Roadmap

- About Python
- Program structure
- Elementary concepts like identifier and statment
- Types of statments
- Decision making and loop
- Simple data type
- Built-in and user-defined functions
- String and list

About PYTHON

- Developed by Guido van Rossum in the late 80s and early 90s at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Licensed as open source under GNU General Public License (GPL).
- Latest version is 3.9.0 (released in October 2020)
- To know python version installed on the system
  - Use Python statement sys.version
  - Use command python3 –version or python3 -V at command prompt
- Python is a case sensitive programming language
  - Fanpower and fanpower are two different identifiers in Python.

Features of Python

- Object-oriented High-level language
- Interpreted and Interactive
- Scripting language
- Portable
- Extendable
- Case-sensitive

Popular and Free IDE for PYTHON

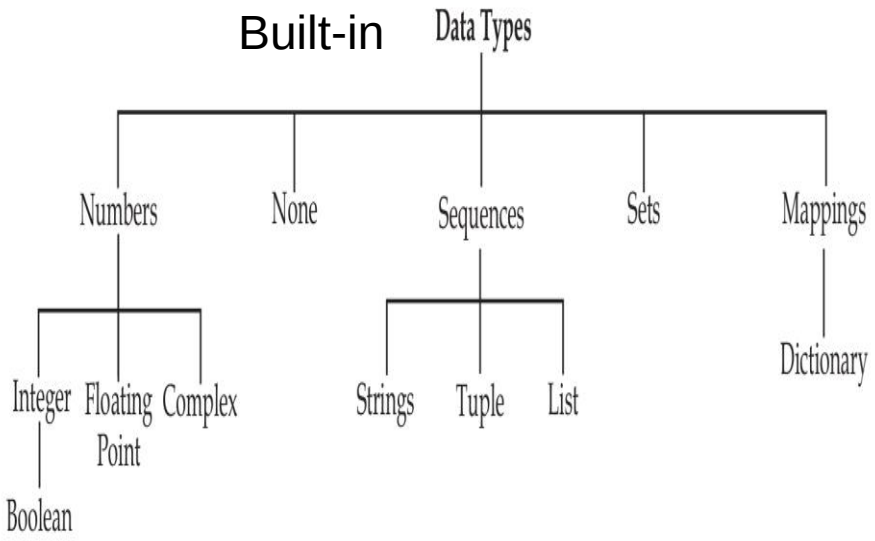
- PyCharm
- IDLE
- Sublime Text 3
- Jupyter Notebook
- Atom
- Google Colab
- ... and many more

Two Modes of Execution

- Interactive Mode
  - Executes one command/statement written after prompt >>>
  - Example:
  - >>> print("Good Morning")
- Script Mode
  - All statements of a program are written in a file (say example.py)
  - Each statement is executed by the interpreter sequentially
  - Used when large code is to be executed
  - Execute code at terminal by writing python3 filename

Data types in Python

- Everything is an object in Python
- Data types are actually classes and variables are instance (object) of these classes.
- Each object has identity that is retrieved using id(obj)
- Each object has type. Use type(obj) to know data type
- Each object has associated value that is assigned using operator =
- Each object is either mutable or immutable
- No need to explicitly declare a variable (object) of a type
- An object of a type is created as per the value assigned



Examples on datatype

Type (Mutable or Immutable)	Meaning	Further classification	Declaration
NUMBER (Immutable)	stores single numeric value	Integer Float Complex	x=int(5) or x=4 y=float(2.3) or y=2.3 z=complex(2,5) or z=2+5j
NONE (NA)	stores single value empty value	NA	a=None
SEQUENCE (Both)	an ordered collection of items, each indexed by positive integers	string (Immutable) list (Mutable) tuple (Immutable)	x="Hello" x=str("Hello") L=[1,2,'x'] L=list([1,2,'x']) T=(2,'x','y')
SETS (Immutable)	unordered collection of values of any type with no duplicate entry	NA	xset = {1.0, "Hello", (1, 2, 3)} yset=set([1,"hello"])
MAPPING (Mutable)	unordered key-value pair	dictionary (key-value) pair	xdict= {1: 'apple', 2: 'ball'}

Program structure

- Lines and identations are used to build program
- Lines at same level of indentation indicate one block
- Multiple nested blocks are at same indentation level

```
if True:
    print "True"
else:
    print "False"
```

Program structure contd..

- Each statement in program is end by EOLN (enter)
- In case line is too long then it is continued using back slash character

```
total = item_one + \
        item_two + \
        item_three
```

Comments

- Statement just for documentation purpose, not to be interpreted
- Use of # for single line comment
- Use of three single quote in the beginning and at the end of mulitle lines comments

Single-line comment

- # This is a comment, too.
- # This is a comment, too.

Multi-lines comment

```
'''not to be executed
... print not to display
... just for documentation'''
```

Logical expressions using logical operators and relational operators

- Check grade is greater than ‘A’ and less than ‘C’
  - (grade>’A’) and (grade <’C’)
- City is ‘Delhi’ and population more than 10000
  - City ==’Delhi’ and population >10000
- Half of salary is less than equal to 3000
  - Salary/2 <= (3000//3)

Types of Statement

- Assignment statement
  - To assign a value to any identifier/variable
  - Angle=90; name='Thapar'
- Input statement
  - To take data from the user
  - name=input("Enter your name")
  - Everthing entered is of string type
- Output statement
  - To display the result : print("name entered is ", name)

Decision Making Statement

- Selection statement to choose actions with alternative courses
- IF-ELSE
- IF-ELSIF

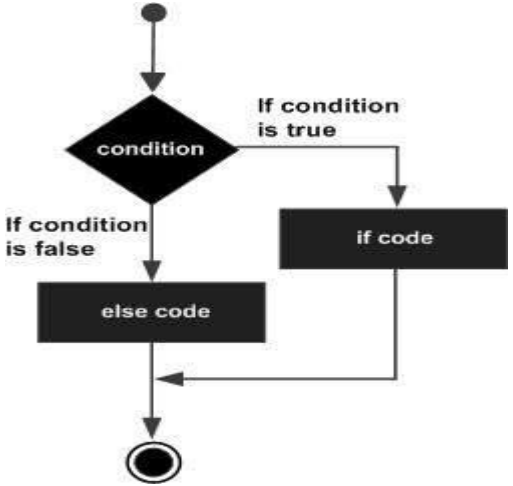
IF else usage

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if..else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
nested if statements	if or else if statement inside another if or else if statement(s).

IF-ELSE

Syntax:

```
if (condition):
statements
   else:
statements
```



Example 1:

```
var1 = 100
if (var1 >= 100):
   print ("1 - Got a true expression value")
   print (var1)
else:
   print(“False value”)
```

indentation

Nested if statement

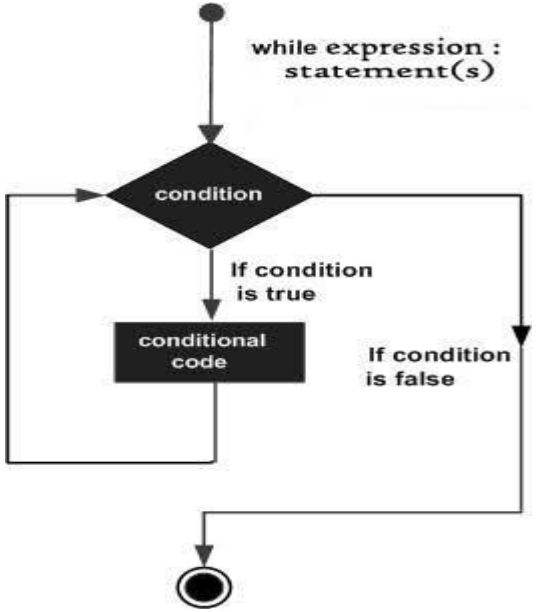
```
age= 30
if age < 18:
  print("Minor")
elif (age>=18) and (age <=60):
    print("adult")
 else:
    print("Senior person")
```

While loop

.For repeatedly executing target statements as long as a given condition is true

.Syntax is

```
while expression:
        statement(s)
```



For loop

To repeat a set of statements fixed number of times

for <variable> in <sequence>:

<statements>

else:

<statements>

```
import math
"""use of function for repeating same task but with different values"""
#define function
def fntot(n):
  total=0
  for i in range(n,0,-1):
     print(i,math.pow(i,2))
     total += math.pow(i,2)
  else: print("i= ",i)
  return total
#calling function
sum=fntot(3)
print("total= ",int(sum))
```

Else with loop

- If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.

- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

Continue statement

.When a continue statement is encountered, the control skips the execution of remaining statements inside the body of loop for the current iteration and jumps to the beginning of the loop for next iteration.

```
num = 0
for num in range(6):
        num = num + 1
        if num == 3:
                continue
print('Num has value ' + str(num))
print('End of loop')
```

Function

- A block of organized, reusable code that is used to perform a single, related action
- Provides better modularity for bigger application and a high degree of code reusing.
- Python gives many built-in functions like print(), etc. but we can also create our own functions called user-defined function using def() statement

User-defined function

- To divide a bigger problem into sub-problems and for modular programming
- Using a function involves two steps:
  - Define a function as

    def fnname():
  - Run the function
  - Two ways:

    I. Run menu-> Run module and then call fnname()

    II. In script itself include (as last statements) if execution is to start from function with name fnname

```
if __name__=='__main__':
    fnname()
```

Slicing

- Retrieving a substring

```
message=‘Good Morning’
message[2:6]
message[-12:-6]
message[:6]
message[:15]
message[0:len(message):2]
message[0:]
message[6:-2]
Message[-6:-2]
```

```
'od M'
'Good M'
'Good M'
'Good Morning'  #even if out of bound
'Go onn'
'Good Morning'
'orni’
‘orni’
```

Membership

- Membership of individual character using IN operator
- ‘m’ in ‘Good morning’ return True
- ‘M’ in ‘Good morning’ return False
- Count letter ‘o’ in given string

```
count=0
for ch in ‘Good Morning’:
        If ch==’o’:
                Count +=1
print(count)
```

Equivalent string function:
‘Good morning’.count(‘o’) -> 3
‘mood morning’.count(‘mo’) -> 2

Strings are immuatable

- Cannot be changed after they are created
- Since strings can't be changed, a new string is constructed to represent computed values.
- For example the expression ('hello' + 'to all') takes in the 2 strings 'hello' and 'to all' and builds a new string 'hellotoall'.
- Can not use ‘\0’ to check for end of string as all data types are implemented using same method
- Say string is my_string = 'programiz' then my_string[5] = 'a' is not permitted as string is immutable

Data structure

- Integer, number etc.. are simple data
- String is a sequence type and made of characters where each character is indexed
  - Immutable data object
- Others are list, set, tuples, dictionary

LIST

- **An ordered sequence of elements seperated by comma.**
- **Non-scalar data type**
- **Syntax : list1=[val1,val2,..valn]**
- **Example: subject=['maths','hindi','physics','english']**
- **Each list element is accessed using index as in case of string**
- **Elements of list may not be same, hence is called Heterogeneous list**
- **list1=[1,2,'abc','xyz']**
- **List is mutable i.e. list1[2]=13 is permitted**
- New contents of list1 is **list1=[1,2,13,'xyz']**

subject=['maths','hindi','physics','english']

- Each list var has an id and is retrieved using id(list1)
- Accessing element of a list and slicing
- seq=list1[start:end:increment]
- e.g.
- subject[3] -> 'english'
- subject[2::2] -> ['physics']
- subject[1::] -> ['hindi', 'physics', 'english']
- subject[::2] -> ['maths', 'physics']
- Aliasing object i.e. temp=subject

Find the output?

```
def fncount(names,l):
  countw=0
  for i in names:
    if len(i)==l:
      countw+=1
  return countw


W=fncount(names,l)
print(w)
```

```
names=['Hello','everyone','welcome']
l=6
```

TUPLE

- Tuples are sequences, just like lists, but is immutable
- Tuples use parentheses (), whereas lists use square brackets []
- an ordered collection and uses indexes for accessing elements as in list ; may be heterogeneous
- Indexing and slicing as in list and string
- (2,1,5,21) ,(2,1,'a',[3,4]) : examples of tuples

Defining a tuple object

- x=tuple() or x=()
- (2,1,’a’,[3,4]) an example of heterogeneous tuple
- tuples are immutable but could be list string etc.

Iterating over sets element

- Just like string and list
- Example:

```
weekdays=('mon','tues','wed','thurs','fri')
for a in weekdays:
  print (a, end='**')
```

OUTPUT:

```
mon**tues**wed**thurs**fri**   #ordered
```

FIND OUTPUT??

```
t=(2,1,'a',[3,4])
for a in t:
    print(type(a),a)
```

OUTPUT:

<class 'int'> 2

<class 'int'> 1

<class 'str'> a

<class 'list'> [3, 4]

FIND OUTPUT??

```
t=(2,1,'a',[3,4])
for a in t:
   if type(a) is list:
         print(type(a), len(a),a[0])
```

OUTPUT:

<class 'list'> 2 3