Experiment No : 01

Aim : Write python programs to understand Basic data types, Operators, expressions and Input Output Statements

Description :

Basic data types:

Python program to demonstrate basic data-type in python Data type defines the kind of a value i.e the type of value whether it is int, string, float etc.

Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

We don't need to define data types in python because python is dynamically typed language, it automatically detects data types.

Python enables us to check the type of the variable used in the program.

Python provides us the type() function, which returns the type of the variable passed.

python data types:

Numbers: Represent numeral values like int, float, complex

Int : It contains positive or negative whole numbers. eg: 1,2,3,-1.....

Float : Contains real floating point representation . eg: 2.0,1.0,8.02....

Complex : Numbers of form Ai+Bj . eg: 2i+3j

Code :

```
x = 20     #int

print(x)

print(type(x))

x = 20.5 #float

print(x)

print(type(x))

x = 1j       #complex

print(x)

print(type(x))

"""
Sequence Type:

String : A string is defined as a collection of one or more characters.
 It is putted in single quotes or double quotes or triple quotes i.e (' ') , (" ") or (' ' '  ' ' ')  .
 for eg: s="PYTHON LAB"

"""
#String

x = "PYTHON LAB"

print(x)

print(type(x))

#List : It is as same as arrays, it contains heterogeneous datatypes. eg: a=[1,2,"a"]

x = ["Go", "Edu", "Hub"]      #list

print(x)

print(type(x))

#Tuple :  Tuples are created by ( ) . it contains heterogeneous datatypes . eg: t=(1,2,3)

x = ("Python", "from", "GoEduHub.com")      #tuple

print(x)

print(type(x))

"""
```

Dictionary: Dictionary is an unordered collection of data. It has key:value pair
which is separated by a colon :, and all keys are separated by a ‘,’. eg : d={'a':1, 'b':2, 'c':3}

"""

x = {"name" : "ABC", "age" : 36} #dict

print(x)

print(type(x))
#Boolean: Booleans represent one of two values: True or False.

x = True #bool

print(x)

print(type(x))

x = b"Hello" #bytes

print(x)

print(type(x))

"""

Set : Set is an unordered and unique collection of data types that mutable and has distinct elements. eg:
s={1,2,3,4,5}

Frozen set : Frozen sets are the sets that are immutable .

"""

x = {"dl", "python", "ml"}   #set

print(x)

print(type(x))

x = frozenset({"opencv", "pandas", "numpy"}) #frozenset

print(x)

print(type(x))

OUTPUT

#Operators:
a=30
b=10
c=0
#print("arithmetic operator")

```
c=a+b
print(c)
c=a-b
print(c)
c=a*b
print(c)
c=a/b
print(c)
c=a%b
print(c)
a=2
b=3
c=a**b
print(c)
a=12
b=5
c=a//b
print(c)


a=30
b=10
#print("comparison operator")
if(a==b):
 print("a is equal to b")
else:
 print("a is not equal to b")
if(a!=b):
 print("a is not equal to b")
else:
 print("a is equal to b")
if(a<b):
 print("a is less than b")
else:
 print("a is not less than b")
if(a>b):
 print("a is greater than b")
else:
 print("a is not greater than b")
a=5
b=10
if(a<=b):
 print("a is either less than or equal to b")
else:
 print("a is neither less than nor equal to b")
if(b>=a):
 print("b is either greater than or equal to a")
else:
 print("b is neither greater than nor equal to a")

print("assignment operator")
a=5
b=10
c=0
```

```
c=a+b                #15
print(c)
c+=a                # c=c+a  20
print(c)
c*=a               # c=c*a  100
print(c)
c/=a
print(c)
c=2
c%=a        # 2/5, o,    2
print(c)
c**=a      # c=c**a  32
print(c)
c//=a       # c=c//a  32/5=6.4
print(c)

print("bitwise operator")    # 32  16 8 4
a=60                #60= 0011 1100
b=13                #13= 0000 1101
c=0
c=a&b                #0000 1100 =12
print(c)
c=a|b                #0011 1101 =61
print(c)
c=a^b              #xor 0011 0001 =49
print(c)
c=~a                #1100 0011
print(c)
c=a<<2              #1111 0000
print(c)
c=a>>2               #0000 1111
print(c)

print(" membership operator  in and not in")
abc="i like cricket"
if "like" in abc:
 print("yes")
if "dislike" not in abc:
 print("yes")
x=2000
y=2000
print(id(x),id(y))
print(x is y)
print( x is not y)
print( x==y)



# OUTPUT


# Expressions:
"""
```

Expressions in Python

An expression is a combination of operators and operands that is interpreted to produce some other value. In any programming language, an expression is evaluated as per the precedence of its operators. So that if there is more than one operator in an expression, their precedence decides which operation will be performed first. We have many different types of expressions in Python. Let's discuss all types along with some exemplar codes :

1. Constant Expressions: These are the expressions that have constant values only.

Example:

```
"""

# Constant Expressions
x = 15 + 1.3

print(x)

"""
```

2. Arithmetic Expressions: An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis. The result of this type of expression is also a numeric value.
The operators used in these expressions are arithmetic operators like addition, subtraction, etc.
Here are some arithmetic operators in Python:

Operators	Syntax	Functioning
+	x + y	Addition
–	x – y	Subtraction
*	x * y	Multiplication
/	x / y	Division
//	x // y	Quotient
%	x % y	Remainder
**	x ** y	Exponentiation

Example:

Exemplar code of arithmetic expressions in Python :

```
"""

# Arithmetic Expressions
x = 40
y = 12

add = x + y
sub = x - y
pro = x * y
div = x / y

print(add)
print(sub)
print(pro)
print(div)

"""
```

3. Integral Expressions: These are the kind of expressions that produce only integer results

after all computations and type conversions.

Example:

"""

# Integral Expressions
a = 13
b = 12.0

c = a + int(b)
print(c)

"""

4. Floating Expressions: These are the kind of expressions which produce floating point numbers as result after all computations and type conversions.

Example:

"""

# Floating Expressions
a = 13
b = 5

c = a / b
print(c)

"""

5. Relational Expressions: In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=). Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a boolean output in the end. These expressions are also called Boolean expressions.

Example:

"""

# Relational Expressions
a = 21
b = 13
c = 40
d = 37

p = (a + b) >= (c - d)
print(p)

"""

6. Logical Expressions: These are kinds of expressions that result in either True or False. It basically specifies one or more conditions. For example, (10 == 9) is a condition if 10 is equal to 9. As we know it is not correct, so it will return False.

Studying logical expressions, we also come across some logical operators
which can be seen in logical expressions most often. Here are some logical operators in Python:

Operator	Syntax	Functioning
and	P and Q	It returns true if both P and Q are true otherwise returns false
or	P or Q	It returns true if at least one of P and Q is true
not	not P	It returns true if condition P is false

Example:

```
"""

P = (10 == 9)
Q = (7 > 5)

# Logical Expressions
R = P and Q
S = P or Q
T = not P

print(R)
print(S)
print(T)

"""
```

7. Bitwise Expressions: These are the kind of expressions in which computations are performed at bit level.

Example:

```
"""

# Bitwise Expressions
a = 12

x = a >> 2
y = a << 1

print(x, y)

"""
```

8. Combinational Expressions: We can also use different types of expressions in a single expression,
and that will be termed as combinational expressions.

Example:

```
"""
# Combinational Expressions
a = 16
b = 12

c = a + (b >> 1)
print(c)
```

"""

But when we combine different types of expressions or use multiple operators in a single expression, operator precedence comes into play.

Multiple operators in expression (Operator Precedence)
It's a quite simple process to get the result of an expression if there is only one operator in an expression. But if there is more than one operator in an expression, it may give different results on basis of the order of operators executed. To sort out these confusions, the operator precedence is defined.
Operator
Precedence simply defines the priority of operators
that which operator is to be executed first. Here we see the operator precedence in Python, where the operator higher in the list has more precedence or priority:

Precedence	Name	Operator
1	Parenthesis	( ) [ ] { }
2	Exponentiation	**
3	Unary plus or minus, complement	-a , +a , ~a
4	Multiply, Divide, Modulo	/ * // %
5	Addition & Subtraction	+ –
6	Shift Operators	>> <<
7	Bitwise AND	&
8	Bitwise XOR	^
9	Bitwise OR	|
10	Comparison Operators	>= <= > <
11	Equality Operators	== !=
12	Assignment Operators	= += -= /= *=
13	Identity and membership operators	is, is not, in, not in
14	Logical Operators	and, or, not

So, if we have more than one operator in an expression, it is evaluated as per operator precedence. For example, if we have the expression "10 + 3 * 4". Going without precedence it could have given two different outputs 22 or 52. But now looking at operator precedence, it must yield 22. Let's discuss this with the help of a Python program:

"""
Multi-operator expression

```
a = 10 + 3 * 4
print(a)

b = (10 + 3) * 4
print(b)

c = 10 + (3 * 4)
print(c)
```

#Hence, operator precedence plays an important role in the evaluation of a Python expression.

OUTPUT

"""

Input and Output statements in Python

How to Take Input from User in Python
Sometimes a developer might want to take user input at some point in the program. To do this Python provides an input() function.

Syntax:

"""
input('prompt')

"""

where prompt is an optional string that is displayed on the string at the time of taking input.

Example 1: Python get user input with a message

"""
Taking input from the user

name = input("Enter your name: ")

# Output
print("Hello, " + name)
print(type(name))

"""
Note: Python takes all the input as a string input by default. To convert it to any other data type we have to convert the input explicitly. For example, to convert the input to int or float we have to use the int() and float() method respectively.

Example 2: Integer input in Python

"""
Taking input from the user as integer

num = int(input("Enter a number: "))

add = num + 1

# Output
print(add)

"""
How to Display Output in Python
Python provides the print() function to display output to the standard output devices.

Syntax: print(value(s), sep= ' ', end = '\n', file=file, flush=flush)

Parameters:
value(s) : Any value, and as many as you like. Will be converted to string before printed
sep='separator' : (Optional) Specify how to separate the objects, if there is more than one.Default :' '
end='end': (Optional) Specify what to print at the end.Default : '\n'

file : (Optional) An object with a write method. Default :sys.stdout
flush : (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Returns: It returns output to the screen.

Example: Python Print Output
"""

```
# Python program to demonstrate
# print() method

print("GFG")

# code for disabling the softspace feature
print('G', 'F', 'G')
```

"""
in the case of the 2nd print statement there is a space between every letter and the print statement always add a new line character at
the end of the string. This is because after every character the sep parameter is printed and at the end of the string the end parameter is printed.
Let's try to change this sep and end parameter.

Example: Python Print output with custom sep and end parameter
"""

```
# Python program to demonstrate
# print() method

print("GFG", end = "@")

# code for disabling the softspace feature
print('G', 'F', 'G', sep="#")
```

"""
Formatting Output
Formatting output in Python can be done in many ways. Let's discuss them below

Using formatted string literals
We can use formatted string literals, by starting a string with f or F before opening quotation marks or triple quotation marks. In this string, we can write Python expressions between { and } that can refer to a variable or any literal value.

Example: Python String formatting using F string

"""

```
# Declaring a variable
name = "Gfg"

# Output
print(f'Hello {name}! How are you?')
```

"""
We can also use format() function to format our output to make it look presentable.
The curly braces { } work as placeholders. We can specify the order in which variables occur in the output.

Example: Python string formatting using format() function
"""

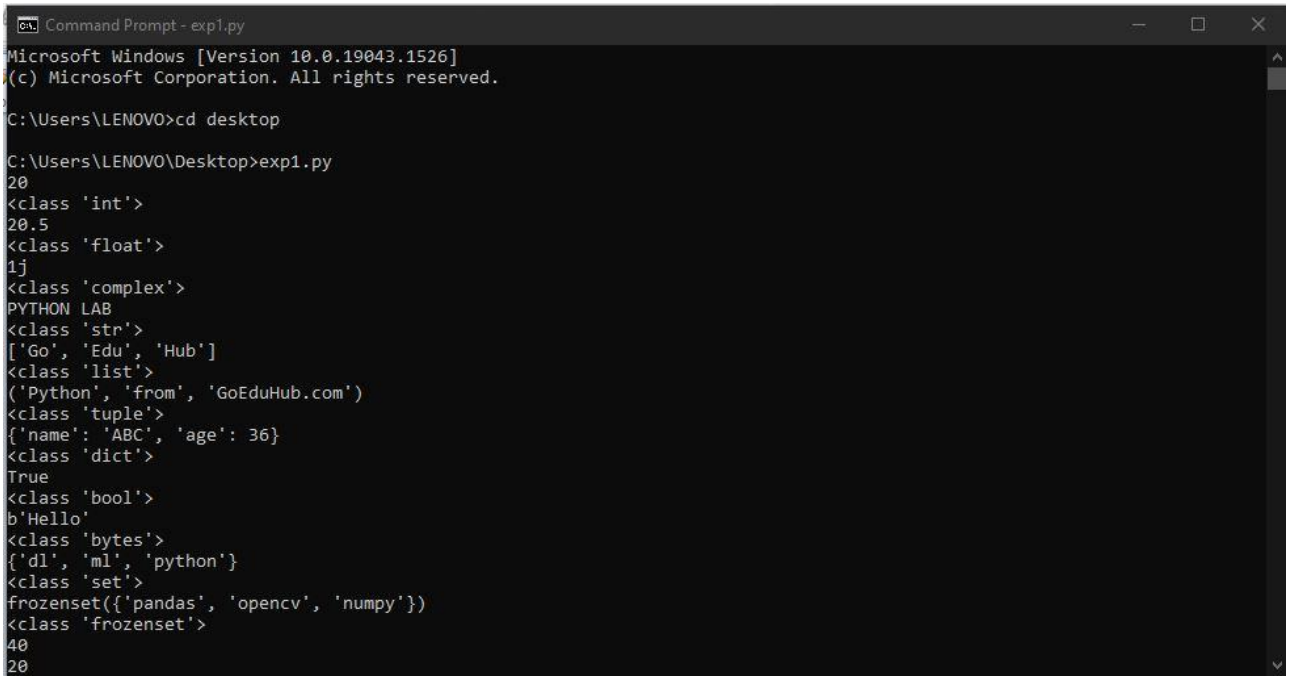
```
# Initializing variables
a = 20
b = 10

# addition
sum = a + b

# subtraction
sub = a- b

# Output
print('The value of a is {} and b is {}'.format(a,b))

print('{2} is the sum of {0} and {1}'.format(a,b,sum))

print('{sub_value} is the subtraction of {value_a} and {value_b}'.format(value_a = a ,
                                                    value_b = b,


                                                sub_value = sub))
```

"""
Using % Operator
We can use '%' operator. % values are replaced with zero or more value of elements. The formatting using % is similar to that of 'printf' in the C programming language.
%d – integer
%f – float
%s – string
%x – hexadecimal
%o – octal
Example:

"""

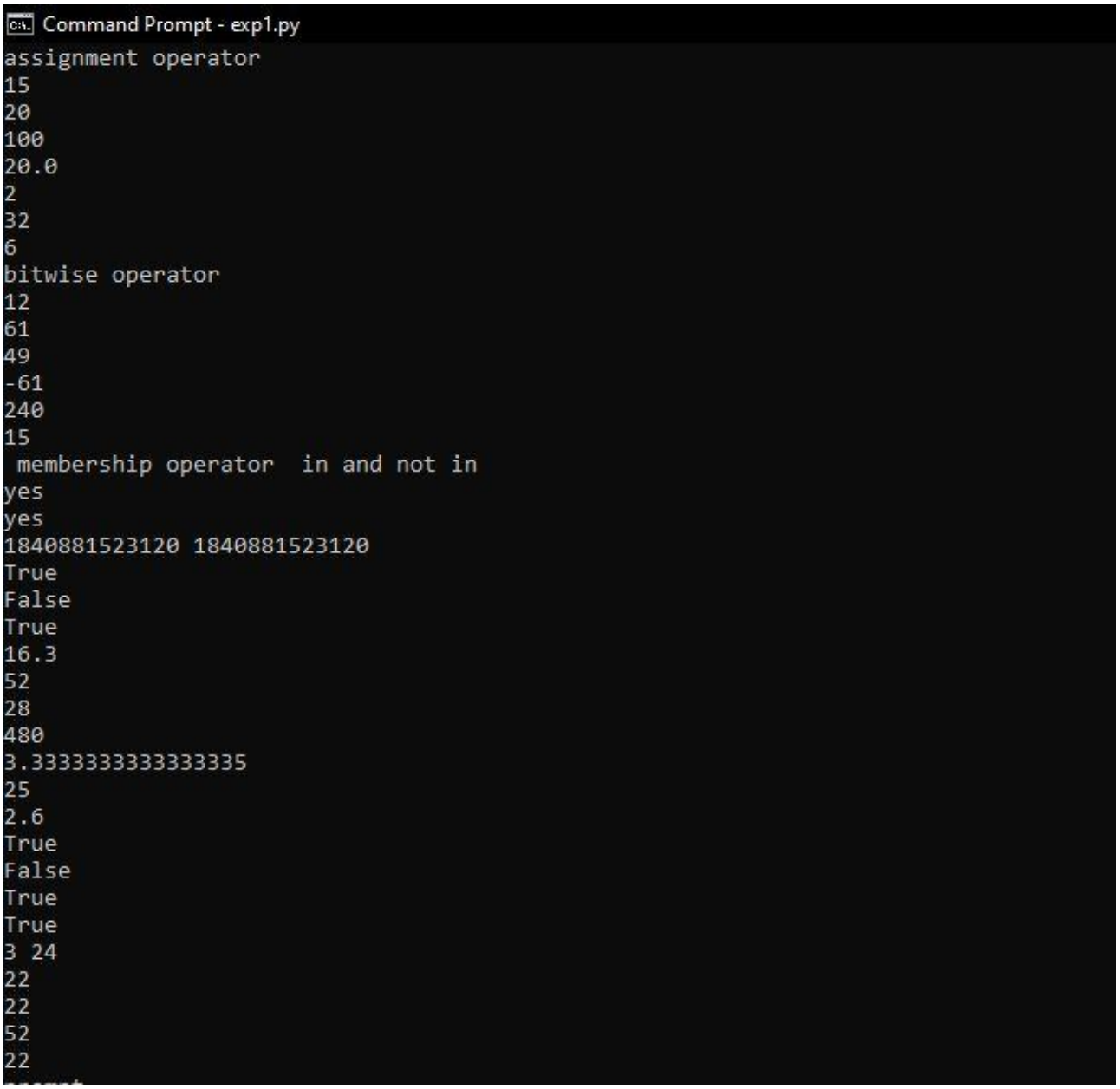
```
# Taking input from the user
num = int(input("Enter a value: "))

add = num + 5
```

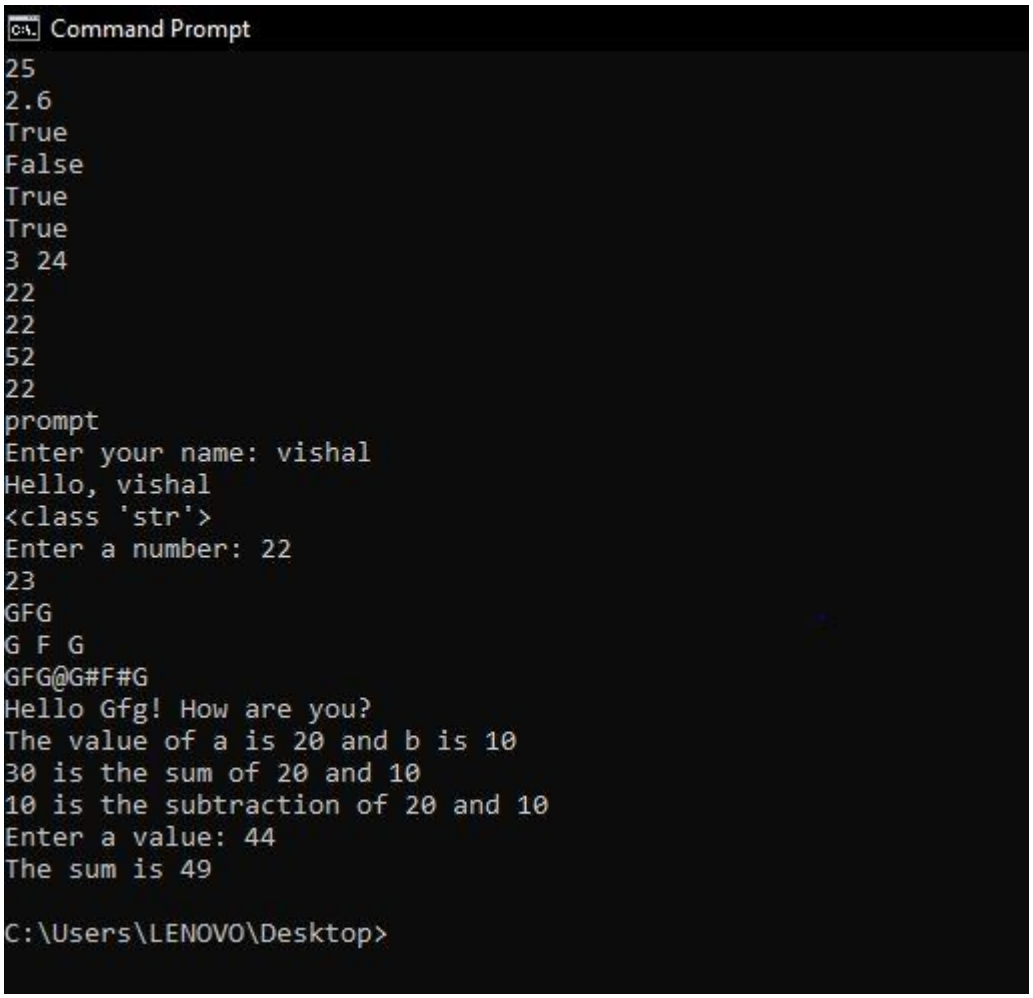
Output:


Command Prompt - exp1.py
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.
C:\Users\LENOVO>cd desktop
C:\Users\LENOVO\Desktop>exp1.py
20
<class 'int'>
20.5
<class 'float'>
1j
<class 'complex'>
PYTHON LAB
<class 'str'>
['Go', 'Edu', 'Hub']
<class 'list'>
('Python', 'from', 'GoEduHub.com')
<class 'tuple'>
{'name': 'ABC', 'age': 36}
<class 'dict'>
True
<class 'bool'>
b'Hello'
<class 'bytes'>
{'dl', 'ml', 'python'}
<class 'set'>
frozenset({'pandas', 'opencv', 'numpy'})
<class 'frozenset'>
40
20

Command Prompt - exp1.py

```
<class 'frozenset'>
40
20
300
3.0
0
8
2
a is not equal to b
a is not equal to b
a is not less than b
a is greater than b
a is either less than or equal to b
b is either greater than or equal to a
assignment operator
15
20
100
20.0
2
32
6
bitwise operator
12
61
49
-61
240
15
 membership operator  in and not in
yes
yes
1840881523120 1840881523120
True
False
True
16.3
52
28
480
3.3333333333333335
25
2.6
True
```


Command Prompt - exp1.py
assignment operator
15
20
100
20.0
2
32
6
bitwise operator
12
61
49
-61
240
15
membership operator in and not in
yes
yes
1840881523120 1840881523120
True
False
True
16.3
52
28
480
3.3333333333333335
25
2.6
True
False
True
True
3 24
22
22
52
22



Conclusion : Therefore we have successfully done the Basic data types, Operators, expressions and Input Output Statements in python programming language.