Python

**Printing**

print(\_\_\_\_)

print(‘hi’, ‘there’) – prints as hi there

print(‘hi\_’ + ‘hello’) – prints as hi hello

print(‘hi’ + ‘hello’) – prints as hihello

print(‘hi’ + 5) or print(5 + ‘hi’)– error because in python, you can’t add a number and a string so we have to use print(‘hi’ + str(5)) (changing the number 5 into a string)- making print(‘hi’ + ‘5’)

print(5 + 8) prints 13

print(int(‘4’)+3) prints 7

print(int(‘4.2’)+3) prints error because 4.2 is not an integer, so we have to use print(float(‘4.2’)+3), which prints 7.2

print(4.4+3) prints 7.4 not 7

**String**

‘Hello’

“Hello” both work

Problem happens when ‘We’re going to the store’ since there is three ‘s which causes an error- so we therefore can use “We’re going to the store” since ‘ and “ is different

Or

We can do ‘We\’re going to the store’ where \ acts like as escape character and We’re going to the store is correctly printed

You can multiply a String with an integer, not float

Ex) “hi” \* 4 equals “hihihihi”

You can check whether a string appears in another string

Ex) ‘Jan’ in ’01 Jan’ – true

‘Jan’ in ‘hi’- false

**String Index**

If you use [], it deals specifically letter by letter

Ex) orange = ‘apple’

Print(orange[1]) prints p

Print(orange[4]) prints e – the number can’t be float and can’t be bigger than length - 1

If you use negative number, it goes from the opposite order

Print(orange[-1]) prints e

Print(orange[-3]) prints p

But you can’t change the letter

Ex)

Orange[1] = ‘C’ # this can’t happen (if you make a string, you can’t change – immutability)

**String Methods**

1. str.capitalize()- returns a copy of the string with the first letter caplitalized and rest lowercase
2. str.count(s)- returns the number of nonoverlappin occurrences of s in the string
3. str.endswith(end)- returns true if the string ends with the characters in the end string
4. str.find(s)- returns the index of the first occurrence of s in the string, or -1 if it doesn’t occur in the strin
5. str.find(s,beg)- returns the index of first occurrence of s at or after index beg in the string, or -1 if it doesn’t occur in the string at or after index beg
6. str.find(s,beg,end)- returns the index of first occurrence of s between indices beg(inclusive) and end(exclusive) in the string, or -1 if it doesn’t occur in the string
7. str.format(<<expression>>)- returns a string made by substitution for placeholder fields in the string-each field is a pair of braces with an integer in between

ex) ‘ ”{0}” is derived from “{1}” ‘.format(‘none’,’no one’)

prints

none is derived from no one

1. str.islower()- returns true if all characters in the string are lowercase
2. str.isupper()- returns true if all characters in the string are uppercase
3. str.lower()-returns a copy of the string with all letters converted to lowercase
4. str.lstrip()- returns a copy of the string with leading whitespace removed
5. str.lstrip(s)- returns a copy of the string with leading occurrences of the characters in s removed
6. str.replace(old,new)- returns a copy of the string with all occurrences of substring old replaced with string new
7. str.rstrip()- returns a copy of the string with trailing whitespace removed
8. str.rstrip(s)- returns a copy of the string with trailing occurrences of the characters in s removed
9. str.split()- returns the whitespace-separated words in the string as a list
10. str.startswith(beginning)- returns true if the string starts with the letters in the string beginning
11. str.strip()-returns a copy of the string with leading and trailing whitespace removed
12. str.strip(s)- returns a copy of the string with leading and trailing occurrences of the characters in s removed
13. str.swapcase()- returns a copy of the string with all lowercase letters capitalized and all uppercase letters made lowercase
14. str.upper()- returns a copy of the string with all letters converted to uppercase

**Math**

+ - \* / (addition, subtraction, multiplication, division)

Comment single line # \_\_\_\_\_ (does not get printed)

Multiple line comment ‘’’ \_\_\_\_\_\_\_\_\_ ‘’’ (does not get printed)

4 + 4 prints 8

3 -1 prints 2

4 \*2 prints 8

8.2 + 3 prints 11.2

5/2 prints the exact number 2.5

5//2 prints the integer division, which gives 2.5 rounded down to 2

5/34 prints the exact decimal and keeps going

4\*\*4 is 4^4 which is 256

4-2\*2 prints 0 (with the order of operations)

**Variables**

Variable names can be example\_Var, exampleVar, and etc (camelcase is most efficient)

We can do anything with variables (without assigning the types)

x = 32 or x = 4+8 (which equals 12)

print(x) prints x so it prints 32

we can even put functions such as x = print(“hello”) which prints hello when is called

we can do x = 3 and z = x \* 2 which is 6 and we can change the x anytime

x = 3

x = 8

If we call x here, 8(the latest one) is called

Ex) z = x + 3 is 11 not 6

We can call variables as x,y = (3,4) – here x is assigned 3 and y is assigned 4

So print(x) prints 3 print(y) prints 5

But you can’t add more variables or numbers and the # of variables and the # of values should match

Ex) x,y = (3,2,4) prints error because there is more numbers than variables

X,y,z = (3,2) prints error because there’s more variables than numbers

You can also have different types of x,y such as x,y = (‘hi’, 3.2) but you can do

x = [312,12,31,35,58,24] and in this case, x equals the list of the numbers

x = 1 and x+=1 makes x equal to 2 because x+=1 means x = x +1

**While Loop**

While (Boolean condition):

(index) (Statements)…

\*\*need statements that will change the boolean condition to false to end the loop

You can use break to achieve this goal – which ends the loop

Ex)

Count = 0

While True:

Print(count)

Count = count + 1

If(count ==3)

Break – this ends the loop

**For Loop**

list = [12,123,1234,12345]

for (any name) in list:

(index) (statements)…

The index or the indentation matters because if there’s no indentation, the statement is not inside the for loop and does not get repeated even though the condition is correct and you cannot index more than once

For (any name) in ‘String’ works as well

Ex)

Ring = ‘jogg

For c in Ring:

print c – prints j, o, g, g (runs for the number of letters of Ring)

**For Loop Specialties**

x = 2

y= [3.5]

for x in y:

print(x)

print(x) – this value of x is not 2, but is 5 in the end because x took the value of 5

Also,

x = 2

y= [3.5]

for z in y:

print(z)

print(z) – even though z was never assigned, z took the value of last number in list y and has the value of 5

When you want to call list inside list,

Ex)

X = [[3,123,4],[44,3223,1233]]

for insideList in X:

for insideInside in insideList:

print(insideInside)—this prints 3, 123, 4, etc

**If else else-if statements**

If (condition):

(index) (statement)…

In python you can do x > y > z and add more than one condition without the or and and function

Ex) x = 2, y = 3, z = 4

If y < z > x:

print(z) – works

if (condition):

(index) (statement1)

else:

(index) (statement2) – else runs only when the condition in if statement is false or when the statement1 at if did not run

if (condition1):

(index) (statement1)

elif (condition2):

(index) (statement2) – elif runs only when the condition1 in the first if statement is wrong and the condition 2 in elif is true

else:

(index) (statement3) – else runs only when both the conditions 1 and 2 are false and if statements 1 and 2 did not run

Ex)

x = 5

y = 6

z = 7

if x > y:

print(x)

elif x > z:

print(x)

elif y > z: (you can have multiple elif statements)

print(y)

else:

print(z)- the last statement in else prints because all the conditions from if and elif are false so 7 gets printed

**Functions in Python**

def (function name) : (def shows that it will identify a function)

(index) (statements)… (has to be indexed or indented to be part of the function)

Ex) def example():

print(1 + 2)

print(1 + 3)

You need to call the function in order for the function to work

**Parameters of Function**

def addTwoNumbers(one, two):

twoNumbers = one + two

print(twoNumbers)- this will print the addition of one and two, the order of the parameters matter and the number of parameters matter

addTwoNumbers(1,3,2) or addTwoNumbers(3) does not work due to the dismatch in number of parameters

You can do addTwoNumbers(two =4, one = 3) and even though the order matters, you directly assigned the two to a value of 4 and one a value of 3 so one goes to the first parameter variable while two goes to the second parameter variable

You cannot also call the variable twoNumbers because its scope is inside the function so outside the function, the variable twoNumbers does not exist

**Default Function Parameter**

You can add default value to the variable

def addToFive(one, two = 5): - here the second variable is automatically assigned to 5

answer = one + two - therefore you don’t need to specify the variable or call it

print(answer)

addToFive(4) works even though it only has one parameter because the second variable two is always 5 so it will print 9 without error

You can always assign new value to the variable ex) addtoFive(4,2) comes out to be 6 without an error- it just assigned the value 2 to the variable two instead of 5

**Global, Local Variable**

Every variable has different scope (variable’s reach is all different)

Global variable: accessed everywhere

Local variable: only be accessed within its frame

Every variable is local until they are stated that they are global variables

In order to make a variable global, you need to do global \_\_\_\_(variable)

When you declare a local variable outside the function, you cannot play with it inside the function because the variable is local.

Ex)

x = 3

def hi():

(index) print(x) – this is okay since it is just using x

(index) print(x + 2) – this is okay since it is just using x

(index) x = x + 2 – not okay since you are changing with local variable

To avoid this, you can make x a global variable

def hi():

(index) global x

(index) print(x) – this is okay since it is just using x

(index) print(x + 2) – this is okay since it is just using x

(index) x = x + 2 – this is okay because the variable x is now global

Or you can assign the value of x to another local variable that is playable

def hi()

(index) y = x

(index) print(y)

(index) print(y + 2)

(index) y = y + 2 – but you cannot access the variable y outside the function so you can return a value to save the variable y

(index) return y – returning

Therefore, you can assign a variable to the function since the function equals to the value of y ex) z = hi()

**Writing To File- saving data to a file**

Ex) saveFile = open(‘exampleFile.txt’ (the name of the file), ‘w’ (abbreviation that the user is going to write the file)- when this is called, the exampleFile is opened and previous exampleFile is cleared

saveFile.write(“hihi”) – what you are going to write at new saveFile

saveFile.close()- important to close because it will hang open for a while

**Appending To File**

Ex) appendMe = ‘new bit of information’

appendFile = open(‘exampleFile.txt’(name of the file), ‘a’ (abbreviation that the user is going to append the file)

appendFile.write(appendMe)

appendFile.close() – here it is adding new information without clearing it

**Read from a file**

readMe = open(‘exampleFile.txt.’(name of the file), ‘r’(abbreviation that the user is going to read the file)

print(readMe) – prints the text or information at exampleFile.txt directly

if do readMe = open(‘exampleFile.txt’, ‘r’).readlines() – it prints the information at the exampleFile as a list

ex) if exampleFile had

“I am Chris”

“I am”

.readlines() and print will [“I am Chris”, “I am”]

**Classes**

Class calculator(name of the class):

(class contains variable and functions)

(index) def addition(x,y):

(index) (index) added = x + y

(index) (index) print(added)

index) def subtraction(x,y):

(index) (index) subtract = x - y

(index) (index) print(subtract)

index) def multiplication(x,y):

(index) (index) multiply = x \* y

(index) (index) print(multiply)

index) def division(x,y):

(index) (index) divide = x / y

(index) (index) print(divide)

How to call class and the functions

Calculator.addition(5,8) prints 13 and etc.

**Random**

Ex)

import random – always need

r = random.random() – random number from 0 to 1

print(r) – prints a random number

Or

r = random.randrange(1,11) – random number from 1 to 10

r = random.randrange(10) means random.randrange(0,10)

**User Input**

x = input(‘What is your name?’) – x stores the input that the user types in (x has a string type- so you can’t add it by an int or float)

print(‘Hello’ + x) – we can then use x as a variable that has the value that the user inserted

**Boolean**

Bool in python

Every integer except 0 is true, 0 is false

Every String is true, unless there is nothing inside String(even if there is a single space, it is true

‘And’ and ‘Not’ and ‘or’ in Python

‘And’ is basically ‘and’

‘Or’ is basically ‘or’

‘Not’ is basically ‘not’

**Type Conversion**

You can’t subtract two Strings or add a String with an integer

Ex)

One = ‘1’

Two = ‘2’

Can’t do Two – One

Need to do… one = int(One), two = int(Two) and then two – one

**List**

You can change things in the list

Ex)

List = [1,2,3]

List[2] = ‘orange’

But you can’t add more items on the list by simple method

Ex)

List[4] = ‘orange’ – since the furthest index is 2, you can’t add item to the fourth index

If you want to add, do append()

Ex)

List.append(‘orange’)

You can delete things in the list by using del()

Ex)

List.del(2) – delete the second index, which is 3 – so list is now 1, 2

You can put list within the list

Ex)

List = [1,3,4,[‘orange’,’apple’]]

You can mutate or change list

Ex) hh = [‘hi’, ‘he’, ‘ha’, ‘hu’]

hh[2] = 0

hh becomes [‘hi’, ‘he’, 0, ‘hu’]

hh = hh + [2]

hh becomes [‘hi’, ‘he’, 0, ‘hu’, 2]

hh = hh \* 2

hh becomes [‘hi’, ‘he’, 0, ‘hu’, 2, ‘hi’, ‘he’, 0, ‘hu’, 2]

del hh[1]

hh becomes [‘hi’, 0, ‘hu’, 2, ‘hi’, ‘he’, 0, ‘hu’, 2]

bb = hh[0:3] # slicing the list, from 0 to (3-1)

bb is [‘hi’,0,’hu’]

bb = hh[3:]

bb is [2, ‘hi’, ‘he’, 0, ‘hu’, 2]

bb = hhh[:3]

bb is [‘hi’,0,’hu’]

alias—when x and y refers to the same list, any change in either x or y changes both

ex)

x = [1,2]

y = x

x = x + [3]

x becomes [1,2,3], y becomes [1,2,3]

\*\*When you import a class or a file, it automatically goes through the file or runs the code – to prevent this, people do if \_\_name\_\_ == ‘\_\_main\_\_’:

**List Methods**

1. L.append(v)- appends value v to the list L
2. L.clear()- removes all items from list L
3. L.count(v)- returns the number of occurrences of v in list L
4. L.extend(v)- appends the items in v to L
5. L.index(v)- returns the index of the first occurrence of v in L – error is raised if v doesn’t occur in L
6. L.index(v, beg)- returns the index of the first occurrence of v at or after index beg in L- error raised if v doesn’t occur in that part of L
7. L.index(v,beg,end)- returns the index of the first occurrence of v between indices beg(inclusive) and end(exclusive) in L; an error is raised if v doesn’t occur in that part of L
8. L.insert(i,v)- inserts value v at index i in list L, shifting subsequent items to make room
9. L.pop()- removes and returns the last item of L (must be nonempty)
10. L.remove(v)- removes the first occurrence of value v from list L
11. L.reverse()- reverses the order of the values in list L
12. L.sort()- sorts the values in list L in ascending order
13. L.sort(reverse=True)- sorts the values in list L in descending order

**Importing Modules**

* gaining access to the variables and functions from a module by importing it
* ex) import math, you can use the built-in functions and can find information about math by doing help(math)
* You can use functions from module by module.function ex)math.sqrt()
* Python automatically executes modules as it imports them, but only once

**Calling Methods**

* When calling methods, ‘brown’.capitalize() and str.capitalize(‘brown’) or abs(-3) and -3.\_\_abs\_\_() is the same
* If there are two underscores before and after the method, those methods are special methods (and are connected with some other syntax in Python)

**Data Type Set**

* Set is similar to list in a way that it stores mutable collections, but without order
* Ex) vowels ={‘a’,’e’,’i’,’o’,’u’}
* If we print out vowels,
* Print(vowels)
* The result might be in different order {‘e’,’o’,’u’,’a’,’i’}
* Also, each item should be distinct- the duplicates(repeats) are ignored
* So {3,2,4} == {3,2,2,2,2,2,3,3,4,4} prints True
* You can change a list to a set
* Ex) set([2,3,3,5]) makes {2,3,5}
* You can’t add two sets – {3,2} + {2,4} is error
* You can add more items by calling set.add(\_\_\_) – and you can include any type within the set
* You cant have set/list with set – set{set{\_\_\_},set{\_\_}, set{list[\_\_\_],list[\_\_]}- can’t happen
* You can have tuple inside set - set{tuple(\_\_\_),tuple(\_\_)}

**Set Methods**

* S.add(v)- adds item v to a set s- this has no effect if v is already in s
* S.clear()- removes all items from the set S
* S.difference(other)- returns a set with items that occur in set S but not in set other (S – other)
* S.intersection(other)- returns a set with items that occur both in sets S and other (S & other)
* S.issubset(other)- returns True only if all of set S’s items are also in set other (S<=other)
* S.issuperset(other)- returns True if set S contains all of set other’s items (S >= other)
* S.remove(v)-removes item v from set S
* S.symmetric\_difference(other)- returns a set with items that are in exactly one of sets s and other- any items that are in both sets are not included in the result (S^Other)
* S.union(other)- returns a set with items that are either in set S or other (S|Other)

**Tuple**

* Tuples- immutable sequence type
* Written using parentheses instead of brackets like strings and lists- can be subscripted, sliced, and looped over
* Ex) bases = (‘a’,’b’,’c’)
* For tuple with one element, it is not simply written as 8 but as 8,(with the comma)
* Ex) bases = (‘a’,) – this is tuple, bases = (‘a’) – this is string
* Since it is immutable, you can’t change the items
* Ex) bases[1] = ‘c’ is not (‘a’,’c’,’c’)- but is error
* But you can change the objects inside if the object is a list, but not when it is tuple
* (tuple can have other tuple, list, set inside them)
* Ex) life = ([2,3],[3,1])
* life[0][1] = 7 is possible since you are mutating a list
* Ex) life = ((2,3),(3,1))
* life[0][1] = 7 is impossible since you are mutating a tuple

**Dictionary**

* Dictionary, known as map, dictionary is unordered mutable collection of key/value pairs
* It associates a key with a value, where the keys form a set(so it can only appear once and is immutable)
* Ex) bird = {‘canada oose’:3, ‘northern fulmar’ :1} and if you type the key, you can the value associated with the key – bird[‘northern fulmar’] gives 1, but error if the key is not found
* You can add a new key/value
* Ex) bird[‘hi’] = 5, makes bird. {‘canada oose’:3, ‘northern fulmar’ :1, ‘hi’:5}
* Key cannot be a list/dictionary/tuple/set but the value can be list/tuple/set/dictionary
* You can change the content of the value even though the value is tuple
* You can use for loop with dictionary- here the variable is attached to the key:
* Ex) for x in bird:

Print(x, bird) – prints Canada oose, 3 … etc

**Dictionary Methods**

* D.clear()- removes all key/value pairs from dictionary D
* D.get(k)- returns the value associated with key k, or None if the key isn’t present
* D.get(k,v)- returns the value associated with key k, or a default value v if the key isn’t present
* D.keys()- returns the value associated with key k, or a default value v if the key isn’t present
* D.items()- returns dictionary D’s(key,value) pairs as set-like obects
* D.pop(k)-removes key k from dictionary D and returns thhe value that was associated with k; if k isn’t in D, an error is raised
* D.pop(k,v)-removes key k from dictionary D and returns thhe value that was associated with k; if k isn’t in D, returns v
* D.setdefault(k)- returns the value associated with key k in D
* D.setdefault(k,v) – returns the value associatd with key k in D; if k isn’t a key in D, adds thhe key k with the value v to D and returns v
* D.values()- returns the dictionary D’s values as a list-like object- entries may or may not be unique
* D.update(other)- updates dictionary D with the contents of dictionary other; for each key in other, if it is also a key in D, replaces that key in D’s value with the value from other; for each key in other, if that key isn’t in D, adds that key/value pair to D

**Searching and Sorting**

* Python already has method ‘index’ ( [‘a’,’b’].index(‘a’) returns the index of a, which is 0) that tells the index of the certain thing, but a sorted list is much faster in speed than an unsorted list to receive the returned variable
* When doing linear search – we look at Index 0, then 1, then 2, and continue
* Invariant of Linear Search - \_\_\_\_\_\_||||\_\_\_\_\_\_\_ (the left part is the part that search has occurred and the right part is the part that search has not occurred yet to find the value)
* Search with While Loop – not efficient because it checks whether the divider in invariant of linear search (||||||) reached the end yet
* Linear Search Running Time –
* First Case: While Loop (0.01) for Loop (0.01) Sentinel (0.01) list.index(0.01)
* Middle Case: While Loop (1261), for (515), sentinel (697), list.index(106)
* Last Case: While Loop(2673), For Loop (1029), Sentinel (1394), list.index(212)
* Linear Search is the simplest way to find a value in a list, but the time required is directly proportional to the length of the list
* Is there a way for faster linear search ? yes, by sorting
* Binary Search – each step divides remaining data into two equal parts and discards one of the two halves – searched in roughly log2N steps
* Binary Search divides and sorts the list by locating the values smaller to the left, bigger to the right, and not searched in the middle
* Binary Search: \_\_\_\_\_(smaller than v)||||||\_\_\_\_\_(not searched)|||||\_\_\_\_\_(larger than v)
* Python’s standard library’s bisect module includes binary search functions has binary search (bisect\_left returns index where item should be inserted in list to keep it in sorted order, assuming it is sorted to begin with)- left means that functions find leftmost(lowest index) position where they can do their jobs
* Bisect\_right, insort\_right- find the rightmost
* Different Sorting
* 1. Selection Sort: searching unknown section for smallest item and moving it to the index i, works by repeatedly finding next smallest item in the unsorted section and placing it at the end of sorted section
* 2. Insertion Sort: Keeps a sorted section at beginning of list and takes the next item from the unsorted section and inserts it where it belongs in the sorted section, increasing the size of sorted section by one
* 3. Mergesort

**Object-Oriented Programming**

* Object-Oriented programming involves:
* 1. Understanding the problem domain (need to know what your customer wants before you write a program)
* 2. Figuring out what types you might want
* 3. Figuring out what features you want your type to have
* 4. Writing a class what represents this type
* 5. Testing the code
* Function isinstance reports whether an object is an instance of a class(whether an object has a particular type)
* Ex) isinstance(‘abc’, str) – since ‘abc’ is a string, it prints true
* Python has a class called object, which is based on every class (even class and functions)
* So… ex) isinstance(‘abc’, object), isinstance(213, object) – all True
* Ex) isinstance(str, object), isinstance(max, object) - True
* This means that every class in Python is derived from class object, so every instance of every class is object
* Object class has few attributes(variables inside a class that refer to methods, functions, variables, or even other classes)- every class naturally has these attributes since every class is a subclass of object class
* If we create a class called Book, we can create an object of the Book from different class
* Ex) x = Book()
* x.title = ‘hi’
* x.authors = [‘d’,’s’] # these title and authors variable are instance variables because they are variables inside an instance of a class
* We can call the method inside Book() to find out some information
* If Book() has numberOfAuthors(x), we can call the method from other class – x.numberOfAuthors(x.authors)
* We do not always have to type the extra assignment statements every time Book() is created – but we can write a method that already does these things as Book() is created
* Ex) class Book:

def \_\_init\_\_ (self, title, author, publisher, price):

#Creates a book with following variables

* So, when we call Book to make a Book object, we can do it simply by book = Book(‘hi’, ‘hey’,[‘s’,’t’,’d’], ‘new york times’, 23.1)
* This \_\_init\_\_ is known as a constructor and is called whenever Book object is created

**Built in functions**

Range(int, int) – goes from first int upto the last int ex) range(1,3) prints 1 and 2 (not 3)

Range(int)- goes from 0 to int ex) range(3) prints 0,1,2

Range(int,int,int) goes from one to the other number with interval last number

Ex)range(2,10,3) prints 2, 2+3=5, 5+3=8

len(String) – the length of the String

len(list) – the length of the list

abs(int, float) – absolute value of the number

pow(\_a\_\_, \_\_b\_\_) – a ^ b or pow(a,b,c) – (a \* b) % c

round(\_\_\_\_\_)- rounding a number

\*\*If you do not know what the function does, you can put help(\_\_function\_\_)