Python Fundamentals for Machine Learning

First program

In [1]:
```
# Program to find simple interest
p = int(input("\n Enter the principal Amount:"))
t = int(input("\n Enter the time period:"))
r = float(input("\n Enter the rate of interest:"))
si = p*t*r/100
print("\n Simple Interest:",si)
```

```
 Enter the principal Amount:1000

 Enter the time period:2

 Enter the rate of interest:8

 Simple Interest: 160.0
```

Output using print

In [2]:
```
print('''This sentence is output to the screen''')
a=5
print("The value of a is:",a)
print('x:',1,2,3,4)
x = 5 ; y = 10
print('The value of x is {} and y is {}'.format(x,y))
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
```

```
This sentence is output to the screen
The value of a is: 5
x: 1 2 3 4
The value of x is 5 and y is 10
I love bread and butter
I love butter and bread
```

In [3]:
```
print('Hello {name}, {greeting}'.format(greeting = 'Good Morning!!',\
                                        name = 'John'))
```

```
Hello John, Good Morning!!
```

In [4]:
```
x = 12.3456789
print('The value of x is %3.2f' %x)
print('The value of x is %3.4f' %x)
```

```
The value of x is 12.35
The value of x is 12.3457
```

In [5]:
```
for x in range(1, 11):
    print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

```
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

In [6]:
```
table = {'Raju': 9480123526, 'Ravi': 9480123527, 'Rahul': 9480123528}
for name, phone in table.items():
    print('{0:10} ==> {1:10d}'.format(name, phone))
```

```
Raju       ==> 9480123526
Ravi       ==> 9480123527
Rahul      ==> 9480123528
```

In [7]:
```
import math
print('The value of PI is approximately %5.3f.' % math.pi)
```

```
The value of PI is approximately 3.142.
```

Input using input

In [8]:
```
x = input('Enter a string: ')
print("The entered string is :{0}".format(x))
y = int(input('Enter a integer: '))
print("The entered integer is :",y)
z = float(input('Enter a floating point number:'))
print("The entered real number  is :",z)
```

```
Enter a string: Apple
The entered string is :Apple
Enter a integer: 10
The entered integer is : 10
Enter a floating point number:2.4
The entered real number  is : 2.4
```

Multiline Statements

In [9]:
```
# Example of implicit line continuation
x = ('1' + '2' +
     '3' + '4')
# Example of explicit line continuation
y = '1' + '2' + \
'11' + '12'
weekdays = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
weekday = {'one':
           'Monday'}
print ('x has a value of', x)
print ('y has a value of', y)
print(weekdays)
print(weekday)
```

```
x has a value of 1234
y has a value of 121112
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
{'one': 'Monday'}
```

In [10]:
```
import os; x = 'Hello'; print(x)
```

```
Hello
```

Conditional Execution

Example code for simple "if" statement

In [11]:
```
var = -1
if var < 0:
    print(var)
    print("the value of var is negative")
# If there is only a single clause then it may go on the same line as the
# header statement
if ( var == -1 ) :
    print("the value of var is negative")
```

```
-1
the value of var is negative
the value of var is negative
```

In [12]:
```
#Example code for ‘if else’ statement
var = 1
if var < 0:
    print("the value of var is negative")
    print(var)
else:
    print("the value of var is positive")
    print(var)
```

```
the value of var is positive
1
```

In [13]:
```
# Example for nested if else
score = 95
if score >= 99:
    print('A')
elif score >=75:
    print('B')
elif score >= 60:
    print('C')
elif score >= 35:
    print('D')
else:
    print('F')
3.0
```

```
B
```

Out[13]:
```
3.0
```

Iterations

Usage of "for" loop

In [14]:
```
# First Example
print("First Example")
for item in [1,2,3,4,5]:
    print('item :', item)
# Second Example
print("Second Example")
letters = ['A', 'B', 'C']
for index in range(len(letters)):
    print('First loop letter :', letters[index])
```

```
First Example
item : 1
item : 2
item : 3
item : 4
item : 5
Second Example
First loop letter : A
First loop letter : B
First loop letter : C
```

In [15]:
```
#While loop: The while statement repeats a set of code until the condition is true

#Example code for while loop statement
count = 0
while (count <3):
    print('The count is:', count)
    count = count + 1
```

```
The count is: 0
The count is: 1
The count is: 2
```

Lists

In [16]:
```
#Python’s lists are the most flexible data type. It can be created by writing a list of comm
a separated values between square brackets. Note that that the items in the list need not be
of the same data type

# Example code for accessing lists
# Create lists
list_1 = ['Statistics', 'Programming', 2016, 2017, 2018]
list_2 = ['a', 'b', 1, 2, 3, 4, 5, 6, 7 ]
# Accessing values in lists
print("list_1[0]: ", list_1[0])
print("list2_[1:5]: ", list_2[1:5])
```

```
list_1[0]:  Statistics
list2_[1:5]:  ['b', 1, 2, 3]
```

In [17]:
```
#Example code for adding new values to lists
print("list_1 values: ", list_1)
# Adding new value to list
list_1.append(2019)
print("list_1 values post append: ", list_1)
```

```
list_1 values:  ['Statistics', 'Programming', 2016, 2017, 2018]
list_1 values post append:  ['Statistics', 'Programming', 2016, 2017, 2018, 2019]
```

In [18]:
```
#Example code for updating existing values of lists
print("Values of list_1: ", list_1)
# Updating existing value of list
print("Index 2 value : ", list_1[2])
list_1[2] = 2015;
print("Index 2's new value : ", list_1[2])
```

```
Values of list_1:  ['Statistics', 'Programming', 2016, 2017, 2018, 2019]
Index 2 value :  2016
Index 2's new value :  2015
```

Example code for basic operations on lists

In [28]:
```
import string
import operator
#Example code for basic operations on lists

print("Length: ", len(list_1))

print("Concatenation: ", [1,2,3] + [4, 5, 6])

print("Repetition :", ['Hello'] * 4)

print("Membership :", 3 in [1,2,3])

print("Iteration :")
for x in [1,2,3]: print(x)

# Negative sign will count from the right
print("slicing :", list_1[-2])

# If you dont specify the end explicitly, all elements from the specified
#start index will be printed
print("slicing range: ", list_1[1:])

print("Max of list: ", max([1,2,3,4,5]))

print("Min of list: ", min([1,2,3,4,5]))

print("Count number of 1 in list: ", [1,1,2,3,4,5,].count(1))

list_1.extend(list_2)

print("Extended :", list_1)

print("Index for Programming:",list_1.index('Programming'))
print (list_1)
print("pop last item in list: ", list_1.pop())
print("pop the item with index 2: ", list_1.pop(2))
list_1.remove('b')
print("removed b from list: ", list_1)
list_1.reverse()
print("Reverse: ", list_1)
list_1 = ['a','c','b']
list_1.sort()
print("Sort ascending: ", list_1)
list_1.sort(reverse = True)
print("Sort descending: ", list_1)
```

```
Length:  5
Concatenation:  [1, 2, 3, 4, 5, 6]
Repetition : ['Hello', 'Hello', 'Hello', 'Hello']
Membership : True
Iteration :
1
2
3
slicing : 2017
slicing range:  ['Programming', 2015, 2017, 2018]
Max of list:  5
Min of list:  1
Count number of 1 in list:  2
Extended : ['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
Index for Programming: 1
['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
pop last item in list:  7
pop the item with index 2:  2015
removed b from list:  ['Statistics', 'Programming', 2017, 2018, 'a', 1, 2, 3, 4, 5, 6]
Reverse:  [6, 5, 4, 3, 2, 1, 'a', 2018, 2017, 'Programming', 'Statistics']
Sort ascending:  ['a', 'b', 'c']
Sort descending:  ['c', 'b', 'a']
```

Tuples

A Python tuple is a sequences or series of immutable Python objects very much similar to the lists. However there exist some essential differences between lists and tuples, which are the following. 1) Unlike list, the objects of tuples cannot be changed. 2) Tuples are defined by using parentheses, but lists are defined by square brackets

In [29]:
```
# Example code for creating tuple
# Creating a tuple
Tuple = ()
print("Empty Tuple: ", Tuple)
Tuple = (1,)
print("Tuple with single item: ", Tuple)
Tuple = ('a','b','c','d',1,2,3)
print("Sample Tuple :", Tuple)
```

```
Empty Tuple:  ()
Tuple with single item:  (1,)
Sample Tuple : ('a', 'b', 'c', 'd', 1, 2, 3)
```

In [30]:
```
#Example code for accessing tuple
# Accessing items in tuple
Tuple = ('a', 'b', 'c', 'd', 1, 2, 3)
print("3rd item of Tuple:", Tuple[2])
print("First 3 items of Tuple", Tuple[0:2])
```

```
3rd item of Tuple: c
First 3 items of Tuple ('a', 'b')
```

In [31]:
```
#Example code for deleting tuple
# Deleting tuple
print("Sample Tuple: ", Tuple)
del Tuple
print(Tuple) # Will throw an error message as the tuple does not exist
```

```
Sample Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-efdc3134feeb> in <module>()
      3 print("Sample Tuple: ", Tuple)
      4 del Tuple
----> 5 print(Tuple) # Will throw an error message as the tuple does not exist

NameError: name 'Tuple' is not defined
```

In [32]:
```
# Example code for basic operations on tupe (not exhaustive)
# Basic Tuple operations
Tuple = ('a','b','c','d',1,2,3)
print("Length of Tuple: ", len(Tuple))
Tuple_Concat = Tuple + (7,8,9)
print("Concatinated Tuple: ", Tuple_Concat)

print("Repetition: ", (1,'a',2, 'b') * 3)
print("Membership check: ", 3 in (1,2,3))
# Iteration
for x in (1, 2, 3): print(x)
print("Negative sign will retrieve item from right: ", Tuple_Concat[-2])
print("Sliced Tuple [2:] ", Tuple_Concat[2:])
# Find max
print("Max of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
max((1,2,3,4,5,6,7,8,9,10)))
print("Min of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
min((1,2,3,4,5,6,7,8,9,10)))
print("List [1,2,3,4] converted to tuple: ", type(tuple([1,2,3,4])))
```

```
Length of Tuple:  7
Concatinated Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3, 7, 8, 9)
Repetition:  (1, 'a', 2, 'b', 1, 'a', 2, 'b', 1, 'a', 2, 'b')
Membership check:  True
1
2
3
Negative sign will retrieve item from right:  8
Sliced Tuple [2:]  ('c', 'd', 1, 2, 3, 7, 8, 9)
Max of the Tuple (1,2,3,4,5,6,7,8,9,10):  10
Min of the Tuple (1,2,3,4,5,6,7,8,9,10):  1
List [1,2,3,4] converted to tuple:  <class 'tuple'>
```

Dictionary

The Python dictionary will have a key and value pair for each item that is part of it. The key and value should be enclosed in curly braces. Each key and value is separated using a colon (:), and further each item is separated by commas (,). Note that the keys are unique within a specific dictionary and must be immutable data types such as strings, numbers, or tuples, whereas values can take duplicate data of any type.

In [33]:
```
# Example code for creating dictionary
# Creating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
```

In [34]:
```
# Example code for accessing dictionary
# Accessing items in dictionary
print("Value of key Name, from sample dictionary:", dict['Name'])
```

```
Value of key Name, from sample dictionary: Jivin
```

In [35]:
```
#Example for deleting dictionary
# Deleting a dictionary

dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict0)
k=1
for i in dict0:
    print(k,i,dict0[i])
    k=k+1
del (dict0['Name']) # Delete specific item

print("Sample dictionary post deletion of item Name:", dict0)

dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
dict0.clear() # Clear all the contents of dictionary
print("dict post dict.clear():", dict0)

dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
del (dict0) # Delete the dictionary
#print(dict0)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
1 Name Jivin
2 Age 6
3 Class First
Sample dictionary post deletion of item Name: {'Age': 6, 'Class': 'First'}
dict post dict.clear(): {}
```

In [36]:
```
#Example code for updating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
dict['Age'] = 6.5
print("Dictionary post age value update: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Dictionary post age value update:  {'Name': 'Jivin', 'Age': 6.5, 'Class': 'First'}
```

In [37]:
```
#Example code for basic operations on dictionary
# Basic operations
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Length of dict: ", len(dict))

# Copy the dict
dict1 = dict.copy()
print("Copy:\n",dict1)

# Retrieve value for a given key
print("Value for Age: ", dict.get('Age'))

# Return items of dictionary
print("dict items: ", dict.items())

# Return items of keys
print("dict keys: ", dict.keys())

# return values of dict
print("Value of dict: ", dict.values())

# Concatenate dicts
dict1 = {'Name': 'Jivin', 'Age': 6}
dict2 = {'Sex': 'male' }
dict1.update(dict2)
print("dict1.update(dict2) = ", dict1)
```

```
Length of dict:  3
Copy:
 {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Value for Age:  6
dict items:  dict_items([('Name', 'Jivin'), ('Age', 6), ('Class', 'First')])
dict keys:  dict_keys(['Name', 'Age', 'Class'])
Value of dict:  dict_values(['Jivin', 6, 'First'])
dict1.update(dict2) =  {'Name': 'Jivin', 'Age': 6, 'Sex': 'male'}
```

User-Defined functions

A user-defined function is a block of related code statements that are organized to achieve a single related action. The key objective of the user-defined functions concept is to encourage modularity and enable reusability of code. Syntax for creating functions without argument: def functoin_name(): 1st block line 2nd block line ...

In [38]:
```
# Example code for creating functions without argument

# Simple function
def someFunction():
    print("Hello World")

# Call the function
someFunction()
```

```
Hello World
```

Syntax for Creating Functions with Argument def functoin_name(parameters): 1st block line 2nd block line ... return [expression]

In [39]:
```
#Example code for creating functions with arguments

# Simple function to add two numbers
def sum_two_numbers(x, y):
        return x + y

# after this line x will hold the value 3
print(sum_two_numbers(1,2))
```

```
3
```

Scope of Variables The availability of a variable or identifier within the program during and after the execution is determined by the scope of a variable. There are two fundamental variable scopes in Python. 1. Global variables 2. Local variables

In [40]:
```
#Example code for defining variable scopes
# Global variable
x = 10
# Simple function to add two numbers
def sum_two_numbers(y):
    return x + y
# Call the function and print result
print(sum_two_numbers(10))
```

```
20
```

In [41]:
```
#Variable Length Arguments
# Example code for passing argumens as *args
# Simple function to loop through arguments and print them

def sample_function(*args):
    for a in args:
        print(a)

# Call the function
sample_function(1,2,3)
```

```
1
2
3
```

In [42]:
```
#Example code for passing argumens as 2D
# Simple function to loop through arguments and print them

def sample_function(**args):
    for a in args:
        print(a, args[a])
# Call the function
sample_function(name='John', age=27)
```

```
name John
age 27
```

In [43]:
```
#Lambda Function
def add(x, y):
    return x + y
print("FUNCTION ADD:\n",add(3,2))

add = lambda x, y : x + y
print("LAMBDA ADD :\n",add(3,2))
```

```
FUNCTION ADD:
 5
```