)
        - else:
            - print(c)
        - #Driver program to test above function
        - AbyB(2.0, 3.0)
        - AbyB(3.0, 3.0)
        - # 5.0 first result, a/b result in 0 second result else clause only fires if try clause is successful
- Finally
    - Thing in exception handling called Finally
    - Finally is code that will be always executed, even if the try fails
    - Can be used in addition to else
        - try:
            - stuff
        - except:
            - stuff
        - else:
            - More stuff if try succeeds
        - finally:
            - Stuff that will always occur
- Error Raising
    - Errors you see in the terminal are raised errors, and you can make your own
        - try:
            - raise NameError(“Hi there”) # Raise Error
        - except NameError:
            - print(“An exception”)
            - raise #to determine whether the exception was raised or not

    - The terminal will print an exception, like regular exception handling does. However, the raise part will print something like
        - traceback (most recent call last):
        - File "/home/d6ec14ca595b97bff8d8034bbf212a9f.py", line 5, in <module>
        - _ raise NameError(“Hi there”) # Raise Error
        - NameError: Hi there

## Classes and Objects

- Python is object oriented
    - Functions, objects, classes, methods
- Uses Traditional scripting (Top Down)
- Class
    - Object constructor or design print
    - class myClass:
        - alias = ‘spider-man’
    - p1 = myClass()
    - print (p1.alias) #returns ‘spider-man’
    - #p1 is object, p1.alias is property
- Functions
    - __init__() #two underscores on each side
    - Initializes values for classes that stay the same. In ex, all heroes have name and alias
    - ex:
    - class Hero:
        - def __init__(self,name,alias):
            - self.name = name
            - self.alias = alias
        - p1 = (‘Peter Parker’, ‘Spider-Man’)
        - print(p1.name)
    - ##returns ‘Peter’, why?
    - #self is a reserved position. e.g. if replaced with test would be test.name
    - Self is a binder that is commonly used by most python programmers
    - ex:
    - def lyric(self):
        - print(self.name, self.name, ‘does whatever a spider can’)
    - p1 = hero(‘Parker’, ‘Spider-Man’)
    - p1.lyric()
    - #returns ‘Spider-Man Spider-Man does whatever a spider can’
- More
    - p1(‘Peter Parker’, ‘Spider-Man’)
    - p1.name = ‘Miles
    - del p1.alias #both this and line above remove ‘Peter Parker’ as name

- ○ Class Hero:
  - ■ pass #lets you not worry about empty function breaking code
- ○ Inheritance
  - ■ Defines a class that inherits all methods & properties from another class
  - ■ Parent class/ base class
    - ● Class spiderVerse:
      - ○ Parent passes its qualities to a child class
      - ○ ex:
      - ○ def __init__(self,name,alias):
        - ■ self.name = name
        - ■ self.alias = alias
      - ○ def printName(self)
        - ■ print(self.name, self.alias)
      - ○ x = spiderverse(‘Peter’, ‘Spider-Man’)
      - ○ x.printName()
      - ○ Class Spider(spiderVerse):
        - ■ pass
        - ■ #Spider is a child of spiderVerse, along with its own logic added on
        - ■ def  __init__(self, name, alias):
          - ● spiderVerse __init__ (self,name,alias):
          - ● self.power = power
          - ● #new __init__ being declared unlinks the old __init__
          - ● super() __init__(name,alias)

- ● Board notes of above:
  - ○ 
Python is object oriented → Functions, objects, Classes
methods.
Traditional Scripting - Top down
class - object constructer, or design print
Class myClass:
alias = 'Spider-Man'
p1 = myClass()
object
print(p1.alias)
property

For Mr poppins

- ● import functionLab
- ● functionLab.addSum(poppinsFood)

**For Final**

- 14 Questions
    - 8 Coding
    - 6 Other
- Can use Spyder
- Write a function
- Call function
- Write an object
- Write a class