Python Programming

- The focus will be on features that are important from data analysis perspective.
  - Declaring Variables
  - Conditional Statements
  - Control Flow
  - Functions
  - Collections (List, Tuple, Sets, Dictionary)

Declaring Variables

- A variable can be declared and then assigned a value without specifying data type.
- Python automatically infers the variable type from values assigned to it.
- A Variable initialized with a value of one type can later be re-assigned as value of a different type.
  - int – Integer type
  - float – Floating point number
  - bool – Boolean value (True/False)
  - str – Textual data

Declaring Variables

Example:

```
var1 = 2
var2 = 5.0
var3 = True
var4 = "Python Programming"
```

Above are the variables creating with various data types (int, float, booleans, string)

Declaring Variables

print() used to print the value of any variables and **type()** function used check the data types of any variables.

```
print(var1)
print(var2)
print(var3)
```

Declaring Variables

print() used to print the value of any variables and **type()** function used check the data types of any variables.

```
print(var1)
print(var2)
print(var3)
```

```
type(var1)  # gives result as int
type(var2)  # gives result as float
type(var3)  # gives result as bool
type(var4)  # gives result as str
```


Conditional Statement

- Python supports if-elif-else for writing conditional statements.
- The condition should be terminated by : (colon) and code block following that must be indented.

Syntax:

```
if condition:
        statements
elif condition:
        statements
else:
        statements
```


Conditional Statement

Example : Simple If

Write conditional statement to check whether variable contains positive value.

```
var1 = 5
if  var1 > 0:
    print("True. Variable contains positive numbers")
```


Conditional Statement

Example : If..Else Statement

Write conditional statement to check then condition and display appropriate message for True & False condition.

```
var2 = -20
if  var2 > 0:
    print("True. Value is positive numbers")
else:
    print("False. Value is not positive numbers")
```

Conditional Statement

Example : if..elif..else

With the help of if..elif..else we can add & check multiple conditions.

```
x = 10
y = 20
if  x>y:
    print("X is greater than Y")
elif y>x:
    print("Y is greater then X")
else:
    print("X & Y are same")
```


Sequence Numbers Generation

range() function used to generate a sequence of numbers. If takes following parameters.

- **start** : starting number of the sequence
- **stop** : Generate numbers up to , but not including this number
- **step** : Difference between each number and default value is 1

Example : Generate sequence number from 1 to 5

```
x = range(1, 5, 1)      #step is 1
Y = range(1, 100, 5)    #step is 5
```


Control Flow Statements

To display value of sequence of numbers by iterating using Control flow statements.

- For Loop
- While Loop

Control Flow Statements

Display sequence of numbers using For loop

```
x = range(1, 5, 1)
for num in x:
    print(num)
```

Output:

```
1
2
3
4
5
```

Control Flow Statements

Display sequence of numbers using While loop

```
i = 1   # Initialize variable i with 1
while i < 5:  # Check the condition
   print(i)
   i = i + 1   # increments
```

Output:
1
2
3
4
5

Functions

- Functions are the most important part of a programming language.
- Functions can be created using *def* keyword.
- The function signature should contain the function name followed by the input parameters enclosed in brackets and must be end with colon (:)
- The functions ends with a return statements. If no return statement implies the function returns Nothing (None).

Functions

Syntax:

```
def function_name(parameters1, parameters2…):
        function-statements
        function-statements
        …
        return statement
```


Functions

Example:

Write a functions to add of two integer numbers:

```
def addElement(a,b):
    y = a + b
    return y

addElement(5,10)
# 15
```

Data Structures/ Collections

- Data Structures/Collections are useful containers to store and manipulate list of homogeneous or heterogeneous elements.
- Following data structures in this sections:
  - List
  - Tuple
  - Set
  - Dictionary

List - Data Structures

- List allows to contain heterogeneous items, that is, a single list can contain items of type int, float, string or object.
- List can be created with square bracket [ ]
- List values can repeat, it allows to store repeated values.
- Lists are mutable and generally initialized with a list of values specified inside square brackets or an empty list.

List - Data Structures

- Creation syntax:

variable_name = [element-1, element-2, element-3..etc]

Example:

- Create list of integer number

```
num_list = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
print(num_list)
```


List - Data Structures

- Creation syntax:

variable_name = [element-1, element-2, element-3..etc]

Example:

- Create list of integer number

```
num_list = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
print(num_list)
```

- Create list of city name

```
city_name = ["Tumkur", "Bangalore", "Mysuru",
             "Mandya", "Davangere", "Shivamogga"]
print(city_name)
```

List - Data Structures

Accessing Elements from a List

- List elements can be access using indexing range separated by colon (:)
- Index range allows from 0 to n-1 element.

Syntax

list-variable-name[index]

or

list-variable-name[start:end] # with range of index

List - Data Structures

num_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(num_list)

Accessing first element
num_list[0]

Accessing list using index range
num_list[3:7] #Index 7 will not be included (4,5,6,7)

Accessing last element
num_list[-1] #Negative index display element from last

Accessing last 3 element from a list
num_list[-3:]

List - Data Structures

Calculate list size (how many elements are present in list)
len(num_list)

List allow to combine another list easily with + (plus) operator
num = [1, 2, 3, 4, 5]
alpha = ['a','b','c','d','e']

combined_list = num + alpha
print(combined_list)

Output:
[1, 2, 3, 4, 5, 'a', 'b' ,'c' ,'d', 'e']

List - Data Structures

List allows to change/update the list elements

num_list = [1,2,3,4,5,6,7,8,9,10]

num_list[4] = 404 #updating 4th index element with new value

num_list[-2] = 99990 #updating 2nd last element of list

print(num_list)

Output:
[1, 2, 3, 4, 404, 6, 7, 8, 99990, 10]

Tuple - Data Structures

- Tuple is also a list, but it is immutable. Once a tuple has been created it cannot be modified.
- Tuple can be created with parenthesis ()
- Tuple can have repeat value, and it allows to store repeated values also.
- Tuple are immutable and generally initialized with a list of values specified inside parenthesis ().

Tuple - Data Structures

- Creation syntax:

variable_name = (element-1, element-2, element-3..etc)

Example:

- Create Tuple of integer number

num_list = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(num_list)

Tuple - Data Structures

- Creation syntax:

variable_name = (element-1, element-2, element-3..etc)

Example:

- Create Tuple of integer number

num_list = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(num_list)

- Create Tuple of city name

city_name = ("Tumkur", "Bangalore", "Mysuru",
"Mandya", "Davangere", "Shivamogga")
print(city_name)

Tuple - Data Structures

Accessing Elements from a Tuple

- Tuple elements can be access using indexing range separated by colon (:)
- Index range allows from 0 to n-1 element.

Syntax

tuple-variable-name[index]

or

tuple-variable-name[start:end] # with range of index

Tuple - Data Structures

num_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(num_tuple)

Accessing first element
num_tuple[0]

Accessing tuple using index range
num_tuple[3:7] #Index 7 will not be included (4,5,6,7)

Accessing last element
num_tuple[-1] #Negative index display element from last

Accessing last 3 element from a tuple
num_tuple[-3:]

Tuple - Data Structures

Calculate list size (how many elements are present in list)
len(num_tuple)

Tuple allow to combine another Tuple easily with + (plus) operator
num_tuple = (1, 2, 3, 4, 5)
alpha_tuple = ('a','b','c','d','e')

combined_tuple = num_tuple + alpha_tuple
print(combined_tuple)

Output:
[1, 2, 3, 4, 5, 'a', 'b' ,'c' ,'d', 'e']

Tuple - Data Structures

#Tuple is immutable, means once it created, we can not change any elements.

num_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(num_tuple)

num_tuple[4] = 404 #This statement will throw error message, bcz tuple is immutable

```
num_tuple[4] = 404
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_15804/3831651665.py in <module>
----> 1 num_tuple[4] = 404

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


Set - Data Structures

- A set is a collection of unique elements, that is the value con not repeat.
- Set can be created with curly brackets {}
- The set automatically removes duplicates and contains only unique list of numbers.
- Syntax:
  variable_name = {element-1, element-2, element-3..etc }

Set - Data Structures

Example:

```
s = {1, 2, 3, 4, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9,10}
print(s)
```

Iterate using For loop.

```
for ele in s:
    print(ele)
```


Set - Data Structures

#Set Operation:

```
s1= {1,2,3,4,5,10,20,30,40}
s2= {4,5,6,7,8,9,10}
```

Union

```
s3 = s1.union(s2)
print(s3)
```


Set - Data Structures

#Set Operation:

```
s1= {1,2,3,4,5,10,20,30,40}
s2= {4,5,6,7,8,9,10}
```

Union

```
s3 = s1.union(s2)
print(s3)
```

Intersection

```
s4=s1.intersection(s2)
print(s4)
```

#difference Set (A-B)

```
s5 =s1.difference(s2)
print(s5)
```


Dictionary - Data Structures

- Dictionary is a list of key and value pairs. All keys in a dictionary are unique.
- Dictionary can be created with curly brackets & key : value
  {"Key": "Value"}
- The value of dictionary can be accessed by using key.
- Syntax:

```
dict_var_name = {
        "key1": "value",
        "key2": "value",
        ...
        }
```

Dictionary - Data Structures

Example:

```
student = {
            "USN":  "1SI22MBA01",
          "NAME":  "John",
           "CITY":  "Tumkur",
          "DEPT":  "MBA",
   "COLLEGE": "SIT, Tumkur"
   }

print(student)
```


Dictionary - Data Structures

```
# Display all the Keys available in Dictionary:
student.keys()

# Display all the value available in Dictionary:
student.values()

# Access element using Key
student['USN']
student['NAME']

# Update value in Dictionary
student['USN']="1SI22MBA02" # Print dictionary after an update
```


Dictionary - Data Structures

```
# Combine two dictionary

sub = {
         'Sub-1': 45,
        'Sub-2': 55,
        'Sub-3': 65
}

student.update(sub)
print(student)
```


Comparison with Python Data structure

Lists	Tuple	Set	Dictionary
Lists are mutable	Tuples are immutable.	Lists are mutable	Lists are mutable
Lists are enclosed within square braces. [ ]	Tuples are enclosed within parenthesis. ( )	Sets are enclosed in curly brackets. { }	Dictionaries are enclosed in curly brackets with key-value pairs. { key : value }
List element can be accessed using index/ range of index	Tuple element can be accessed using index/range of index	Have use iteration like for, while loop to access element	Dictionary element can be accessible using its key
len() function used to get length/size of list	len() function used to get length/size of Tuple	len() function used to get length/size of set	len() function used to get length/size of dictionary
Easy to combine two or more lists with + (plus) operator	Easy to combine two or more Tuple with + (plus) operator	union to be used for combining two set	update function used to combine two dictionary

Exercise

Create a List with given element [10,20,30,40,50,60,70,80,90,100]

Write a python code for :

1. Create a List
2. Print element using print()
3. Print element using iteration (For loop)
4. Multiply list elements with number 2
5. Display first element of list
6. Display last element of list
7. Display first 3 elements of list
8. Display last 3 elements of list

Exercise

Create a dictionary for Employee data

employee_Name : John

employee_City : Bangalore

employee_Mobile: 9876512345

employee_Email : john@gmail.com

Write a python code for :

1. Create a employee dictionary
2. Display all the key present in dictionary
3. Display all the value present in dictionary
4. Print Dictionary element
5. Access dictionary element using employee_Name
6. Access dictionary element using employee_Email