# Chapter 1

- Mrs. Apurwa A. Barve

## Why Python?

- Very famous and in demand programming language :
  - Easy to learn
  - Easy to set up
  - Simple syntax
  - Interpreted language which means that the code is executed as soon as it is typed.
  - Can be considered as procedural, object oriented and even functional programming language.
  - Large ecosystem - support of python developers in the form of libraries and features
  - Flexible when it comes to syntax, data types, etc so molds itself as per the requirement
  - Current version - 3

# Scope of Python

- Multi functional :
  - Machine learning - TensorFlow, Keras, PyTorch
  - Data science - NumPy, Pandas, Matplotlib, SciPy, etc
  - Web development - Django, Flask, Tornado, etc
  - Automation - Libraries to automate DevOps tasks, scripts for automated backups, clean ups, automating tasks while working on excel sheets or other tasks, etc
  - Artificial Intelligence - Face recognition, Voice recognition
  - Web scraping

## Set up & Configuration

- Python setup - https://www.python.org/downloads/
- PyCharm setup - https://www.jetbrains.com/pycharm/ (go for professional setup)
- Refer PyCharmGuide.docx to understand how to start using the IDE.
- How to use commandline for executing python - https://www.geeksforgeeks.org/how-to-use-cmd-for-python-in-windows-10/

## Activity 1.1

1. Open your IDE and create a new project. Call it chapter1Prj
2. Add a new python file in this project.What is the extension of the python file which you created above?
3. Copy the following lines in the file and see the output :

```
import sys

print(sys.version)
```

What is **sys?**

1. Based on the inputs given in the previous slide, execute the above lines of code on the command line.

## Indentation, white spaces

- Indentation - space/tab at the beginning of the line of code
- White spaces - Used to deliver indentation
- Both the above play an important role in defining scopes of loops, functions and classes in Python. In other programming languages, this is done using curly braces {}

## Activity 1.2

1. Copy the following code and check the output

```
if 8>9:
   print("This will never be printed")
if 8<9 :
print("This too will never be printed")
```

What is the error in the above code? What do you observe?

2. Correct the above lines of code and observe the output

3. Copy the following code and observe the output

```
if 8<9 :
      print("Will this be printed?")
```

Do more than one tabs work in Indentation?

## Comments

- ❖ Comments are used to add explanation about the code
- ❖ They make a code more readable
- ❖ They explain the underlying functionality and additional details about the business logic
- ❖ It could be marking those lines of code which you do not want to be executed
- ❖ Consider following examples:

1.
```
#This is a comment
print("Python programming comment demo")
```

2.
```
print("Python programming comment demo") #This too is a comment
```

3.
```
#This is a comment
 #on multiple lines
 #This is a comment
print("Python programming multi line comment demo")
```

## Activity 1.3

Consider the below code. Copy it and execute it. Discuss the output.

```
"""

This is a comment sample

how does it work??

"""

print("Only this line is executed")
```

Why does the above code work? Is the multiline string a style of commenting in Python?

## Variables

- ❖ These are the containers that hold values
- ❖ A variable is created the moment you first assign a value to it.
- ❖ They have some datatype based on the value which they are holding.
- ❖ In Python the variables are defined using following syntax :

  variable_name = variableValue

  Observe that there is no datatype defined at the time of defining the variable. This means that the variables are **DYNAMICALLY TYPED and NOT STATIC TYPED** as is the case with languages such as Java, C, C++,etc
- ❖ If the variable name has multiple words then the convention says to separate those words using _ (underscore)

  student_roll_number

## Variables - Naming rules

| **Allowed in a name** | **Not allowed in a name** |
|---|---|
| Begin with a letter or underscore | Can not begin with a number |
| Can only contain alpha-numeric characters and underscores | Can not have reserved words |
| Names are case sensitive | |

# Variables - Local v/s Global

| Local Variables | Global Variables |
|---|---|
| Declared within a function. Scope is limited to the function | Declared outside the function but used/ modified within a function. Variables that are only referenced inside a function are implicitly global. |
| #understanding the scope of the variables in Python<br><br>#we are defining addition() using def keyword<br>def addition():<br>   num1,num2=30.50,40.25<br>   sumAdd = num1+num2<br>   print("Num1 is ", num1)<br>   print("Num2 is", num2)<br>   print("Addition result is:",sumAdd)<br>#we are invoking addition()<br>addition()<br>print("Num1 is ",num1) #what will be the output of this line?<br>print("Num2 is", num2) #what will be the output of this line?<br>print("Addition result is:",sumAdd) #what will be the output of this line? | num1, num2 = 30.50, 40.25<br>def addition():<br>   print("Inside addition")<br>   global num1,num2<br>   print("Num1 is ", num1)<br>   print("Num2 is", num2)<br>   num1, num2 = 50.75, 120.25<br>   sumAdd = num1+num2<br>   print("Addition result is:",sumAdd)<br>#we are invoking addition()<br>addition()<br>print("Num1 is ",num1) #what will be the output of this line?<br>print("Num2 is", num2) #what will be the output of this line?<br>print("Addition result is:",sumAdd) #what will be the output of this line? |

## Activity 1.4

1. Copy the code given below and observe the output

```
# different ways of declaring variables
x=10
st1='Allowed'  #string value in single quotes
_st2="allowed"  #string value in double quotes
ST1 = "Case sensitive" #ST1 is different from st1
print("Value of x is",x)
print("Value of st1 is",st1)
print("Value of ST1 is",ST1)
print("Value of st2 is", _st2)
x="reassignment"
print("Reassigned value of x is",x)
```

## Activity 1.4

2. Copy the code given below and add the missing lines

```
val_int = int(x) #this will hold value in int format
val_float = float(x) #this will hold value in float format
val_str = str(x) #this will hold value in string format

#complete the following code to print the values of
# val_int
#val_float
#val_str
```

The above example is of **casting.** Using the syntax given above we can determine the way the values are stored in the given variable.

# Activity 1.4

3. Copy the code snippet given below and execute it. Observe the output and discuss the difference in print() and print(f)

```
 #multiple variables assigned different values in a single line
v1,v2,v3 = 10.00,20.50,12.75
sum = v1+v2+v3
#what will this print?
print("Sum of the variables is ",sum)
#what will the following line print?
print(f"Sum of {v1,v2,v3} is {sum}")
```

4. What is the use of type() ? Explain with a code example

## Activity 1.4

5. Copy the below given code snippet and observe the output. Check which all lines will execute and which wont. Discuss why the lines which do not get executed are causing problem.

```
num1 = 20
num2 = 30
st1 = 'Hello'
st2 = 'World'
#guess the output of the following
print(num1)
print(num1)
print(num1+num2)
print(st1)
print(st2)
print(st1+st2)
print(st1,st2)
print(num1,st1)
print(num1+st1) #will this line execute?
```

## Data types

| **Name of the data type** | **Type of data stored** | **Syntax** |
|---|---|---|
| str (These are immutable) | Text | str = “Hello world” (or) str = ‘Hello World’ (or) str = ‘’’ Hello world’’’ |
| int (Unlimited length) | Numeric | num = 2 |
| float | | num = 2.0 |
| complex ( Complex numbers are written with a "j" as the imaginary part) | | num = 3j |
| True / False (Any integer, floating-point number, or complex number having zero as a value is considered as False, while if they are having value as any positive or negative number then it is considered as True.) | Boolean | st_task_done = True (or) st_task_done = False |

## Activity 1.5

1. You are making a housie game app to generate random numbers. Write a Python program to help you generate these numbers 6 times for a given housie ticket.

## Operators

This topic will be covered as a part of Students’ Presentation

## Object Types - List

- There are four collection data types in the Python programming language: List, Tuple, Set, Dictionary.
- Syntax for List creation:
    1. nameOfTheListVariable = [ ]   — This will create a blank list
    2. nameOfTheListVariable = [element1,element2,... ]  — List containing n number of elements
    3. nameOfTheListVariable = list((“apple”,20,True,”banana”)) – All of these are allowed. We are using list() constructor here.
- Items in the list are ordered i.e they will retain the sequence in which they are inserted into the list.
- They are changeable and duplicates are allowed
- Individual item in the list is identified using the index number of that element. Index numbers begin with 0
- They are similar to dynamically sized arrays.
- They can have heterogeneous elements.

## Activity 1.6

1. Complete the code given below:

```
#add missing months' names below to complete the list
monthNames = list(("Jan","Feb","Mar",,,,,,,"Oct","Nov","Dec"))

#take a number from the user between 1 to 12.
# complete the code statement here  input("Enter any number between 1 to 12")
#return the name of the month corresponding to that number.
#eg : if the number is 4 code should return Apr
print("The month corresponding to the given number is:",list[variableNumber])
```

1. Execute the following code snippets and observe the output
   a. 
   ```
   daysOfWeek= ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
   print(daysOfWeek[:3])
   ```

## Activity 1.6

b. daysOfWeek= ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]

```
print(daysOfWeek[3:])
```

c. rainfallData = [[1,"Arunachal Pradesh"],[2,"Sikkim"],[3,"West Bengal"], [4,"Odisha"], [5,"Uttarakhand"]]

```
print(rainfallData[3][0],rainfallData[3][1])
```

## Object Types - Dictionaries

- There are four collection data types in the Python programming language: List, Tuple, Set, Dictionary.
- Syntax for Dictionary creation:

```
varName = {key1:value1,key2:value2,...,keyN:valueN} (or)

varName = dict(key1=value1,key2=value2,...,keyN=valueN) #this is using constructor
```

- Dictionary is used to store the data in key:value pair format.
- Since Python 3.7 dictionary data type became **ordered**.
- Elements in the dictionary are changeable.
- Duplicates are **not** allowed.
- The elements can be heterogeneous

# Activity 1.7

1. Execute the code given below and discuss the output

   a.
   ```
   myDictionary = {1:"Apple", 2:"Cherry",3:"Banana",4:"Peach"}
   print(myDictionary)
   ```

   b.
   ```
   print(len(myDictionary))
   ```

   c.
   ```
   studentRecord = {'name':'ABC',
                    'age':20,
                    'contactNumbers':[123456,987653]
                   }
   print(studentRecord)
   print(studentRecord['name'])
   print(studentRecord['contactNumbers'])
   print(type(studentRecord))
   ```

## Activity 1.7

2. Execute the following code snippet and observe the results:

```
studentRecord = {'name':'ABC',
      'age':20,
      'contactNumbers':[123456,987653]
      }
oneMoreStudentRecord = studentRecord
print('The new dictionary looks like:',oneMoreStudentRecord)
```

## Activity 1.7

3. Add the following lines to the above code snippet. Execute the modified code and observe the results:

```
studentRecord = {'name':'ABC',
      'age':20,
      'contactNumbers':[123456,987653]
      }
oneMoreStudentRecord = studentRecord
print('The new dictionary looks like:',oneMoreStudentRecord)
studentRecord.update({'address':'Mumbai'})
print('Modified studentRecord dictionary looks like:',studentRecord)
print('The new dictionary looks like:',oneMoreStudentRecord)
```

# Object Types - Tuples

- There are four collection data types in the Python programming language: List, Tuple, Set, Dictionary.
- Syntax for Tuple creation:

```
myTuple=(element1,element2,..,elementN) (OR)

myTuple=tuple((element1, element2,...,elementN)) #this is using constructor (or)

myTuple = (element1,) #tuple having ONLY one element. Ends with ,
```

- Elements in a tuple are ordered and NOT changeable.
- Tuples are used to store IMMUTABLE objects.
- Duplicates are not allowed.
- The elements can be heterogeneous
- Elements are accessed using their index values. Index starts from 0.

# Activity 1.8

Execute the below given code and observe the output

```
firstTuple=() # this is an empty tuple
print('First tuple',firstTuple)
print(type(firstTuple))
secondTuple=("mon","tue","wed","thu")
print('Second tuple',secondTuple)
print(type(secondTuple))
thirdTuple=(1,2,3)
print('Third tuple',thirdTuple)
print(type(thirdTuple))
fourthTuple=tuple(("fri","sat","sun"))
print('Fourth tuple',fourthTuple)
print(type(fourthTuple))
fifthTuple=(secondTuple,fourthTuple) #NESTEDtuple
print('Fifth tuple',fifthTuple)
print(type(fifthTuple))
myList=[10,20,30]
sixthTuple=tuple(myList)
print('Sixth tuple',sixthTuple)
seventhTuple=(40,)
print('Seventh tuple',seventhTuple)
print(type(seventhTuple))
eightTuple=(90)
print('Eighth tuple',eightTuple)
print(type(eightTuple))
ninthTuple=tuple(range(1,11))
print('Ninth tuple',ninthTuple)
```

## Object Types - Unpacking Tuples

- A was used to assign or extract the tuple values on the right to the variables on the left is called UNPACKING of a tuple.
- The method of assigning the values to a tuple is called Packing a tuple.
- If the number of variables on the left do not match the number of values in the tuple then we can use ASTERISK with any one of the variable on the left. This special variable will hold all the remaining values within the tuple.
- Packing and unpacking of tuples - Refer Packing_UnpackingTuplesEx.py

## Object Types - Set

- Will be covered as a part of Video presentation assignment

## Python Blocks

- A group of code statements which are indented together forms a BLOCK in python. It is executed as a single unit.
- Block defines the scope of the variable and control the flow of execution of the program.
- Examples of blocks:
  - Functions
  - Conditional statements
  - Class

## Python Flow Control - if,elif,else

❖ The if, elif, else conditions are used in isolation or combination to compare operands and determine the flow of execution of the program.

❖ Ternary operators can also be used to with if,elif and else conditions to reduce the length of code and get the same result based on evaluation of conditions.

# Activity 1.9

1. Execute the below given code and observe the output

Using if,elif,else

```
age = int(input('Enter your age'))
if(age < 18):
    print('You are not an adult yet')
elif(age == 18):
    print('Just 18!')
else:
    print('Legally adult')
```

# Activity 1.9

2. Execute the below given code and observe the output

Using ternary operators

```
age = int(input('Enter your age'))
print('You are not an adult yet' if age<18 else 'Just 18!' if age == 18  else 'Legally
adult')
```

3. Execute the following code and observe the output. Understand the use of **pass statement**

```
age=input('Enter your age')
 if age <= 18:
        pass
else:
        print('This is else')
```

# Python Flow Control - switch case

- Compare one value with multiple options. Only ONE matching option is executed. If nothing matches, there is a provision to give default option.
- Syntax :

```
match valueToBeMatched:
    case option1:
        statement(s) for case1
    case option2:
        statement(s) for case2
    case _:
        #this is the default case using an underscore to match the option
```

## Activity 1.10

Execute the below given code and observe the output

```
day = input('Enter a day of the week')
match day:
    case 'Mon'|'mon'|'Monday'|'monday':
        print('Monday')
    case 'Tue' | 'tue' | 'Tuesday' | 'tuesday':
        print('Tuesday')
    case 'Wed' | 'wed' | 'Wednesday' | 'wednesday':
        print('Wednesday')
    case 'Thu' | 'thu' | 'Thursday' | 'thursday':
        print('Thursday')
    case #complete the code for friday
        #complete the code
    case #complete the code for saturday
        #complete the code
    case 'Sun' | 'sun' | 'Sunday' | 'sunday':
        print('Sunday')
    case _:
        #add a print statement here
```

## Loops

- while loop - Executes the given set of statements for as long as the condition is TRUE.
- for loop - To execute the given set of statements ‘n’ number of times / iterations. It iterates over list, tuple, dictionary, set and string.

## Activity 1.11

1. Execute the below given code and observe the output. Understand the use of **break and continue** statements in flow control

```
while (True):
    num = int(input("Enter a number: "))
    if(num % 2 == 0):
        print('Even number')
        continue
    else:
        print('Odd number')
        break
```

## Activity 1.11

2. Using for loop write a program in Python to print all the prime till the nth number. Take the nth number from the user.

```
val = int(#missingKeyword('Enter the number till which you want to find prime numbers'))
primeSt = False

for i in #missingKeyword(2,#missingvariable):
    if val % i == #missingvalue:
        #missingKeyword #terminate the loop here
else:
    primeSt = True
if(primeSt):
    print(f'{val} is a prime number')
else:
    print(f'{val} is not a prime number')
```

## Activity 1.11

3.Execute the code snippets given below and observe the output

a.
```
message = 'Hello world'
 for i in message:
        print(i)
```

b.
```
myList = ['ABC', 1, 2, 'PQR', True]
 for i in myList:
        print(i)
```

c.
```
for i in range(1,15,2):
        print(i)
```

d.
```
myTelephoneDiary = {"ABC":1234,"PQR":5678,"XYZ":2345,"QWE":7834}
 for ky in myTelephoneDiary:
      print(f'The value for {ky} is {myTelephoneDiary[ky]}')
```

# Activity 1.11

4. Execute the code snippet given below and understand the use of exit()

```
import sys
#missingKeyword True:
    status = input('Enter Y to continue or N to exit')
    if status == #missingValue:
        num = int(input('Enter a number to determine if it is odd or even'))
        if #missingCondition == 0:
            print(num,'is even number.')
        else:
            print(num,'is odd number.')
    elif status == 'N':
        print('Thank you! See you again')
        sys.exit()
```

# File Handling basics

- File handling allows us to perform a wide range of operations such as creating,opening, reading, writing on to file.
- Python allows a file to be operated in text or binary format.
- When any file has to be handled it has to be loaded into the primary memory first.
- Each file is given a FILE HANDLER using which it can be operated upon.
- Syntax for **opening** a file:

  ```
  fileObject = open(r”filename”,”accessmode”)
  ```

  The letter r before the filename prevents the characters in the file name to be treated as special characters by Python interpreter. If the file is in the same directory as the file handling program then the r can be omitted.

- Syntax for **closing** a file:

  ```
  fileObject.close()
  ```

  This command closes the file and frees the memory from the primary memory space which was occupied by the said file.

## File Handling Modes

- ❖ File handling allows us to perform a wide range of operations such as creating,opening, reading, writing on to file.
- ❖ Python allows a file to be operated in text or binary format.

| File mode | Description |
|---|---|
| **r** | Read operation |
| **r+** | Read and write to a file without overwriting existing data. |
| **w** | Write mode. Create the file if it does not exist. Overwrite the file if it exists. |
| **w+** | Write and read to a file. Overwrites existing file. Reduces the file size if there is no data in the file. Creates a new file if previous does not exist. |
| **a** | Append mode. Write to an existing file without overwriting |
| **a+** | Append and read to an existing file without overwriting it. |

# File Operations - reading file

❖ To perform any operation on a given file, one first has to open it using the open().

| Read methods | Description |
|---|---|
| **read()** | Read the entire file in one operation. **Refer fileReadEx.py** |
| **readline()** | Read the file line by line. Useful in conserving memory especially in case of large files. **Refer fileReadLineEx.py** |
| **readlines()** | Read the file entirely. Content is stored as a list. |

# File Operations - writing to a file

- To perform any operation on a given file, one first has to open it using the open() and set the mode to w or a .

| **Write methods** | **Description** |
|---|---|
| **write()** | Writes the content in the file without adding any extra characters not even new line, \n. **Refer writeFileEx.py** |
| **wrtielines()** | Writes list of strings to a file in one go. Each string is separated using \n. **Refer writeLinesEx.py** |

Thank you