**Numbers:**

**Python Notes**

Eg:

my\_income = 50000

tax\_rate = 12.0

my\_taxes = my\_income \* tax\_rate / 100

print(my\_taxes)

#output: 6000.0

>>>2\*3 #6

>>>2\*\*3 #8

>>>6%2 #0

>>>6/2 #3.0

>>>6//2 #3

**Lists:**

#basic

Eg: mylist = [1,2,3]

print(mylist) # [1, 2, 3]

#as you can see, we not only insert integers but we can also insert mixed datatypes

Eg:

>>> mixed\_list = ["dog",123, True,20.5, ['a', 'b', 'c']]

>>> print(mixed\_list)

print(mixedlist) # ['dog', 123, True, 20.5, ['a', 'b', 'c']]

Eg:

#finding length of the list

print(len(mixedlist)) # 5

print(len(mylist)) # 3

Eg:

#indexing the list

>>> print(mylist [2]) # 3

>>> print(mixed\_list [4][0]) # a

>>> print(mixed\_list[-2]) # 20.5

>>> print(mixed\_list[::2]) # ['dog', True, ['a', 'b', 'c']]

>>> print(mixed\_list[::2][1]) # True

Eg:

# lists are mutable

>>> mycars = ["Ferrari", "Mercedes", "BMW"]

>>> print(mycars) # ['Ferrari', 'Mercedes', 'BMW']

>>> mycars [2] = "Ford Mustang"

>>> print(mycars) # ['Ferrari', 'Mercedes', 'Ford Mustang']

Eg:

#List methods

#this will add the item to the last of the list

>>> mycars.append("Lamborghini")

>>> print(mycars) # ['Ferrari', 'Mercedes', 'Ford Mustang', 'Lamborghini']

>>> mycars1 = ["Ford", "Toyota", "Jaguar"]

>>> mycars.extend(mycars1)

>>> print(mycars) # ['Ferrari', 'Mercedes', 'Ford Mustang', 'Lamborghini', 'Ford'

, 'Toyota', 'Jaguar']

Eg:

# this will remove last item present in the list

>>> mycars.pop() # 'Jaguar'

>>> mycars.pop(1) # 'Mercedes'

>>> print(mycars) # ['Ferrari', 'Ford Mustang', 'Lamborghini', 'Ford', 'Toyota']

Eg:

# this will reverse the list

>>> mycars.reverse()

>>> print(mycars) # ['Toyota', 'Ford', 'Lamborghini', 'Ford Mustang', 'Ferrari']

Eg:

# this will sort the list

>>> mycars.sort()

>>> print(mycars) # ['Ford', 'Lamborghini', 'Toyota', 'Ferrari', 'Ford Mustang']

Eg:

#matrices in lists

>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> print(matrix[0][0]) # 1

#List Comprehension

>>> first\_col = [row[0] for row in matrix]

>>> print(first\_col) # [1, 4, 7]

>>> second\_col = [row[1] for row in matrix]

>>> print(second\_col) # [2, 5, 8]

**Strings:**

Eg:

#basics

>>> print("hello") # hello

>>> print("I\'m Cool") # I'm Cool

>>> print("I'm a Python Developer") # I'm a Python Developer

Eg:

#indexing

>>> my\_var = 'ABCDEFGH'

>>> print(my\_var[2]) # C

#If i want last item then we have to use negative indexing

>>> print(my\_var[-1]) # H

>>> print(my\_var[-2]) # G

Eg:

#slicing

#Which means it will take the string after B

>>> print(my\_var[2:]) # CDEFGH

>>> print(my\_var[:2]) # AB

#which means it will take the string after skiping the middle item between them

>>> print(my\_var[3:5]) # DE

>>> print(my\_var[::-2]) #HFDB

>>> my\_var[2] = "Z"

#this will produce error by saying type error because strings are immutable

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

Eg:

#stings methods

>>> my\_string = 'i am a python developer'

#convers the string to upper case

>>> my\_string.upper() # 'I AM A PYTHON DEVELOPER'

#converts the string to lower case

>>> my\_string.lower() # 'i am a python developer'

#it only capitalize the first letter of the string

>>> my\_string.capitalize() # 'I am a python developer'

# It splits the string into parts

>>> my\_string.split() # ['i', 'am', 'a', 'python', 'developer']

# It splits the strings by eliminating the word ‘o’ from the string and splits

>>> my\_string.split('o') # ['i am a pyth', 'n devel', 'per']

Eg:

#print formatting

>>> print("My name is :{}".format("Areeb"))

# My name is :Areeb

>>> print("I have a {a} and her name is {b}".format(a="dog",b="julie"))

# I have a dog and her name is julie

>>> print("my name is {a}...{a} Shaik".format(a="areeb"))

# my name is areeb...areeb Shaik

**Sets:**

#in sets elements are added individually

Eg:

#in sets the result will be in unsorted order

>>> x = set()

>>> x.add(0)

>>> x.add(1)

>>> x.add(2)

>>> print(x) # {0, 1, 2}

Eg:

#set accepts only one argument at a time

>>> x.add(1,4)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: add() takes exactly one argument (2 given)

Eg:

#In sets duplicates elements are not allowed

#as you can see i'm repeating the elements present in the set

>>> x=set([1,1,1,1,2,2,2,2,3,3,3,3])

>>> print(x) # {1, 2, 3}

**Tuples:**

#tuples are immutable

#tuples will work exact same as lists

#the only difference is that you cannot change the values from the tuples

Eg:

>>> my\_tuples = (1,2,3)

>>> print(my\_tuples)

# (1, 2, 3)

Eg:

#tuples doesn’t support item assignment

>>> my\_tuples[1] = 5

Error:

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

Eg:

#It also support mixed datatypes

>>> my\_tups = ("asdaf",2341,True,2035.05)

>>> print(my\_tups)

# ('asdaf', 2341, True, 2035.05)print(mytuple[2])

>>> my\_tups[2]

# True

**Dictionaries:**

#Dictionaries in python are similar to Objects in JavaScript

# dictionaries are mutable

Eg:

my\_dic = {"key1":"value1","key2":"value2","key3":"value3"}

print(my\_dic['key1'])

# {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

Eg:

#we can also have mixed datatypes in Dictionaries as well

>>> my\_dict = {"key1":123,"abc":"Strings",879:['a','b','c'],"key2":{'123':['Another one','Catch me']}}

>>> print(my\_dict)

# {'key1': 123, 'abc': 'Strings', 879: ['a', 'b', 'c'], 'key2': {'123': ['Another one', 'Catch me']}}

>>> print(my\_dict['key2']['123'][1])

# Catch me

Eg:

#we can also convert the resulted string to upper case

>>> print(my\_dict['key2']['123'][1].upper())

# CATCH ME

Eg:

>>> my\_food = {'Breakfast':'Idli','Lunch':'Biryani'}

>>> print(my\_food)

# {'Breakfast': 'Idli', 'Lunch': 'Biryani'}

Eg:

#adding items in dictionaries

>>> my\_food['Dinner'] = "Juice"

>>> print(my\_food)

# {'Breakfast': 'Idli', 'Lunch': 'Biryani', 'Dinner': 'Juice'}

Eg:

#printing the required key from the dictionary

print(my\_food['Dinner'])

# Juice

Eg:

#changing the value from the dictionary

my\_food['Dinner'] = 'Shwarma'

print(my\_food)

# Shwarma

**Functions:**

###### Problem -1 #########

#i'm defining a function with my\_func name and passing the dafult paramter as Areeb

def my\_func(param="Areeb"):

# as you can see we've used '''/""" will be used to describe your function

'''

I'm assigning the default parameter in functions argument

'''

#i'm printing the text by using format method

print("My name is {}".format(param))

my\_func()

# ouput : My name is Areeb

####### Problem -2 #############

# defining the function and passing the arguments

def add\_num(num1,num2):

#return will add the passing arguments

return num1+num2

#as you can we've passed the arguments as strings

result = add\_num("2", "3")

print(result)

#the ouput : 23

######### problem -3 #############

#i'm defining the function to check the type of arguments passed

def type\_check(num1,num2):

#by using if condition we are checking the type of arguments passed

if type(num1) == type(num2) == type(10):

#here by using return keyword we are performing addition operation

return num1+num2

#using else statement to print

else:

#this will get the output if the passed arguments are not integers

return "Sorry! I need integers"

result = type\_check("sad",2)

print(result)

#the ouput : Sorry! I need integers

result = type\_check(2,3)

print(result)

#output : 5

######## Problem-4 ############

#####LAMBDA EXPRESSION#########

#I'm writing a lambda expression to print even numbers from 0 to 10

#i have assign a list ranges from 0 to 11

mylist = range(11)

#lambda expression

even = filter(lambda num:num%2==0,mylist)

#print the ouput by assigning with list because we initialize the items with list

print(list(even))

#ouput : [0,2,4,6,8,10]

###some methods######

tweet = "Hi! I'm playing #cricket game"

res = tweet.split('#')

print(res)

#ouput : ["Hi! I'm playing","cricket game"]

res = tweet.split(' ')[2]

print(res)

#ouput : playing

###printing boolean values

print('x' in ['a','b','x']) #output : True

print('x' in ['a','b','c']) #ouput : False

**Control Flow:**

#########################

#### CONTROL FLOW #######

#########################

# In this lecture we will cover Control Flow in Python, basically how to dictate

# our code behaves in whatever manner we want. Let's start with basic comparison

# Operators:

###########################

## COMPARISON OPERATORS ###

###########################

# Greater than

1 > 2

# Less than

1 < 2

# Greater than or Equal to

1 >= 1

# Less than or Equal to

1 <= 4

# Equality

1 == 1

1 == "1"

'hi' == 'bye'

# Inequality

1 != 2

###########################

### LOGICAL OPERATORS #####

###########################

# AND

(1 > 2) and (2 < 3)

# OR

(1 > 2) or (2 < 3)

# Multiple logical operators

(1 == 2) or (2 == 3) or (4 == 4)

##################################

### if,elif, else Statements #####

##################################

# Indentation is extremely important in Python and is basically Python's way of

# getting rid of enclosing brackets like {} we've seen in the past and are common

# with other languages. This adds to Python's readability and is huge part of the

# "Zen of Python". It is also a big reason why its so popular for beginners. Any

# text editor or IDE should be able to auto-indent for you, but always double check

# this if you ever get errors in your code! Code blocks are then noted by a colon (:).

# Now let's show some examples of if, elif, and else statements:

if 1 < 2:

print('Yep!')

if 1 < 2:

print('yep!')

# If Else - Make sure to line up the else with the if statement to "connect" them

if 1 < 2:

print('first')

else:

print('last')

###

###

if 1 > 2:

print('first')

else:

print('last')

# To add more conditions (like else if) you just use a single phrase "elif"

if 1 == 2:

print('first')

elif 3 == 3:

print('middle')

else:

print('Last')

################################################################################

####################-----------------------------###############################

####################-----------LOOPS-------------###############################

####################-----------------------------###############################

################################################################################

# Time to review loops with Python, such as For Loops and While loops

# Python is unique in that is discards parenthesis and brackets in favor of a

# whitespace system that defines blocks of code through indentation, this forces

# the user to write readable code, which is great for future you looking back at

# your older code later on!

#####################

### FOR LOOPS #######

#####################

# Use For Loops for any sequence of elements. If you try to use a for loop with

# a mapping like a dictionary, it will still work, but it won't loop with any

# order. Let's walk through some examples of how a for loop behaves with the

# various data structures we've learned about!

## For Loop with a list

# Perform an action with each element

seq = [1,2,3,4,5]

for item in seq:

print(item)

# Perform an action for every element but doesn't actually involve the elements

for item in seq:

print('Yep')

# You can call the loop variable whatever you want:

for jelly in seq:

print(jelly+jelly)

## For Loop with a Dictionary

ages = {"Sam":3,"Frank":4,"Dan":29}

for key in ages:

print("This is the key")

print(key)

print("This is the value")

print(ages[key])

print("\n")

# A list of tuple pairs is a very common format for functions to return data in

# Because it is so common we can use tuple un-packing to deal with this, example:

mypairs = [(1,10),(3,30),(5,50)]

# Normal

for tup in mypairs:

print(tup)

# Tuple un-packing

for item1,item2 in mypairs:

print(item1)

print(item2)

#######################

### WHILE LOOPS #######

#######################

# While loops allow us to continually perform and action until a condition

# becomes true. For example:

i = 1

while i < 5:

print('i is: {}'.format(i))

i = i+1

#####################

### OTHER TOPICS ####

#####################

# RANGE FUNCTION

# range() can quickly generate integers for you, based on a starting and ending point

# Note that its a generator:

range(5)

list(range(5))

for i in range(5):

print(i)

# Start and ending

range(1,10)

# Third argument for step-size

range(0,10,2)

# List Comprehension

# This technique allows you to quickly create lists with a single line of code.

# You can think of this as deconstructing a for loop with an append(). For Example:

# Starting with:

x = [1,2,3,4]

# We could do this:

out = []

for item in x:

out.append(item\*\*2)

print(out)

# Written in List Comprehension Form

[item\*\*2 for item in x]

# List Comprehension is a great tool, but remember its not always approriate for

# every situation, don't sacrafice readability for a list Comprehension. It's

# speed is very comparable to the for loop.

#####################################

#### PART 6: EXERCISE REVIEW #######

#####################################

# Time to review all the basic data types we learned! This should be a

# relatively straight-forward and quick assignment.

###############

## Problem 1 ##

###############

# Given the string:

s = 'django'

# Use indexing to print out the following:

# 'd'

print(s[0])

# 'o'

print(s[-1])

# 'djan'

print(s[:4])

# 'jan'

print(s[1:4])

# 'go'

print(s[-2:])

# Bonus: Use indexing to reverse the string

print(s[::-1])

###############

## Problem 2 ##

###############

# Given this nested list:

l = [3,7,[1,4,'hello']]

# Reassign "hello" to be "goodbye"

l[2] = "goodbye"

print(l)

###############

## Problem 3 ##

###############

# Using keys and indexing, grab the 'hello' from the following dictionaries:

d1 = {'simple\_key':'hello'}

print(d1['simple\_key'])

d2 = {'k1':{'k2':'hello'}}

print(d2['k1']['k2'])

d3 = {'k1':[{'nest\_key':['this is deep',['hello']]}]}

print(d3['k1'][0]['nest\_key'][1][0])

###############

## Problem 4 ##

###############

# Use a set to find the unique values of the list below:

mylist = [1,1,1,1,1,2,2,2,2,3,3,3,3]

print(set(mylist))

###############

## Problem 5 ##

###############

# You are given two variables:

age = 4

name = "Sammy"

# Use print formatting to print the following string:

"Hello my dog's name is Sammy and he is 4 years old"

print("Hello my dog's name is {a} and he {b} is years old".format(a="sammy",b=4))