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
```