

Chapter 2: Fundamentals of Python

2.1 Command Line Argument and Flow control

Mathematical Functions (math Module)

A Module is collection of functions, variables and classes etc. A math is a module that contains several functions to perform mathematical operations.

If we want to use any module in Python, first we have to import that module.

```
import math
```

Once we import a module then we can call any function of that module:

```
import math
print(math.sqrt(16))
print(math.pi)
```

```
>>> 4.0
>>> 3.141592653589793
```

We can create alias name by using as keyword.

```
import math as m
```

Once we create alias name, by using that we can access functions and variables of that module:

```
import math as m
print(m.sqrt(16))
print(m.pi)
```

We can import a particular member of a module explicitly as follows

```
from math import sqrt
from math import sqrt, pi
```

If we import a member explicitly then it is not required to use module name while accessing.

```
from math import sqrt, pi
print(sqrt(16))
print(pi)
print(math.pi) → NameError: name 'math' is not defined.
```

Important functions of math module

- `ceil(x)`
- `floor(x)`
- `pow(x,y)`
- `factorial(x)`
- `trunc(x)`
- `gcd(x,y)`
- `sin(x)`
- `cos(x)`
- `tan(x)`

Important variables of math module

- `Pi` → 3.14
- `E` → 2.71
- `inf` → infinity
- `nan` → not a number

Input And Output Statements

Reading dynamic input from the keyboard

input():

`input()` function can be used to read data directly in our required format. We are not required to perform type casting.

`x=input("Enter Value")`

`type(x)`

`10` → `int`

`"durga"` → `str`

`10.5` → `float`

`True` → `bool`

Command Line Arguments

`Argv` is not Array it is a List. It is available `sys` Module.

The Argument which are passing at the time of execution are called Command Line Arguments.

Eg: `D:\Python_classes py test.py 10 20 30`

Within the Python Program this Command Line Arguments are available in `argv`, which is present in `SYS` Module.

test.py	10	20	30
---------	----	----	----

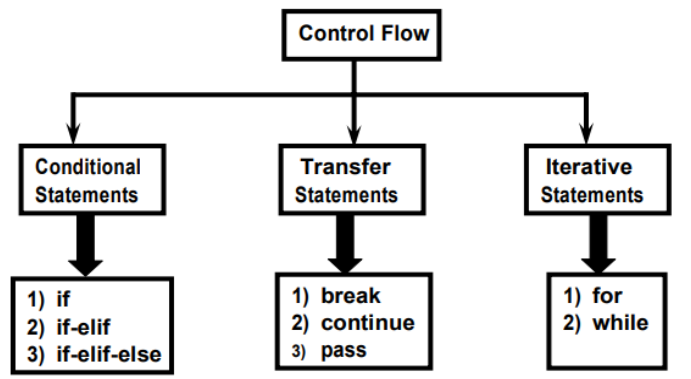
Note: argv[0] represents Name of Program. But not first Command Line Argument. argv[1] represent First Command Line Argument.

Output statements:

We can use print() function to display output.

Flow Control

Flow control describes the order in which statements will be executed at run time.



Conditional Statements

1) *if*:

if condition : statement (or)

if condition :

statement-1

statement-2

statement-3

If condition is true then statements will be executed.

2) *if-else*:

if condition :

Action-1

else :

Action-2

if condition is true then Action-1 will be executed otherwise Action-2 will be executed.

3) *if-elif-else*:

Syntax:

if condition1:

Action-1

```
elif condition2:  
    Action-2  
elif condition3:  
    Action-3  
elif condition4:  
    Action-4 ...  
else:  
    Default Action
```

Based on the condition, corresponding action will be executed.

Iterative Statements

Iterative statements are used to write a program without explicitly programmed.

If we want to execute a group of statements multiple times then we should go for Iterative statements.

Python supports 2 types of iterative statements:

1) *for loop:*

If we want to execute some action for every element present in some sequence (it may be string or collection) then we should go for **for loop**.

Syntax:

```
for x in sequence:  
    body
```

where sequence can be string or any collection.

Body will be executed for every element present in the sequence.

2) *while loop:*

If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

Syntax:

```
while condition:  
    body
```

3) *Nested Loops:*

Sometimes we can take a loop inside another loop, which are also known as nested loops.

Transfer Statements/Jump Statements

Jump statements are used to come out from the loop without Stopping loop

- **break:** We can use break statement inside loops to break loop execution based on some condition.
- **continue:** We can use continue statement to skip current iteration and continue next iteration.

- ***pass statement:*** pass is a keyword in Python.

Patterns

Patterns are different shapes.



2.2 Data Structures

A Data Structure is a special way of organizing and storing data in a computer so that it can be used efficiently.

In python we have four types of data structures.

- 1) List
- 2) Tuple
- 3) Set
- 4) dict

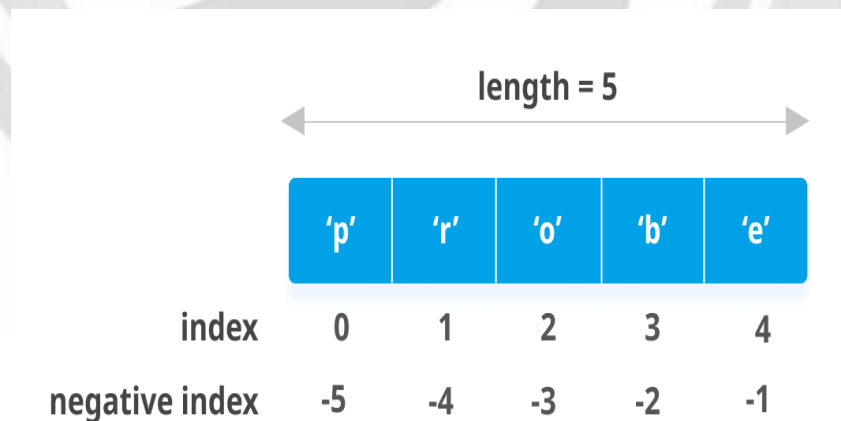
List

List is a collection which is ordered and changeable. Allows duplicate members. It's a mutable data type.

If we want to represent a group of individual objects as a single entity where insertion order is preserved and duplicates are allowed, then we should go for List.

In list, Insertion order preserved, duplicate objects are allowed, and heterogeneous objects are allowed. List is dynamic because based on our requirement we can increase the size and decrease the size.

In List the elements will be placed within square brackets and with the comma separator. We can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role. Python supports both positive and negative indexes. +ve index means from left to right where as negative index means right to left



List objects are mutable. i.e., we can change the content.

Note: Sometimes we can take list inside another list, such type of lists is called nested lists.

Ex: [10,20, [30,40]]

List Operations

To get information about list:

1. *len() function:*

Returns the number of elements present in the list

Eg: n = [10,20,30,40] print(len(n))

[Output] → 4

2. *count() function:*

It returns the number of occurrences of specified item in the list

n=[1,2,2,2,2,3,3]

print(n.count(1))

print(n.count(2))

print(n.count(3))

print(n.count(4))

[Output] → 1

[Output] → 4

[Output] → 2

[Output] → 0

3. *index() function:*

Returns the index of first occurrence of the specified item.

Eg: n=[1,2,2,2,2,3,3]

print(n.index(1)) → 0

print(n.index(2)) → 1

print(n.index(3)) → 5

print(n.index(4)) → ValueError: 4 is not in list

4. *append() function:*

We can use append() function to add item at the end of the list.

5. *insert() function:*

To insert item at specified index position.

6. *extend() function:*

To add all items of one list to another list.

7. *remove()* function:

We can use this function to remove specified item from the list. If the item present multiple times, then only first occurrence will be removed.

8. *pop()* function:

It removes and returns the last element of the list. This is only function which manipulates list and returns some element.

9. *reverse()*:

We can use to reverse() order of elements of list.

10. *sort()* function:

In list by default insertion order is preserved. If we want to sort the elements of list according to default natural sorting order then we should go for sort() method.

Tuple

- Tuple is exactly same as List except that it is immutable. i.e., once we create Tuple object, we cannot perform any changes in that object. Hence Tuple is read only version of List.
- If our data is fixed and never changes then we should go for Tuple.
- Insertion Order is preserved.
- Duplicates are allowed.
- Heterogeneous objects are allowed.
- We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.
- Tuple support both +ve and -ve index. +ve index means forward direction (from left to right) and -ve index means backward direction (from right to left).
- We can represent Tuple elements within Parenthesis and with comma separator. Parenthesis are optional but recommended to use.

Differences between List and Tuple:

List and Tuple are exactly same except small difference:

List objects are mutable whereas Tuple objects are immutable. In both cases insertion order is preserved, duplicate objects are allowed, heterogeneous objects are allowed, index and slicing are supported.

List	Tuple
List is a Group of Comma separated values within Square Brackets and Square Brackets are mandatory. Eg: <code>i = [10, 20, 30, 40]</code>	Tuple is a Group of Comma separated Values within Parenthesis and Parenthesis are optional. Eg: <code>t = (10, 20, 30, 40)</code> <code>t = 10, 20, 30, 40</code>
List Objects are Mutable i.e., once we create List Object, we can perform any changes in that Object. Eg: <code>i[1] = 70</code>	Tuple Objects are Immutable i.e., once we create Tuple Object, we cannot change its content. <code>t[1] = 70</code> → <code>ValueError: tuple object does not support item assignment.</code>
If the Content is not fixed and keep on changing then we should go for List.	If the content is fixed and never changes then we should go for Tuple.
List Objects cannot be used as Keys for dictionaries because Keys should be hash able and Immutable.	Tuple Objects can be used as Keys for dictionaries because Keys should be hash able and Immutable.

Set

- If we want to represent a group of unique values as a single entity then we should go for set.
- Duplicates are not allowed.
- Insertion order is not preserved. But we can sort the elements.
- Indexing and slicing not allowed for the set.
- Heterogeneous elements are allowed.
- Set objects are mutable i.e., once we create set object, we can perform any changes in that object based on our requirement.
- We can represent set elements within curly braces and with comma separation.
- We can apply mathematical operations like union, intersection, difference etc on set objects.

Mathematical operations on the Set

1. *union()*:

`x.union(y)` → We can use this function to return all elements present in both sets
`x.union(y)` or `x|y`

2. *intersection()*:

`x.intersection(y)` or `x&y`
Returns common elements present in both x and y

3. *difference()*:

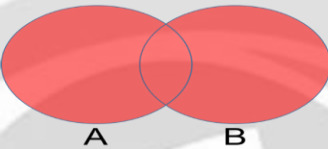
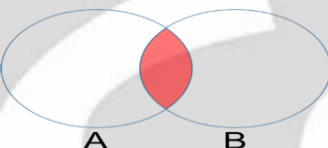
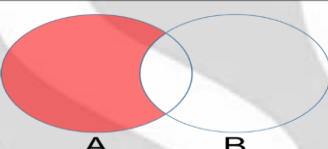
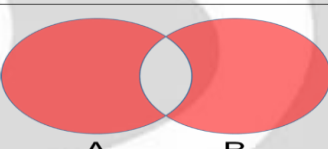
`x.difference(y)` or `x-y`

Returns the elements present in x but not in y

4. *symmetric_difference()*:

`x.symmetric_difference(y)` or `x^y`

Returns elements present in either x or y but not in both

Set Operation	Venn Diagram	Interpretation
Union		$A \cup B$, is the set of all values that are a member of A, or B, or both.
Intersection		$A \cap B$, is the set of all values that are members of both A and B.
Difference		$A \setminus B$, is the set of all values of A that are not members of B
Symmetric Difference		$A \triangle B$, is the set of all values which are in one of the sets, but not both.

Dictionary

We can use List, Tuple and Set to represent a group of individual objects as a single entity. If we want to represent a group of objects as key-value pairs then we should go for Dictionary.

Eg: rollno --name

phone number--address

ip address - domain name

- Duplicate keys are not allowed but values can be duplicated.
- Heterogeneous objects are allowed for both key and values.
- Insertion order is not preserved.

- Dictionaries are mutable.
- Dictionaries are dynamic
- Indexing and slicing concepts are not applicable

Deep Copy and Shallow Copy

In python, Assignment statements do not copy objects, they create bindings between a target and an object. When we use the = operator, it only creates a new variable that shares the reference of the original object. In order to create “real copies” or “clones” of these objects, we can use the copy module in python.

Syntax for deep copy: `copy.deepcopy(x)`

Syntax for shallow copy: `copy.copy(x)`

The copy() returns a shallow copy of the list, and deepcopy() returns a deep copy of the list.

Shallow copy-->any changes made to a copy of object do reflect in the original object.

Deep copy --> Any changes made to a copy of object do not reflect in the original object.

List Comprehensions

- List compression is a feature, which one can use within a single line of code to construct powerful functionality.
- List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc.
- A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

Syntax: `list = [expression for item in list if condition]`

2.3 String Operations and Format Specifiers

String

The most commonly used object in any project and in any programming, language is String. Any sequence of characters within either single quotes or double quotes is considered as a String.

Doc String/Documentation string

To get the string as it is which is with in the triple quotations

The docstrings are declared using '''triple single quotes''' or """triple double quotes"""

(Or)

We can define multi-line String literals by using triple single or double quotes.

String Operations

1. **len():**

We can use len() function to find the number of characters present in the string.

2. **rstrip():**

To remove spaces at right hand side.

3. **lstrip():**

To remove spaces at left hand side.

4. **strip():**

To remove spaces both sides.

5. **find():**

Returns index of first occurrence of the given substring.

6. **upper():**

To convert all characters to upper case.

7. **lower():**

To convert all characters to lower case

8. **swapcase():**

Converts all lower case characters to upper case and all upper case characters to lower case.

9. **title():**

To convert all character to title case. i.e., first character in every word should be upper case and all remaining characters should be in lower case.

10. **capitalize():**

Only first character will be converted to upper case and all remaining characters can be converted to lower case.

Formatting the Strings

The main objective of format() method is to format the string into meaningful output form. We can format the strings with variable values by using replacement operator {} and format() method.

Formatting string using % It is the oldest method of string formatting. Here we use the modulo % operator. The modulo % is also known as the “string-formatting operator”.

Formatting string using format() method

Format() method was introduced with Python3 for handling complex string formatting more efficiently. Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces {} into a string and calling the str.format(). The value we wish to put into the placeholders and concatenate with the string passed as parameters into the format function.

Syntax: *'String here {} then also {}'.format('something1','something2')*

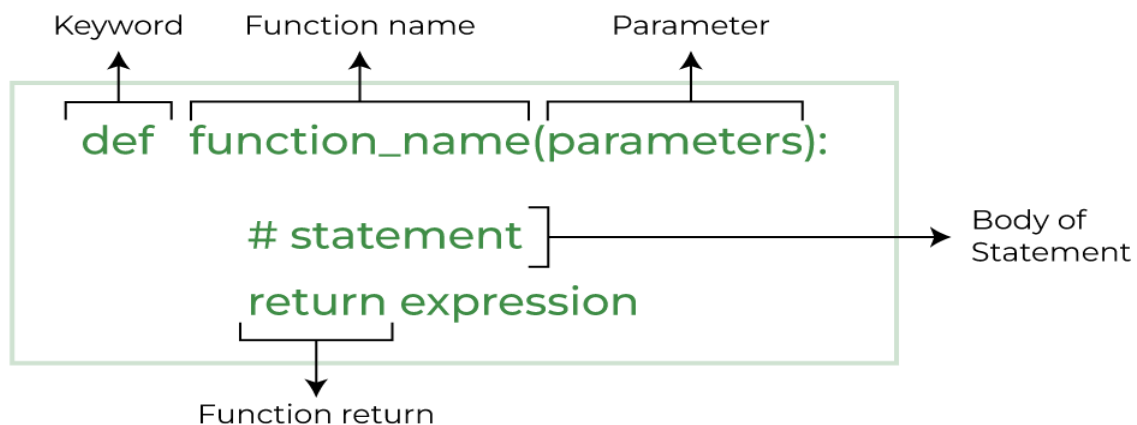
2.4 Python Functions and Generators

Function

Function is a block of related statements which is designed to perform a logical and evaluative task. We can create a Python function using the def keyword.

We need to keep the following points in mind while calling functions:

- In the case of passing the keyword arguments, the order of arguments is important.
- There should be only one value for one parameter.
- The passed keyword name should match with the actual keyword name.
- In the case of calling a function containing non-keyword arguments, the order is important.



Python supports 2 types of functions

User-defined functions

- The functions which are developed by programmer explicitly according to business requirements, are called user defined functions.

In-built functions

- The functions which are coming along with Python software automatically, are called built in functions or pre-defined functions.
- All data types, conversion functions, input and output functions are called inbuilt functions
- Eg: id() type() input() eval() etc..

When we define and call a Python function, the term parameter and argument is used to pass information to the function.

Parameter: It is the variable listed inside the parentheses in the function definition.

Argument: It is a value sent to the function when it is called. It is data on which function performs some action and returns the result.

Types of Function

1) Required Argument Function

Required arguments are the arguments passed to a function in correct positional order. The number of arguments in the function call should match exactly with the function definition.

2) Default Argument Function

In this function, arguments can have default values. We assign default values to the argument using the '=' (assignment) operator at the time of function definition. You can define a function with any number of default arguments.

The default value of an argument will be used inside a function if we do not pass a value to that argument at the time of the function call. Due to this, the default arguments become optional during the function call.

3) Variable Length Argument Function

In Python, there is a situation where we need to pass multiple arguments to the function. Such types of arguments are called arbitrary arguments or variable-length arguments. We use variable-length arguments if we don't know the number of arguments needed for the function in advance.

Types of Arbitrary Arguments:

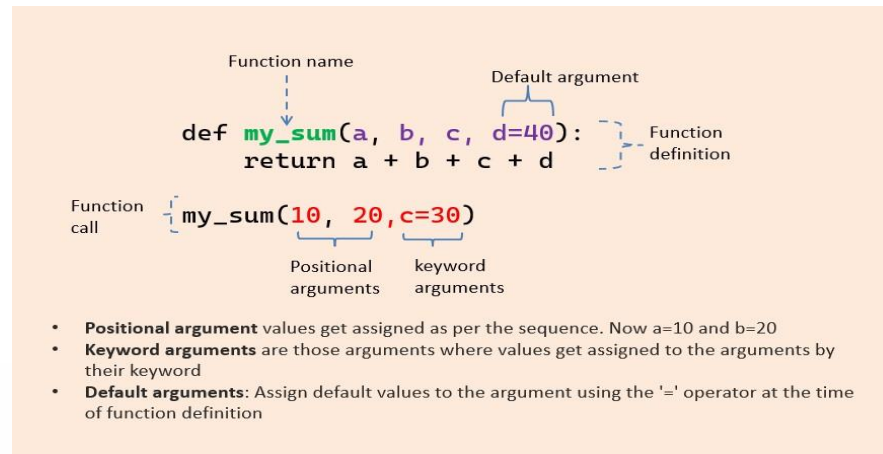
*(a) Arbitrary positional arguments (*args)*

We can declare a variable-length argument with the * (asterisk) symbol. Place an asterisk (*) before a parameter in the function definition to define an arbitrary positional argument. we can pass multiple arguments to the function. Internally all these values are represented in the form of a tuple.

*(b) Arbitrary keyword arguments (**kwargs)*

The **kwargs allow you to pass multiple keyword arguments to a function. Use the **kwargs if you want to handle named arguments in a function.

Use the unpacking operator (**) to define variable-length keyword arguments. Keyword arguments passed to a kwargs are accessed using key-value pair (same as accessing a dictionary in Python).



4) Anonymous Functions

An anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword. Anonymous functions are also called lambda functions.

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

5) Filter Functions

We can use filter() function to filter values from the given sequence based on some condition. The Python built-in filter() function extracts only particular elements from an iterable (like list, tuple, dictionary, etc.). It takes a function and an iterable as arguments and applies the function passed as an argument to each element of the iterable; once this process of filtering is completed, it returns an iterator as a result. An iterable is a Python object that can be iterated over.

Syntax: filter(function, sequence)

function argument is responsible to perform conditional check

sequence can be list or tuple or string.

6) Map Functions

Map in Python is a function that works as an iterator to return a result after applying a function to every item of an iterable (tuple, lists, etc.). It is used when you want to apply a single transformation function to all the iterable elements. The iterable and function are passed as arguments to the map in Python.

Syntax: map (function, iterables)

function: It is the transformation function through which all the items of the iterable will be passed.

iterables: It is the iterable (sequence, collection like list or tuple) that you want to map.

7) Reduce Functions

The reduce() function is defined in the functools module. Like the map and filter functions, the reduce() function receives two arguments, a function and an iterable. However, it doesn't return another iterable, instead it returns a single value.

Syntax: reduce(function,sequence)

To use the reduce() function, you need to import it from the functools module

```
from functools import reduce
```

Generator

Generators in Python are used to create iterators and return a traversal object. It helps in traversing all the items one at a time with the help of the keyword yield. It is indicated with 'yield' keyword. It is used instead of return statement. gen fun contains one or more yield function. when called, it returns the iterator but does not execute immediately. methods like next() are implemented automatically, so we can iterate the items using next() function. Once the function is yields, the function is paused and control is transferred to the caller. Local variables and their states are remembered between successive calls. Finally, when the function terminates stop iteration is raised automatically on further calls.

```
def gen_fun():  
    yield 10  
    yield 20  
    yield 30  
for i in gen_fun():  
    print(i)
```

>>>> Output: 10
 20
 30

2.5 File Handling and Exception Handling

Files are identified locations on a disk where associated data is stored. Working with files will make your programs fast when analyzing masses of data. *Exceptions* are special objects that any programming language uses to manage errors that occur when a program is running.

Modules

A *Python module* is a file containing Python definitions and statements. A module can define functions, classes, and variables. A file containing Python code, for example: example.py, is called a module, and its module name would be example. We use modules to break down large programs into small manageable and organized files. Modules provide reusability of code.

Let us create a module. Type the following and save it as example.py.

Python Module example

```
def add(a, b):  
    """This program adds two  
    numbers and return the result"""  
    result = a + b  
    return result
```

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum. We use the import keyword to do this. To import our previously defined module example, we type the following in the Python prompt.

```
import example  
example.add(4,5.5)  
9.5
```

Library

A *Python library* is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. For example, using Seaborn, you can generate visualizations with just one line of code. To create a chart from an object, you'd have to write a lot of code without a library like this. Some of the important library in python are NumPy, Pandas, Matplotlib, SciPy etc.

File handling

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g., hard disk). Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them. When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0s and 1s).

Text files: In this type of file, each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in python by default.

Binary files: In this type of file, there is no terminator for a line and the data is stored after converting it into machine-understandable binary language.

Types of modes

r: open an existing file for a read operation.

w: open an existing file for a write operation. If the file already contains some data, then it will be overridden.

a: open an existing file for append operation. It won't override existing data.

r+: To read and write data into the file. The previous data in the file will be overridden.

w+: To write and read data. It will override existing data.

a+: To append and read data from the file. It won't override existing data.

OS module

Python OS module provides the facility to establish the interaction between the user and the operating system. It offers many useful OS functions that are used to perform OS-based tasks and get related information about operating system. The OS comes under Python's standard utility modules. This module offers a portable way of using operating system dependent functionality.

OS module has path sub module which contains is File () function to check whether a particular file exists or not?

```
os.path.isfile(fname)
```

Pickling & Unpickling

Python pickle module is used for serializing and de-serializing python object structures. The process to convert any kind of python objects (list, dict, etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshallng. We can convert the byte stream (generated through pickling) back into python objects by a process called as unpickling.

In real world scenario, the use pickling and unpickling are used to easily transfer data from one server/system to another and then store it in a file or database. It is advisable not to unpickle data received from an untrusted source as they may pose security threat. After importing pickle module, we can do pickling and unpickling. Importing pickle can be done using the following command

```
import pickle
```

Exception handling

Exception Handling is a mechanism to handle runtime errors such as 'ClassNotFoundException', 'IOException', 'SQLException', 'RemoteException' etc. Error in Python can be of two types i.e., Syntax errors and Exceptions. Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.

Syntax Error: This error is caused by the wrong syntax in the code. It leads to the termination of the program.

Exceptions: Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program, it changes the normal flow of the program.

Exception handling is done the below parameters.

- try
- except
- finally

Try and except statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause. Both keywords are followed by indented blocks.

Finally, keyword is always executed after the try and except blocks. The final block always executes after normal termination of try block or after try block terminates due to some exception.

Decorators

Decorators are used to modify the behavior of the function. Decorators provide the flexibility to wrap another function to expand the working of wrapped function, without permanently modifying it. In Decorators, functions are passed as an argument into another function and then called inside the wrapper function.

We use a decorator when we need to change the behavior of a function without modifying the function itself. A few good examples are when we want to add logging, test performance, perform caching and verify permissions. We can also use one when we need to run the same code on multiple functions. This avoids us writing duplicating code.

To create a decorator function in Python, we create an outer function that takes a function as an argument. There is also an inner function that wraps around the decorated function. Here is the syntax for a basic Python decorator:

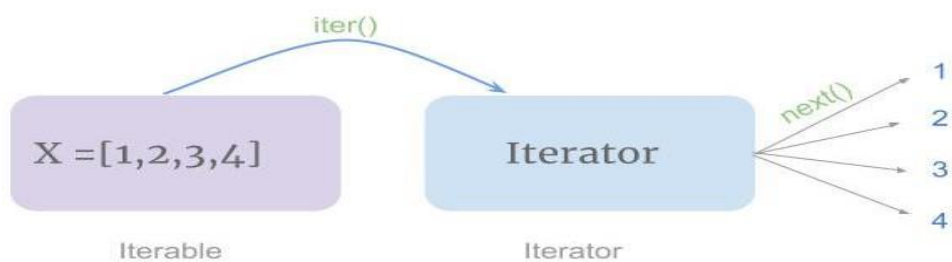
```
def my_decorator_func(func):  
    def wrapper_func():  
        # Do something before the function.  
        func()  
        # Do something after the function.  
    return wrapper_func
```

To use a decorator, we attach it to a function like you see in the code below. We use a decorator by placing the name of the decorator directly above the function we want to use it on. You prefix the decorator function with an @ symbol.

```
@my_decorator_func  
def my_func():  
    pass
```

Iterators

An iterator in Python is an object that contains a countable number of elements that can be iterated upon. In simpler words, we can say that Iterators are objects that allow you to traverse through all the elements of a collection and return one element at a time. Iterable is an object, which one can iterate over. It generates an Iterator when passed to the iter() method. Lists, tuples, dictionaries, strings and sets are all iterable objects. They are iterable containers that you can convert into an iterator.



2.6 Regular Expressions and Date, Time

A Regular Expressions (RegEx) is a special sequence of characters that uses a search pattern to find a string or set of strings. It can detect the presence or absence of a text by matching it with a particular pattern, and also can split a pattern into one or more sub-patterns.

If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions. Regular Expressions is a declarative mechanism to represent a group of Strings according to particular format/pattern.

The main important application areas of Regular Expressions are:

- To develop validation frameworks/validation logic.
- To develop Pattern matching applications (ctrl-f in windows).
- To develop Translators like compilers, interpreters etc.
- To develop digital circuits.

We can develop Regular Expression Based applications by using python module: re (import re). This module contains several inbuilt functions to use Regular Expressions very easily in our applications.

Regular expressions are inbuilt functions. Important functions of re module

- Search
- Match
- Finditer
- Findall
- Sub
- Subn
- Split
- Compile

1. Match function

Match function of re in Python will search the regular expression pattern and return the first occurrence. The Python RegEx Match method checks for a match only at the beginning of the string. So, if a match is found in the first line, it returns the match object. But if a match is found in some other line, the Python RegEx Match function returns null.

Syntax: re.match(pattern,string)

2. Search function

Search function will search the regular expression pattern and return the first occurrence. Unlike Python `re.match()`, it will check all lines of the input string. The Python `re.search()` function returns a match object when the pattern is found and “null” if the pattern is not found.

Syntax: `re.search(pattern,string)`

3. Finditer function

The `finditer()` function matches a pattern in a string and returns an iterator that yields the Match objects of all non-overlapping matches. If the search is successful, the `finditer()` function returns an iterator yielding the Match objects. Otherwise, the `finditer()` also returns an iterator that will yield no Match object.

Syntax: `re.finditer(pattern,string)`

pattern is regular expression that you want to search for in the string.

string is the input string.

4. Findall function

`Findall()` function is used to search for “all” occurrences that match a given pattern. In contrast, `Search()` function will only return the first occurrence that matches the specified pattern. `findall()` will iterate over all the lines of the file and will return all non-overlapping matches of pattern in a single step.

Syntax: `re.finditer(pattern,string)`

5. Sub function

The `re.sub()` function is used to replace occurrences of a particular sub-string with another sub-string. This function takes as input the following:

- The sub-string to replace
- The sub-string to replace with
- The actual string

Syntax: `re.sub(regex,replacement,targetstring)`

6. Split function

The `re.split()` function splits the string at every occurrence of the sub-string and returns a list of strings which have been split.

Syntax: `string.split(separator, maxsplit)`

Separator is optional. It specifies the separator to use when splitting the string. By default, any whitespace is a separator.

Maxsplit is optional. It specifies how many splits to do. Default value is -1, which is "all occurrences".

7. Compile function

The `compile()` function takes source code as input and returns a code object which can later be executed by `exec()` function.

Syntax: `compile(source, filename, mode, flag, dont_inherit, optimize)`

source - normal string, a byte string, or an AST (Abstract Syntax Trees) object.

filename - File from which the code is read.

- mode - mode can be either `exec` or `eval` or `single`.
- eval - if the source is a single expression.
- exec - if the source is block of statements.
- single - if the source is single statement.

flags and dont_inherit - Default Value= 0. Both are optional parameters. It monitors that which future statements affect the compilation of the source.

optimize (optional) - Default value -1. It defines the optimization level of the compiler.

Date and Time module

The `datetime` module supplies classes for manipulating dates and times in both simple and complex ways. Python `Datetime` module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and `datetime` are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

1) We can use Python `datetime` module to get the current date and time of the local system.

```
from datetime import datetime
# Current date time in local system
print(datetime.now())
```

2) If you want current date of the local system, you can use the datetime date() method.

```
print(datetime.date(datetime.now()))
```

3) If you want only time in the local system, use time() method by passing datetime object as an argument.

```
print(datetime.time(datetime.now()))
```

4) Python Current Date Time in timezone – pytz

Most of the times, we want the date in a specific timezone so that it can be used by others too. Python pytz is one of the popular modules that can be used to get the timezone implementations. You can install this module using the following PIP command.

```
pip install pytz

import pytz
utc = pytz.utc
pst = pytz.timezone('America/Los_Angeles')
ist = pytz.timezone('Asia/Calcutta')
```

5) We can get the year, month, and date attributes from the date object using the year, month and date attribute of the date class.

```
from datetime import date
# date object of today's date
today = date.today()
print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

6) We can create date objects from timestamps by using the fromtimestamp() method. The timestamp is the number of seconds from 1st January 1970 at UTC to a particular date.

```
from datetime import datetime
# Getting Datetime from timestamp
date_time = datetime.fromtimestamp(1887639468)
print("Datetime from timestamp:", date_time)
```

7) Python timedelta class is used for calculating differences in dates and also can be used for date manipulations in Python. It is one of the easiest ways to perform date manipulations.

Syntax: class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

Returns: Date

Finding age

Calculating age is a very simple process. We will have to find the difference between the current year and the birth year. Python provides various approaches to calculate the age.

Python provides an inbuilt datetime module. It has several classes which provide a number of functions to deal with dates, times, and time intervals. To calculate the age, we simply subtract the birth year from the current year. To implement this, we need to check if the current month and date are less than the birth month and date. If it is, subtract 1 from the age, otherwise 0. For example

```
from datetime import date
def calculateAge(dob):
    today = date.today()
    age = today.year - dob.year - (((today.month, today.day) <
    (dob.month, dob.day))
    return age
print(calculateAge(date(2017, 9, 18)), " years old")
```