Name:- Divyang Bagla

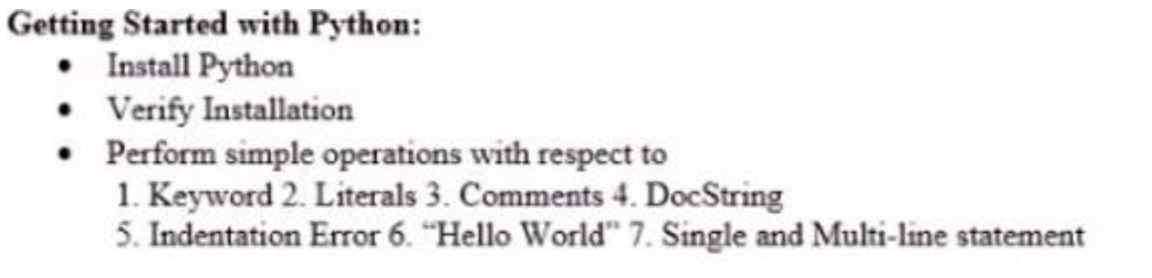
ERP ID:- 1032180739

Batch A2

Roll No. :- PC33

Lab Assignment No. 1

Aim:-



Getting Started with Python:

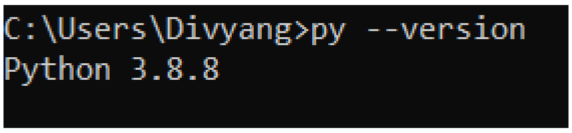
- Install Python
- Verify Installation
- Perform simple operations with respect to
  1. Keyword 2. Literals 3. Comments 4. DocString
  5. Indentation Error 6. "Hello World" 7. Single and Multi-line statement

Theory:-

Steps to install python in windows 10:-

1. Step 1 − Select Version of Python to Install. ...
2. Step 2 − Download Python Executable Installer. ...
3. Step 3 − Run Executable Installer. ...
4. Step 4 − Verify Python is installed on Windows. ...
5. Step 5 − Verify Pip was installed.

Verify Installation :-



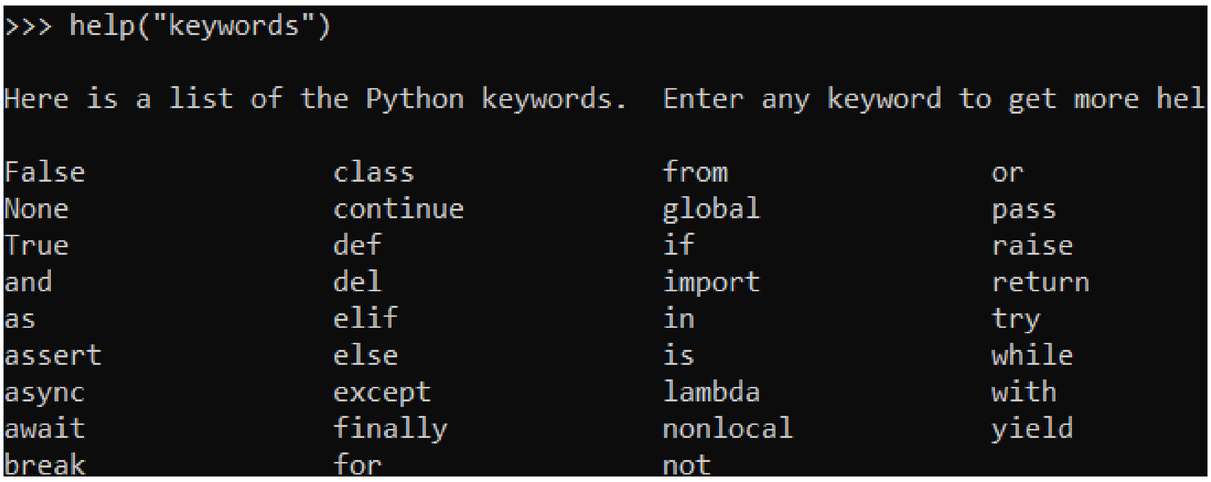
```
C:\Users\Divyang>py --version
Python 3.8.8
```

Define Following :-

Keywords :- Python keywords are special reserved words that have specific meanings and purposes and can't be used for anything but those specific purposes. These keywords are always available—you'll never have to import them into your code.

Python keywords are different from Python's **built-in functions and types**. The built-in functions and types are also always available, but they aren't as restrictive as the keywords in their usage.

An example of something you *can't* do with Python keywords is assign something to them. If you try, then you'll get a **SyntaxError**. You won't get a SyntaxError if you try to assign something to a built-in function or type, but it still isn't a good idea.



```
>>> help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more hel

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Literals :- Literals are a notation for representing a fixed value in source code. They can also be defined as raw value or data given in variables or constants.

Numeric literals

```
x = 24
y = 24.3
z = 2+3j
print(x, y, z)
```

Comments :- A comment in Python **starts with the hash character, # , and extends to the end of the physical line**. A hash character within a string value is not seen as a comment, though. To be precise, a comment can be written in three ways - entirely on its own line, next to a statement of code, and as a multi-line comment block.

DocString:- Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods.

It's specified in source code that is used, like a comment, to document a specific segment of code. Unlike conventional source code comments, the docstring should describe what the function does, not how.

What should a docstring look like?

- The doc string line should begin with a capital letter and end with a period.
- The first line should be a short description.
- If there are more lines in the documentation string, the second line should be blank, visually separating the summary from the rest of the description.
- The following lines should be one or more paragraphs describing the object's calling conventions, its side effects, etc.

Example:-

```
def my_function():

    '''Demonstrates triple double quotes
    docstrings and does nothing really.'''

    return None

print("Using __doc__:")
print(my_function.__doc__)

print("Using help:")
help(my_function)
```

Output:-

```
Using __doc__:

Demonstrates triple double quotes

    docstrings and does nothing really.

Using help:

Help on function my_function in module __main__:


my_function()

    Demonstrates triple double quotes

    docstrings and does nothing really.
```

Indentation Error:- Python is a procedural language. The indentation error can occur when the spaces or tabs are not placed properly. There will not be an issue if the interpreter does not find any issues with the spaces or tabs. If there is an error due to indentation, it will come in between the execution and can be a show stopper.

Example:-

```
site = 'edu'
if site == 'edu':
print('Logging in to EduCBA!')
else:
print('Please type the URL again.')
print('You are ready to go!')
```

In above there is an indentation error is present.

Conclusion:- Installed python and learned about the keywords, literals, single and multi line comments, indentation error etc.

Lab Assignment 1Code

Lab Assignment No. 1

KeyWords

In [1]:
```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Literals

In [2]:
```
x = 24
y = 24.3
z = 2+3j
print(x, y, z)
```

```
24 24.3 (2+3j)
```

In [4]:
```
s = 'python'

# in double quotes
t = "python"

# multi-line String
m = '''geek
           for
               geeks'''

print(s)
print(t)
print(m)
```

```
python
python
geek
           for
               geeks
```

Comments in Python

In [5]:
```
# Single Line Comments
```

In [9]:
```
'''
THIS IS A MULTILINE COMMENT
USING STRING LITERALS!
'''
```

Out[9]:
```
'\nDO NOT FORGET TO PROPERLY\nINDENT THE  STARTING OF STRING \nLITERALS WITHIN YOUR CODE! '
```

DocString

In [10]:
```
def my_function():
    '''Demonstrates triple double quotes
    docstrings and does nothing really.'''

    return None

print("Using __doc__:")
print(my_function.__doc__)

print("Using help:")
help(my_function)
```

```
Using __doc__:
Demonstrates triple double quotes
    docstrings and does nothing really.
Using help:
Help on function my_function in module __main__:

my_function()
    Demonstrates triple double quotes
    docstrings and does nothing really.
```

Indentation Error

In [11]:
```
n = 10
for i in range(0,n):
print(i)
```

```
  File "<ipython-input-11-0ce7dd5839d8>", line 3
    print(i)
    ^
IndentationError: expected an indented block
```

Print "Hello World"

In [13]:
```
print("Hello World !")
```

```
Hello World !
```

Single And Multi Line Statements

In [14]:
```
#single line statement
print("This is a trial code")
#multi line statement
print('''Welcome!!
This is Jupter Notebook
Assign 1''')
```

```
This is a trial code
Welcome!!
This is Jupter Notebook
Assign 1
```

In [ ]:
```

```

Name:- Divyang Bagla

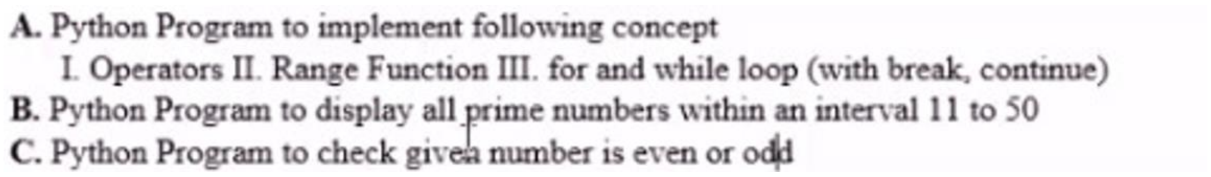
ERP ID:- 1032180739

Batch A2

Roll No. :- PC33

Lab Assignment No. 2

Aim:-



A. Python Program to implement following concept

I. Operators II. Range Function III. for and while loop (with break, continue)

B. Python Program to display all prime numbers within an interval 11 to 50

C. Python Program to check given number is even or odd

Theory:-

Operators

Definition: Operators in general are used to perform operations on values and variables in Python. These are standard symbols used for the purpose of logical and arithmetic operations.

Types of Operators:

- Arithmetic
- Logical
- Relational
- Bitwise
- Assignment
- Special
- Membership

Arithmetic operators: Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

- Addition [+]: adds two operands x + y
- Subtraction [-]: subtracts two operands x - y
- Multiplication [*]: multiplies two operands x * y
- Division (float) [/]: divides the first operand by the second x / y
- Division (floor) [//]: divides the first operand by the second x // y
- Modulus [%]: returns the remainder when first operand is divided by the second x % y
- Power [* *]: Returns first raised to power second x * * y

Relational Operators: Relational operators compares the values. It either returns True or False according to the condition.

- Greater than [>]: True if left operand is greater than the right x > y
- Less than [<]: True if left operand is less than the right x < y
- Equal to [==]: True if both operands are equal x == y
- Not equal to [!=]: True if operands are not equal x != y
- Greater than or equal to [>=]: True if left operand is greater than or equal to the right x >= y
- Less than or equal to [<=]: True if left operand is less than or equal to the right x <= y

Logical operators: Logical operators perform Logical AND, Logical OR and Logical NOT operations.

- Logical AND [and]: True if both the operands are true x and y
- Logical OR [or]: True if either of the operands is true x or y
- Logical NOT [not]: True if operand is false not x

Bitwise operators: Bitwise operators acts on bits and performs bit by bit operation.

- Bitwise AND [&]: x & y
- Bitwise OR [|]: x | y
- Bitwise NOT [~]: ~x
- Bitwise XOR [^]: x ^ y
- Bitwise right shift [>>]: x>>
- Bitwise left shift [<<]: x<<

Assignment operators: Assignment operators are used to assign values to the variables.

- Assign value of right side of expression to left side operand: **x = y + z**
- Add AND: Add right side operand with left side operand and then assign to left operand: **a+=b or a=a+b**
- Subtract AND: Subtract right operand from left operand and then assign to left operand: **a-=b or a=a-b**
- Multiply AND: Multiply right operand with left operand and then assign to left operand: **a*=b or a=a * b**
- Divide AND: Divide left operand with right operand and then assign to left operand: **a/=b or a=a/b**
- Modulus AND: Takes modulus using left and right operands and assign result to left operand: **a%=b or a=a%b**
- Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand: **a//=b or a=a//b**
- Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand: **a ** =b or a=a ** b**
- Performs Bitwise AND on operands and assign value to left operand: **a&=b or a=a&b**
- Performs Bitwise OR on operands and assign value to left operand: **a|=b or a=a|b**
- Performs Bitwise xOR on operands and assign value to left operand: **a^=b or a=a^b**
- Performs Bitwise right shift on operands and assign value to left operand: **a>>=b or a=a>>b**
- Performs Bitwise left shift on operands and assign value to left operand: **a<<=b or a=a<<b**

Special operators: There are some special type of operators like-

Identity operators- is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

- **is:** True if the operands are identical
- **is not:** True if the operands are not identical.

Membership operators: in and not in are the membership operators; used to test whether a value or variable is in a sequence.

- **in:** True if value is found in the sequence
- **not in:** True if value is not found in the sequence

Range Function

Definition: The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax: range(start, stop, step)

Parameter Values:

start (Optional): An integer number specifying at which position to start. Default is 0 stop (Required): An integer number specifying at which position to stop (not included). step (Optional): An integer number specifying the incrementation. Default is 1.

For and While Loop

What is Loop?

Loops can execute a block of code number of times until a certain condition is met. Their usage is fairly common in programming. Unlike other programming language that have For Loop, while loop, dowhile, etc.

What is For Loop?

For loop is used to iterate over elements of a sequence. It is often used when you have a piece of code which you want to repeat "n" number of time.

Syntax:

for iterator_var in sequence:

---statements(s)

What is While Loop?

While Loop is used to repeat a block of code. Instead of running the code block once, It executes the code block multiple times until a certain condition is met.

Syntax:

while expression:

---statement(s)

Break and Continue in For Loop

Break Statement: Allows you to break or terminate the execution of the for loop. It brings control out of the loop.

Continue Statement: It returns the control to the beginning of the loop.

Conclusion:- Thus learned the different operators and loops in python.

Lab Assignment 2 Code

Lab Assignment 2

Operators

Airthmetic Operators

In [25]:

```
#Arithmetic Operators
a = 5
b = 10

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b

# Multiplication of number
mul = a * b

# Division(float) of number
div1 = a / b

# Division(floor) of number
div2 = a // b

# Modulo of both number
mod = a % b

# Power
pow = a ** b

# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
print(pow)
```

```
15
-5
50
0.5
0
5
9765625
```

Relational Operators

In [26]:

```
#Relational Operators
a = 10
b = 5

# a > b is False
print(a > b)

# a < b is True
print(a < b)

# a == b is False
print(a == b)
```

```
# a != b is True
print(a != b)

# a >= b is False
print(a >= b)

# a <= b is True
print(a <= b)
```

```
True
False
False
True
True
False
```

Logical operators

In [27]:

```
#Logical Operators
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

```
False
True
False
```

Bitwise operators

In [28]:

```
#Bitwise operators
a = 10
b = 4

# Print bitwise AND operation
print(a & b)

# Print bitwise OR operation
print(a | b)

# Print bitwise NOT operation
print(~a)

# print bitwise XOR operation
print(a ^ b)

# print bitwise right shift operation
print(a >> 2)

# print bitwise left shift operation
print(a << 2)
```

```
0
14
-11
14
```

```
2
40
```

Special operators

In [29]:

```
#Identity operators
a1 = 3
b1 = 3
a2 = 'Python'
b2 = 'Python'
a3 = [1,2,3]
b3 = [1,2,3]

print(a1 is not b1)

print(a2 is b2)

# Output is False, since lists are mutable.
print(a3 is b3)
```

```
False
True
False
```

Membership operators

In [36]:

```
#Membership operator
x = 'Divyang Bagla'
y = {3:'a',4:'b'}

print('V' in x)

print('D' not in x)

print('DIV' in x)

print('Div' not in x)

print(3 in y)

print('b' in y)
```

```
False
False
False
False
True
False
```

Range Function

In [31]:

```
a = range(6)
for i in a:
    print(i)

print('*****')

b = range(3, 6)
for j in b:
    print(j)

print('*****')
```

```
c = range(3, 20, 2)
for k in c:
    print(k)
```

```
0
1
2
3
4
5
*****
3
4
5
*****
3
5
7
9
11
13
15
17
19
```

For and while loop

In [32]:

```
#FOR LOOP

x=0
for x in range(2,7):
    print(x)

#use a for loop over a collection
Months = ["Jan","Feb","Mar","April","May","June"]
for m in Months:
    print(m)
```

```
2
3
4
5
6
Jan
Feb
Mar
April
May
June
```

Break and Continue in For Loop

In [33]:

```
#use the break and continue statements
for x in range (10,20):
    if (x == 15): break
    if (x % 2 == 0) : continue
    print(x)
```

```
11
13
```

Python Program to display all prime numbers between 11 and 50.

In [34]:

```
start = 11
```

```
end = 50

for i in range(start, end+1):
      if i>1: #All prime numbers are greater than 1
        for j in range(2,i):
            if(i % j==0): #remainder zero, means not prime, hence terminate.
                break
        else:
            print(i)
```

```
11
13
17
19
23
29
31
37
41
43
47
```

Python program to check if a number is even or odd

In [35]:
```
num = int(input("Enter a number: "))

if (num % 2) == 0:
    print(num, "is an even number")
else:
    print(num, "is an odd number")
```

```
Enter a number: 11
11 is an odd number
```

Name:- Divyang Bagla

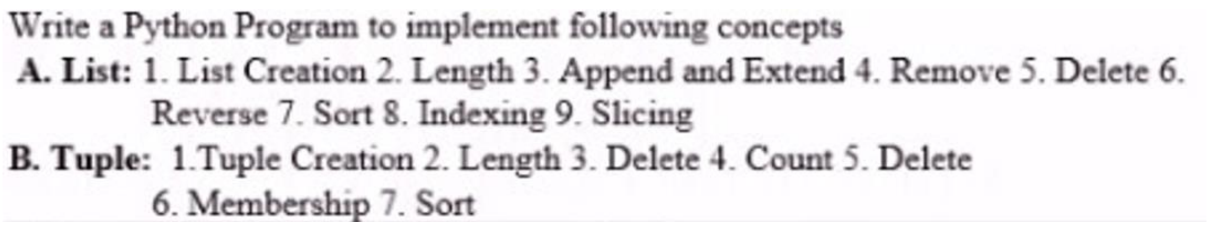
ERP ID:- 1032180739

Batch A2

Roll No. :- PC33

Lab Assignment No. 3

Aim:-



Write a Python Program to implement following concepts

A. List: 1. List Creation 2. Length 3. Append and Extend 4. Remove 5. Delete 6. Reverse 7. Sort 8. Indexing 9. Slicing

B. Tuple: 1.Tuple Creation 2. Length 3. Delete 4. Count 5. Delete 6. Membership 7. Sort

Theory: -

List

Definition: Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets []

List Features:

1. Ordered
2. Mutable
3. Duplicates Allowed

Tuples

Definition: Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

Tuples are created using parentheses ()

Tuple Features:

1. Ordered
2. Immutable
3. Duplicates Allowed

Lab Assignment 3 Code

Lab Assignment 3

List

In [1]:

```
#List Creation
my_list = []

my_list = [1, 2, 3] # list of integers

my_list1 = [1, "Hello", 3.4] # list with mixed data types

my_list2 = ["mouse", [8, 4, 6], ['a']] # nested list
```

In [2]:

```
#List indexing
my_list = [1,2,3,4,5,6,7,8,9,10]

print(my_list[0])

print(my_list[2])

print(my_list[7])

print(my_list[-1])

#Nested List Indexing
n_list = ["Happy", [2, 0, 1, 5]]

print(n_list[0])

print(n_list[0][1])

print(n_list[1][-2])

#print(my_list[4.0]) #Erroneous code as only integer index is allowed
```

```
1
3
8
10
Happy
a
1
```

In [3]:

```
#List editing
#Correcting mistake values in a list
my_list = [1, 4, 6, 8]

print(my_list)

# change the 1st item
my_list[0] = 7

print(my_list)

# change 2nd to 4th items
my_list[1:4] = [3, 5, 7]

print(my_list)
```

```
[1, 4, 6, 8]
```

```
[7, 4, 6, 8]
[7, 3, 5, 7]
```

In [4]:

```
#List Append, Extend and Concatenate
my_list = [1, 2, 5]

my_list.append(7) #Append

print(my_list)

my_list.append([1,2,3,4,5])

my_list.extend([9, 11, 13]) #Extend

print(my_list)

print(my_list + [9, 7, 5]) #Concatenate using +

print(['mylist'] * 3)
```

```
[1, 2, 5, 7]
[1, 2, 5, 7, [1, 2, 3, 4, 5], 9, 11, 13]
[1, 2, 5, 7, [1, 2, 3, 4, 5], 9, 11, 13, 9, 7, 5]
['mylist', 'mylist', 'mylist']
```

In [5]:

```
#insert() in List
odd = [1,3,4,5,6]
odd.insert(1,2)
print(odd)

odd[6:7] = [7, 8]
print(odd)
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [6]:

```
#Delete, Remove, Pop, Clear
my_list = [1,2,3,4,5,6,7,8,9,10]

del my_list[2] #Delete one list item

print(my_list)

del my_list[1:5] #Delete multiple list items

print(my_list)

#del my_list #Delete the entire list
#print(my_list) #Erroneous code as the list is deleted

print('------------')

my_list = ['d','i','v','y','a','n','g']

my_list.remove('i')

print(my_list)

print(my_list.pop(1))

print(my_list)

print(my_list.pop())
print(my_list)

my_list.clear()

print(my_list)
```

```
[1, 2, 4, 5, 6, 7, 8, 9, 10]
[1, 7, 8, 9, 10]
------------
['d', 'v', 'y', 'a', 'n', 'g']
v
['d', 'y', 'a', 'n', 'g']
g
['d', 'y', 'a', 'n']
[]
```

In [7]:

```
#Sort, Count, Reverse
my_list = [2, 7, 5, 8, 1, 6, 0, 8, 4]

print(my_list.index(8))

print(my_list.count(8))

my_list.sort()

print(my_list)

my_list.sort(reverse=True)

print(my_list)

my_list.reverse()

print(my_list)
```

```
3
2
[0, 1, 2, 4, 5, 6, 7, 8, 8]
[8, 8, 7, 6, 5, 4, 2, 1, 0]
[0, 1, 2, 4, 5, 6, 7, 8, 8]
```

In [8]:

```
#List slicing in Python

my_list = ['p','y','t','h','o','n']

# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

```
['t', 'h', 'o']
['p']
['n']
['p', 'y', 't', 'h', 'o', 'n']
```

Tuples

In [9]:
```
my_tuple = ('Divyang','Bagla', 'Python', 1, 2, 3, 4, 5);

print(my_tuple)
```
```
('Divyang', 'Bagla', 'Python', 1, 2, 3, 4, 5)
```

In [10]:
```
#To write a tuple containing a single value you have to include a comma, even though
tup1 = (50,);
print(tup1)
```
```
(50,)
```

In [11]:
```
#Accesing Values in a Tuple
print(my_tuple[0])
print(my_tuple[1][2])
```
```
Divyang
g
```

In [12]:
```
#Update in Tuple
my_tuple = ('Divyang','Bagla', 'Python', 1, 2, 3, 4, 5);
tup1 = (1, 2, 3, 'abc', 'xyz');

#tup1[0] = 100; #Erroneous code since updation is not valid for tuples

# So let's create a new tuple as follows
tup3 = tup1 + my_tuple
print(tup3)
```
```
(1, 2, 3, 'abc', 'xyz', 'Divyang', 'Bagla', 'Python', 1, 2, 3, 4, 5)
```

In [14]:
```
#Delete and Remove
my_tuple = ('Divyang', 'Python', 1, 2, 3, 4, 5)
print(my_tuple)
del my_tuple
#print('After deleting my_tuple:')
#print(my_tuple) #Erroneous code since tuple is deleted

my_tuple1= (1,2,3,4,5,6,7)
#my_tuple1.remove(2) #Erroneous code since tuple is immutable
```
```
('Divyang', 'Python', 1, 2, 3, 4, 5)
```

In [15]:
```
#Length of Tuple
my_tuple = ('Divyang','Bagla', 'Python', 1, 2, 3, 4, 5)
len(my_tuple)
```
Out[15]:
```
8
```

In [16]:
```
#Sort and Count
my_tuple = (1, 5, 7, 1, 9, 10, 2, 6)
#my_tuple.sort() #erroneous since sort() method doesnt work with immutable data type
print(sorted(my_tuple)) #will return sorted list not a tuple
print(my_tuple.count(1)) #will count number of 1's in the tuple
```
```
[1, 1, 2, 5, 6, 7, 9, 10]
2
```

In [18]:
```
#Membership test in tuple
my_tuple = ('d','i','v','y','a','n','g')
```

```
# In operation
print('a' in my_tuple)
print('b' in my_tuple)

# Not in operation
print('g' not in my_tuple)
```

```
True
False
False
```

```
In [ ]:
```

Name:- Divyang Bagla

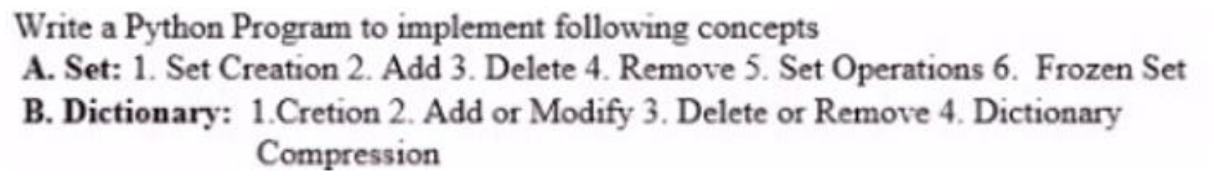
ERP ID:- 1032180739

Batch A2

Roll No. :- PC33

Lab Assignment No. 4

Aim:-



Write a Python Program to implement following concepts

A. Set: 1. Set Creation 2. Add 3. Delete 4. Remove 5. Set Operations 6. Frozen Set

B. Dictionary: 1.Cretion 2. Add or Modify 3. Delete or Remove 4. Dictionary Compression

Theory:-

Sets

Definition: Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is both unordered and unindexed.

Sets are written with curly brackets {}.

Set Features:

1. Unordered
2. Unindexed
3. Mutable
4. Duplicates Allowed

Dictionary

Definition: Dictionary in Python is an ordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds key:value pair.

Key-value is provided in the dictionary to make it more optimized.

Dictonary is written with curly brackets {}.

Dictionary Features:

1. Dictionaries are unordered. A dictionary contains key-value pairs but does not possess an order for the pairs.
2. Keys are unique. Dictionary keys must be unique.
3. Keys must be immutable.

Lab Assignment 4 Code

Lab Assignment 4

Sets

In [1]:
```
#Set Creation
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set1 = {1.0, "Hello", (1, 2, 3)}
print(my_set1)
```

```
{1, 2, 3}
{1.0, (1, 2, 3), 'Hello'}
```

In [2]:
```
#Set cannot have duplicates
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)

my_set = set([1, 2, 3, 2]) #We can make set from a list
print(my_set)

#my_set1 = {1, 2, [3, 4]} #Erroneous Code as set cannot have mutable items. Here [3,
#print(my_set1)
```

```
{1, 2, 3, 4}
{1, 2, 3}
```

In [3]:
```
#Distinguish set and dictionary while creating empty set
#initialize a with {}
a = {}

#check data type of a
print(type(a))

#initialize a with set()
a = set()

#check data type of a
print(type(a))
```

```
<class 'dict'>
<class 'set'>
```

In [4]:
```
#Modifying a Set: Add, Update.
my_set = {1, 3}
print(my_set)

#my_set[0] #Erroneous Code: TypeError as 'set' object does not support indexing

my_set.add(2) # add an element
print(my_set)

my_set.update([2, 3, 4]) # add multiple elements
print(my_set)

my_set.update([4, 5], {1, 6, 8}) # add list and set
print(my_set)
```

```
{1, 3}
```

```
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

In [5]:
```
#Discard and Remove

my_set = {1, 3, 4, 5, 6}
print(my_set)

my_set.discard(4) #Discard an element
print(my_set)

my_set.remove(6) #Remove an element
print(my_set)

my_set.discard(2) #Discard an element not present in my_set
print(my_set)

#my_set.remove(2) #Erroneous Code
```

```
{1, 3, 4, 5, 6}
{1, 3, 5, 6}
{1, 3, 5}
{1, 3, 5}
```

In [6]:
```
#Pop and Clear
my_set = set("HelloWorld")
print(my_set)

print(my_set.pop()) # pop an element

my_set.pop() # pop another element
print(my_set)

my_set.clear() # clear my_set
print(my_set)
```

```
{'e', 'd', 'r', 'W', 'H', 'l', 'o'}
e
{'r', 'W', 'H', 'l', 'o'}
set()
```

In [7]:
```
#Set Operations

A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

print(A | B) #Set Union
print(B.union(A))
print(A & B) #Set Intersection
print(A.intersection(B))
print(A-B) #Set Difference
print(A.difference(B))
print(A ^ B) #Set Symmetric Difference

A.isdisjoint(B) #To check for disjoint sets
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{1, 2, 3, 4, 5, 6, 7, 8}
{4, 5}
{4, 5}
{1, 2, 3}
{1, 2, 3}
{1, 2, 3, 6, 7, 8}
```

```
Out[7]: False
```

```
In [8]:
#Frozen Set

vowels = ('a', 'e', 'i', 'o', 'u') #Tuple of vowels

fSet = frozenset(vowels)
print('The frozen set is:', fSet)

#fSet.add('v') #Erroneous Code as frozensets are immutable
```

```
The frozen set is: frozenset({'u', 'e', 'a', 'i', 'o'})
```

Dictionary

```
In [10]:
#Dictionary Creation
my_dict = {}

my_dict = {1: 'Divyang', 2: 'Bagla'} # dictionary with integer keys

my_dict1 = {'name': 'Python', 1: [2, 4, 3]} # dictionary with mixed keys

my_dict2 = dict({1:'Python', 2:'Lab'}) # using dict()

my_dict3 = dict([(1,'Sets'), (2,'Dicts')]) # from sequence having each item as a pai

print(my_dict)
print(my_dict1)
print(my_dict2)
print(my_dict3)
```

```
{1: 'Divyang', 2: 'Bagla'}
{'name': 'Python', 1: [2, 4, 3]}
{1: 'Python', 2: 'Lab'}
{1: 'Sets', 2: 'Dicts'}
```

```
In [11]:
#get vs [] for retrieving elements

my_dict = {'firstname': 'Divyang', 'lastname': 'Bagla', 'age': 20}

print(my_dict['firstname'])

print(my_dict.get('age'))

print(my_dict.get('address'))

#print(my_dict['address']) #Erroneous Code: The address key doesn't exist.
```

```
Divyang
20
None
```

```
In [12]:
#Changing and adding Dictionary Elements
my_dict = {'firstname': 'Divyang', 'lastname': 'Bagla', 'age': 20}

print(my_dict)

my_dict['age'] = 24 #Update age value

print(my_dict)

my_dict['address'] = 'Downtown Los Angeles' #Add item
```

```
print(my_dict)
```

```
{'firstname': 'Divyang', 'lastname': 'Bagla', 'age': 20}
{'firstname': 'Divyang', 'lastname': 'Bagla', 'age': 24}
{'firstname': 'Divyang', 'lastname': 'Bagla', 'age': 24, 'address': 'Downtown Los An
geles'}
```

In [13]:

```
#Removing elements from a dictionary

squares= {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

print(squares.pop(4)) #Remove a particular item, returns its value

print(squares)

print(squares.popitem()) #Remove an arbitrary item, return (key,value)

print(squares)

squares.clear() #Remove all items

print(squares)

del squares #Delete the entire dictionary

#print(squares) #Erroneous code as dictionary doesnt exist.
```

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
```

In [14]:

```
square_dict = dict()
for num in range(1, 11):
    square_dict[num] = num*num
print(square_dict)

# Dictionary Comprehension
squares = {x: x*x for x in range(1, 11)}

print(squares)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

In [15]:

```
#2nd Example
#item price in dollars
old_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
print(old_price)

dollar_to_pound = 0.76
new_price = {item: value*dollar_to_pound for (item, value) in old_price.items()}
print(new_price)
```

```
{'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
{'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}
```

In [16]:

```
#3rd Example: Conditional Dictionary Comprehension

original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}
print(original_dict)
```

```
even_dict = {k: v for (k, v) in original_dict.items() if v % 2 == 0} #only the items
#because of the if clause in the dictionary comprehension.
print(even_dict)

new_dict = {k: v for (k, v) in original_dict.items() if v % 2 != 0 if v < 40} #only
#of less than 40 have been added to the new dictionary.
print(new_dict)

new_dict_1 = {k: ('old' if v > 40 else 'young')
    for (k, v) in original_dict.items()}
print(new_dict_1)
```

```
{'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}
{'jack': 38, 'michael': 48}
{'john': 33}
{'jack': 'young', 'michael': 'old', 'guido': 'old', 'john': 'young'}
```

In [ ]: