Why Python?

(C/C++/Java)
Python

✓ R

✓ — very simple language to learn

(*) — Best packages for AI : matplotlib,
numpy, scipy,
✓ ← scikit-learn,
deep learning ✓ ← tensorflow

✓ — iPython Notebooks for interactive data analysis & modelling

✓ — Extensively used in the industry.

Python Keywords

Keywords are the reserved words in python

We can't use a keyword as variable name, function name or any other identifier

Keywords are case sentive

In [3]:
```
#Get all keywords in python 3.6

import keyword

print(keyword.kwlist)

print("\nTotal number of keywords: ", len(keyword.kwlist))
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'conti
nue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from',
 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
```

Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

Rules for Writing Identifiers:

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.

In [6]:
```
abc_12 = 12;
global = 1
```

```
  File "<ipython-input-6-461be8cb845a>", line 2
    global = 1
           ^
SyntaxError: invalid syntax
```

Python Comments

Comments are lines that exist in computer programs that are ignored by compilers and interpreters.

Including comments in programs makes code more readable for humans as it provides some information or explanation about what each part of a program is doing.

In general, it is a good idea to write comments while you are writing or updating a program as it is easy to forget your thought process later on, and comments written later may be less useful in the long term.

In Python, we use the hash (#) symbol to start writing a comment.

In [1]:
```
#Print Hello, world to console
print("Hello, world")
```

```
Hello, world
```

Multi Line Comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line.

In [2]:
```
#This is a long comment
#and it extends
#Multiple lines
```

Another way of doing this is to use triple quotes, either ''' or """.

In [3]:
```
"""This is also a
perfect example of
multi-line comments"""
```

Out[3]:
```
'This is also a\nperfect example of\nmulti-line comments'
```

Python Indentation

1. Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.
2. A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.
3. Generally four whitespaces are used for indentation and is preferred over tabs.

In [14]:
```
for i in range(10): # we will look at for loop in detail later.
    print(i)
```

```
0
1
2
3
4
5
6
7
```

Python Statement

Instructions that a Python interpreter can execute are called statements.

Examples:

In [17]:
```
a = 1  #single statement
```

Multi-Line Statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character ().

In [18]:
```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8
```

In [1

In [19]:
```
print(a)
```

```
36
```

In [21]:
```
#another way is to use paranthesis
a = (1 + 2 + 3 +
     4 + 5 + 6 +
     7 + 8)
print(a)
```

```
36
```

In [22]:
```
a = 10; b = 20; c = 30    #put multiple statements in a single line using ;
```

In [ ]:

In [ ]:

Variables

A variable is a location in memory used to store some data (value).

They are given unique names to differentiate between different memory locations. The rules for writing a variable name is same as the rules for writing identifiers in Python.

We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of the variable. This is handled internally according to the type of value we assign to the variable.

Variable Assignments

In [1]:
```
#We use the assignment operator (=) to assign values to a variable

a = 10
b = 5.5
```

In [34]:
```
#We use the assignment operator (=) to assign values to a variable

a = 10
b = 5.5
c = "ML"
```

Multiple Assignments

In [2]:
```
a, b, c = 10, 5.5, "ML"
```

In [3]:
```
a = b = c = "AI" #assign the same value to multiple variables at once
```

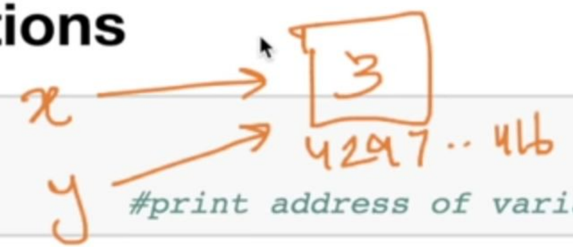
Storage Locations

In [4]:
```
x = 3

print(id(x))                    #print address of variable x
```

```
4297636416
```

In [5]:

Storage Locations



In [4]:
```
x = 3

print(id(x))                    #print address of variable x
```

```
4297636416
```

In [5]:
```
y = 3

print(id(y))                    #print address of variable y
```

```
4297636416
```

Observation:

x and y points to same memory location

In [6]:
```
y = 2
print(id(y))                    #print address of variable y
```

Data Types

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

Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

In [7]:
```
a = 5                                    #data type is implicitly set to integer
print(a, " is of type", type(a))
```

```
5  is of type <class 'int'>
```

In [

are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

In [7]:
```
a = 5                                    #data type is implicitly set to integer
print(a, " is of type", type(a))
```

```
5  is of type <class 'int'>
```

In [8]:
```
a = 2.5                                  #data type is changed to float
print(a, " is of type", type(a))
```

```
2.5  is of type <class 'float'>
```

In [9]:
```
a = 1 + 2j                               #data type is changed to complex number
print(a, " is complex number?")
print(isinstance(1+2j, complex))
```

```
(1+2j)  is complex number?
True
```

Boolean

Boolean

Boolean represents the truth values False and True

In [10]:
```
a = True                              #a is a boolean type
print(type(a))
```

```
<class 'bool'>
```

Python Strings

String is sequence of Unicode characters.

We can use single quotes or double quotes to represent strings.

Multi-line strings can be denoted using triple quotes, ''' or """.

A string in Python consists of a series or sequence of characters - letters, numbers, and special characters.

Strings can be indexed - often synonymously called subscripted as well.

Python Strings

String is sequence of Unicode characters.

We can use single quotes or double quotes to represent strings.

Multi-line strings can be denoted using triple quotes, ''' or """.

A string in Python consists of a series or sequence of characters - letters, numbers, and special characters.

Strings can be indexed - often synonymously called subscripted as well.

Similar to C, the first character of a string has the index 0.

In [11]:
```
s = "This is Online AI course"
print(s)
```

```
This is Online AI course
```

In [12]:
```
print(s[0])
```

characters.

Strings can be indexed - often synonymously called subscripted as well.

Similar to C, the first character of a string has the index 0.

In [42]:
```
s = '''This is Online AI
        course'''
print(type(s))
```

```
<class 'str'>
```

In [48]:
```
print(s[len(s)-1])
#last char s[len(s)-1] or s[-1]
```

```
e
```

In [13]:
```
#slicing
s[5:]
```

Out[13]:
```
'is Online AI course'
```

Python List

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is , Items separated by commas are enclosed within brackets [ ].

In [52]:
```
a = [10, 20.5, "Hello"]
print(a[2])                #print 1st index element
```

```
Hello
```

Lists are mutable, meaning, value of elements of a list can be altered.

In [15]:
```
a[1] = 30.7
print(a)
```

```
[10, 30.7, 'Hello']
```

Python Tuple

Python Tuple

Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

parenthesis

list

In [16]:
```
t = (1, 1.5, "ML")
```

In [17]:
```
print(t[1]) #extract particular element
```

```
1.5
```

In [18]:
```
t[1] = 1.25
```

```
---------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-18-0ddc671fae60> in <module>()
----> 1 t[1] = 1.25

TypeError: 'tuple' object does not support item assignment
```

Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

[ ] - list
{ } - set
( ) - tuples

In [19]:
```
a = {10, 30, 20, 40, 5}
print(a)
```

```
{5, 40, 10, 20, 30}
```

In [20]:
```
print(type(a))              #print type of a
```

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

We can perform set operations like union, intersection on two sets. Set have unique values.

In [21]:
```
s = {10, 20, 20, 30, 30, 30}
print(s)                    #automatically set won't consider duplicate ele
```

<class 'set'>

We can perform set operations like union, intersection on two sets. Set have unique values.

In [21]:
```
s = {10, 20, 20, 30, 30, 30}
print(s)                        #automatically set won't consider duplicate ele
```

```
{10, 20, 30}
```

In [22]:
```
print(s[1]) #we can't print particular element in set because
            #it's unorder collections of items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
<ipython-input-22-ad7511dba6cd> in <module>()
----> 1 print(s[1]) #we can't print particular element in set because
      2             #it's unorder collections of items

TypeError: 'set' object does not support indexing
```

Python Dictionary

Dictionary is an unordered collection of key-value pairs.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

key value

In [58]:
```
d = {'a': "apple", 'b': "bat"}
print(d['a'])
```

```
apple
```

Conversion between Datatypes

We can convert between different data types by using different type conversion functions like int(), float(), str() etc.

In [2

Conversion between Datatypes

We can convert between different data types by using different type conversion functions like int(), float(), str() etc.

```
In [25]: float(5)      #convert interger to float using float() method
Out[25]: 5.0

In [26]: int(100.5)    #convert float to integer using int() method
Out[26]: 100

In [27]: str(20)       #convert integer to string
Out[27]: '20'
```

Conversion to and from string must contain compatible values.

```
In [28]: int('10p')
```

```
---------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-62-b6ad1e4c556d> in <module>()
----> 1 int('10p')

ValueError: invalid literal for int() with base 10: '10p'
```

In [31]:
```
user = "satish"
lines = 100

print("Congratulations, " + user + "! You just wrote " + str(lines) + " lin
#remove str and gives error
```

```
Congratulations, satish! You just wrote 100 lines of code
```

We can convert one sequence to other

In [28]:
```
a = [1, 2, 3]

print(type(a))      #type of a is list
```

```
#remove str and gives error
```

```
Congratulations, satish! You just wrote 100 lines of code
```

We can convert one sequence to other

In [66]:
```
a = [1, 2, 3]

print(type(a))       #type of a is list

s = set(a)           #convert list to set using set() method

print(type(s))       #now type of s is set
```

```
<class 'list'>
<class 'set'>
```

In [29]:
```
list("Hello")        #convert String to list using list() method
```

Out[29]:
```
['H', 'e', 'l', 'l', 'o']
```

Python Input and Output

Python Output

We use the print() function to output data to the standard output device

In [3]:
```
print("Hello World")
```

```
Hello World
```

In [4]:
```
a = 10
print("The value of a is", a) #python 3
print("The value of a is " + str(a))
```

```
The value of a is 10
The value of a is 10
```

Output Formatting

In [5]:

Output Formatting

In [5]:
```
a = 10; b = 20 #multiple statements in single line.

print("The value of a is {} and b is {}".format(a, b))    #default
```

```
The value of a is 10 and b is 20
```

In [6]:
```
a = 10; b = 20  #multiple statements in single line

print("The value of b is {1} and a is {0}".format(a, b)) #specify position o
```

```
The value of b is 20 and a is 10
```

In [7]:
```
can use keyword arguments to format the string
t("Hello {name}, {greeting}".format(name="satish", greeting="Good Morning"))
```

```
Hello satish, Good Morning
```

In [8]:
```
#we can combine positional arguments with keyword arguments
```

Output Formatting

In [5]:
```
a = 10; b = 20 #multiple statements in single line.

print("The value of a is {} and b is {}".format(a, b))    #default
```

```
The value of a is 10 and b is 20
```

In [6]:
```
a = 10; b = 20  #multiple statements in single line

print("The value of b is {1} and a is {0}".format(a, b)) #specify position o
```

```
The value of b is 20 and a is 10
```

In [7]:
```
can use keyword arguments to format the string
t("Hello {name}, {greeting}".format(name="satish", greeting="Good Morning"))
```

```
Hello satish, Good Morning
```

In [8]:
```
#we can combine positional arguments with keyword arguments
```

Output Formatting

In [5]:
```
a = 10; b = 20 #multiple statements in single line.

print("The value of a is {} and b is {}".format(a, b))    #default
```

```
The value of a is 10 and b is 20
```

In [6]:
```
a = 10; b = 20  #multiple statements in single line

print("The value of b is {1} and a is {0}".format(a, b)) #specify position o
```

```
The value of b is 20 and a is 10
```

In [7]:
```
#we can use keyword arguments to format the string
print("Hello {name}, {greeting}".format(name="satish", greeting="Good Morni
```

```
Hello satish, Good Morning
```

In [8]:

In [6]:
```
a = 10; b = 20  #multiple statements in single line

print("The value of b is {1} and a is {0}".format(a, b)) #specify position o
```

```
The value of b is 20 and a is 10
```

In [7]:
```
#we can use keyword arguments to format the string
print("Hello {name}, {greeting}".format(name="satish", greeting="Good Morni
```

```
Hello satish, Good Morning
```

In [8]:
```
#we can combine positional arguments with keyword arguments
print('The story of {0}, {1}, and {other}'.format('Bill', 'Manfred',
                                                  other='Georg'))
```

```
The story of Bill, Manfred, and Georg
```

Python Input

want to take the input from the user. In Python, we have the input() function to allow this.

In [1

```
print('The story of {0}, {1}, and {other}'.format('Bill', 'Manfred',
                                                  other='Georg'))
```

```
The story of Bill, Manfred, and Georg
```

Python Input

want to take the input from the user. In Python, we have the input() function to allow this.

In [*]:
```
num = input("Enter a number: ")
print(num)
```

```
Enter a number:
```

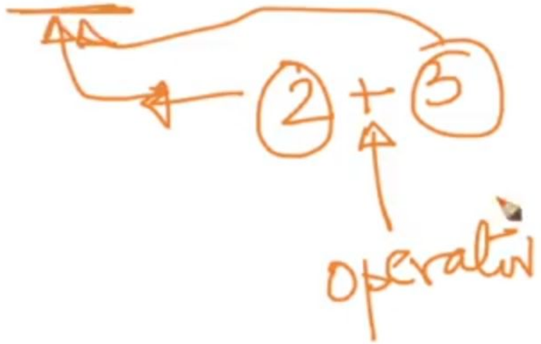
In [ ]:

In [ ]:

In [ ]:

Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

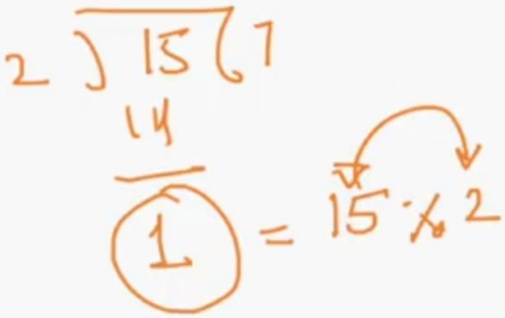


Operator Types

1. Arithmetic operators
2. Comparison (Relational) operators
3. Logical (Boolean) operators
4. Bitwise operators
5. Assignment operators
6. Special operators

Example:

In [27]:
```
x, y = 10, 20

#addition
print(15%2)

#subtraction(-)

#multiplication(*)

#division(/)

#modulo division (%)

#Floor Division (//)

#Exponent (**)
```




```
1
```

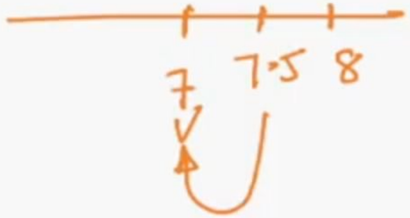
Comparision Operators

Example:

In [29]:
```
x, y = 10, 20

#addition
print(15//2)

#subtraction(-)

#multiplication(*)

#division(/)

#modulo division (%)

#Floor Division (//)

#Exponent (**)
```

$\frac{15}{2} = 7.5$

$\lfloor 7.5 \rfloor = 7$



7

Comparision Operators

Comparison operators are used to compare values. It either returns True or False according to the condition.

>, <, ==, !=, >=, <= are comparision operators

In [22]:
```
a, b = 10, 20

print(a < b)  #check a is less than b

#check a is greater than b

#check a is equal to b

#check a is not equal to b (!=)

#check a greater than or equal to b

#check a less than or equal to b
```

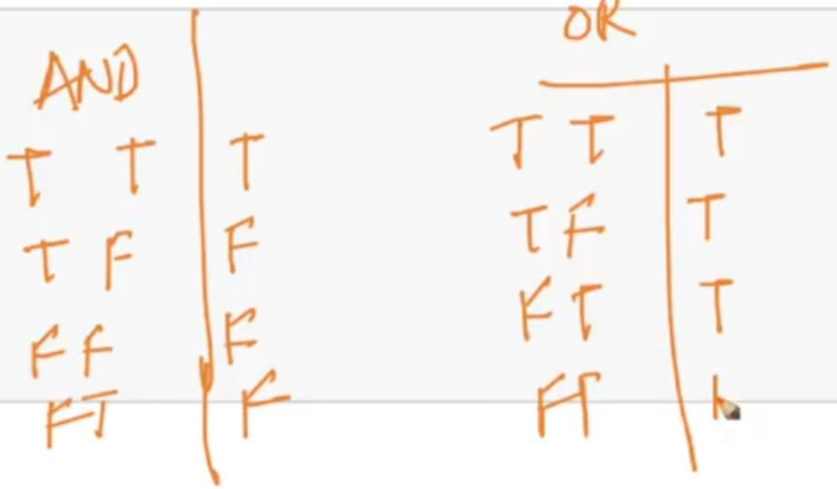
```
True
```

Logical Operators

Logical operators are **and, or, not** operators.

In [7]:
```
a, b = True, False

#print a and b
print(a and b)

#print a or b

#print not b
```

```
False
```



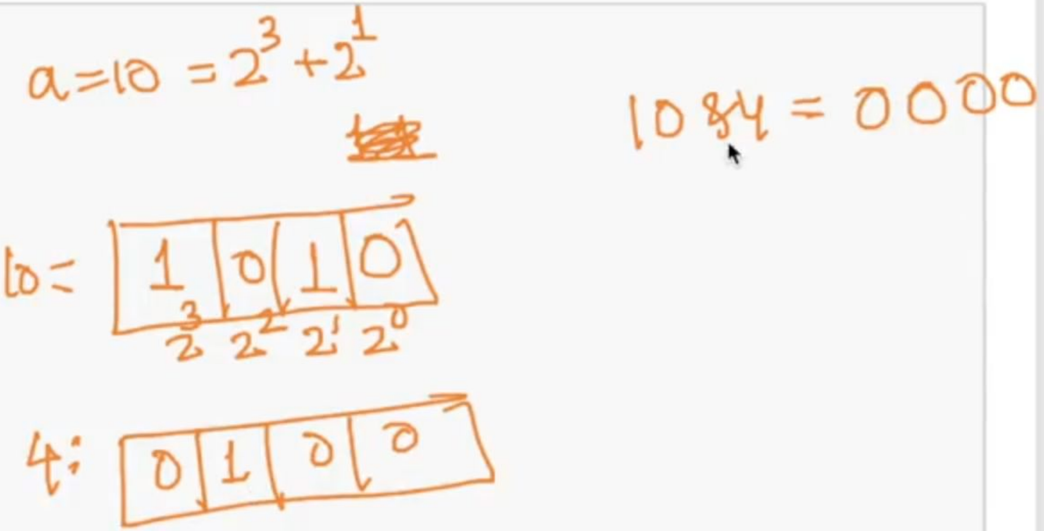
Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit

`&, |, ~, ^, >>, << are Bitwise operators`

In [13]:
```
a, b = 10, 4

#Bitwise AND
print(a & b)

#Bitwise OR


#Bitwise NOT


#Bitwise XOR


#Bitwise rightshift
```



python-basics-master 2/Introc | 3_comments_indentation_state | 4_input_output | 5_operators | python - ipython notebook is h | Chekuri Srikan...

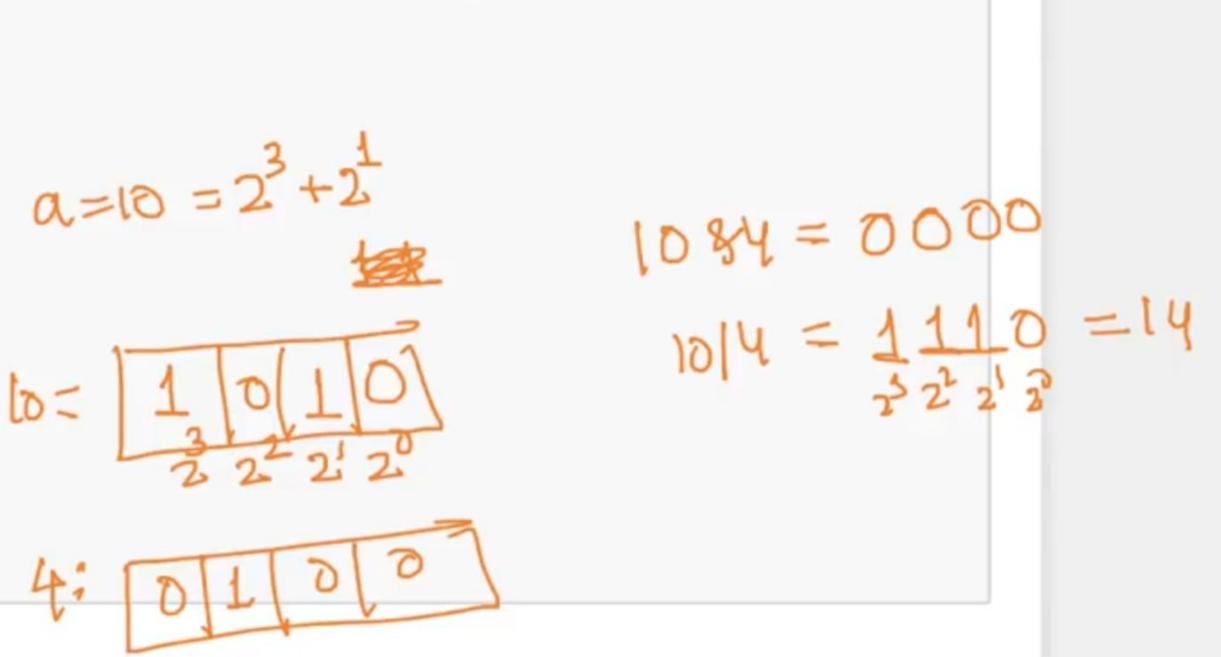
localhost:8888/notebooks/Google%20Drive/OnlineVideos/2/python-basics-master%202/Introduction/5_operators.ipynb

File Edit View Insert Cell Kernel Help | Trusted | Python 3

Markdown

In [34]:
```
a, b = 10, 4

#Bitwise AND
print(a | b)

#Bitwise OR

#Bitwise NOT

#Bitwise_XOR

#Bitwise rightshift

#Bitwise Leftshift
```

```
14
```



Assignment operators

115 / 115

Assignment operators

Assignment operators are used in Python to assign values to variables.

a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

```
=,  +=,  -=,  *=,  /=,  %=,  //=,  **=, &=,  |=,  ^=,  >>=,  <<= ar
e Assignment operators
```

In [14]:
```
a = 10

a += 10          #add AND
print(a)

#subtract AND (-=)

#Multiply AND (*=)
```

In [14]:
```
a = 10

a += 10          #add AND
print(a)

#subtract AND (-=)

#Multiply AND (*=)

#Divide AND (/=)

#Modulus AND (%=)

#Floor Division (//=)

#Exponent AND (**=)
```

```
20
```

a = a + 10

a = a - 10

a = a * 10

Special Operators

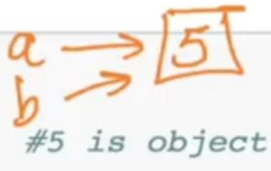
Identity Operators

is and is not are the identity operators in Python.

They are used to check if two values (or variables) are located on the same part of the memory.

In [15]:
```
a = 5
b = 5
print(a is b)    #5 is object created once both a and b points to same objec

#check is not
```

```
True
```



In [16]:
```
l1 = [1, 2, 3]
l2 = [1, 2, 3]
print(l1 is l2)
```

In [15]:
```
a = 5
b = 5
print(a is b)    #5 is object created once both a and b points to same objec

#check is not
```

```
True
```

In [16]:
```
l1 = [1, 2, 3]
l2 = [1, 2, 3]
print(l1 is l2)
```

```
False
```

In [35]:
```
s1 = "AAIC"
s2 = "AAIC"
print(s1 is s2)
```


```
True
```

MemberShip Operators

MemberShip Operators

in and not in are the membership operators in Python.

They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

In [19]:
```
lst = [1, 2, 3, 4]
print(1 in lst)        #check 1 is present in a given list or not

#check 5 is present in a given list
```

```
True
```

In [20]:
```
d = {1: "a", 2: "b"} # this is a dictionary
print(1 in d)
```

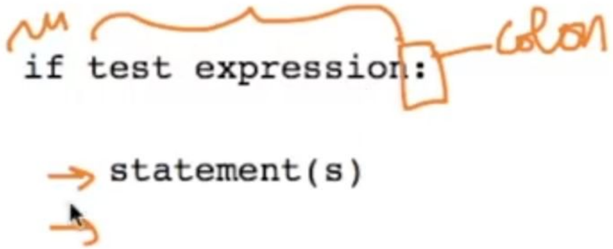
```
True
```

In [ ]:

Python if ... else Statement

The **if...elif...else** statement is used in Python for decision making.

if statement syntax



The program evaluates the test expression and will execute statement(s) only if the text expression is True.

If the text expression is False, the statement(s) is not executed.

Example

In [5]:
```
num = 10

# try 0, -1 and None
if num > 10:
    print("Number is positive")
print("This will print always")      #This print statement always print

#change number
```

```
This will print always
```

if ... else Statement

Syntax:

Flow-Control/1_if-else/ | if-elif-else | Chekuri Srikan...

localhost:8888/notebooks/Google%20Drive/OnlineVideos/2/python-basics/Flow-Control/1_if-else/if-elif-else.ipynb

File Edit View Insert Cell Kernel Help | Trusted | Python 3

Code

Example

In [6]:
```
num = 10
if num > 0:
    print("Positive number")
else:
    print("Negative Number")
```

```
Positive number
```

if...elif...else Statement

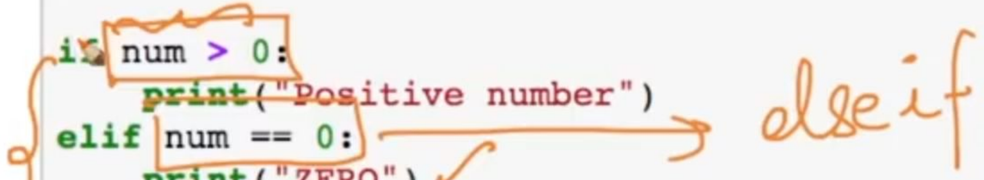
Syntax:

```
if test expression:
```

Example:

In [15]:
```
num = 0

if num > 0:
    print("Positive number")
elif num == 0:
    print("ZERO")
else:
    print("Negative Number")
```

```
ZERO
```

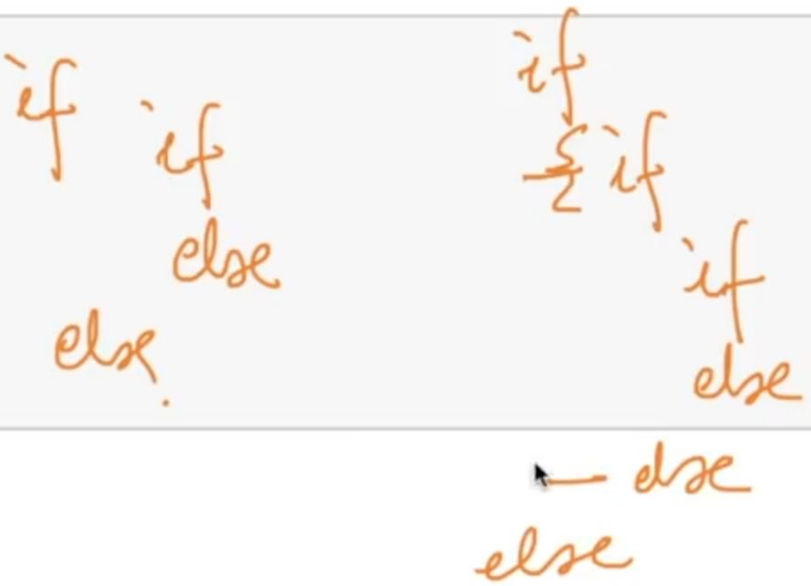


Nested if Statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Example:

In [8]:
```
num = 10.5

if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative Number")
```




```
Positive number
```

Python program to find the largest element among three Numbers

In

Python while Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

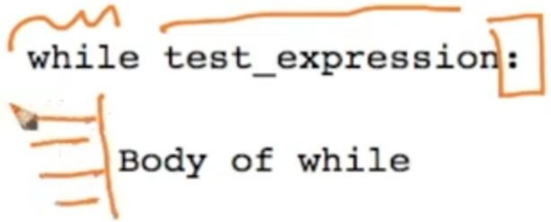
Syntax:

```
while test_expression:

    Body of while
```

Python While Loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

Syntax:



The body of the loop is entered only if the test_expression evaluates to True.

After one iteration, the test expression is checked again.

This process continues until the test_expression evaluates to False.

Example

In [1]:
```
#Find product of all numbers present in a list

lst = [10, 20, 30, 40, 50]

product = 1
index = 0

while index < len(lst):
    product *= lst[index]
    index += 1

print("Product is: {}".format(product))
```

```
Product is: 12000000
```

while Loop with else

The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement.

In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

In [2]:
```
numbers = [1, 2, 3]

#iterating over the list
index = 0
while index < len(numbers):
    print(numbers[index])
    index += 1
else:
    print("no item left in the list")
```

```
1
2
3
no item left in the list
```

Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.

Iterating over a sequence is called traversal.

Syntax:

```
for element in sequence :

    Body of for
```

Example

In [1]:
```
#Find product of all numbers present in a list
lst = [10, 20, 30, 40, 50]

product = 1
#iterating over the list
for ele in lst:
    product *= ele

print("Product is: {}".format(product))
```

```
Product is: 12000000
```

range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

In [11]:
```
#print range of 10
for i in range(10):
    print(i)
```

i i i ---- i
0, 1, 2, ---- ,9

```
0
1
2
3
4
5
6
7
8
9
```

In [12]:
```
#print range of numbers from 1 to 20 with step size of 2
for i in range(1, 20, 2):
```

In [12]:
```
#print range of numbers from 1 to 20 with step size of 2
for i in range(1, 20, 2):
    print(i)
```

start
step-size
end

```
1
3
5
7
9
11
13
15
17
19
```

1
3
20

In [4]:
```
lst = ["satish", "srinu", "murali", "naveen", "bramha"]

#iterate over the list using index
for index in range(len(lst)):
```

```
6
7
8
9
```

In [14]:
```
#print range of numbers from 1 to 20 with step size of 2
for i in range(1, 20, 1):
    print(i)
```


```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

0 - - - - 19

1 - - - - - 19

```
6
7
8
9
```

In [16]:
```
#print range of numbers from 1 to 20 with step size of 2
for i in range(0, 20, 5):
    print(i)
```

```
0
5
10
15
```

In [4]:
```
lst = ["satish", "srinu", "murali", "naveen", "bramha"]

#iterate over the list using index
for index in range(len(lst)):
    print(lst[index])
```

```
satish
srinu
```

```
0
5
10
15
```

0 1 2 3 4

In [4]:
```
lst = ["satish", "srinu", "murali", "naveen", "bramha"]

#iterate over the list using index
for index in range(len(lst)):
    print(lst[index])
```

len(lst) = 5
range(5) = 0,1,2,3,4

```
satish
srinu
murali
naveen
bramha
```

for loop with else

```
0
5
10
15
```

In [17]:
```
lst = ["satish", "srinu", "murali", "naveen", "bramha"]

#iterate over the list using index
#for index in range(len(lst)):
#    print(lst[index])
for ele in lst:
    print(ele)
```

```
satish
srinu
murali
naveen
bramha
```