Dharmsinh Desai University, Nadiad
Faculty of technology,
Department of Computer Engineering
Subject : Software Project
Lab Manual

Lab 1 : Basic operations and control structures in Python.

Aim : To introduce and learn basic operations and control structures in Python.

Requirements : Python

- **Python supports the following types of operators**:
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators
  - Arithmetic Operators

- In python, we can use following arithmetic operators.

	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponent
//	Floor Division[#]

[#] Floor division will remove the digits after decimal point in answer and if any value in operation is negative than in resultant value is floored.

Example:

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4 2017, 14:34:30)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+5
8
>>> 5-2
3
>>> 5*5
25
>>> 5/3
1.6666666666666667
>>> 5**2
25
>>> 5%2
1
>>> 5//3
1
>>>
```

❖ Comparison Operators

==	Equal to
!= or <>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Example:

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4 2017, 14:34:30)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1==1
True
>>> 1!=1
False
>>> 1<2
True
>>> 1>4
False
>>> 1>=1
True
>>> 1<=2
True
```

❖ Assignment Operator:

=	Assign a value of right side variable to left side

	variable
+=	Shorthand +
-=	Shorthand -
*=	Shorthand *
/=	Shorthand /
%=	Shorthand modulo
**=	Shorthand exponent
//=	Shorthand floor division

Example :

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4 2017, 14:34:30)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=5
>>> a
5
>>> a+=3
>>> a
8
>>> a**=2
>>> a
64
>>> a//=4
>>> a
16
>>>
```

❖ **Logical Operator:**

and	Logical And
or	Logical Or
not	Logical Not

Example :

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4 2017, 14:34:30)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> True and False
False
>>> True or False
True
>>> not True
False
>>>
```

❖ **Bitwise Operator:**

&	Binary AND
|	Binary OR
^	Binary EX-OR
~	Binary Ones Complement
<<	Binary Left Shift
>>	Binary Right Shift

Example :

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4 2017, 14:34:30)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> 3&5
1
>>> 3|6
7
>>> 2^6
4
>>> ~7
-8
>>>
```

❖ **Membership Operator:**

In	Evaluates to true if it finds a variable in the specified sequence and false otherwise.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

Example :

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4 2
[GCC 4.8.5 20150623 (Red Hat 4.
Type "help", "copyright", "cred
>>> 'hello' in "hello india"
True
```

❖ **Identity Operator:**

is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

Example :

```
[vivek@localhost ~]$ python3.5
Python 3.5.1 (default, Dec  4
[GCC 4.8.5 20150623 (Red Hat 4
Type "help", "copyright", "cre
>>> 'abc' is 'abc'
True
```

❖ **Variable**

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

Example:

- **Number:** Python supports four different types of numbers : Integer, Long, Float and Complex. Some examples of number are as follow

int	long	float	complex
10	51924361L	0.0	2.1+3.14j
100	-0x19323L	15.20	1+2j
-786	0122L	-21.9	9.322e-36j

- **String**

In Python, String are defined either with a single quote or a double quote.

i.e. a=”Hello” or a='hello'

The difference between the two is that using double quotes makes it easy to include apostrophes (whereas these would terminate the string if using single quotes)

i.e. mystring = "Don't worry about apostrophes"

Example:

```
>>> "'Isn\'t," she said.'
"'Isn\'t," she said.'
>>> print("'Isn\'t," she said.')
"Isn't," she said.
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

•If you don’t want characters prefaced by \ to be interpreted as special characters, you can use *raw strings* by adding an r before the first quote:

```
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

•String literals can span multiple lines. One way is using triple-quotes: """..."""" or '''…'''

```
>>> print(''' CE Department ,
                Dharmsinh Desai University,
                College Road, Nadiad''')
```

•Strings can be *indexed* (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one.

•Indices may also be negative numbers, to start counting from the right. -1 is assigned to last letter

```
>>>name=”Hello DDU”
>>>name[1]
'e'
>>>name[-1]
'U'
```

•Even Slicing is also supported by python

```
>>>name[1:3]    # First index is included and second index is excluded
'el'
>>>name[:3]
'Hel'
>>>name[6:]
'DDU'
```

➢ Python strings cannot be changed — they are immutable. Therefore, assigning to an indexed position in the string results in an error

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

```
>>>name[1]='l'
```

❖ The built-in function len() returns the length of a string.

```
>>>len(name)
9
```

Control Structure:

❖ **if :** Syntax of if statement in python is as below

```
if condition :
        statement
```

Example :

```
n=input("Enter number")
if int(n)>0:
        print(str(n)+" is positive")
```

❖ **if else:** Syntax of if else statement is as below

```
if condition :
        statement
else:
        statement
```

Example:

```
n=input("Enter number")
if int(n)>0:
        print(n+" is positive")
else:
        print(n+" is negative")
```

❖ **elif:** Syntax of elif statement is as below

```
if condition :
        statement
elif condition :
        statement
else :
        statement
```

Example :

```
n=input("Enter number")
if int(n)>0:
        print(n+" is positive")
elif int(n)<0:
        print(n+" is negative")
else:
        print(n)
```

❖ **Nested if else:** Syntax for nested if else is as below (Here else part is of outer if statement):

```
if condition :
        if condition:
                statement
        else :
                statement
```

Example:

```
n=input("Enter number")
if int(n)>0:
        if int(n)%2==0 :
                print(n+" is positive even no")
        else:
                print(n+" is positive odd no")
elif int(n)<0:
        print(n+" is negative")
else:
        print(n)
```

- In python indentation is important for control statement. Either if or else block contains multiple statements then also braces are not required with proper indentation we can put multiple statements inside if or else block

```
if condition:
        statement-1
        statement-2
statement-3
```

Here statement-1 and statement-2 executes if condition becomes true means both statement (1 and 2) are in if block. While statement-3 is outside of if block.(statement-3 executes whether condition is true or false).

❖ **Looping statements**

Python provides two ways for executing the loops.

- While loop
- For loop

❖ **While Loop**

In python, while loop is used to execute a block of statements repeatedly until a given a condition is satisfied. And when the condition becomes false, the line immediately after the loop in program is executed.

Syntax:

```
while expression:
        statements
```

Example:

```
n=0
while n<10:
        print(n)
        n=n+1
print("End of loop")
```

- Same as if statement in loop also indentation is important. In above example, statements indented by same space are consider as loop body.
- In python we can use else statement with while loop. As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed. The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

Syntax:

```
while condition :
        statements
else :
        statements
```

Example:

```
n=0
while n<10:
        print(n)
        n=n+1
else :
        print("count is greater than 9")
print("End of loop")
```

- **For Loop:** For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is "for in" loop which is similar to for each loop in other languages. Let us learn how to use for in loop for sequential traversals.

Syntax :

for index in sequence :
statements

Example :

```
for index in [1,2,3,4,5]:
        print(index)
```

- We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution. Syntax for same is as below

Syntax:

for index in sequence :
statements
else:
statements

Example:

```
for index in [1,2,3,4,5]:
        print(index)
else:
        print("Inside Else block")
```

- **Nested Loop:** Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax :

for var in sequence:
for var in sequence:
statements

while expression:
while expression:

statements

Example:

```
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

- **Loop Control Statements:** Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.
  - **Continue Statement:** it returns the control to the beginning of loop

    Example:

```
for letter in 'Helloworld':
        if letter=='l':
                continue;
        print(letter)
```

  - **Break Statement:** It brings control out of the loop

    Example:

```
for letter in 'Helloworld':
        if letter=='l':
                break;
        print(letter)
```

  - **Pass Statement:**

    We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

```
for letter in 'helloworld':
    pass
print 'Last Letter :', letter
```

Excercise:

[1]. Write a program which accepts a number from user and display that number if and only if it is positive value.

[2]. Write a program which accepts average marks of student and if average is greater than 40 then it will display a message “Congratulation!!! You pass this exam successfully” else it will display “Sorry!!! Better luck next time”.

[3]. Write a program which accepts marks of five subject and based on average of those marks it will display appropriate grade.

[4]. Write a program to find largest number out of three numbers entered by user.

[5]. Check the output of following code and try to justify the output:

i).

```
a=-5
if a>0:
        if a%2==0 :
                print(str(a) + " is positive even number")
else:
        print(str(a) + " is negative number")
```

ii).

```
a=5
if a>0:
        if a%2==0 :
                print(str(a) + " is positive even number")
        else:
                print(str(a) + " is odd number")
```

- For the below elements, print binary of modulo 64 (remainder) result.(Hint: Use function bin and operator %).

  1111,3333,1235,2378,1212,1456,2134,2345,1111,8231,2222,9999

Expected Output:

1111 23 0b10111
3333 5 0b101
1235 19 0b10011
2378 10 0b1010
1212 60 0b111100
1456 48 0b110000
2134 22 0b10110
2345 41 0b101001
1111 23 0b10111
8231 39 0b100111
2222 46 0b101110
9999 15 0b1111

- Write a program to check whether entered number is prime or not.
- Write a program to display armstrong numbers between 1 to n.(where value of n will be provided by user).