Python Basics Concepts

Why Python?

- Presence of Third Party Module
- Extensive Support Libraries.
- Open Source and Community Development.
- Learning Ease and Support Available.
- User-friendly Data Structures.
- Productivity and Speed.

What are Identifiers ?

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers.Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Valid identifiers:

- ab10c: contains only letters and numbers
- abc_DE: contains all the valid characters
- _: surprisingly but Yes, underscore is a valid identifier
- _abc: identifier can start with an underscore

Invalid identifers:

- 99: identifier can’t be only digits
- 9abc: identifier can’t start with number
- x+y: the only special character allowed is an underscore
- for: it’s a reserved keyword

Worried about to confirm whether an identifier is ok or not??

There's a built in method to confirm that.

Let's see

In [2]:

```
# Run this cell and see the answer in boolean way(True, False) and don't worry if you don't
# Just keep in mind, there's a way to cross check the identifier.
print("abc".isidentifier())  # True
print("99a".isidentifier())  # False
print("_".isidentifier())  # True
print("for".isidentifier())  # True - wrong output
```

```
True
False
True
True
```

Reserved words in Python

Reserved words (also called keywords) are defined with predefined meaning and syntax in the language. These keywords have to be used to develop programming instructions. Reserved words can't be used as identifiers for other programming elements like name of variable, function etc.

Following is the list of reserved keywords in Python 3

and, except, lambda, with, as, finally, nonlocal, while, assert, false, None, yield, break, for, not, class, from, or, continue ,global, pass, def, if, raise, del, import, return, elif, in, True, else, is, try

To see what each keyword means here, you can refer this link [Click here (https://www.w3schools.com/python/python_ref_keywords.asp)](https://www.w3schools.com/python/python_ref_keywords.asp)

There's another way to check reserved keywords in Python

In [1]:

```
# Run this command and you will see reserved keywords in a list(list is a data structure yo
import keyword
keyword.kwlist
```

Out[1]:

```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```


Variables

In [6]:

```
# creation of a variable
# You can use either single or double quote while defining string variable

x = 5 # int type(default)
y = "CloudyML" # string type
print(x)
print(y)
print(type(x)) # to check type of variable
print(type(y))
```

```
5
CloudyML
<class 'int'>
<class 'str'>
```

In [1]:

```
# You can also assign multiple variables at the same time like this
a, b, c = 'Hack', 'weekly', 'group'
print(a)
print(b)
print(c)
```

```
Hack
weekly
group
```

In [2]:

```
# How to unpack things ??? Let's see
# Unpacking a list
fruits = ['Python', 'Java', 'C++']
a, b, c = fruits
print(a)
print(b)
print(c)
```

```
Python
Java
C++
```

In [7]:

```
# Casting of a variable
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
print(x)
print(y)
print(z)
print(type(x))
print(type(y))
print(type(z))
```

```
3
3
3.0
<class 'str'>
<class 'int'>
<class 'float'>
```

In [10]:

```
# Variables names are case sensitive
a = 4
A = "cloud"
#A will not overwrite a
print(a)
print(A)
```

```
4
cloud
```

In [11]:

```
# If you define variable with same name then the variable's old value will be replaced by n
A = "ML"
print(A) # previously it was A = "cloud"
```

```
ML
```

In [4]:

```
# Would you like to add two variables ? Obviously it's gonna happen so let's learn what can

var1 = 'Python is ' # observe the space before last comma
var2 = 'awesome'
# You can add two strings
sum = var1 + var2
print(sum) # you could simple print(var1 + var2) also
```

```
Python is awesome
```

In [10]:

```
# Now let's add two integers
num1 = 19
num2 = 47
sum_num = num1 + num2
print("Adding while my variables are in integer form --->", sum_num)

# suppose if I want to type cast integer to string and then add?? How will I do this??
num1_str = str(num1)
num2_str = str(num2)
print("After type casting integer to string, and then adding two numbers --->",num1_str+num
```

```
Adding while my variables are in integer form ---> 66
After type casting integer to string, and then adding two numbers ---> 1947
```

In [11]:

```
# Suppose you want to add string with numbers or vice versa... I mean different variables..
str1 = "Python version is "
version = 3.8
# print(str1 + version)         # it will throw TypeError, You can try running the code
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-3f0ffd7e9c9e> in <module>
      2 str1 = "Python version is "
      3 version = 3.8
----> 4 print(str1 + version)

TypeError: can only concatenate str (not "float") to str
```

In [12]:

```
# But you can add string with integer by type casting integer to string
print(str1 + str(version))
```

```
Python version is 3.8
```

In [19]:

```
# What is local and global variable??

x = "awesome"  # global variable and it is not function dependant

def myfunc():
    x = "fantastic" # Local variable because it is defined inside function and it has scope
    print("CloudyML is " +  x) # here local variable gets priority when name of local and g

myfunc()

print("CloudyML is " + x)
```

```
CloudyML is fantastic
CloudyML is awesome
```

In [20]:

```
# What if you want to call global variable with same name as local inside a function??

# Use global keyword


x = "awesome"  # global variable and it is not function dependant

def myfunc():
    global x
    x = "fantastic" # Local variable because it is defined inside function and it has scope
    print("CloudyML is " +  x) # here global variable gets priority because I used global k

myfunc()

print("CloudyML is " + x)
```

```
CloudyML is fantastic
CloudyML is fantastic
```

In [29]:

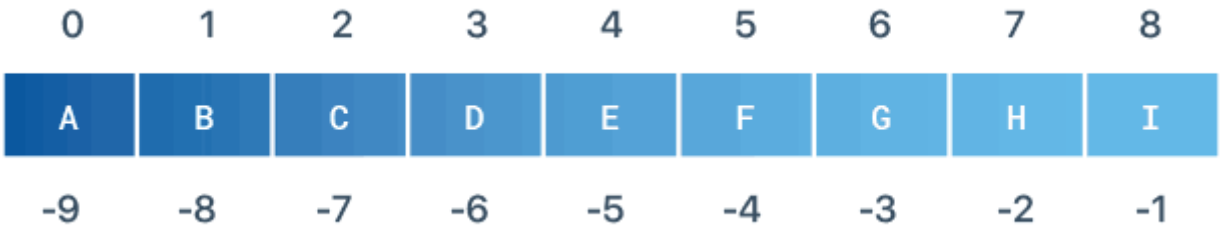
```
# String formatting method

# Method 1
quantity = 5
item_name = "Mango"
price = 10
order = "I want {} pieces of {} for {} dollars." # {} is called placeholder, so 3 placehold
print(order.format(quantity, item_name, price))

# Method 2
order =  "I want {0} pieces of {1} for {2} dollars."  # here 0,1,2 tells the position of qu
print(order.format(quantity, item_name, price))

# Method 3
print(f"I want {quantity} pieces of {item_name} for {price} dollars.")  # Observe the f out
                                                                        # names passed in

# Method 4
print("I want %d pieces of %s for %.0f dollars."%(quantity, item_name, price)) # %d for int
                                                                              # with 0 de
```

```
I want 5 pieces of Mango for 10 dollars.
I want 5 pieces of Mango for 10 dollars.
I want 5 pieces of Mango for 10 dollars.
I want 5 pieces of Mango for 10 dollars.
```

String slicing

Syntax
let's say we have a string 'a' and we want to use indexing to get something from the string

a[start:stop:step] -----> start means starting index number, stop is the last index no which won't be included, # step is the jump from start



Observe the index numbering from left to right and right to left which we will use in next cell

In [3]:

```
a = "Data Scientist is the sexiest job of 21st century"
print(a[:10]) # start here is not mentioned means 0, stop is the 10th index which is 't', s
print(a[:10:2]) # this time every 2nd element from a till 10th index no
print(a[::2])    # every 2nd element from beginning to end in a
print(a[::-1])   # reversing a (if you start indexing from the end of the string, it starts
print(a[-7:])  # extracting century
```

```
Data Scien
Dt ce
Dt cets stesxetjbo 1tcnuy
yrutnec ts12 fo boj tseixes eht si tsitneicS ataD
century
```

Boolean value concept

- Almost any value is evaluated to True if it has some sort of content.
- Any string is True, except empty strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.

Next few commands where we are using some equality operators will return boolean True or False

In [8]:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

In [9]:

```
# these print will return False
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
False
False
False
False
False
False
False
```

In [11]:

```
# These print will return True
print(bool("abc"))
print(bool(123))
print(bool(["apple", "cherry", "banana"]))
```

```
True
True
True
```

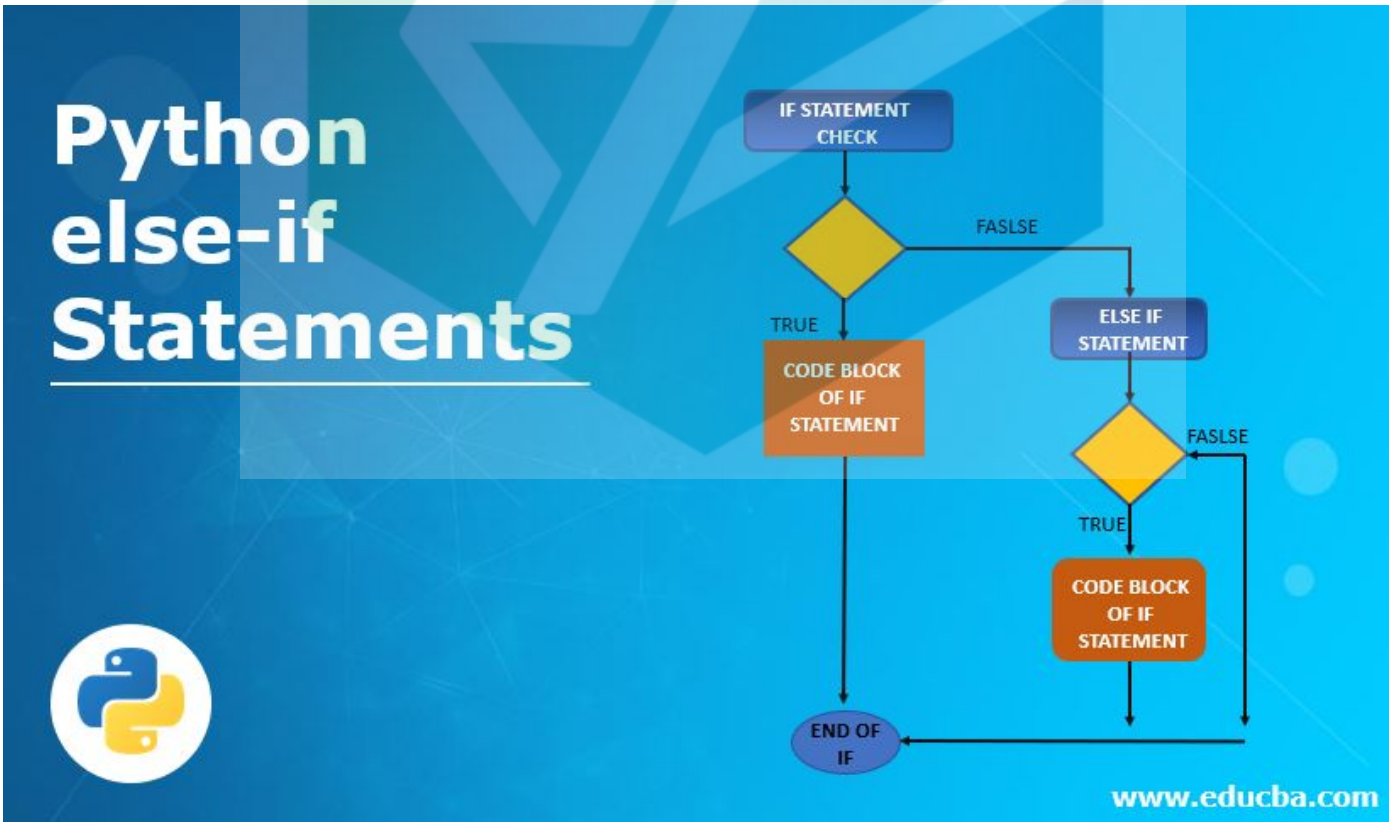
Python Arithmetic operators

In [14]:

```
x = 15
y = 4


print('x + y =',x+y)    # Output: x + y = 19

print('x - y =',x-y)    # Output: x - y = 11

print('x * y =',x*y)    # Output: x * y = 60

print('x / y =',x/y)    # Output: x / y = 3.75

print('x // y =',x//y)   # Output: x // y = 3

print('x ** y =',x**y)   # Output: x ** y = 50625

print('x % y =', x % y)  # output  x % y = 3
```

```
x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x // y = 3
x ** y = 50625
x % y = 3
```

If elif, else statement



These are the conditions used basically in if, elif, and else statement

- Equals: a == b

- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

In [15]:

```
# simple if statement
a = 100
b = 101
if b > a:  # observe the ':' after if block
  print("b is greater than a")  # Observe the print statment line indentation(4 spaces gap
```

```
b is greater than a
```

In [18]:

```
a = 100
b = 50
if b > a:
  print("b is greater than a")
elif a == b:                    # elif keyword is pythons way of saying "if the previous
  print("a and b are equal")    # conditions were not true, then try this condition".

else:        # else keyword catches anything which isn't caught by the preceding conditions
  print("a is greater than b")
```

```
a is greater than b
```

Explanation

1. First if condition is checked, if it meets the requirement then code under if block is executed.
2. If it fails the requirement of first if condition, then it goes to next condition which is elif.
3. If it meets the requirement of elif condition, it executes code inside elif, otherwise it goes to final else block.

For Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

In [19]:

```
# Simplest for loop over a list

Country = ["India", "China", "USA", "UK", "Russia"]
for x in Country:  # iterating each item in the list
  print(x)
```

```
India
China
USA
UK
Russia
```

In [20]:

```
# Iterating over a string(each letter of a string)

for i in "Galaxy":
    print(i)
```

```
G
a
l
a
x
y
```

In [21]:

```
# Using break in the statement with if condition

for i in "Galaxy":
    if i == 'x':          # once i reaches x in 'Galaxy', it will come out of the loop
        break                  #  break stop the loop before it has looped through all the i
    else:
        print(i)
```

```
G
a
l
a
```

In [22]:

```
# Using continue in the statement with if condition

# continue statement---> we can stop the current iteration of the loop, and continue with t

for i in "Galaxy":
    if i == 'x':          # once i reaches x in 'Galaxy', it will skip the current iteration
        continue
    else:
        print(i)
```

```
G
a
l
a
y
```

While loop

While loop ---> we can execute a set of statements as long as a condition is true.

In [4]:

```
# Let's see an example of while loop

i = 0
while i < 5:
    print("Python is cool language.")
    i += 1
```

```
Python is cool language.
Python is cool language.
Python is cool language.
Python is cool language.
Python is cool language.
```

In [5]:

```
# We can use boolean term in while loop

count = 0
while True:
    print("Java is also cool.")
    count += 1
    if count == 5:
        break
```

```
Java is also cool.
Java is also cool.
Java is also cool.
Java is also cool.
Java is also cool.
```

In [6]:

```
# We can also use else statement with while statement

i = 0
while i < 5:
    print("Python is cool language.")
    i += 1
else:
    print("Python is more than cool language.") # With the else statement we can run a bloc
                                                # once when the condition no longer is tru
```

```
Python is cool language.
Python is cool language.
Python is cool language.
Python is cool language.
Python is cool language.
Python is more than cool language.
```

Functions in Python

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

In [7]:

```
# Function with no parguments

def cloudyfunc():  # Use def keyword to define your own function
    print("AI for all is a an affordable course")

# I have used print in the function so I don't need to define a variable here, just call fu
cloudyfunc()
```

```
AI for all is a an affordable course
```

In [8]:

```
# Here I will use return in the same function cloudyfunc and see how it works

def cloudyfunc():  # Use def keyword to define your own function
    return "AI for all is a an affordable course"

cloudy = cloudyfunc()  # storing returned value from cloudyfunc()

cloudy # print cloudy here(in jupyter or colab, no need to use print in such condition)
```

Out[8]:

```
'AI for all is a an affordable course'
```

In [10]:

```
# Function with arguments

def MLfunc(first_arg, second_arg): # You can define as many arguments
    print(first_arg, second_arg)

MLfunc("cloudy", "ML")
```

```
cloudy ML
```

In [12]:

```
# One recursion example using function for fun

def recursive_function(some_num):
    if some_num > 0:
        result = some_num + recursive_function(some_num - 1)
    else:
        result = 0
    return result

result = recursive_function(5)
result

# Explanation
# WHen i passed 5, 5>0, so if block code will be executed and result = 5 + recursive_functi
# for value 4 as argument, 4>0, again if block will be executed, result = 4 + recursive_fun
# for value 3 as argument, 3>0, again if block will be executed, result = 3 + recursive_fun
# for value 2 as argument, 2>0, again if block will be executed, result = 2 + ecursive_func
# for value 1 as argument, 1>0, again if block will be executed, result = 1 + ecursive_func
# for 0, if block condition will fail and else block will be executed where result = 0

# so finally let's calculate final result
# value 1 will give result = 1
# value 2 will give result = 2 + value1 = 2 + 1 = 3
# value 3 will give result = 3 + value2 = 3 + 3 = 6
# value 4 will give result = 4 + value3 = 4 + 6 = 10
# value 5 will give result = 5 + value4 = 5 + 10 = 15

# Hope you enjoyed recursion function ......
```

Out[12]:

```
15
```

List comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

newlist = [expression for item in iterable if condition == True]

In [17]:

```
# some simple list comprehension examples

num_list = [i for i in range(10)]
letter_list = [i for i in "India"]

print(num_list)
print(letter_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['I', 'n', 'd', 'i', 'a']
```

In [18]:

```
# list comprehension with if condition
vowel_list = [i for i in "UnitedStatesofAmerica" if i in'aeiouAEIOU']
vowel_list
```

Out[18]:

```
['U', 'i', 'e', 'a', 'e', 'o', 'A', 'e', 'i', 'a']
```

In [20]:

```
# showing how to convert normal method to list comprehension

name = "India, United States, Canada, South Africa"
my_list = []
# Let's say I want to add letters 'a' and 's' in my_list

for i in name:
    if i in ['a', 's']:
        my_list.append(i)

print("Normal method way =", my_list)



# That was a little long approach
# I could do that in 1 line

my_new_list = [i for i in name if i in ['a', 's']]    # 4 line of code merged in this one l
print("Short approach using list comp =",my_new_list)
```

```
Normal method way = ['a', 'a', 's', 'a', 'a', 'a', 'a']
Short approach using list comp = ['a', 'a', 's', 'a', 'a', 'a', 'a']
```

In [29]:

```
# Let's see some math library functions
import math

some_num = 10
another_num = 5.6

print(math.exp(some_num))  # exponentional function

print(math.sqrt(some_num))  # square root of some_num

print(math.floor(another_num))  # Rounds a number down to the nearest integer

print(math.ceil(another_num))   # rounds a number up to the nearest integer

ist_of_num = [1,2,3,4]
print(math.prod(list_of_num)) # returns product of all numbers in an iterable

print(math.pow(2, 3))    # returns value of x to the power of y , here x = 2, y = 3

# there are many such functions in math library, for more information click the below link.
```

```
22026.465794806718
3.1622776601683795
5
6
24
8.0
```

For more math library function, click here --> Click here (https://www.w3schools.com/python/module_math.asp)

That's all for Python basics. For any kind of feedback regarding this notes, you can talk to your Mentors.

In [ ]:

```

```