Python Basics

In [1]:

```
# What is your name! print your name!
# Only use one print function

print("Kunal Joshi")
```

```
Kunal Joshi
```

- `()` <= Parentheses
- `''` <= Single Quotes
- `""` <= Double Quotes
- `\n` <= New_line
- `#` <= Used to comment inside code

In [2]:

```
# define variables named as with values: mukesh=7, z=6, rohan=5, longitude=4
manish=7
z=6
rohan=5
longitude=4
```

In [3]:

```
# print required variable
# output - 5
rohan
```

Out[3]:

```
5
```

Variable Assignment: **`Variable_Name = Value`**

Variables Naming Rules:

- `Python is case-senstive` => x=5 is different from X=5 (one is lowe and other is upper case)
- `var name can't start with special character except underscore(_)` => _X = 7 is `valid` , @X = 7 is `invalid`
- `var name can't start with number` => 9X = 7 is `invalid` , X9 = 7 is `valid`
- can't use `keywords` as a variable name *

Declaring a Variable

In [4]:

```
# declare 4 variables with values as: ur_age 21,ur_weight 50.6, ur_first_name = 'Mukesh'
ur_age = 20
ur_weight = 50.6
ur_first_name = 'Manish'
ur_last_name = "Manral"
```

Data Type(Type of variable)

Name	Type	Description
Integers	int	Integer number, like 34,-56 ...
Float	float	Decimal number, like 3.4,-5.6 ...
String	str	Ordered sequence of characters, like 'your name'
Boolean	bool	Logical values indicating True or False only

In [5]:

```
# print type of ur_age,ur_weight,ur_first_name,ur_last_name variables
print(type(ur_age))
print(type(ur_weight))
print(type(ur_first_name))
print(type(ur_last_name))
```

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

In [6]:

```
# print values of ur_age,ur_weight,ur_first_name,ur_last_name variables
print(ur_age)
print(ur_weight)
print(ur_first_name)
print(ur_last_name)
```

```
20
50.6
Manish
Manral
```

In [7]:

```
# make 2 variables with values as: ur_first_name 'Mukesh',ur_last_name'Mukesh'

# make a variable TrueOrFalse which will have comparison of variables ur_last_name == ur_
ur_first_name = 'Mukesh'
ur_last_name = 'Mukesh'

TrueOrFalse = ur_last_name == ur_first_name

TrueOrFalse
```

Out[7]:

```
True
```

In [8]:

```
# define a variable name "x" and assign value 777 and print it
x = 777
print(x)
```

```
777
```

- To view some data on screen, python have `print` function
  - Using `print` function we can control view on output screen

In [ ]:

```

```

`Operators` : Symbols that represent mathematical or logical tasks

Example:
`700 + 77`

- `+` <= Operator
- `700 & 77` <= Operands

In [9]:

```
# Initialize variables [x,y,z,zz] with values
## x as 7 =>int ,
## y as 77 =>int,
## z as 77.7 => float,
## zz as 'Hi' => string
x = 7
y = 77
z = 77.7
zz = 'Hi'
```

Arithmetic Operators

In [10]:

```
# add x and z
add = x + y
add
```

Out[10]:

```
84
```

In [11]:

```
# subtract z and y
sub = z - y
sub
```

Out[11]:

```
0.7000000000000028
```

In [12]:

```
# Multiply x and z
mul=x*z
mul
```

Out[12]:

```
543.9
```

In [13]:

```
# Exponent (raise the power or times) x times z
exp = x**z
exp
```

Out[13]:

```
4.614426248242042e+65
```

In [14]:

```
# division on x and z
div=x/z
div
```

Out[14]:

```
0.09009009009009009
```

`//` => divides and returns integer value of quotient

- It will dump digits after decimal

In [15]:

```
# floor division(ignores decimal) on x and z (gives quotient)
fdiv = x // z
fdiv
```

Out[15]:

```
0.0
```

In [17]:

```
# Modulo(gives remainder) on x and z
mod = x % z
mod
```

Out[17]:

```
7.0
```

Comparison Operators

In [16]:

```
# comapre and see if x is less then z
# can use '<' symbol
com = x < z
com
```

Out[16]:

```
True
```

In [22]:

```
# check the type of above comaprison where it says comapre and see if x is less then z
com1 = x <z
print(type(com1))
```

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

- `Bool` => takes two values, either `True` or `False`

In [24]:

```
# compare and see if x is less then or equall to z
# can use '<=' symbol
com1 = x <= z
com1
```

Out[24]:

```
True
```

In [20]:

```
# comapre and see if x equall to z
# can use '==' symbol
com2 = x == z
com2
```

Out[20]:

```
False
```

In [23]:

```
# comapre and see if x is greater than z
# can use '>' symbol
com3 = x > z
com3
```

Out[23]:

```
False
```

In [25]:

```
# comapre and see if x is greater than or equall to z
# can use '>=' symbol
com4 = x >= z
com4
```

Out[25]:

```
False
```

In [26]:

```
# comapre and see if x is Not equall to z
# can use '!=' symbol
com5 = x != z
com5
```

Out[26]:

```
True
```

Logical Operators

In [27]:

```
# compare if 108 is equall to 108, 21 is equall to 21 using logical and
# equall to => '=='
# logical and => and
# in and both condition must be True to get a True
com6 = 108 == 108 and 21 == 21
com6
```

Out[27]:

```
True
```

In [29]:

```
# how above condition can give False as output show all those conditions
```

In [28]:

```
# compare if 108 is equall to 108, 21 is equall to 11 using logical or
# equall to => '=='
# logical or => or
# in or Only one condition need to be True to get a True
com7 = 108 == 108 or 21 == 11
com7
```

Out[28]:

```
True
```

In [31]:

```
# this is for you to understand it
(108 == 108) or (21 == 11) or (108 <= 11)
```

Out[31]:

```
True
```

In [ ]:

```

```

`if` --- `else` => to handle single condition

`if` --- `elif` --- `else` => to handle Multiple condition

`Observe in Python code:`

- `if` => statement in python
- `else` => statement in python
- `:` => colon => denotes start of if block i.e. any line written after colon belong to if condition
- `....` => see then as indentation i.e. 4 spaces => indentation indicates all code belong to only if and then another indentation indicates code for only else block

In [29]:

```
# make variable with value as : money 100000
# see output of money > 2000
money = 100000
if money > 2000:
    print(money)
```

```
100000
```

In [30]:

```
# assign money variable value of 10000
##### say you have this much ammount in your account
# start of if condition
# if money is greater then 1000 which is data science course free
# if money > 1000 is false i.e. you have less money then 1000 in your account then else 
money = 10000
if money > 1000:
    print(" data science course free")
else:
    print("you have less money then 1000 in your account then else will work for now onl
```

```
 data science course free
```

In [31]:

```
# take a test_score variable with 80 in it.
# if test_score greater then 80 then print A grade
# elif test_score greater then 60 and less then 80 print B grade
# else print Nothing for you
test_score = 80
if test_score >= 80:
    print("A grade")
elif (test_score >= 60) and(test_score < 80):
    print(" B grade")
else:
    print("Nothing for you")
```

```
A grade
```

In [ ]:

```

```

Python Loops

In [39]:

```
"""
for iterating_variable in sequence:
    statement(s)
"""
```

Out[39]:

```
'\nfor iterating_variable in sequence:\n    statement(s)\n'
```

In [40]:

```
for iterating_variable in range(10):
    print(iterating_variable)
```

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

In [32]:

```
# print 'I love sports' 10 times using for loop
for i in range(10):
    print("I love sports")
```

```
I love sports
I love sports
I love sports
I love sports
I love sports
I love sports
I love sports
I love sports
I love sports
I love sports
```

`10 => stoping criteria` of, for loop

- `in` => keyword
- `sequence` => on which to itterate
- `:` => colon , start of for loop

`!=` = not equall to => `behaves as a stoping criteria`

In [42]:

```
# Syntax of while loop
"""
while comparison:
    statements(s)
"""
```

Out[42]:

```
'\nwhile comparison:\n    statements(s)\n'
```

In [33]:

```
# while loop
# save 0 in variable number
# print till 10 using while loop
i = 0
while i < 11:
    print(i)
    i+=1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

- Initialized variable `number = 0` and then increment it's value in each iteration
- Loop will only continue to run only if value is less than 10

Type of Jump Statements

`Break Statement` `Continue Statement`

Break Statement

In [34]:

```
# example that uses break statement in a for loop
# take range(10) and print 'The number is' + value
# break when num equals 5
for i in range(10):
    print("The number is",i)
    if i==5:
        break
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

Continue Statement

In [35]:

```
# Using same `for loop program` as in Break Statement section above
# Use a continue statement rather than a break statement
# take range(10) and print 'The number is' + value
# continue when num equals 5
# Using same `for loop program` as in Break Statement section above
# Use a continue statement rather than a break statement
# take range(10) and print 'The number is' + value
# continue when num equals 5
for i in range(10):
    if i==5:
        continue
    print("The number is",i)
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 6
The number is 7
The number is 8
The number is 9
```

In [ ]:

```

```

String Manipulation

In [49]:

```
string_ = '' or "" or """ """
```

In [36]:

```
# define a string variable with "We are creating next generation data science eco-system
s = "We are creating next generation data science eco-system at CollegeRanker"
```

In [37]:

```
# Find length of string including spaces
len(s)
```

Out[37]:

```
72
```

In [38]:

```
# Access characters in a string with indexing i.e string[0]

s[0]
```

Out[38]:

```
'W'
```

In [39]:

```
# Access characters with negative indexing i.e string[-1]
s[-1]
```

Out[39]:

```
'r'
```

String Slicing

In [40]:

```
# select string from first to 6th element i.e string[:6]

s[:6]
```

Out[40]:

```
'We are'
```

In [42]:

```
# select string from 7th to negative 10th element i.e string[7:-10]

s[7:-10]
```

Out[42]:

```
'creating next generation data science eco-system at Col'
```

Count of a particular `character` in a string

In [43]:

```
s.count("data")
```

Out[43]:

```
1
```

Count of a particular `sub-string` in a string

In [44]:

```
s.count("s")
```

Out[44]:

```
3
```

Find a substring in string using `find` and `index` function

In [46]:

```
# .find() => if present it will return starting index, not found then it will return -1
# .index() => if present it will return starting index, not found then it will give erro
print(s.find("s"))
print(s.index("s"))
```

```
37
37
```

In [47]:

```
### Checking whether string `startswith` or `endswith` a particular substring or not
start = s.startswith('We')
end = s.endswith('CollegeRanker')
start, end
```

Out[47]:

```
(True, True)
```

In [ ]:

```

```

In [49]:

```
### Converting string to upper case ###
txt = "kuna joshi"
a = txt.upper()
a
```

Out[49]:

```
'KUNA JOSHI'
```

In [50]:

```
### Converting only first character of string to upper case
b = txt.capitalize()
b
```

Out[50]:

```
'Kuna joshi'
```

In [51]:

```
### Checking if string is in lower case or upper case
c = txt.islower()
print(c)
d = txt.isupper()
print(d)
```

```
True
False
```

In [52]:

```
### Checking if string is digit, alpabetic, alpha-numeric
e = txt.isdigit()
f = txt.isalpha()
g = txt.isalnum()

print(e)
print(f)
print(g)
```

```
False
False
False
```

In [53]:

```
# assign "C++ is easy to learn" to a new_str variable
new_str = "C++ is easy to learn"
new_str
```

Out[53]:

```
'C++ is easy to learn'
```

In [54]:

```
### Replace C++ with Python

result = new_str.replace("C++", "Python")
print(result)
```

```
Python is easy to learn
```

In [55]:

```
### Use Split function on new_str ###
h = new_str.split(',')
print(h)
```

```
['C++ is easy to learn']
```

Python Functions

In [76]:

```
"""
def function_name():
    stetement(s)
"""
```

Out[76]:

```
'\ndef function_name():\n    stetement(s)\n'
```

In [56]:

```
# define a function with welcome_message(name) and body 'Welcome to Functions !!!'
def welcome_message(name):
    print(name,'Welcome to Functions !!!')
```

In [57]:

```
# call a function with your name

welcome_message("Kunal")
```

```
Kunal Welcome to Functions !!!
```

- `def` Keyword marking start of function
- `function name` to uniquely identify function
    - `function naming` follows same `rules of writing identifiers`
- `parameters` (arguments) to pass values to a function => totally optional
- `()` paranthesis
- `colon (:)` start of function
- `documentation string` (docstring) describe's what function does => totally optional
- `return statement` returns a value from function => totally optional
- inside colon is `function definition` it should always be present before function call or get an error

In [58]:

```
# Write a function to add two number which are as 3 and 4
# in total variable store adition of 3 + 4
# print total variable
def add():
    total = 3 + 4
    print(total)
add()
```

```
7
```

Positional Arguments

Most arguments are identified by their position in function call

- Say `print(x,y)` will give different results from `print(y,x)`

What ever sequence is given while defining a function values must be taken in that sequence only

- Otherwise use argument name **`(keyword arguments)`** to take values
- We first define `positional argument` and then `keyword arguments`

In [59]:

```
## Create substraction_function(small_number,large_number) and return difference between

def substraction_function(small_number,large_number):
    diff = large_number - small_number
    return diff
```

In [60]:

```
# pass arguments in right order

substraction_function(5,10)
```

Out[60]:

```
5
```

In [61]:

```
# always pass arguments using there name(keyword arguments) then order does not matter

substraction_function(small_number = 5, large_number = 10)
```

Out[61]:

```
5
```

Scope of Variables means that part of program where we can access particular variable

- `Local Variable` => variables defined inside a function and can be only accessed from inside of that particular function
- `Global Variable` => variables defined outside a function and can be accessed throughout program

Let's define a global variable, `"global_variable"` outside function

- We will return its value using a function `"randome_function"` and see that we would be able to access its value using that function also

In [62]:

```
#### Observe every output from here onwords #####
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # accessing variable which is outside of this function
    return global_variable
```

In [63]:

```
random_function()
```

Out[63]:

```
'variable outside of function'
```

```
See we can acess the data of golbal variable from Inside of the Function
```

=> Let's see what will happen if we try to change value of global variable from Inside of the Function

In [64]:

```
#### Observe every output from here onwords #####
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # changing value of global variable from inside of the function
    global_variable = 'changing variable outside of function from inside of function'
    # accessing variable which is outside of this function
    return global_variable
```

In [65]:

```
print(random_function())
print(global_variable)
```

```
changing variable outside of function from inside of function
variable outside of function
```

In [66]:

```
global_var = "Hi! I am from Global RFM team"

def rfm():
  return global_var

rfm()
```

Out[66]:

```
'Hi! I am from Global RFM team'
```

In [67]:

```
global_variable = 23

def rfm():
  global_variable = 25
  return global_variable

print(rfm())
print(global_variable)
```

```
25
23
```

In [ ]:

```
```