UNIT 1 - INTRODUCTION TO PYTHON PROGRAMMING

Why Python?

1. web development (server-side)
2. software development
3. mathematics
4. system scripting
5. Python works on different platforms (Windows, Mac, ubantu, Linux, Raspberry Pi, etc).
6. Python has a simple syntax similar to the English language.
7. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
8. Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

What can Python do?

1. Python can be used on a server to create web applications.
2. Python can be used alongside software to create workflows.
3. Python can connect to database systems. It can also read and modify files.
4. Python can be used to handle big data and perform complex mathematics.
5. Python can be used for rapid prototyping, or for production-ready software development.

To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself with the help of following - on the Windows, Mac or Linux command line:

```
C:\Users\[redacted] python    OR   C:\Users\[redacted]>py
```

Python Syntax

1. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
2. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Comments

Python has commenting capability for the purpose of in-code documentation. Comments start with a #, and Python will render the rest of the line as a comment

```
print("Hello, World!")  #This is a comment
```

Multiline Comments

TYPE 1

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

TYPE 2

```
" " "This is a comment written in
more than just one line " " "
print("Hello, World!")
```

Variables

1. Python has no command for declaring a variable.
2. A variable is created the moment you first assign a value to it.

```
x =  15          # x is of type int
x =  'Welcome'   # x is now of type str
print(x)
```

3. if you want to specify the data type of a variable, this can be done with casting.

```
x =  str(7)        # x will be '7'
y =  int(6)        # y will be 6
z =  float(8)    # z will be 8.0
```

4. String variables can be declared either by using single or double quotes

```
x =  "welcome'
# is the same as
x =  "welcome"
```

5. Variable names are case-sensitive.

```
a =  14
A =  'welcome'
#A will not overwrite a both are different
```

Variable Names

1. A variable name must start with a letter or the underscore character
2. A variable name cannot start with a number
3. A variable name can only contain alpha-numeric characters and underscores
4. A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
5. A variable name cannot be any of the Python keywords.

myvar my_var _my_var myVar MYVAR myvar2
salary SALARY Salary _salary _Salary _SALARY SALARY_
Salary_ salary_
my_variable_name

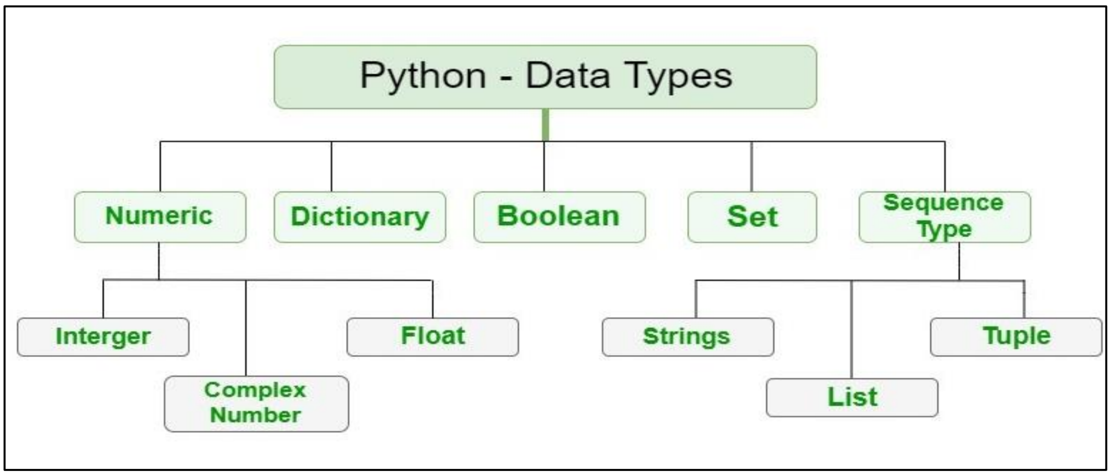
Assigining the values to variables

1. we can assign the multiple values to multiple variables at a time
2. Python allows you to assign values to multiple variables in one line
   x, y, z = "ABC", "XYZ", "LMN"
3. we can also assign the one value to multiple variables at a time
   x = y = z = 'ABC'
4. assignining multiple values to a set of variable is calles as collection which is refered as list or tuple
5. Python allows you to extract the values into variables. This is called *unpacking*.
   fruits = ["apple", "banana", "cherry"]
   x, y, z = fruits
6. then x, y and z will be apple, banana and cheery respectively and we can take the output of it as **print(x, y, z) [output will be** apple banana cherry] **or**
   print(x + y + z) [output will be applebananacherry]
7. spaces will onlybe printed in between the two strings when seperated by comma or if we have put the space while writting them in quotation marks.
8. if we have x = 15 and y = 'welcome' and we take the output as follows then
   ```
   print(x+y)    #error as both the variables having diffrent data types
   print(x,y)    # o/p will be 15 welcome
   ```

Global Variables

1. Variables that are created outside of a function (as in all of the examples above) are known as global variables.
2. Global variables can be used by everyone, both inside of functions and outside.
   ```
   x = 'morning' #now x is global variable
   def myfun():
   print( 'good ' + x)
   myfun() #fun call
   ```
   o/p : good morning
   ```
   x = 'morning'
   def myfun():
   x = 'afternoon'
   print( 'good ' + x)
   myfun() #fun call
   print( 'good ' + x)
   ```
   o/p : good afternoon
   good morning
3. To create a global variable inside a function, you can use the global keyword.
   ```
   def myfunc():
       global x
      x = "fantastic"
   myfunc()
   print("Python is " + x)
   ```
4. in the following program x is overwritten
   ```
   x = "awesome"
   def myfunc():
       global x
      x = "fantastic"
   myfunc()
   print("Python is " + x)
   ```

DATA TYPES



```
# DataType Output: str
        x = "Hello World"
# DataType Output: int
        x = 50
# DataType Output: float
        x = 60.5
# DataType Output: complex
        x = 3j
# DataType Output: list
    x = ["geeks", "for", "geeks"]
# DataType Output: tuple
    x = ("geeks", "for", "geeks")
# DataType Output: range
    x = range(10)
# DataType Output: dict
    x = {"name": "Suraj", "age": 24}
```

```
# DataType Output: set
    x = {"geeks", "for", "geeks"}
# DataType Output: frozenset
    x = frozenset({"geeks", "for",
"geeks"})
# DataType Output: bool
    x = True
# DataType Output: bytes
    x = b"Geeks"
# DataType Output: bytearray
    x = bytearray(4)
# DataType Output: memoryview
    x = memoryview(bytes(6))
# DataType Output: NoneType
    x = None
```

Numeric Data Type in Python

1. Integer (12, 6, ...)
2. float (1.2, 15.7, 2567.589, ...)
3. complex numbers(3+i,5+7i, ...)

Python Operators

Python divides the operators in the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

Assignment operators are used to compute the values:

Operator	Name	Example	Explaination
+	Addition	x + y	sum
-	Subtraction	x - y	diffrence
*	Multiplication	x * y	product
/	Division	x / y	quotient
%	Modulus	x % y	remainder
**	Exponentiation	x ** y	to the power
//	Floor division	x // y	round off to nearest

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y

!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y

~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

The precedence order is described in the table below, starting with the highest precedence at the top:

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT (increment/decrement)
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise operators L,R, AND,XOR, OR
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
~, &&, ||	Logical operators NOT, AND, OR

<u>BUILD IN DATA TYPES</u>

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- <u>Tuple</u> is a collection which is ordered and unchangeable. Allows duplicate members.
- <u>Set</u> is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- <u>**Dictionary**</u> is a collection which is ordered** and changeable. No duplicate members.

<u>List</u>

1. Listsprint(thislist) are used to store multiple items in a single variable.
2. Lists are created using square brackets: `mylist = ["apple", "banana", "cherry"]`
3. List items are ordered, changeable, and allow duplicate values.
4. List items are indexed, the first item has index [0], the second item has index [1]
5. List is changeable, meaning that we can change, add, and remove items in a list after it has been created.

6. print(len(mylist)), len function prints the length of the list
7. A list can contain different data types: list1 = ["abc", 34, True, 40, "male"]
8. Negative indexing means start from the end -1 refers to the last item, -2 refers to the second last item etc. print(list1[-1]) will give the o/p as male
9. print(thislist[2:5]) will print the index 2 (included) and end at index 5 (not included). **[index : position]**
   ```
   print(thislist[:4])  #
   print(thislist[2:])  #
   print(thislist[-4:-1])   #
   ```
10.
    ```
    thislist = ["apple", "banana", "cherry"]
    if "apple" in thislist:
            print("Yes, 'apple' is in the fruits list")
    ```
11. Insert "watermelon" as the third item:
    ```
    thislist = ["apple", "banana", "cherry"]
    thislist.insert(2, "watermelon")
    print(thislist)
    ```
12. To append elements from *another list* to the current list, use the extend() method. (MERGING)
    ```
    thislist = ["apple", "banana", "cherry"]
    tropical = ["mango", "pineapple", "papaya"]
    thislist.extend(tropical)
    ```

13. The remove() method removes the specified item.
    thislist.remove("banana")
14. The pop() method removes the specified index.
    thislist.pop(1)
15. If you do not specify the index, the pop() method removes the last item.
16. The del keyword also removes the specified index:
    ```
    del thislist[0]
    del thislist
    ```
17. The clear() method empties the list.
    thislist.clear()

Tuple

1. A tuple is a collection which is ordered and **unchangeable**.
2. Tuples are written with round brackets.
3. Tuple items are ordered, unchangeable, and allow duplicate values.
4. Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
5. Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

```
tuple1 = ("abc", 34, True, 40, "male")
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

6. Access tuple items by referring to the index number, inside square brackets.
   thistuple[1]
7. Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
8. But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Set

1. Sets are used to store multiple items in a single variable.
2. A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
3. Set *items* are unchangeable, but you can remove items and add new items.
4. Sets are written with curly brackets.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

5. Set items are unordered, unchangeable, and do not allow duplicate values.
6. Sets cannot have two items with the same value.
7. True and 1 is considered the same value
8. Once a set is created, you cannot change its items, but you can add or remove new items.

```
thisset.add("orange")
```

9. To add items from another set into the current set, use the update() method

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
    'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papaya'}
```

10. To remove an item in a set, use the remove(), or the discard() method.
11. we can Remove "banana" by using the remove() method

```
thisset.remove("banana")
```

12. we have to specify the variable to remove, if variableis not present then it will produce the error.
13. if the item to remove does not exist, discard() will **NOT** raise an error.

```
    thisset.discard("banana")
```

14. We can also use the pop() method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed.
15. The return value of the pop() method is the removed item.

```
    thisset = {"apple", "banana", "cherry"}
    x = thisset.pop()
    print(x)  # x contains the poped item
```

16. The clear() method empties the set. thisset.clear()
17. The del keyword will delete the set completely del thisset

Dictionary

1. Dictionaries are used to store data values in key:value pairs.
2. A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

```
thisdict = {
  "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict)
        {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
print(thisdict["brand"])
Ford
```

3. dictionaries are ordered, it means that the items have a defined order, and that order will not change.
4. Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
5. Dictionaries cannot have two items with the same key

```
  thisdict = {
      "brand": "Ford",
      "model": "Mustang",
      "year": 1964,
      "year": 2020
  }
  print(thisdict)
  # in the above program only overwritten value is considered
```

LOOPING STATEMENTS

While Loop

```
    while expression:
        statement(s)
    # Python program to illustrate while loop
    count = 0
    while (count < 3):
        count = count + 1
        print("Hello Geek")
```

While Loop with else statement

```
while condition:
     # execute these statements
else:
     # execute these statements
```

# Python program to illustrate combining else with while

```
count = 0
while (count < 3):
        count = count + 1
        print("Hello Geek")
else:
        print("In Else Block")
```

For Loop

For loops are used for sequential traversal

```
   for iterator_var in sequence:
        statements(s)
```

# Python program to illustrate Iterating over range 0 to n-1

```
n = 4
for i in range(0, n):
        print(i)
```

# Python program to illustrate combining else with for

```
list = ["geeks", "for", "geeks"]
for index in range(len(list)):
        print(list[index])
else:
        print("Inside Else Block")
```

Nested Loops

Python programming language allows to use one loop inside another loop.

NESTED FOR LOOP

```
for iterator_var in sequence:
   for iterator_var in sequence:
       statements(s)
   statements(s)
```

NESTED WHILE LOOP

```
while expression:
   while expression:
       statement(s)
   statement(s)
```

Loop Control Statements

Continue Statement

the continue statement in Python returns the control to the beginning of the loop.

```
# Prints all letters except 'e' and 's'
for letter in 'geeksforgeeks':
        if letter == 'e' or letter == 's':
                continue
        print('Current Letter :', letter)
```

Break Statement

The break statement in Python brings control out of the loop.

```
for letter in 'geeksforgeeks':

        # break the loop as soon it sees 'e'
        # or 's'
        if letter == 'e' or letter == 's':
                break
print('Current Letter :', letter)
```

Pass Statement

We use pass statement in Python to write empty loops. Pass is also used for empty control statements, functions and classes.

```
# An empty loop
for letter in 'geeksforgeeks':
        pass
print('Last Letter :', letter)
```

Python Module

1. A Python module is a file containing Python definitions and statements.
2. A module can define functions, classes, and variables.
3. A module can also include runnable code.
4. Grouping related code into a module makes the code easier to understand and use.
5. It also makes the code logically organized.

create a simple calc.py in which we define two functions, one add and another subtract.

```
# A simple module, calc.py
def add(x, y):
        return (x+y)

def subtract(x, y):
        return (x-y)
```

6. When the interpreter encounters an import statement, it imports the module if the module is present in the search path.
7. A search path is a list of directories that the interpreter searches for importing a module.
8. For example, to import the module calc.py, we need to put the following command at the top of the script.
   ```
   # importing module calc.py
   import calc
   print(calc.add(10, 2))
   ```
9. To access the functions inside the module the dot(.) operator is used.
10. Import Specific Attributes from a Python module
    ```
    from math import sqrt, factorial
    ```
11. The use of * has its advantages and disadvantages. If you know exactly what you will be needing from the module, it is not recommended to use *, else do so.
    ```
    # importing sqrt() and factorial from the module math
    ```

```
from math import *
# if we simply do "import math", then math.sqrt(16) and math.factorial() are required.
print(sqrt(16))
print(factorial(6))
```

12. We can rename the module while importing it using the keyword.

```
import math as mt
print(mt.sqrt(16))
print(mt.factorial(6))
```

Functions

1. User-defined Functions :

Functions that we define ourselves to do certain specific task are referred as user-defined functions.

2. Built-in Functions :

Python has several functions that are readily available for use. These functions are called built-in functions.

3. Lambda Functions :

1. They are called as **anonymous function** that are defined without a name.
2. While normal functions are defined using the **def** keyword in Python, anonymous functions are defined using the **lambda** keyword.

4. Recursion Functions

1. A recursive function is a function defined in terms of itself via **self-referential** expressions.
2. This means that the function will continue to call itself and repeat its behavior until some condition is met to return a result

PRACTICE EXAMPLES

```
# Python program to add two numbers
num1 = 15
num2 = 12
# Adding two nos
sum = num1 + num2
# printing values
print("Sum of", num1, "and", num2 , "is", sum)

#To define a function that take two integers and return the sum of those two numbers
def add(a,b):
return a+b
#initializing the variables
num1 = 10
num2 = 5
#function calling and store the result into sum_of_twonumbers
sum_of_twonumbers = add(num1,num2)
#To print the result
print("Sum of {0} and {1} is {2};" .format(num1, num2, sum_of_twonumbers))
```

#Simple interest formula is given by: Simple Interest = (P x T x R)/100

```
# Python3 program to find simple interest principal amount, time and
# rate of interest taken from user.

def simple_interest(p,t,r):
    print('The principal is', p)
    print('The time period is', t)
    print('The rate of interest is',r)
    si = (p * t * r)/100
    print('The Simple Interest is', si)
# Driver code
P = int(input("Enter the principal amount :"))
T = int(input("Enter the time period :"))
R = int(input("Enter the rate of interest :"))
simple_interest(P,T,R)

# Python program to find Area of a circle
def findArea(r):
    PI = 3.142
    return PI * (r*r);
# Driver method
print("Area is %.6f" % findArea(5));

# Python program to find Area of a circle using inbuild library
import math
```

```
def area(r):
    area = math.pi* pow(r,2)
   return print('Area of circle is:' ,area)
area(4)
```

#Python Program for Sum of squares of first n natural numbers $1^2 + 2^2 + 3^2 + ….. + N^2$.

```
def squaresum(n):
    # Iterate i from 1 and n finding square of i and add to sum.
   sm = 0
   for i in range(1, n+1):
      sm = sm + (i * i)
    return sm
  # Driven Program
n = 4
print(squaresum(n))
```

# program to swap elements at given positions

```
# Swap function
def swapPositions(lis, pos1, pos2):
    temp=lis[pos1]
    lis[pos1]=lis[pos2]
    lis[pos2]=temp
    return lis
# Driver function
List = [23, 65, 19, 90]
pos1, pos2 = 1, 3
print(swapPositions(List, pos1-1, pos2-1))
```

# Reversing a list using slicing technique

```
def Reverse(lst):
    new_lst = lst[::-1]
    return new_lst
lst = [10, 11, 12, 13, 14, 15]
print(Reverse(lst))
```

# Python program to find smallest number in a list

```
# list of numbers
list1 = [10, 20, 4, 45, 99]
# sorting the list
list1.sort()
# printing the first element
print("Smallest element is:", list1[0])
```

# Python program to find smallest number in a list

```
# list of numbers
list1 = [10, 20, 1, 45, 99]
```

```
# printing the minimum element
print("Smallest element is:", min(list1))
```

# Python program to print Even Numbers in a List

```
# list of numbers
list1 = [10, 21, 4, 45, 66, 93]
# iterating each number in list
for num in list1:
    # checking condition
    if num % 2 == 0:
        print(num, end=" ")
```

# Python program to print positive Numbers in a List

```
list1 = [-10, 21, -4, -45, -66, 93]
num = 0
# using while loop
while(num < len(list1)):
    # checking condition
    if list1[num] >= 0:
        print(list1[num], end = " ")
        # increment num
    num += 1
```

# Python code To reverse words in a given string

```
# input string
string = "geeks quiz practice code"
# reversing words in a given string
s = string.split()[::-1]   # split breaks the words from sentences
l = []   #empty list
for i in s:
    # appending reversed words to l
    l.append(i)
# printing reverse words
print(" ".join(l))      #join function combines the words and make sentence
```

#Adding Tuple to List and vice – versa

```
my_list = [5, 6, 7]
my_tuple = (9, 10)
index = 3
my_list.insert(index, my_tuple)
print("List after addition: ", my_list)
```

Write a program to find 2nd largest from list

Write a program to print odd numbers from list

Write a program to print negative numbers from list

#program to add two nos using function

```
def add(num1: int, num2: int) -> int:
    """Add two numbers"""
    num3 = num1 + num2
    return num3
# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} results {ans}.")
```

Write a program using function to print that input no is even or odd