Introduction to Python

Topics

- Basic Rules
- Declaring & Printing variables
- Using built-in functions
- Basic Operators: Arithmetic using binary operators
- Dealing with Strings
- Python Operations
- Control Flow with if, elif, else
- Loop
- Working with Collections - List, Tuple, Set & Dictionary
- Functions – User defined functions
- Lambda functions

Basic Rules

- **Case Sensitivity**

In python language ‘e’ and ‘E’ are different. Every python object is case sensitive

- **Comments:**

Anything preceded by the hash mark (#) is ignored by the Python interpreter.

- **Whitespace Formatting (Indentation):**

Curly braces, {} is used for delimiting the loops/blocks of codes in many languages. Python uses whitespace indentation (a tab space) for the same.

A colon denotes the start of an indented code block after which all of the code must be indented by the same amount

Python uses indentation for blocks, instead of curly braces. Both tabs and spaces are supported.

Declaring & Printing variables

- Variable are dynamically typed, so no need to mention the variable types. Python interpreter can automatically infer the type when the variables are initialized. The simplest directive in Python is the "print" directive - it simply prints out a line

```
In [1]: var1 = 2
        var2 = 5.0

In [2]: var1
Out[2]: 2

In [3]: type( var1 )
Out[3]: int

In [4]: type( var2 )
Out[4]: float
```

```
In [5]: print( var1 )
        2

In [6]: mystring = 'This is python'
        print( mystring )
        This is python

In [7]: print( var1, var2, mystring )
        2 5.0 This is python
```

Using built-in functions

- Functions comes with python base version, called built in functions.

Example: round()

```
In [12]: round( 1.234 )
Out[12]: 1
```

Round upto a number of decimal values

```
In [13]: round( 1.234, 2 )
Out[13]: 1.23
```

- To invoke some functions from the package
- For example import a math function

```
In [14]: import math

In [15]: math.ceil( 1.2 )
Out[15]: 2

In [16]: math.floor( 1.2 )
Out[16]: 1
```

```
In [17]: abs( -1.2 )
Out[17]: 1.2

In [18]: # Get the variable type
         type( var1 )
Out[18]: int

In [19]: pow( var1 , 2 )
Out[19]: 4
```

Basic Operators: Arithmetic using binary operators

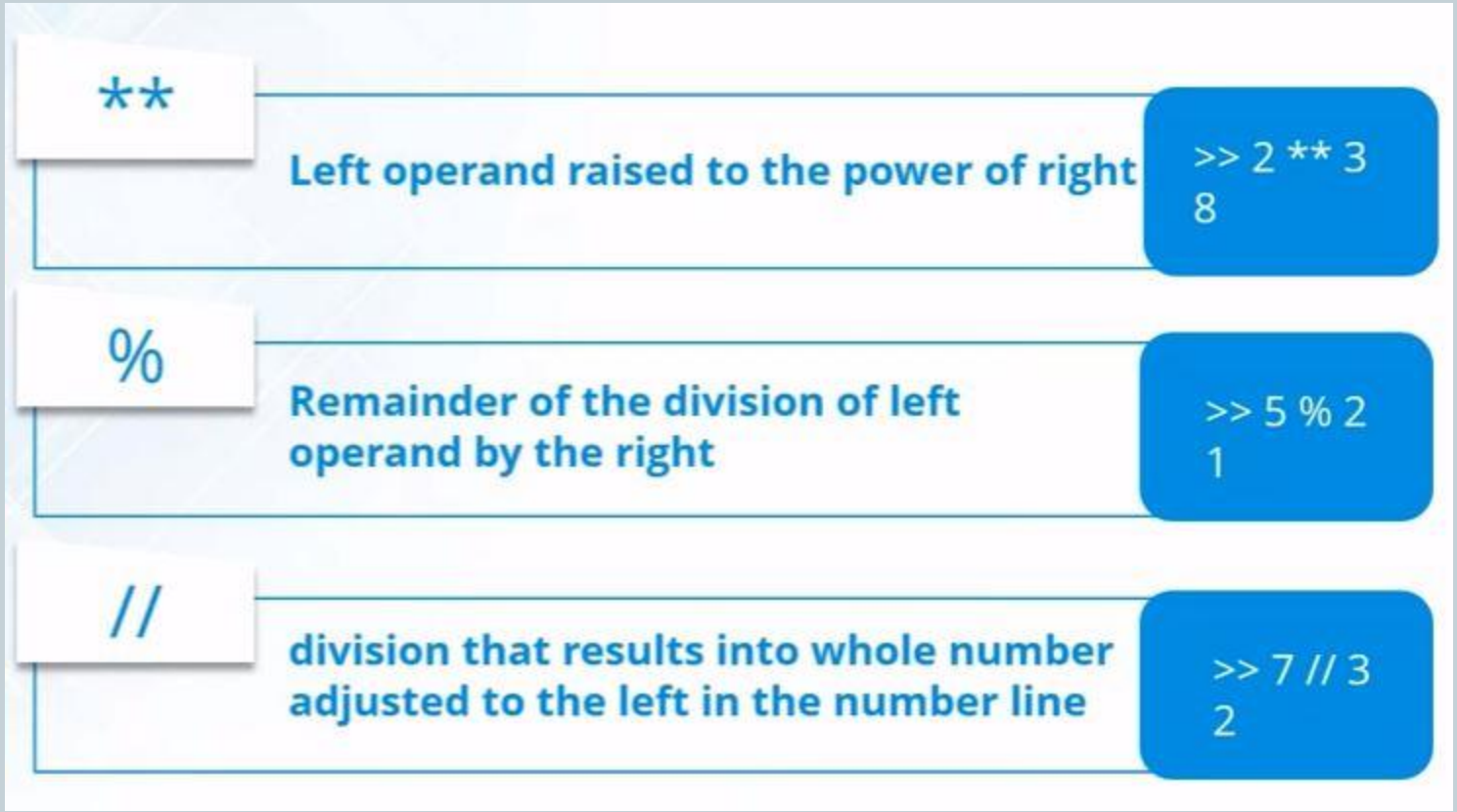
- x+y : add x and y
- x-y : y less x
- x*y : x multiplies y
- x/y : y divides x
- x//y : quotient (integer) part of x/y
- x%y : modulus of x/y
- x**y : x to the power y
- x|y : x or y
- x&y : x and y
- x==y : x equals to y
- x!=y : x is not equal to y
- x<y, x>y, x<=y, x>=y : x lt t, x gt y, x le y, x ge y

Dealing with Strings

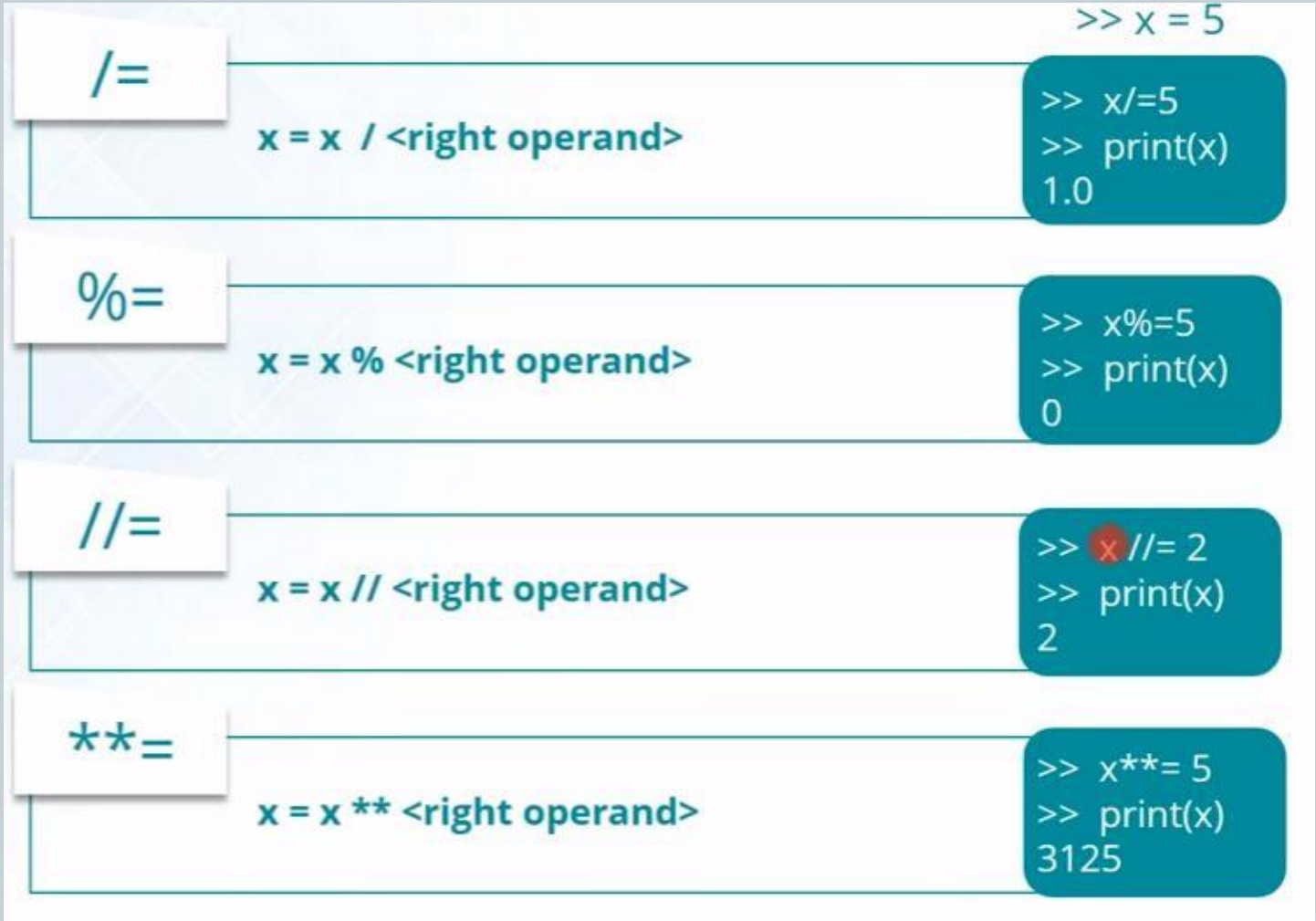
```
In [91]: string0 = 'python'
         string1 = "Data Science"
         string2 = '''This is Data science
                 workshop
                 using Python'''

In [92]: print( string0, string1, string2)

python Data Science This is Data science
       workshop
       using Python

In [93]: string2.find( "Python" )

Out[93]: 53

In [94]: string0.capitalize()

Out[94]: 'Python'

In [95]: string0.upper()

Out[95]: 'PYTHON'

In [96]: len( string2 )

Out[96]: 59

In [97]: string2.split()

Out[97]: ['This', 'is', 'Data', 'science', 'workshop', 'using', 'Python']

In [98]: string2.replace( 'Python', 'R')

Out[98]: 'This is Data science \n        workshop\n        using R'
```

Python Operator

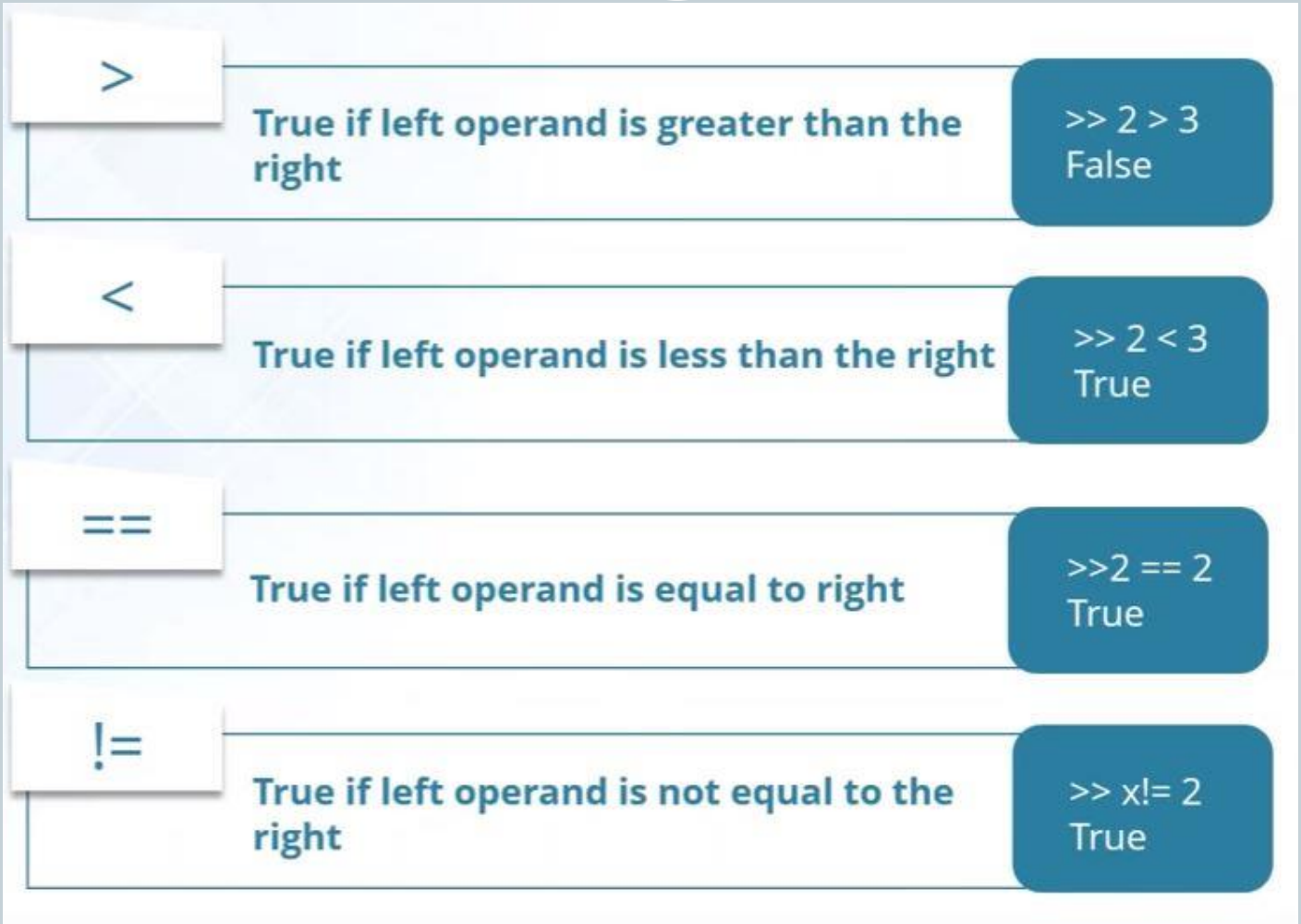
Arithmetic Operator



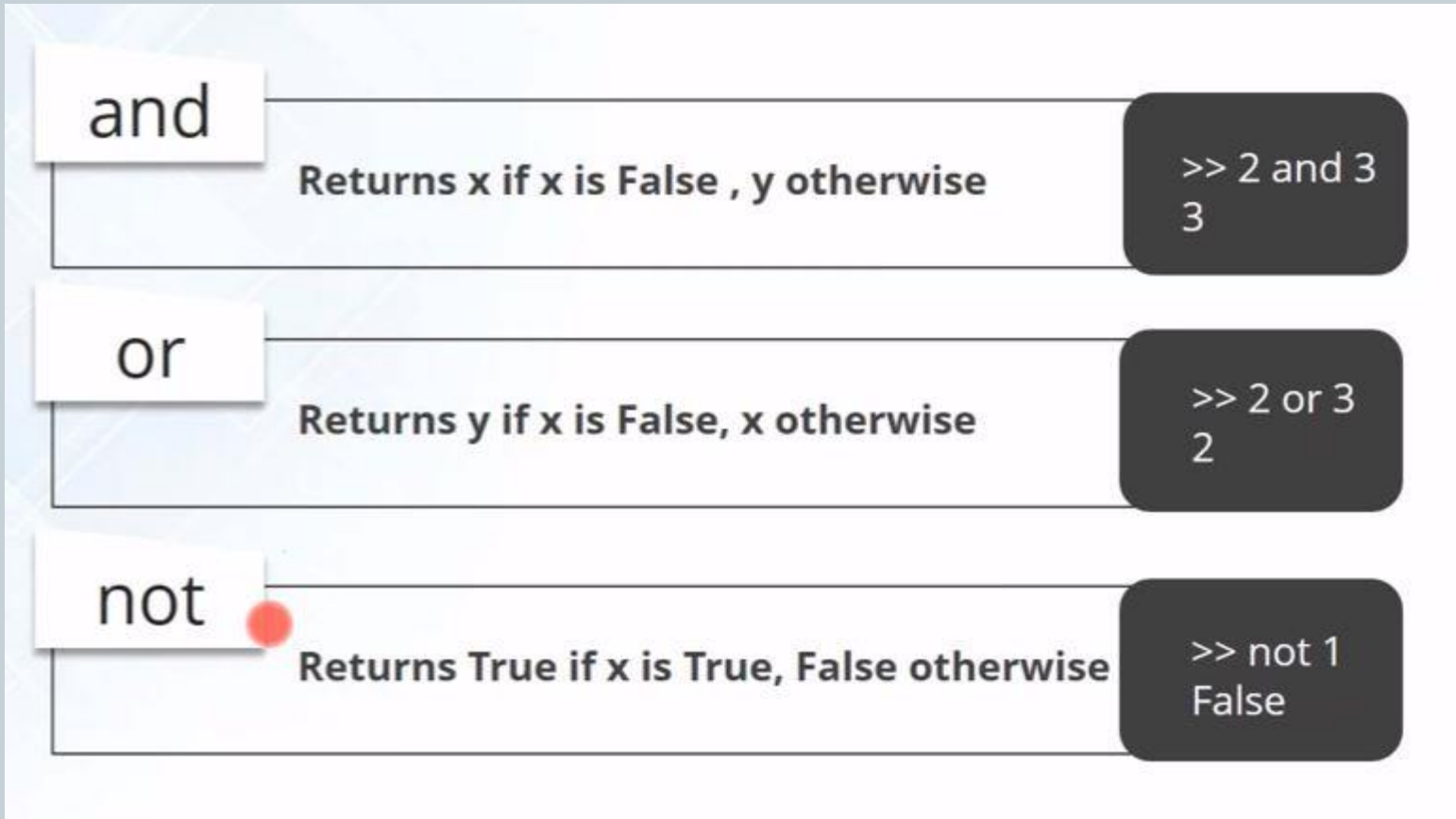
Assignment operator



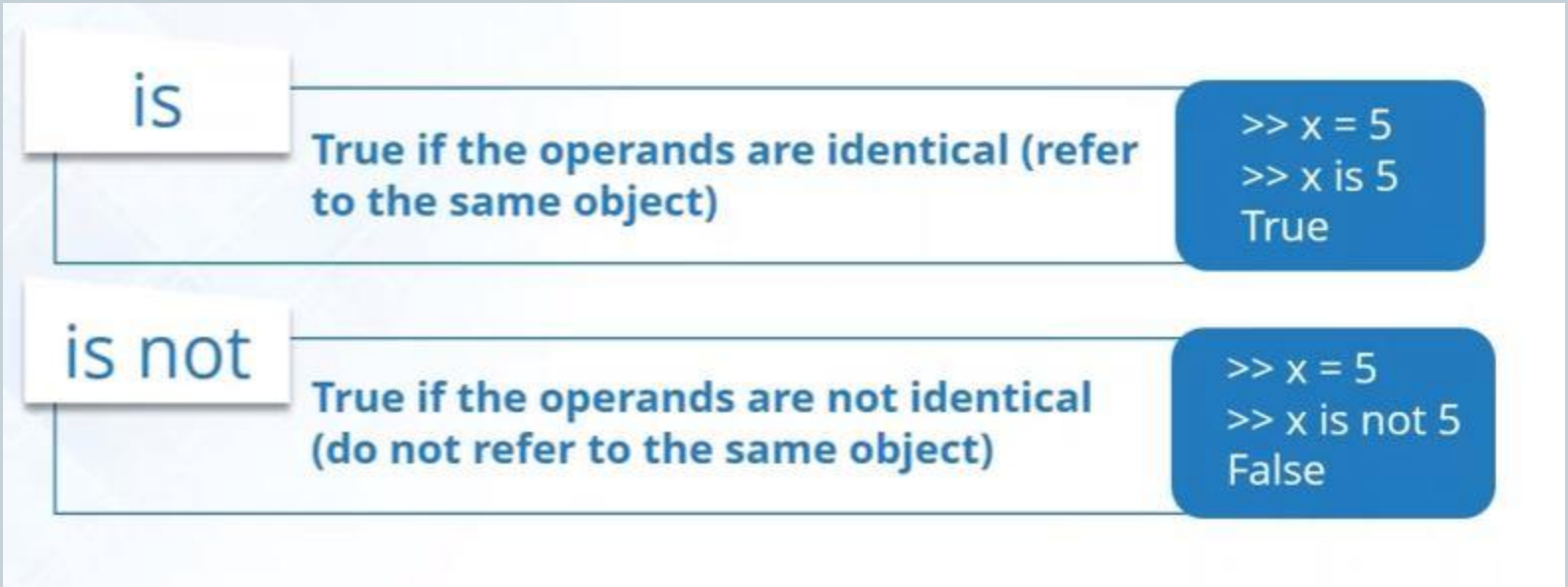
Comparison Operator



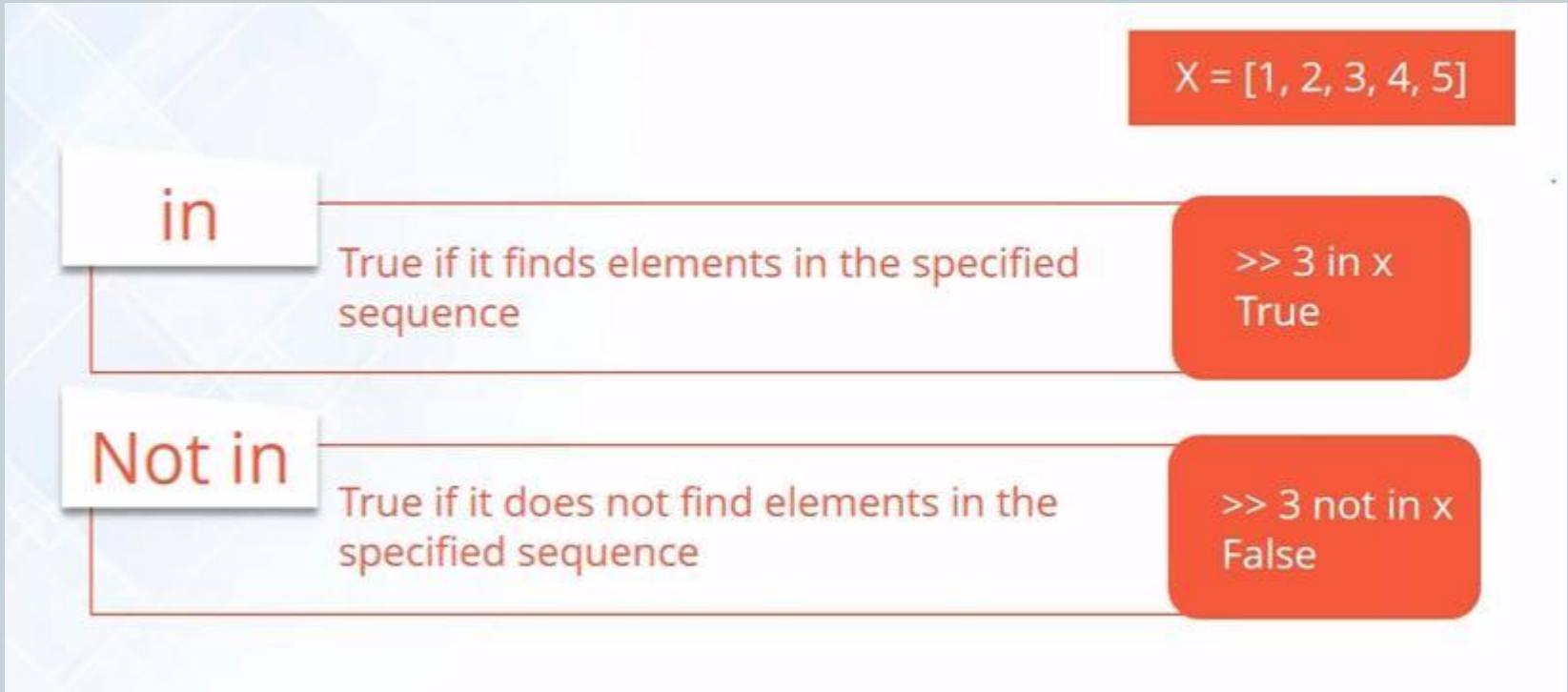
Logical Operator



Identity operator



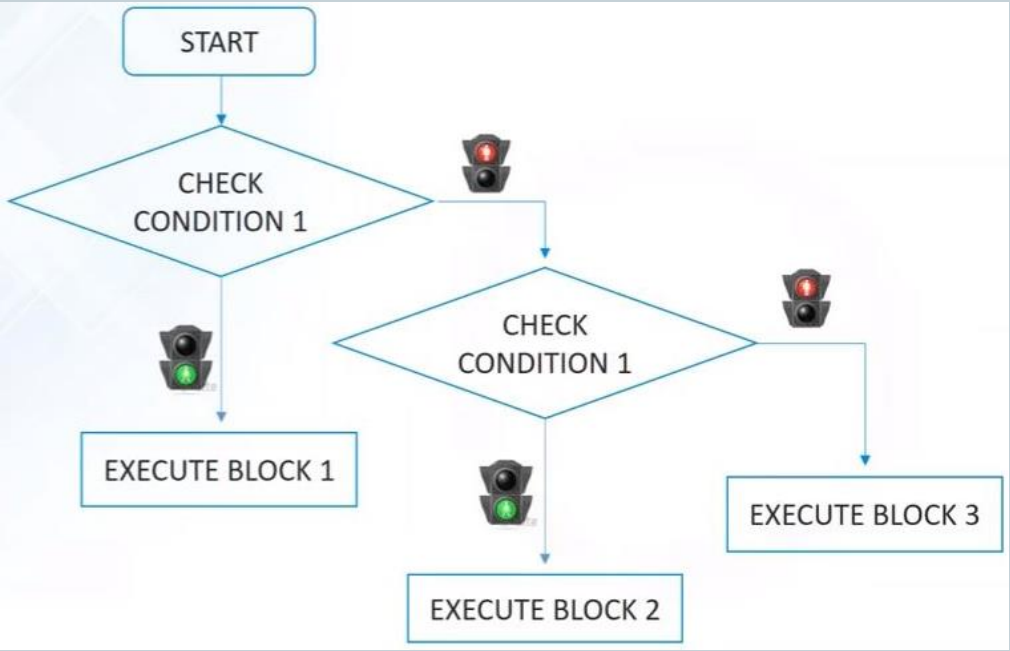
Membership Operator



Control Flow with if, elif, else

» Like other languages Python uses boolean outcome to evaluate conditions. The boolean values True and False are returned when an expression is compared or evaluated.

"The if statement is one of the most well-known control flow statement types. It checks a condition which, if True, evaluates the code in the block that follows:



Syntax:

```
if (condition 1):
    statements 1 ...
elif (condition 2):
    statements 2 ...
else
    statements 3 ...
```

In [28]:

```
x = 10
y = 12
if x > y:
  print ("x>y")
elif x < y:
  print ("x<y")
else:
  print ("x=y")
```

```
x<y
```

For Loop

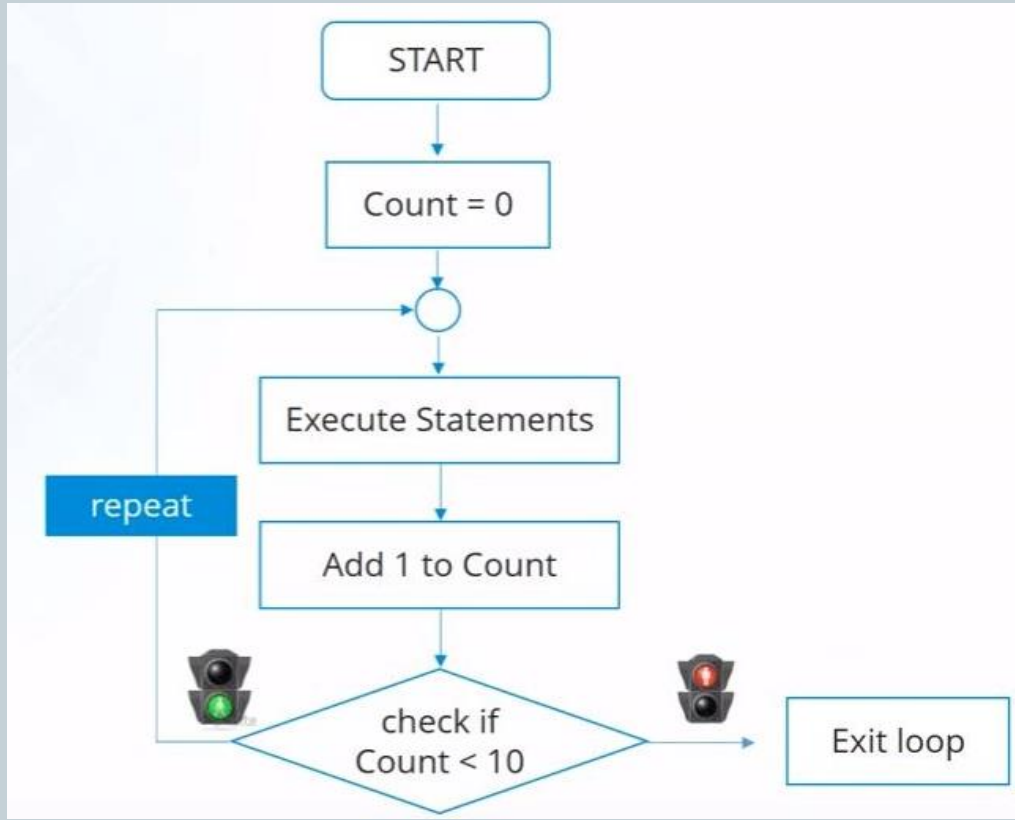
- **There are two types of loops in Python, for and while.**
- **for Loops:** For loops iterate over a given sequence **.**
- Loops are meant for iteration tasks over a range

Syntax:

```
for value in range:
    Execute the action
```

Example: Here we print out the squares of the first 10 natural numbers

```
for x in range(1,10)
    print(x**2)
```



While Loop

» **while Loops:** While loops repeat as long as a certain boolean condition is met.

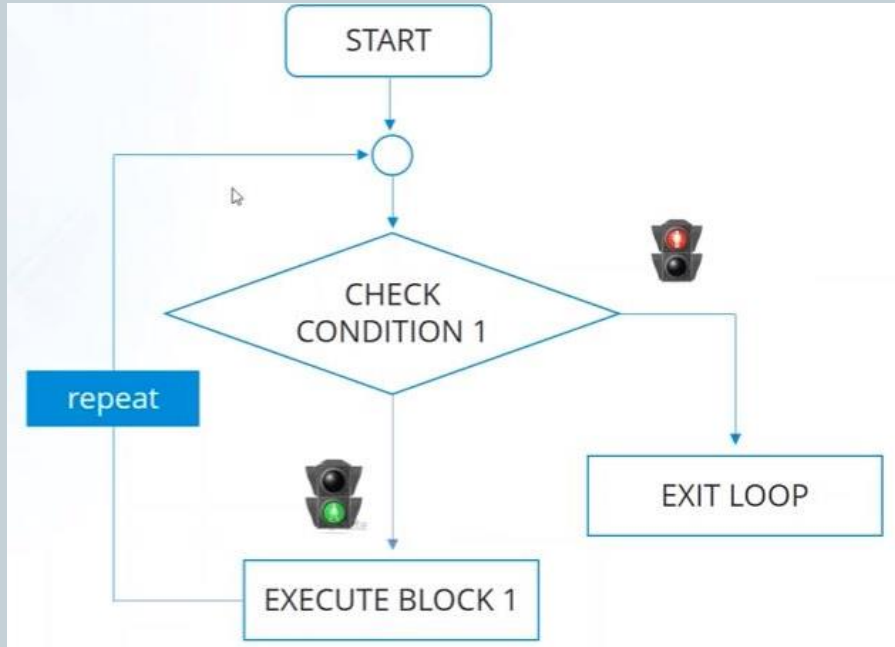
Python has a while loop as well, which works similar to other programing language.

```
x = 0
while x < 10:
    print (x, "is less than 10" x += 1)
```

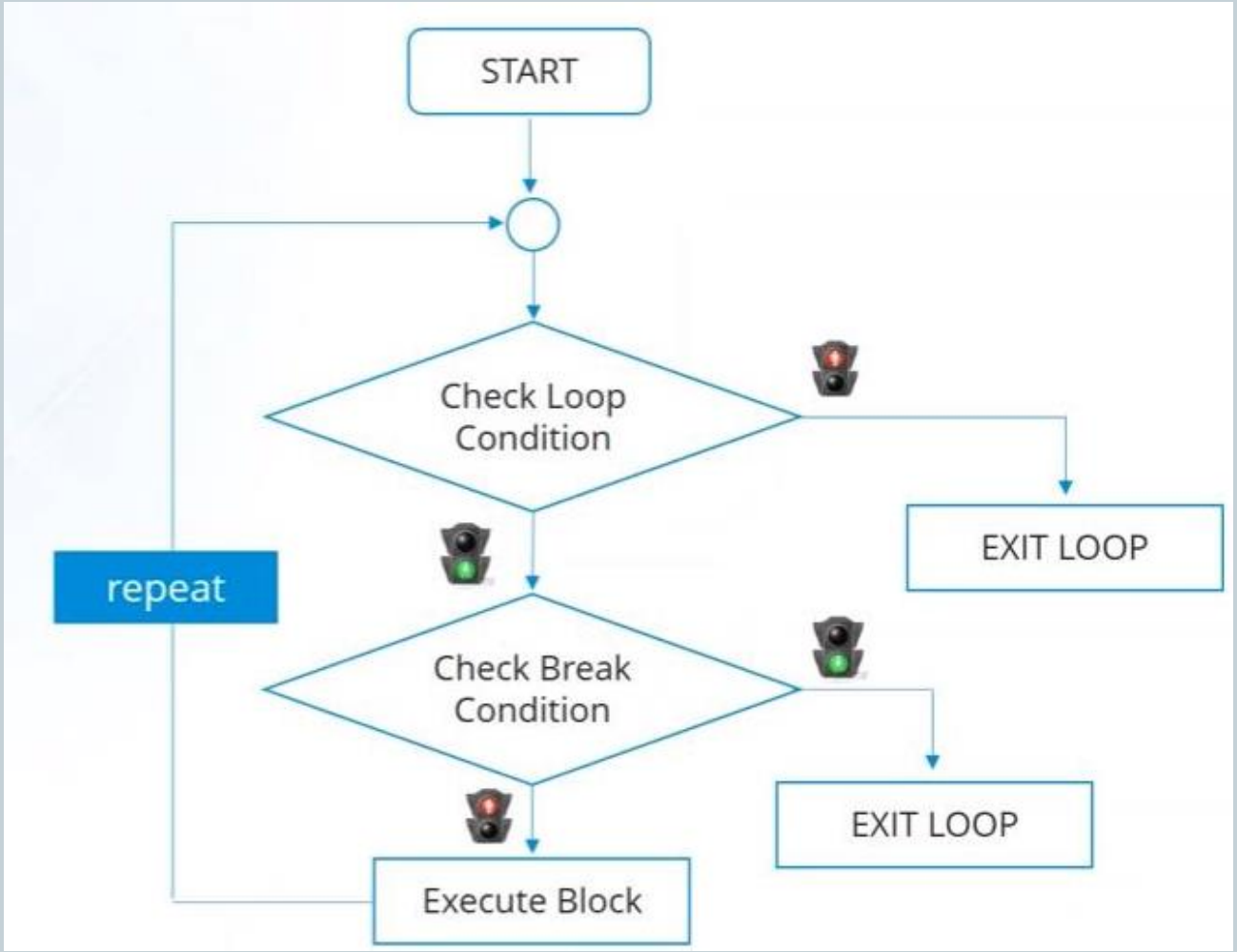
```
In [32]: i = 1
         while i < 5:
           print(i)
           i = i+1
         print('Bye')

1
2
3
4
Bye
```

```
In [33]: i = 1
         while i < 5:
           print(i)
           i = i+1
           if i == 4:
               break
         print('Bye')

1
2
3
Bye
```

```
In [34]: i = 1
         while i < 5:
           i = i+1
           if i == 3:
               continue
           print(i)
         print('Bye')

2
4
5
Bye
```

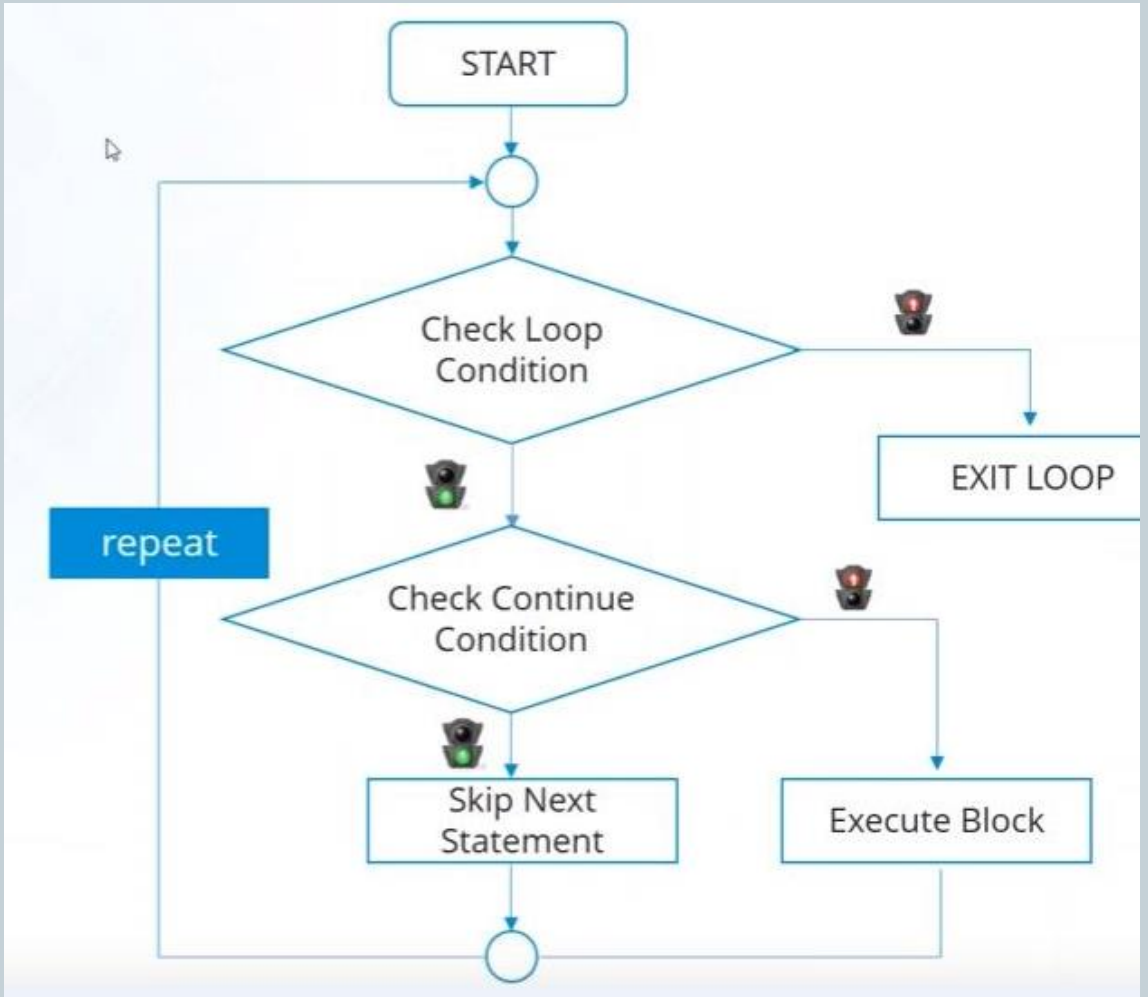


Break in the Loop




```
#break
count =0
while True:
    print(count)
    count+=1
    if (count > 10):
        break
```

Continue the Loop




```
#continue

for x in range(20):
    if (x%2)==0:
        continue
    print(x)
```

User defined function

- Functions are a convenient way to save some piece of executable statements with some name.
- A programmer can call out the defined function any time for the relevant purpose
- **Functions in Python**:
  - Functions in python are defined using the keyword "def", followed with the function's name as the block's name
  - As a standard rule in python, variable or arguments' type not to be declared
  - Functions may also receive arguments (variables passed from the caller to the function)
  - Functions may return a value to the caller, using the keyword- 'return'. Functions can return multiple parameter
  - If only one or few returned parameters need to be captured and other ignored
  - Simply write the function's name followed by (), placing any required arguments within the brackets.

User defined function

```
In [99]: def addElements( a, b ):
           return a + b

In [100]: addElements( 2, 3 )
Out[100]: 5

In [101]: addElements( 2.3, 4.5 )
Out[101]: 6.8

In [102]: addElements( "python", "workshop" )
Out[102]: 'pythonworkshop'
```

```
In [103]: def addElements( a, b ):
            return a, b, a + b

In [104]: x, y, z = addElements( 2, 3 )

In [105]: addElements( 2.3, 4.5 )
Out[105]: (2.3, 4.5, 6.8)
```

```
In [108]: def addElements( a, b = 4 ):
            return a + b

In [109]: addElements( 2 )
Out[109]: 6

In [110]: addElements( 2, 5 )
Out[110]: 7
```

```
In [111]: def add_n(*args):
            sum = 0
            for arg in args:
                sum = sum + arg
            return sum

In [112]: add_n( 1, 2, 3 )
Out[112]: 6

In [113]: add_n( 1, 2, 3, 4, 5, 6 )
Out[113]: 21

In [114]: add_n()
Out[114]: 0
```

Lambda Function

- Lambda functions in python are functions that can be passed as parameters to another functions.
- The functions are anonymous and defined inline, while passing as a parameter.
- Primarily used to deal with collections, to apply a function or operations on each individual elements of python

```
In [115]: a = lambda x: x * x

In [116]: a( 2 )
Out[116]: 4

In [117]: a( 2 ) * a( 2 )
Out[117]: 16
```

```
In [126]: list( filter( lambda x : x < 5, listprods ) )
Out[126]: [1, 4]
```

```
In [118]: mylist = [1,2,3,4,5,6,7,8,9]

In [119]: xsquare = []

          for x in mylist:
            xsquare.append( pow( x, 2 ) )

          print( xsquare )

          [1, 4, 9, 16, 25, 36, 49, 64, 81]

In [120]: map( lambda x: pow( x, 2 ), mylist)
Out[120]: <map at 0x45c9fd0>

In [121]: xsquare1 = list( map( lambda x: pow( x, 2 ), mylist) )

In [122]: print( xsquare1 )

          [1, 4, 9, 16, 25, 36, 49, 64, 81]

In [123]: mylist1 = [1,2,3,4,5,6,7,8,9]

In [124]: listprods = list( map( lambda x, y: x * y, mylist, mylist1 ) )

In [125]: listprods
Out[125]: [1, 4, 9, 16, 25, 36, 49, 64, 81]
```