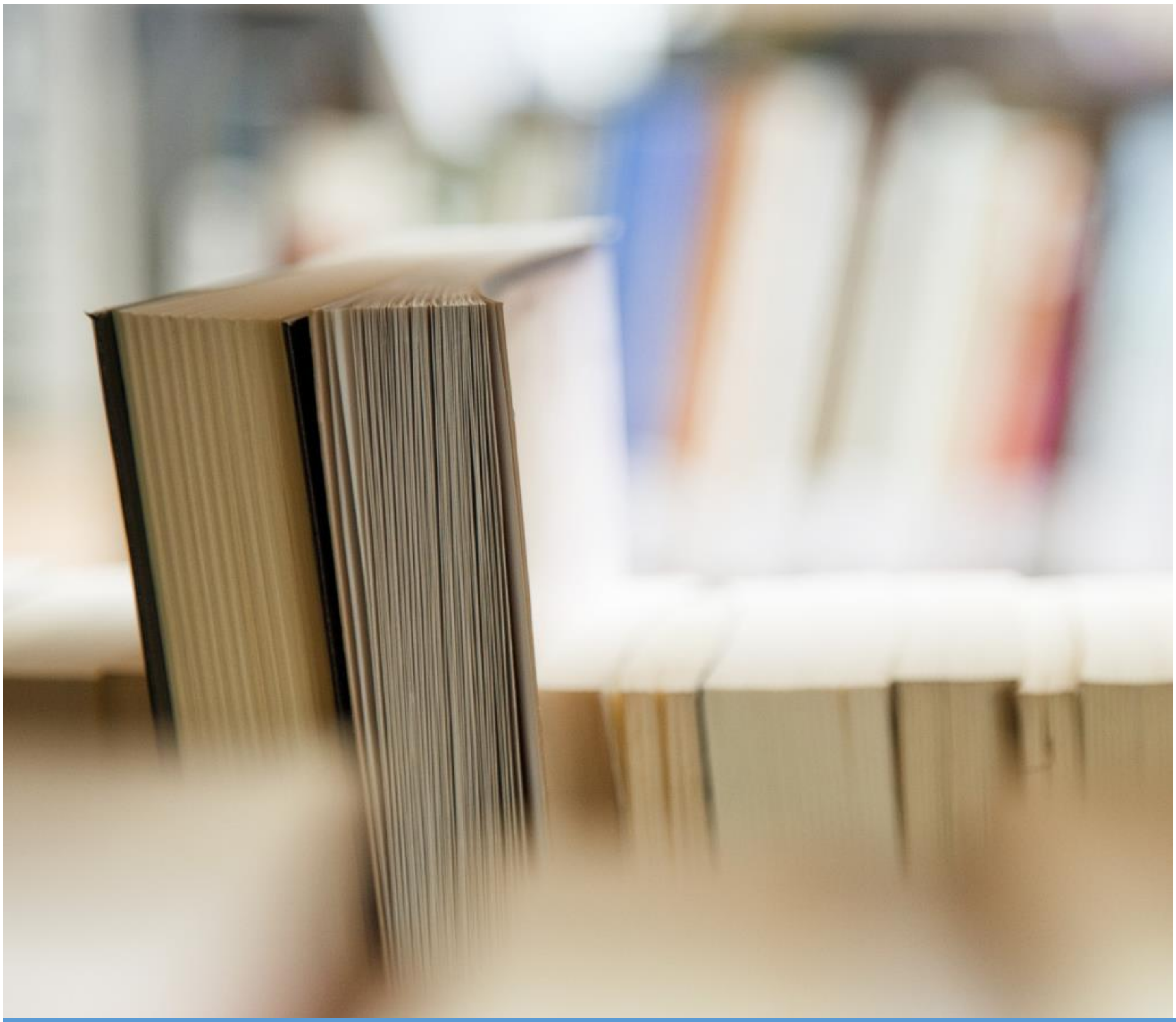


Python Hand Book

Core Python

Prepared By Deepak Sir

Chapter 1:- Python Introduction

- Python tutorial provides basic and advanced concepts of Python.
- This Python tutorial is designed for beginners and professionals.
- Python is a simple, general purpose, high level, and object-oriented programming language.
- Python is an interpreted scripting language also.
- **Guido Van Rossum is known as the founder of Python programming**.

What is Python?

- **Python** is a general purpose, dynamic, high level, and interpreted programming language. It supports Object Oriented programming approach to develop applications.
- It is simple and easy to learn and provides lots of high-level data structures.
- Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.
- Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.
- Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.
- Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.
- We don't need to use data types to declare variable because it is *dynamically typed* so we can write **a=10** to assign an integer value in an integer variable.
- Python makes the development and debugging *fast* because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

Python 2 vs. Python 3

In most of the programming languages, whenever a new version releases, it supports the features and syntax of the existing version of the language, therefore, it is easier for the projects to switch in the newer version.
However, in the case of Python, the two versions Python 2 and Python 3 are very much different from each other.

Differences between Python 2 and Python 3 are given below:

1. Python 2 uses **print** as a statement and used as print "something" to print some string on the console. On the other hand, Python 3 uses **print** as a function and used as print("something") to print something on the console.
2. Python 2 uses the function raw_input() to accept the user's input. It returns the string representing the value, which is typed by the user. To convert it into the integer, we need to use the int() function in Python. On the other hand, Python 3 uses input() function which automatically interpreted the type of input entered by the user. However, we can cast this value to any type by using primitive functions (int(), str(), etc.).
3. In Python 2, the implicit string type is ASCII, whereas, in Python 3, the implicit string type is Unicode.
4. Python 3 doesn't contain the xrange() function of Python 2. The xrange() is the variant of range() function which returns a xrange object that works similar to Java iterator. The range() returns a list for example the function range(0,3) contains 0, 1, 2.

Python Features

Python provides lots of features that are listed below.

1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5) Free and Open Source

Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.

6) Object-Oriented Language

Python supports object oriented language and concepts of classes and objects come into existence.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8) Large Standard Library

Python has a large and broad library and provides rich set of module and functions for rapid application development.

9) GUI Programming Support

Graphical user interfaces can be developed using Python.

10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

Python Applications

Python is known for its general purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

Here, we are specifing applications areas where python can be applied.

1) Web Applications

We can use Python to develop web applications.
It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser etc.
It also provides Frameworks such as Django, Pyramid, Flask etc to design and develop web based applications.
Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

2) Desktop GUI Applications

Python provides Tk GUI library to develop user interface in python based application.
Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms.
The Kivy is popular for writing multitouch applications.

3) Software Development

Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

4) Scientific and Numeric

Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

5) Business Applications

Python is used to build Business applications like ERP and e-commerce systems. Tryton is a high level application platform.

6) Console Based Application

We can use Python to develop console based applications. For example: **IPython**.

7) Audio or Video based Applications

Python is awesome to perform multiple tasks and can be used to develop multimedia applications.
Some of real applications are: TimPlayer, cplay etc.

8) 3D CAD Applications

To create CAD application Fandango is a real application which provides full features of CAD.

9) Enterprise Applications

Python can be used to create applications which can be used within an Enterprise or an Organization.
Some real time applications are: OpenErp, Tryton, Picalo etc.

10) Applications for Images

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.
There are several such applications which can be developed using Python

Chapter 2: Python Variables, Data Type & Operators

Python Variables

- Variable is a name which is used to refer memory location.
- Variable also known as identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.
- Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for variable name.
- Rahul and rahul both are two different variables.

Identifier Naming

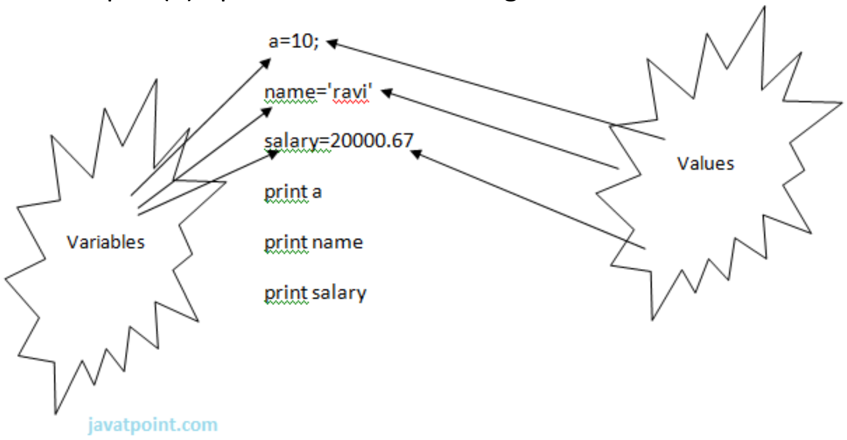
Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive for example my name, and MyName is not the same.

Examples of valid identifiers : a123, _n, n_9, etc.
Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

- Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.
- We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.
- The equal (=) operator is used to assign value to a variable.



```
a=10;
name='ravi'
salary=20000.67
print a
print name
print salary
```

Output:

```
1.	>>>
2.	10
3.	ravi
4.	20000.67
5.	>>>
```

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.
We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables.

Lets see given examples

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print x
print y
print z
```

Output:

```
>>>
50
50
50
>>>
```

2.Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print a
print b
print c
```

Output:

```
>>>
5
10
15
>>>
```

The values will be assigned in the order in which variables appears.

Basic Fundamentals:

This section contains the basic fundamentals of Python like:

I) Tokens and their types.
ii) Comments

Tokens:

- Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
- Token is the smallest unit inside the given program.
  There are following tokens in Python:
- Keywords.
- Identifiers.
- Literals.
- Operators.

Python Data Types

- Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it.
- The interpreter implicitly binds the value with its type.
- Python enables us to check the type of the variable used in the program.
- Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
 A=10
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.
Python provides various standard data types that define the storage method on each of them.
The data types defined in Python are given below.

1. **Numbers**
2. **String**
3. **List**
4. **Tuple**
5. **Dictionary**

Numbers

Number stores numeric values. Python creates Number objects when a number is assigned to a variable.
For example;

```
1.	a = 3 , b = 5  #a and b are number objects
```

Python supports 4 types of numeric data.

1. int (signed integers like 10, 2, 29, etc.)
2. long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.
A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

Python Keywords

- Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation.
- These keywords can't be used as variable.

Following is the List of Python Keywords.

True	**False**	**None**	**and**	**as**
asset	**def**	**class**	**continue**	**break**
else	**finally**	**elif**	**del**	**except**
global	**for**	**if**	**from**	**import**
raise	**try**	**or**	**return**	**pass**
nonlocal	**in**	**not**	**is**	**lambda**

Python Literals

Literals can be defined as a data that is given in a variable or constant.
Python support the following literals:

I. String literals:

- String literals can be formed by enclosing a text in the quotes.
- We can use both single as well as double quotes for a String.

Eg: "Aman" , '12345'

Types of Strings:

There are two types of Strings supported in Python:

a).Single line String-

Strings that are terminated within a single line are known as Single line Strings.

Eg:
```
>>> text1='hello'
```

b).Multi line String-

A piece of text that is spread along multiple lines is known as Multiple line String.

2).Using triple quotation marks:-

Eg:

```
>>> str2='''welcome
to
SSSIT'''
>>> print str2
welcome
to
SSSIT
>>>
```

II.Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)	
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. eg: 3.14j	

III. Boolean literals:

A Boolean literal can have any of the two values: True or False.

IV. Special literals.

Python contains one special literal **i.e., None.**

None is used to specify to that field that is not created. It is also used for end of lists in Python.

Eg:

```
>>> val1=10
>>> val2=None
>>> val1
10
>>> val2
>>> print val2
None
>>>
```

V. Literal Collections.

Collections such as tuples, lists and Dictionary are used in Python.

Python Operators

- The operator can be defined as a symbol which is responsible for a particular operation between two operands.
- Operators are the pillars of a program on which the logic is built in a particular programming language.

Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators

Arithmetic operators are used to perform arithmetic operations between two operands.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to second operand.
// (Floor division)	`It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator

Comparison operators are used to comparing the value of the two operands and returns boolean true or false accordingly.

The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Python assignment operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the reminder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Logical Operators

The logical operators are used primarily in the expression evaluation to make a decision.
Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true then not (a) will be false and vice versa.

Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both side do not point to the same object.

Python Comments

- Comments in Python can be used to explain any program code.
- It can also be used to hide the code as well.
- Comments are the most helpful stuff of any program.
- It enables us to understand the way, a program works.
- In python, any statement written along with # symbol is known as a comment.
- The interpreter does not interpret the comment.

Python supports two types of comments:

1) Single Line Comment:

In case user wants to specify a single line comment, then comment must start with ?#?

Eg:

```
1.	# This is single line comment.
2.	print "Hello Python"
```

Output:

```
Hello Python
```

2) Multi Line Comment:

Multi lined comment can be given inside triple quotes.

eg:

```
1.	""" This
2.	    Is
3.	    Multipline comment'''
```

Chapter 3: Python Conditional Statement (If)

Python If-else statements

- Decision making is the most important aspect of almost all the programming languages.
- As the name implies, decision making allows us to run a particular block of code for a particular decision.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

- For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code.
- In Python, indentation is used to declare a block.
- If two statements are at the same indentation level, then they are the part of the same block.
- Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.
- Indentation is the most used part of the python language since it declares the block of code. A
- ll the statements of one block are intended at the same level indentation.
- We will see how the actual indentation takes place in decision making and other stuff in python.

The if statement

- The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block.
- The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

Example 1

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```

Output:

```
enter the number?10
Number is even
```

Example 2 : Program to print the largest of the three numbers.

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

Output:

```
Enter a? 100
Enter b? 120
Enter c? 130
c is largest
```

The if-else statement

- The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.
- If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

Example 1 : Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:

```
Enter your age? 90
You are eligible to vote !!
```

Example 2: Program to check whether a number is even or not.

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even...")
else:
    print("Number is odd...")
```

Output:

```
enter the number?10
Number is even
```

The elif statement

- The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.
- We can have any number of elif statements in our program depending upon our need.
- However, using elif is optional.

Example 1

```
number = int(input("Enter the number?"))
if number==10:
   print("number is equals to 10")
elif number==50:
   print("number is equal to 50");
elif number==100:
   print("number is equal to 100");
else:
   print("number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15
number is not equal to 10, 50 or 100
```

Example 2

```
marks = int(input("Enter the marks? "))
if marks > 85 and marks <= 100:
  print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
  print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
  print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
  print("You scored grade C ...")
else:
  print("Sorry you are fail ?")
```

Chapter 4: Python Loops

- The flow of the programs written in any programming language is sequential by default.
- Sometimes we may need to alter the flow of the program.
- The execution of a specific code may need to be repeated several numbers of times.
- For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times

Why we use loops in python?

- The looping simplifies the complex problems into the easy ones.
- It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times.
- For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

There are the following loop statements in Python.

Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

Python for loop

- The for **loop in Python** is used to iterate the statements or a part of the program several times.
- It is frequently used to traverse the data structures like list, tuple, or dictionary.

Example

```
i=1
n=int(input("Enter the number ?"))
for i in range(0,10):
print(i,end = ' ')
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Python for loop example : printing the table of the given number

```

i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
print("%d X %d = %d"%(num,i,num*i));
```

Output:

```
Enter a number:10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Nested for loop in python

- Python allows us to nest any number of for loops inside a for loop.
- The inner loop is executed n number of times for every iteration of the outer loop.

```
n = int(input("Enter the number of rows you want to print?"))
i,j=0,0
for i in range(0,n):
    print()
    for j in range(0,i+1):
        print("*",end="")
```

Output:

```
Enter the number of rows you want to print?5
*
**
***
****
*****
```

Using else statement with for loop

Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

Example 1

```
for i in range(0,5):
    print(i)
else:print("for loop completely exhausted, since there is no break.");
```

In the above example, for loop is executed completely since there is no break statement in the loop. The control comes out of the loop and hence the else block is executed.

Output:

```
0
1
2
3
4
for loop completely exhausted, since there is no break.
```

Example 2

```
for i in range(0,5):
    print(i)
    break;
else:print("for loop is exhausted");
print("The loop is broken due to break statement...came out of loop")
```

In the above example, the loop is broken due to break statement therefore the else statement will not be executed. The statement present immediate next to else block will be executed.

Output:

```
0
The loop is broken due to break statement...came out of loop
```

Python while loop

- The while loop is also known as a pre-tested loop.
- In general, a while loop allows a part of the code to be executed as long as the given condition is true.
- It can be viewed as a repeating if statement.
- The while loop is mostly used in the case where the number of iterations is not known in advance.

Example 1

```
i=1;
while i<=10:
   print(i);
   i=i+1;
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example 2

```
i=1
number=0
b=9
number = int(input("Enter the number?"))
while i<=10:
   print("%d X %d = %d \n"%(number,i,number*i));
   i = i+1;
```

Output:

```
Enter the number?10

10 X 1 = 10

10 X 2 = 20

10 X 3 = 30

10 X 4 = 40

10 X 5 = 50

10 X 6 = 60

10 X 7 = 70

10 X 8 = 80

10 X 9 = 90

10 X 10 = 100
```

Infinite while loop

- If the condition given in the while loop never becomes false then the while loop will never terminate and result into the infinite while loop.
- Any non-zero value in the while loop indicates an always-true condition whereas 0 indicates the always-false condition.
- This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

Example 1

```
while (1):
    print("Hi! we are inside the infinite while loop");
```

Output:

```
Hi! we are inside the infinite while loop
(infinite times)
```

Example 2

```
var = 1
while var != 2:
    i = int(input("Enter the number?"))
    print ("Entered value is %d"%(i))
```

Output:

```
Enter the number?102
Entered value is 102
Enter the number?102
Entered value is 102
Enter the number?103
Entered value is 103
Enter the number?103
(infinite loop)
```

Using else with Python while loop

- The else block is executed when the condition given in the while statement becomes false.
- Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.

```
i=1;
while i<=4:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
```

Output:

```
1
2
3
4
The while loop exhausted
```

Example 2

```
i=1;
while i<=5:
    print(i)
    i=i+1;
    if(i==3):
        break;
else:print("The while loop exhausted");
```

Output:

```
1
2
```

Python break statement

- Break is used to abort the current execution of the program and the control goes to the next line after the loop.
- The break is commonly used in the cases where we need to break the loop for a given condition.

Example 1

```
list =[1,2,3,4]
count = 1;
for i in list:
    if i == 4:
        print("item matched")
        count = count + 1;
        break
print("found at",count,"location");
```

Output:

```
item matched
found at 2 location
```

Example 2

```
str = "python"
for i in str:
    if i == 'o':
        break
    print(i);
```

Output:

```
p
y
t
h
```

Example 3: break statement with while loop

```
i = 0;
while 1:
   print(i," ",end=""),
   i=i+1;
   if i == 10:
      break;
print("came out of while loop");
```

Output:

```
0 1 2 3 4 5 6 7 8 9 came out of while loop
```

Python continue Statement

- The continue statement in python is used to bring the program control to the beginning of the loop.
- The continue statement skips the remaining lines of code inside the loop and start with the next iteration.
- It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

Example 1

```
i = 0;
while i!=10:
   print("%d"%i);
   continue;
   i=i+1;
```

Output:

```
infinite loop
```

Example 2

```
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
   if i==5:
      continue;
   print("%d"%i);
```

Output:

```
1
2
3
4
6
7
8
9
10
```

Chapter 5: Python Modules

- A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module. We may have a runnable code inside the python module.
- Modules in Python provides us the flexibility to organize the code in a logical way.
- To use the functionality of one module into another, we must have to import the specific module.

Example

In this example, we will create a module named as file.py which contains a function that contains a code to print some message on the console.

Let's create the module named as **file.py.**

```
#displayMsg prints a message to the name being passed.
def displayMsg(name)
    print("Hi "+name);
```

Here, we need to include this module into our main module to call the method displayMsg() defined in the module named file.

Loading the module in our python code

We need to load the module in our python code to use its functionality. Python provides two types of statements as defined below.

1. The import statement
2. The from-import statement

The import statement

The import statement is used to import all the functionality of one module into another.
We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times, it has been imported into our file.

The syntax to use the import statement is given below.

```
import module1,module2,........ module n
```

Hence, if we need to call the function displayMsg() defined in the file file.py, we have to import that file as a module into our module as shown in the example below.

Example:

```
import file;
name = input("Enter the name?")
file.displayMsg(name)
```

Output:

```
Enter the name?John
Hi John
```

The from-import statement

Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module. This can be done by using from? import statement. The syntax to use the from-import statement is given below.

```
from < module-name> import <name 1>, <name 2>..,<name n>
```

Consider the following module named as calculation which contains three functions as summation, multiplication, and divide.

calculation.py:

```
#place the code in the calculation.py
def summation(a,b):
   return a+b
def multiplication(a,b):
   return a*b;
def divide(a,b):
   return a/b;
```

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b)) #we do not need to specify the module name while accessing summation()
```

Output:

```
Enter the first number10
Enter the second number20
Sum =  30
```

The from...import statement is always better to use if we know the attributes to be imported from the module in advance. It doesn't let our code to be heavier. We can also import all the attributes from a module by using *.

Consider the following syntax.

```
from <module> import *
```

Renaming a module

Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in our python source file.
The syntax to rename a module is given below.

```
import <module-name> as <specific-name>
```

Example

```
#the module calculation of previous example is imported in this example as cal.
import calculation as cal;
a = int(input("Enter a?"));
b = int(input("Enter b?"));
print("Sum = ",cal.summation(a,b))
```

Output:

```
Enter a?10
Enter b?20
Sum =  30
```

Using dir() function

The dir() function returns a sorted list of names defined in the passed module. This list contains all the sub-modules, variables and functions defined in this module.
Consider the following example.

Example

```
import json

List = dir(json)

print(List)
```

Output:

```
['JSONDecoder', 'JSONEncoder', '__all__', '__author__', '__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__',
'_default_decoder', '_default_encoder', 'decoder', 'dump', 'dumps', 'encoder', 'load',
'loads', 'scanner']
```

The reload() function

As we have already stated that, a module is loaded once regardless of the number of times it is imported into the python source file. However, if you want to reload the already imported module to re-execute the top-level code, python provides us the reload() function. The syntax to use the reload() function is given below.

```
reload(<module-name>)
```

for example, to reload the module calculation defined in the previous example, we must use the following line of code.

```
reload(calculation)
```

Chapter 6: Python Functions

- Functions are the most important aspect of an application.
- A function can be defined as the organized block of reusable code which can be called whenever required.
- Python allows us to divide a large program into the basic building blocks known as function.
- A function can be called multiple times to provide reusability and modularity to the python program.
- In other words, we can say that the collection of functions creates a program.
- The function is also known as procedure or subroutine in other programming languages.
- Python provide us various inbuilt functions like range() or print().
- Although, the user can create its functions which can be called user-defined functions.

Advantage of Functions in Python

There are the following advantages of Python functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call python functions any number of times in a program and from any place in a program.
- We can track a large python program easily when it is divided into multiple functions.
- Reusability is the main achievement of python functions.
- However, Function calling is always overhead in a python program.

Creating a function

In python, we can use **def** keyword to define the function.
The syntax to define a function in python is given below.

```
def my_function():
    function-code
    return <expression>
```

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.
A function can accept any number of parameters that must be the same in the definition and function calling.

Function calling

- In python, a function must be defined before the function calling otherwise the python interpreter gives an error.
- Once the function is defined, we can call it from another function or the python prompt.
- To call the function, use the function name followed by the parentheses.

A simple function that prints the message "Hello Word" is given below.

```
def hello_world():
    print("hello world")

hello_world()
```

Output:

```
hello world
```

Parameters in function

- The information into the functions can be passed as the parameters.
- The parameters are specified in the parentheses. We can give any number of parameters, but we have to separate them with a comma.

The following example which contains a function that accepts a string as the parameter and prints it.

Example 1

```
#defining the function
def func (name):
   print("Hi ",name);

#calling the function
func("Ayush")
```

Example 2

```
#python function to calculate the sum of two variables
#defining the function
def sum (a,b):
   return a+b;

#taking values from the user
a = int(input("Enter a: "))
b = int(input("Enter b: "))

#printing the sum of a and b
print("Sum = ",sum(a,b))
```

Output:

```
Enter a: 10
Enter b: 20
Sum =  30
```

There may be several types of arguments which can be passed at the time of function calling.

1. **Required arguments**
2. **Keyword arguments**
3. **Default arguments**
4. **Variable-length arguments**

Required Arguments

- Till now, we have learned about function calling in python. However, we can provide the arguments at the time of function calling.
- As far as the required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition.

- If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Consider the following example.

Example 1

```
#the argument name is the required argument to the function func
def func(name):
    message = "Hi "+name;
    return message;
name = input("Enter the name?")
print(func(name))
```

Output:

```
Enter the name?John
Hi John
```

Example 2

```
#the function simple_interest accepts three arguments and returns the simple interest accordingly
def simple_interest(p,t,r):
    return (p*t*r)/100
p = float(input("Enter the principle amount? "))
r = float(input("Enter the rate of interest? "))
t = float(input("Enter the time in years? "))
print("Simple Interest: ",simple_interest(p,r,t))
```

Output:

```
Enter the principle amount? 10000
Enter the rate of interest? 5
Enter the time in years? 2
Simple Interest:  1000.0
```

Default Arguments

Python allows us to initialize the arguments at the function definition. If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

Example 1

```
def printme(name,age=22):
    print("My name is",name,"and age is",age)
printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function
```

Output:

```
My name is john and age is 22
```

Variable length Arguments

In the large projects, sometimes we may not know the number of arguments to be passed in advance. In such cases, Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.
However, at the function definition, we have to define the variable with * (star) as *<variable - name >.
Consider the following example.

Example

```python
def printme(*names):
    print("type of passed argument is ",type(names))
    print("printing the passed arguments...")
    for name in names:
        print(name)
printme("john","David","smith","nick")
```

Output:

```
type of passed argument is  <class 'tuple'>
printing the passed arguments...
john
David
smith
nick
```

Scope of variables

- The scopes of the variables depend upon the location where the variable is being declared.
- The variable declared in one part of the program may not be accessible to the other parts.

In python, the variables are defined with the two types of scopes.

1. Global variables
2. Local variables

The variable defined outside any function is known to have a global scope whereas the variable defined inside a function is known to have a local scope.
Consider the following example.

Example 1

```python
1.  def print_message():
2.      message = "hello !! I am going to print a message." # the variable message is local to the function
3.      print(message)
4.  print_message()
5.  print(message) # this will cause an error since a local variable cannot be accessible here.
```

Output:

```
hello !! I am going to print a message.
  File "/root/PycharmProjects/PythonTest/Test1.py", line 5, in
    print(message)
NameError: name 'message' is not defined
```

Chapter 7. Python Lambda, Map & Filter Functions

- Python allows us to not declare the function in the standard manner, i.e., by using the def keyword. Rather, the anonymous functions are declared by using lambda keyword.
- However, Lambda functions can accept any number of arguments, but they can return only one value in the form of expression.
- The anonymous function contains a small piece of code.

Example 1

```
x = lambda a:a+10
# a is an argument and a+10 is an expression which got evaluated and returned.
print("sum = ",x(20))
```

Output:

```
sum =  30
```

Example 2

Multiple arguments to Lambda function

```
x = lambda a,b:a+b
# a and b are the arguments and a+b is the expression that evaluated and returned.
print("sum = ",x(20,10))
```

Output:

```
sum =  30
```

Why use lambda functions?

- The main role of the lambda function is better described in the scenarios when we use them anonymously inside another function.
- In python, the lambda function can be used as an argument to the higher order functions as arguments. Lambda functions are also used in the scenario where we need a Consider the following example.

Example 1

```
#the function table(n) prints the table of n
def table(n):
   return lambda a:a*n;
# a will contain the iteration variable i and a multiple of n is returned at each function call
n = int(input("Enter the number?"))
b = table(n)
 #the entered number is passed into the function table. b will contain a lambda function which is called again and again with the iteration variable i
for i in range(1,11):
   print(n,"X",i,"=",b(i));
 #the lambda function b is called with the iteration variable i,
```

Output:

```
Enter the number?10
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Example 2

Use of lambda function with filter

```
#program to filter out the list which contains odd numbers
List = {1,2,3,4,10,123,22}
Oddlist = list(filter(lambda x:(x%3 == 0),List)) # the list contains all the items of the li
st for which the lambda function evaluates to true
print(Oddlist)
```

Output:

```
[3, 123]
```

Example 3

Use of lambda function with map

```
#program to triple each number of the list using map
List = {1,2,3,4,10,123,22}
new_list = list(map(lambda x:x*3,List)) # this will return the triple of each item of th
e list and add it to new_list
print(new_list)
```

Output:

```
[3, 6, 9, 12, 30, 66, 369]
```

Chapter 8: Python String

What is String in Python?

- A string is a sequence of characters.
- A character is simply a symbol. For example, the English language has 26 characters.
- Computers do not deal with characters, they deal with numbers (binary).
- Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.
- This conversion of character to a number is called encoding, and the reverse process is decoding.
- ASCII and Unicode are some of the popular encoding used.
- In Python, string is a sequence of Unicode character.
- Unicode was introduced to include every character in all languages and bring uniformity in encoding.

How to create a string in Python?

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
        the world of Python"""
print(my_string)
```

How to access characters in a string?

- We can access individual characters using indexing and a range of characters using slicing.
- Index starts from 0.
- Trying to access a character out of index range will raise an IndexError.
- The index must be an integer. We can't use float or other types, this will result into TypeError.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

```
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

If we try to access index out of the range or use decimal number, we will get errors.

```
# index must be in range
>>> my_string[15]
...
IndexError: string index out of range

# index must be an integer
>>> my_string[1.5]
...
TypeError: string indices must be integers
```

How to change or delete a string?

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

Python String Operations

There are many operations that can be performed with string which makes it one of the most used datatypes in Python.

Concatenation of Two or More Strings

- Joining of two or more strings into a single one is called concatenation.
- The + operator does this in Python. Simply writing two string literals together also concatenates them.
- The * operator can be used to repeat the string for a given number of times.

```
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

Iterating Through String

Using for loop we can iterate through a string. Here is an example to count the number of 'l' in a string.

```
count = 0
for i in 'Hello World':
    if(i == 'l'):
        count += 1
print(count,'letters found')
```

String Membership Test

We can test if a sub string exists within a string or not, using the keyword in.

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

Built-in functions to Work with Python

- Various built-in functions that work with sequence, works with string as well.
- Some of the commonly used ones are enumerate() and len().
- The enumerate()function returns an enumerate object. It contains the index and value of all the items in the string as pairs.
- This can be useful for iteration.
- Similarly, len() returns the length (number of characters) of the string.

```
str = 'cold'

# enumerate()

s = list(enumerate(str))
print(“Answer = ”, s)

#character count
print(“length = “, len(str))

output
Answer =  [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
length =  4
```

Python String Formatting

Escape Sequence

If we want to print a text like -He said, "What's there?"- we can neither use single quote or double quotes. This will result into SyntaxError as the text itself contains both single and double quotes.

```
>>> print("He said, "What's there?"")
...
SyntaxError: invalid syntax
>>> print('He said, "What's there?"')
...
SyntaxError: invalid syntax
```

- One way to get around this problem is to use triple quotes.
- Alternatively, we can use escape sequences.
- An escape sequence starts with a backslash and is interpreted differently.
- If we use single quote to represent a string, all the single quotes inside the string must be escaped.
- Similar is the case with double quotes. Here is how it can be done to represent the above text.

```
# using triple quotes
print('''He said, "What's there?"''')
```

```
# escaping single quotes
print('He said, "What\'s there?"')

# escaping double quotes
print("He said, \"What's there?\"")
```

```
>>> print("This is printed\nin two lines")
This is printed
in two lines
```

Raw String to ignore escape sequence

- Sometimes we may wish to ignore the escape sequences inside a string.
- To do this we can place r or R in front of the string.
- This will imply that it is a raw string and any escape sequence inside it will be ignored.

```
>>> print("This is \x61 \ngood example")
This is a
good example
>>> print(r"This is \x61 \ngood example")
This is \x61 \ngood example
```

The format() Method for Formatting Strings

- The format() method that is available with the string object is very versatile and powerful in formatting strings.
- Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.
- We can use positional arguments or keyword arguments to specify the order.

```
# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```

```
OUTPUT:-
--- Default Order ---
John, Bill and Sean

--- Positional Order ---
Bill, John and Sean

--- Keyword Order ---
Sean, Bill and John
```

Old style formatting

We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

```
>>> x = 12.3456789
>>> print('The value of x is %3.2f' %x)
The value of x is 12.35
>>> print('The value of x is %3.4f' %x)
The value of x is 12.3457
```

Common Python String Methods

- **There are numerous methods available with the string object.**
- **The format()method that we mentioned above is one of them.**
- **Some of the commonly used methods are lower(), upper(), join(), split(), find(), replace() etc.**

```
>>> "PrOgRaMiZ".lower()
'programiz'

>>> "PrOgRaMiZ".upper()
'PROGRAMIZ'

>>> "This will split all words into a list".split()
['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']

>>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])
'This will join all words into a string'

>>> 'Happy New Year'.find('ew')
7

>>> 'Happy New Year'.replace('Happy','Brilliant')
'Brilliant New Year'
```

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Example

Consider the following example to understand the real use of Python operators.

```
str = "Hello"
str1 = " world"
print(str*3) # prints HelloHelloHello
print(str+str1)# prints Hello world
print(str[4]) # prints o
print(str[2:4]); # prints ll
print('w' in str) # prints false as w is not present in str
print('wo' not in str1) # prints false as wo is present in str1.
print(r'C://python37') # prints C://python37 as it is written
print("The string str : %s"%(str)) # prints The string str : Hello
```

Output:

```
HelloHelloHello
Hello world
o
ll
False
False
C://python37
The string str : Hello
```

Python Formatting operator

- Python allows us to use the format specifiers used in C's printf statement.
- The format specifiers in python are treated in the same way as they are treated in C.
- However, Python provides an additional operator % which is used as an interface between the format specifiers and their values.
- In other words, we can say that it binds the format specifiers to the values.

Consider the following example.

```
1.	Integer = 10;
2.	Float = 1.290
3.	String = "Ayush"
4.	print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am str
ing ... My value is %s"%(Integer,Float,String));
```

Output:

```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Ayush
```

Chapter 9: Python datetime

In this article, you will learn to manipulate date and time in Python with the help of examples.
Python has a module named **datetime** to work with dates and times.
Let's create a few simple programs related to date and time before we dig deeper.

Example 1: Get Current Date and Time

```python
import datetime

datetime_object = datetime.datetime.now()
print(datetime_object)
```

When you run the program, the output will be something like:

```
2018-12-19 09:26:03.478039
```

- Here, we have imported **datetime** module using `import datetime` statement.
- One of the classes defined in the `datetime` module is `datetime` class.
- We then used `now()` method to create a `datetime` object containing the current local date and time.

Example 2: Get Current Date

```python

import datetime

date_object = datetime.date.today()
print(date_object)
```

When you run the program, the output will be something like:

```
2018-12-19
```

In this program, we have used `today()` method defined in the `date` class to get a `date` object containing the current local date.

What's inside datetime?

We can use dir() function to get a list containing all attributes of a module.

```python
import datetime

print(dir(datetime))
```

When you run the program, the output will be:

```

['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', '_divide_and_round', 'date', 'datetime',
'datetime_CAPI', 'time', 'timedelta', 'timezone', 'tzinfo']
```

Commonly used classes in the datetime module are:

- date Class
- time Class
- datetime Class
- timedelta Class

---

datetime.date Class

- You can instantiate date objects from the date class.
- A date object represents a date (year, month and day).

Example 3: Date object to represent a date

```

import datetime

d = datetime.date(2019, 4, 13)
print(d)
```

When you run the program, the output will be:

```
2019-04-13
```

If you are wondering, date() in the above example is a constructor of the date class. The constructor takes three arguments: year, month and day.
The variable a is a date object.

We can only import date class from the datetime module. Here's how:

```

from datetime import date

a = date(2019, 4, 13)
print(a)
```

Example 4: Get current date

You can create a date object containing the current date by using a classmethod named today(). Here's how:

```

from datetime import date

today = date.today()

print("Current date =", today)
```

Example 5: Get date from a timestamp

We can also create date objects from a timestamp. A Unix timestamp is the number of seconds between a particular date and January 1, 1970 at UTC. You can convert a timestamp to date using fromtimestamp() method.

```python

from datetime import date

timestamp = date.fromtimestamp(1326244364)
print("Date =", timestamp)
```

When you run the program, the output will be:

```
Date = 2012-01-11
```

Example 6: Print today's year, month and day

We can get year, month, day, day of the week etc. from the date object easily. Here's how:

```python

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)
```

---

datetime.time

A time object instantiated from the time class represents the local time.

Example 7: Time object to represent time

```python

from datetime import time

# time(hour = 0, minute = 0, second = 0)
a = time()
print("a =", a)

# time(hour, minute and second)
b = time(11, 34, 56)
print("b =", b)

# time(hour, minute and second)
c = time(hour = 11, minute = 34, second = 56)
```

```python
print("c =", c)

# time(hour, minute, second, microsecond)
d = time(11, 34, 56, 234566)
print("d =", d)
```

When you run the program, the output will be:

```
a = 00:00:00
b = 11:34:56
c = 11:34:56
d = 11:34:56.234566
```

Example 8: Print hour, minute, second and microsecond

Once you create a time object, you can easily print its attributes such as hour, minute etc.

```python

from datetime import time

a = time(11, 34, 56)

print("hour =", a.hour)
print("minute =", a.minute)
print("second =", a.second)
print("microsecond =", a.microsecond)
```

When you run the example, the output will be:

```
hour = 11
minute = 34
second = 56
microsecond = 0
```

Notice that we haven't passed microsecond argument. Hence, its default value 0 is printed.

---

datetime.datetime

The datetime module has a class named datetime that can contain information from both **date** and **time** objects.

Example 9: Python datetime object

```python

from datetime import datetime

#datetime(year, month, day)
a = datetime(2018, 11, 28)
```

```python
print(a)

# datetime(year, month, day, hour, minute, second, microsecond)
b = datetime(2017, 11, 28, 23, 55, 59, 342380)
print(b)
```

When you run the program, the output will be:

```
2018-11-28 00:00:00
2017-11-28 23:55:59.342380
```

The first three arguments year, month and day in the datetime() constructor are mandatory.

Example 10: Print year, month, hour, minute and timestamp

```python

from datetime import datetime

a = datetime(2017, 11, 28, 23, 55, 59, 342380)
print("year =", a.year)
print("month =", a.month)
print("hour =", a.hour)
print("minute =", a.minute)
print("timestamp =", a.timestamp())
```

When you run the program, the output will be:

```


year = 2017
month = 11
day = 28
hour = 23
minute = 55
timestamp = 1511913359.34238
```

---

datetime.timedelta

A timedelta object represents the difference between two dates or times.

Example 11: Difference between two dates and times

```python

from datetime import datetime, date

t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)
t3 = t1 - t2
print("t3 =", t3)

```

```
t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9, second = 33)
t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55, second = 13)
t6 = t4 - t5
print("t6 =", t6)

print("type of t3 =", type(t3))
print("type of t6 =", type(t6))
```

When you run the program, the output will be:

```
t3 = 201 days, 0:00:00
t6 = -333 days, 1:14:20
type of t3 = <class 'datetime.timedelta'>
type of t6 = <class 'datetime.timedelta'>
```

Notice, both t3 and t6 are of <class 'datetime.timedelta'> type.

Example 12: Difference between two timedelta objects

```

from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)
t3 = t1 - t2

print("t3 =", t3)
```

When you run the program, the output will be:

```
t3 = 14 days, 13:55:39
```

Here, we have created two timedelta objects t1 and t2, and their difference is printed on the screen.

Example 13: Printing negative timedelta object

```

from datetime import timedelta

t1 = timedelta(seconds = 33)
t2 = timedelta(seconds = 54)
t3 = t1 - t2

print("t3 =", t3)
print("t3 =", abs(t3))
```

When you run the program, the output will be:

```
t3 = -1 day, 23:59:39
t3 = 0:00:21
```

Example 14: Time duration in seconds

You can get the total number of seconds in a timedelta object using total_seconds() method.

```python

from datetime import timedelta

t = timedelta(days = 5, hours = 1, seconds = 33, microseconds = 233423)
print("total seconds =", t.total_seconds())
```

When you run the program, the output will be:

```
total seconds = 435633.233423
```

You can also find sum of two dates and times using + operator. Also, you can multiply and divide a timedelta object by integers and floats.

Python format datetime

The way date and time is represented may be different in different places, organizations etc. It's more common to use mm/dd/yyyy in the US, whereas dd/mm/yyyy is more common in the UK.
Python has strftime() and strptime() methods to handle this.

Python strftime() - datetime object to string

The strftime() method is defined under classes date, datetime and time. The method creates a formatted string from a given date, datetime or time object.

Strftime syntax

Strftime accepts two parameters.

- The first compulsory parameter is the format string
- The second optional parameter is the time to format.
- If the second parameter isn’t provided, it defaults to the current time.

```
time.strftime(format[, t])
```

Common Date Format Strings

Here are a few other common examples of character strings and their equivalent outputs

Format String	**Output**
%m/%d/%Y	09/21/2015
%d/%m/%Y	21/09/2015
%Y-%m-%d	2015-09-21
%I:%M %p	08:58 AM
%b %d %Y	Sep 21 2015
%d %B, %Y	21 September, 2015
%c	Mon, Sep 21 2015 08:58:12 (Locale Appropriate DateTime)
%x	09/21/15 (Locale Appropriate Date)
%X	08:58:12 (Locale Appropriate Time)

Complete Character Code List

The table below shows all the codes that you can use to create the date format string.

Code	Meaning	Example
%a	Weekday as locale’s abbreviated name.	Mon
%A	Weekday as locale’s full name.	Monday
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	1
%d	Day of the month as a zero-padded decimal number.	30
%-d	Day of the month as a decimal number. (Platform specific)	30
%b	Month as locale’s abbreviated name.	Sep
%B	Month as locale’s full name.	September
%m	Month as a zero-padded decimal number.	09
%-m	Month as a decimal number. (Platform specific)	9
%y	Year without century as a zero-padded decimal number.	13

Code	Meaning	Example
%Y	Year with century as a decimal number.	2013
%H	Hour (24-hour clock) as a zero-padded decimal number.	07
%-H	Hour (24-hour clock) as a decimal number. (Platform specific)	7
%I	Hour (12-hour clock) as a zero-padded decimal number.	07
%-I	Hour (12-hour clock) as a decimal number. (Platform specific)	7
%p	Locale’s equivalent of either AM or PM.	AM
%M	Minute as a zero-padded decimal number.	06
%-M	Minute as a decimal number. (Platform specific)	6
%S	Second as a zero-padded decimal number.	05
%-S	Second as a decimal number. (Platform specific)	5
%f	Microsecond as a decimal number, zero-padded on the left.	000000
%z	UTC offset in the form +HHMM or -HHMM (empty string if the the object is naive).	
%Z	Time zone name (empty string if the object is naive).	
%j	Day of the year as a zero-padded decimal number.	273
%-j	Day of the year as a decimal number. (Platform specific)	273
%U	Week number of the year (Sunday as the first day of the week) as a zero padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0.	39
%W	Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0.	39
%c	Locale’s appropriate date and time representation.	Mon Sep 30 07:06:05 2013
%x	Locale’s appropriate date representation.	09/30/13
%X	Locale’s appropriate time representation.	07:06:05
%	A literal '%' character.	%

Example 15: Format date using strftime()

```python

from datetime import datetime

# current date and time
now = datetime.now()

t = now.strftime("%H:%M:%S")
print("time:", t)

s1 = now.strftime("%m/%d/%Y, %H:%M:%S")
# mm/dd/YY H:M:S format
print("s1:", s1)

s2 = now.strftime("%d/%m/%Y, %H:%M:%S")
# dd/mm/YY H:M:S format
print("s2:", s2)
```

When you run the program, the output will be something like:

```
time: 04:34:52
s1: 12/26/2018, 04:34:52
s2: 26/12/2018, 04:34:52
```

Python strptime() - string to datetime

The strptime() method creates a datetime object from a given string (representing date and time).

Example 16: strptime()

```python
from datetime import datetime

date_string = "21 June, 2018"
print("date_string =", date_string)

date_object = datetime.strptime(date_string, "%d %B, %Y")
print("date_object =", date_object)
```

When you run the program, the output will be:

```
date_string = 21 June, 2018
date_object = 2018-06-21 00:00:00
```

The strptime() method takes two arguments:

1. a string representing date and time
2. format code equivalent to the first argument

By the way, %d, %B and %Y format codes are used for day, month(full name) and year respectively.

Python sleep time

- The sleep() method of time module is used to stop the execution of the script for a given amount of time.
- The output will be delayed for the number of seconds given as float.

Consider the following example.

Example

```python
import time
for i in range(0,5):
    print(i)
    #Each element will be printed after 1 second
    time.sleep(1)
```

Output:

```
0
1
2
3
4
```

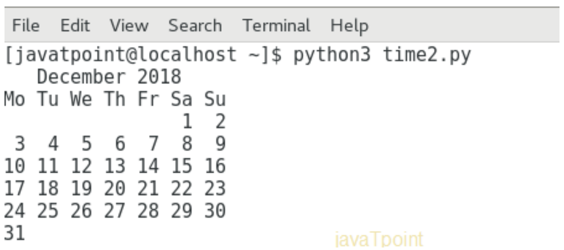
The calendar module

Python provides a calendar object that contains various methods to work with the calendars.
Consider the following example to print the Calendar of the last month of 2018.

Example

```python
import calendar;
cal = calendar.month(2018,12)
#printing the calendar of December 2018
print(cal)
```

Output:



```
[javatpoint@localhost ~]$ python3 time2.py
   December 2018
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

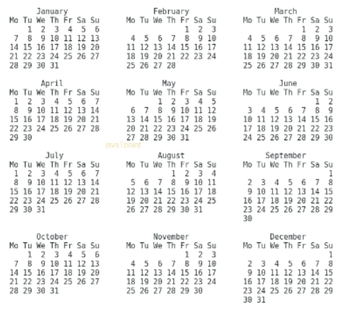
Printing the calendar of whole year

The prcal() method of calendar module is used to print the calendar of the whole year. The year of which the calendar is to be printed must be passed into this method.

Example

```python
import calendar

#printing the calendar of the year 2019
calendar.prcal(2019)

```

Output:



```
      January                   February                   March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                   1  2  3                   1  2  3
 7  8  9 10 11 12 13       4  5  6  7  8  9 10       4  5  6  7  8  9 10
14 15 16 17 18 19 20      11 12 13 14 15 16 17      11 12 13 14 15 16 17
21 22 23 24 25 26 27      18 19 20 21 22 23 24      18 19 20 21 22 23 24
28 29 30 31               25 26 27 28               25 26 27 28 29 30 31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7             1  2  3  4  5                      1  2
 8  9 10 11 12 13 14       6  7  8  9 10 11 12       3  4  5  6  7  8  9
15 16 17 18 19 20 21      13 14 15 16 17 18 19      10 11 12 13 14 15 16
22 23 24 25 26 27 28      20 21 22 23 24 25 26      17 18 19 20 21 22 23
29 30                     27 28 29 30 31            24 25 26 27 28 29 30

        July                     August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7                1  2  3  4                         1
 8  9 10 11 12 13 14       5  6  7  8  9 10 11       2  3  4  5  6  7  8
15 16 17 18 19 20 21      12 13 14 15 16 17 18       9 10 11 12 13 14 15
22 23 24 25 26 27 28      19 20 21 22 23 24 25      16 17 18 19 20 21 22
29 30 31                  26 27 28 29 30 31         23 24 25 26 27 28 29
                                                    30

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                   1  2  3                         1
 7  8  9 10 11 12 13       4  5  6  7  8  9 10       2  3  4  5  6  7  8
14 15 16 17 18 19 20      11 12 13 14 15 16 17       9 10 11 12 13 14 15
21 22 23 24 25 26 27      18 19 20 21 22 23 24      16 17 18 19 20 21 22
28 29 30 31               25 26 27 28 29 30         23 24 25 26 27 28 29
                                                    30 31
```

Chapter 10: Python List [ ]

- List in python is implemented to store the sequence of various type of data.
- A list can be defined as a collection of values or items of different types.
- The items in the list are separated with the comma (,) and enclosed with the square brackets [].

A list can be defined as follows.

1. **L1 = ["John", 102, "USA"]**
2. **L2 = [1, 2, 3, 4, 5, 6]**
3. **L3 = [1, "Ryan"]**

List indexing and splitting

- The indexing are processed in the same way as it happens with the strings.
- The elements of the list can be accessed by using the slice operator [:].

The index starts from 0 and goes to length - 1.

The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

Consider the following example.

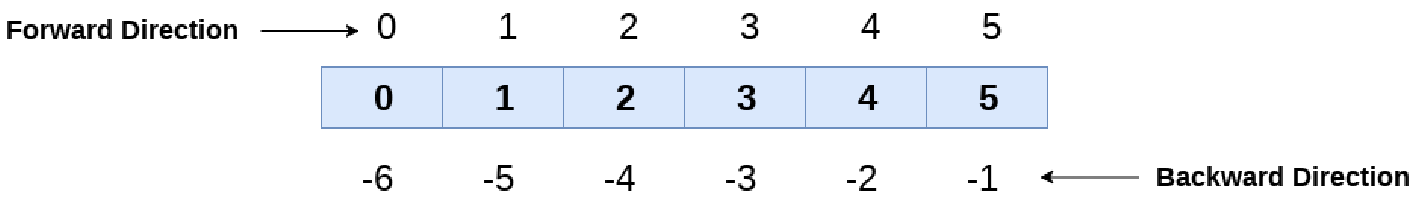
List = [0, 1, 2, 3, 4, 5]

0	**1**	**2**	**3**	**4**	**5**

```
List[0] = 0            List[0:] = [0,1,2,3,4,5]

List[1] = 1            List[:] = [0,1,2,3,4,5]

List[2] = 2            List[2:4] = [2, 3]

List[3] = 3            List[1:3]  = [1, 2]

List[4] = 4            List[:4] = [0, 1, 2, 3]

List[5] = 5
```

- Unlike other languages, python provides us the flexibility to use the negative indexing also.
- The negative indices are counted from the right.
- The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.

List = [0, 1, 2, 3, 4, 5]



Updating List values

Lists are the most versatile data structures in python since they are **mutable** and their values can be updated by using the slice and assignment operator.

Consider the following example to update the values inside the list.

```
List = [1, 2, 3, 4, 5, 6]
print(List)
List[2] = 10;
print(List)
List[1:3] = [89, 78]
print(List)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
```

- The list elements can also be deleted by using the **del** keyword.
- Python provides us the remove() method if we do not know which element to be deleted from the list.

Consider the following example to delete the list elements.

```
List = [0,1,2,3,4]
print(List)
del List[0]
print(List)
del List[3]
print(List)
```

Output:

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3]
```

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings can be iterated as follows.

```
List = ["John", "David", "James", "Jonathan"]
for i in List:
print(i);
```

Output:

```
John
David
James
Jonathan
```

Python List Operations

The concatenation (+) and repetition (*) operator work in the same way as they were working with the strings.
Lets see how the list responds to various operators.

Consider a List
l1 = [1, 2, 3, 4],
l2 = [5, 6, 7, 8]

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) **Output** 1 2 3 4
Length	It is used to get the length of the list	len(l1) = 4

Adding elements to the list

- Python provides append() function by using which we can add an element to the list.
- However, the append() method can only add the value to the end of the list.

Consider the following example in which, we are taking the elements of the list from the user and printing the list on the console.

```
l =[];
n = int(input("Enter the number of elements in the list"));
for i in range(0,n):
l.append(input("Enter the item?"));
print("printing the list items....");
for i in l:
print(i, end = "  ");
```

Output:

```
Enter the number of elements in the list 5
Enter the item?1
Enter the item?2
Enter the item?3
Enter the item?4
Enter the item?5
printing the list items....
1  2  3  4  5
```

Removing elements from the list

```
List = [0,1,2,3,4]
print("printing original list: ");
for i in List:
print(i,end=" ")
List.remove(0)
print("\nprinting the list after the removal of first element...")
for i in List:
print(i,end=" ")
```

Output:

```
printing original list:
0 1 2 3 4
printing the list after the removal of first element...
1 2 3 4
```

Python List Built-in functions

Python provides the following built-in functions which can be used with the lists.

SN	Function	Description
1	len(list)	It is used to calculate the length of the list.
2	max(list)	It returns the maximum element of the list.
3	min(list)	It returns the minimum element of the list.
4	list(seq)	It converts any sequence to the list.

Examples:-

```
L1=[1,2,13,4,5]
print (len(L1))
print(max(L1))
print(min(L1))
a=12,13,44  # any sequence
print(list(a))
```

```
Output
5
13
[12, 13, 44]
```

Python List built-in methods

Python List sort() Method

- Python **sort()** method sorts the list elements.
- It also sorts the items into descending and ascending order.
- It takes an optional parameter 'reverse' which sorts the list into descending order.
- By default, list sorts the elements into ascending order.

It is a simple example which sorts two lists in ascending order. See the example below.

```python
apple = ['a', 'p', 'p', 'l', 'e'] # Char list
even = [6,8,2,4] # int list
print(apple)
print(even)
apple.sort()
even.sort()
# Displaying result
print("\nAfter Sorting:\n",apple)
print(even)
```

Output:

```
['a', 'p', 'p', 'l', 'e']
[6, 8, 2, 4]
After Sorting:
 ['a', 'e', 'l', 'p', 'p']
[2, 4, 6, 8]
```

Python List sort() Method Example Descending

This example sorts the list into descending order.

```python
# Python list sort() Method
# Creating a list
even = [6,8,2,4] # int list
# Calling Method
#apple.sort()
even.sort(reverse=True)    # sort in reverse order
# Displaying result
print(even)
```

Output:

```
[8, 6, 4, 2]
```

Python List reverse() Method

- Python **reverse()** method reverses elements of the list.
- If the list is empty, it simply returns an empty list.
- After reversing the last index value of the list will be present at 0 index.

Let's first see a simple example to reverse the list.
It prints all the elements in reverse order.

```
# Python list reverse() Method
# Creating a list
apple = ['a','p','p','l','e']
# Method calling
apple.reverse() # Reverse elements of the list
# Displaying result
print(apple)
```

Output:

```
['e', 'l', 'p', 'p', 'a']
```

Python List pop() Method

- Python **pop()** element removes an element present at specified index from the list.
- It returns the popped element.

Let's first see a simple example of popping element from the list.
Element present at the index 2 is popped, see the example below.

```
# Python list pop() Method
# Creating a list
list = ['1','2','3']
for l in list:  # Iterating list
    print(l)
list.pop(2)
print("After poping:")
for l in list:  # Iterating list
    print(l)
```

Output:

```
1
2
3
After poping:
1
2
```

Python List insert(i,x) Method

- Python **insert()** method inserts the element at the specified index in the list.
- The first argument is the index of the element before which to insert the element.

Let's see an example to insert an element at 3 index of the list.

```
# Python list insert() Method
# Creating a list
list = ['1','2','3']
for l in list:  # Iterating list
    print(l)
list.insert(3,4)
print("After extending:")
for l in list:  # Iterating list
    print(l)
```

Output:

```
1
2
3
After extending:
1
2
3
4
```

Python List clear() Method

- Python **clear()** method removes all the elements from the list.
- It clear the list completely and returns nothing.

Let's see a simple example in which clear() method is used to clear a list.

```
# Python list clear() Method
# Creating a list
list = ['1','2','3']
for l in list:  # Iterating list
    print(l)
list.clear()
print("After clearing:")
for l in list:  # Iterating list
    print(l)
```

Output:

```
1
2
3
After clearing:
```

Python List copy() Method

Python **copy()** method copies the list and returns the copied list.

A Simple example which copy a list to another and makes an new list.

```
# Python list copy() Method
# Creating a list
evenlist = [6,8,2,4] # int list
copylist = []
# Calling Method
copylist = evenlist.copy()
# Displaying result
print("Original list:",evenlist)
print("Copy list:",copylist)
```

Output:

```
Original list: [6, 8, 2, 4]
Copy list: [6, 8, 2, 4]
```

Python List count() Method

- Python **count()** method returns the number of times element appears in the list.
- If the element is not present in the list, it returns 0.

It is a simple example to understand how a count method works.

```
# Python list count() Method
# Creating a list
apple = ['a','p','p','l','e']
# Method calling
count = apple.count('p')
# Displaying result
print("count of p :",count)
```

Output:

```
count of p : 2
```

Python List index() Method

- Python **index()** method returns index of the passed element.
- This method takes an argument and returns index of it. If the element is not present, it raises a ValueError.
- If list contains duplicate elements, it returns index of first occurred element.

Let's first see a simple example to get index of an element using index() method.

```
# Python list index() Method
# Creating a list
apple = ['a','p','p','l','e']
# Method calling
index = apple.index('p')
# Displaying result
print("Index of p :",index)
```

Output: **Index of p : 1**

Chapter 11 :Python Tuples ( )

- Python Tuple is used to store the sequence of **immutable** python objects.
- Tuple is similar to lists since the value of the items stored in the list can be changed
- whereas the tuple is immutable and the value of the items stored in the tuple cannot be changed.

A tuple can be written as the collection of comma-separated values enclosed with the small brackets.

A tuple can be defined as follows.

```
T1 = (101, "Ayush", 22)
T2 = ("Apple", "Banana", "Orange")
```

Example

```
tuple1 = (10, 20, 30, 40, 50, 60)
print(tuple1)
count = 0
for i in tuple1:
    print(i));
```

Output:

```
(10, 20, 30, 40, 50, 60)
10
20
30
40
50
60
```

Example 2

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
    print(i));
```

Output:

```
Enter the tuple elements ...12345
('1', '2', '3', '4', '5')
 1
 2
 3
 4
 5
```

- **However, if we try to reassign the items of a tuple, we would get an error as the tuple object doesn't support the item assignment.**
- **An empty tuple can be written as follows.**

```
T3 = ()
```

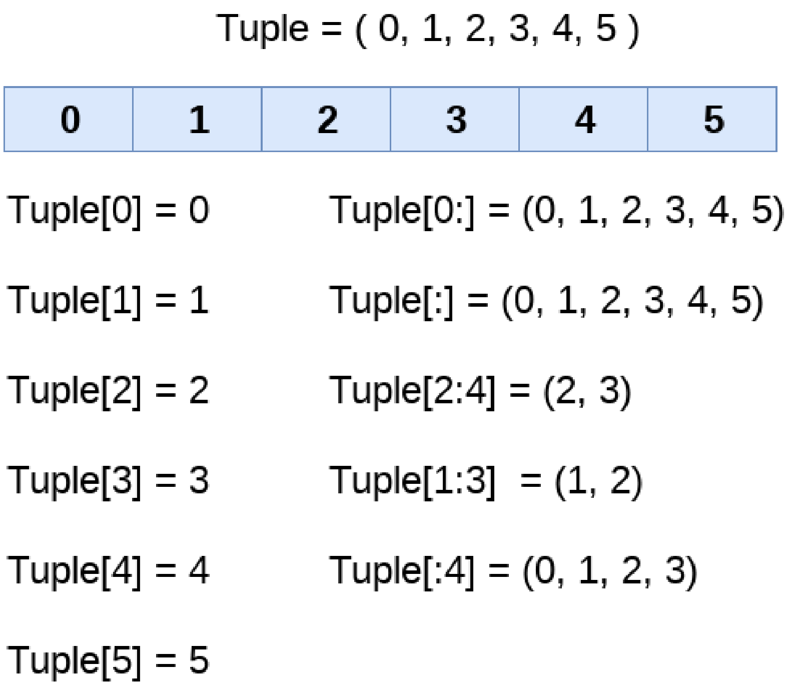
- **The tuple having a single value must include a comma as given below.**

```
T4 = (90,)
```

- A tuple is indexed in the same way as the lists.
- The items in the tuple can be accessed by using their specific index value.

Tuple indexing and splitting

The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.
The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.
Consider the following image to understand the indexing and slicing in detail.



Tuple = (0, 1, 2, 3, 4, 5)

0	1	2	3	4	5

Tuple[0] = 0	Tuple[0:] = (0, 1, 2, 3, 4, 5)
Tuple[1] = 1	Tuple[:] = (0, 1, 2, 3, 4, 5)
Tuple[2] = 2	Tuple[2:4] = (2, 3)
Tuple[3] = 3	Tuple[1:3] = (1, 2)
Tuple[4] = 4	Tuple[:4] = (0, 1, 2, 3)
Tuple[5] = 5	

- Unlike lists, the tuple items can not be deleted by using the del keyword as tuples are immutable.
- To delete an entire tuple, we can use the del keyword with the tuple name.

Consider the following example.

```
tuple1 = (1, 2, 3, 4, 5, 6)
print(tuple1)
del tuple1[0]
print(tuple1)
del tuple1
print(tuple1)
```

Output:

```
(1, 2, 3, 4, 5, 6)
Traceback (most recent call last):
  File "tuple.py", line 4, in <module>
    print(tuple1)
NameError: name 'tuple1' is not defined
```

Like lists, the tuple elements can be accessed in both the directions. The right most element (last) of the tuple can be accessed by using the index -1. The elements from left to right are traversed using the negative indexing.

Consider the following example.

```
1.	tuple1 = (1, 2, 3, 4, 5)
2.	print(tuple1[-1])
3.	print(tuple1[-4])
```

Output:

```
5
2
```

Basic Tuple operations

The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.

t = (1, 2, 3, 4, 5)
t1 = (6, 7, 8, 9).

Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false.	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) **Output** 1 2 3 4 5
Length	It is used to get the length of the tuple.	len(T1) = 5

Python Tuple inbuilt functions

SN	Function	Description
1	len(tuple)	It calculates the length of the tuple.
2	max(tuple)	It returns the maximum element of the tuple.
3	min(tuple)	It returns the minimum element of the tuple.
4	tuple(seq)	It converts the specified sequence to the tuple.

Where use tuple

Using tuple instead of list is used in the following scenario.

1. Using tuple instead of list gives us a clear idea that tuple data is constant and must not be changed.
2. Tuple can simulate dictionary without keys.

   Consider the following nested structure which can be used as a dictionary.
   [(101, "John", 22), (102, "Mike", 28), (103, "Dustin", 30)]
   Tuple can be used as the key inside dictionary due to its immutable nature.

List VS Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

Nesting List and tuple

We can store list inside tuple or tuple inside the list up to any number of level.

Lets see an example of how can we store the tuple inside the list.

```
Employees = [(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 34)]
print("----Printing list----");
for i in Employees:
   print(i)
Employees[0] = (110, "David",22)
print();
print("----Printing list after modification----");
for i in Employees:
   print(i)
```

Output:

```
----Printing list----
(101, 'Ayush', 22)
(102, 'john', 29)
(103, 'james', 45)
(104, 'Ben', 34)

----Printing list after modification--
--
(110, 'David', 22)
(102, 'john', 29)
(103, 'james', 45)
(104, 'Ben', 34)
```

Chapter 12: Python Dictionary { }

- Dictionary is used to implement the key-value pair in python.
- The dictionary is the data type in python which can simulate the real-life data arrangement where some specific value exists for some particular key.
- In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any python object whereas the **keys are the immutable python object**, i.e., Numbers, string or tuple.

Creating the dictionary

- The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:).
- The collections of the key-value pairs are enclosed within the curly braces {}.

The syntax to define the dictionary is given below.

```
Dict = {"Name": "Ayush","Age": 22}
```

In the above dictionary **Dict**, The keys **Name**, and **Age** are the string that is an immutable object.

Let's see an example to create a dictionary and printing its content.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Output

```
<class 'dict'>
printing Employee data ....
{'Age': 29, 'salary': 25000, 'Name': 'John', 'Company': 'GOOGLE'}
```

Accessing the dictionary values

We have discussed how the data can be accessed in the list and tuple by using the indexing. However, the values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.
The dictionary values can be accessed in the following way.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print("Names" +Employee["Name"])
print("Age : ",Employee["Age"])
print("Salary :",Employee["salary"])
print("Company :"+Employee["Company"])
```

Output:

```
<class 'dict'>
printing Employee data ....
Name : John
Age : 29
Salary : 25000
Company : GOOGLE
```

Updating dictionary values

The dictionary is a mutable data type, and its values can be updated by using the specific keys. Let's see an example to update the dictionary values.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Enter the details of the new employee....");
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

Output:

```
<class 'dict'>
printing Employee data ....
{'Name': 'John', 'salary': 25000, 'Company': 'GOOGLE', 'Age': 29}
Enter the details of the new employee....
Name: David
Age: 19
Salary: 8900
Company:JTP
printing the new data
{'Name': 'David', 'salary': 8900, 'Company': 'JTP', 'Age': 19}
```

Deleting elements using del keyword

The items of the dictionary can be deleted by using the del keyword as given below.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Deleting some of the employee data")
del Employee["Name"]
del Employee["Company"]
print("printing the modified information ")
print(Employee)
print("Deleting the dictionary: Employee");
del Employee
print("Lets try to print it again ");
print(Employee)
```

Output:
```
<class 'dict'>
printing Employee data ....
{'Age': 29, 'Company': 'GOOGLE', 'Name': 'John', 'salary': 25000}
Deleting some of the employee data
printing the modified information
{'Age': 29, 'salary': 25000}
Deleting the dictionary: Employee
Lets try to print it again
Traceback (most recent call last):
  File "list.py", line 13, in <module>
    print(Employee)
NameError: name 'Employee' is not defined
```

Iterating Dictionary

A dictionary can be iterated using the for loop as given below.

Example 1

# for loop to print all the keys of a dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
    print(x);
```

Output:
```
Name
Company
salary
Age
```

Example 2

#for loop to print all the values of the dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
    print(Employee[x]);
```

Output:
```
29
GOOGLE
John
25000
```

Example 3

#for loop to print the values of the dictionary by using values() method.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee.values():
    print(x);
```

Output:
```
GOOGLE
25000
John
29
```

Example 4

#for loop to print the items of the dictionary by using items() method.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee.items():
    print(x);
```

Output:

```
('Name', 'John')
('Age', 29)
('salary', 25000)
('Company', 'GOOGLE')
```

Properties of Dictionary keys

In the dictionary, we can not store multiple values for the same keys. If we pass more than one values for a single key, then the value which is last assigned is considered as the value of the key.

Consider the following example.

```
Employee = {"Name": "John", "Age": 29, "Salary":25000,"Company":"GOOGLE","Name":"Johnn"}

for x,y in Employee.items():
    print(x,y)
```

```
Output:
Salary 25000
Company GOOGLE
Name Johnn
Age 29
```

- In python, the key cannot be any mutable object. We can use numbers, strings, or tuple as the key
- we can not use any mutable object like the list as the key in the dictionary.

Consider the following example.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE",
[100,201,301]:"Department ID"}
for x,y in Employee.items():
    print(x,y)
```

```
Output:
Traceback (most recent call last):
  File "list.py", line 1, in
    Employee = {"Name": "John", "Age": 29,
"salary":25000,"Company":"GOOGLE",[100,201,301]:"Department ID"}
TypeError: unhashable type: 'list'
```

Built-in Dictionary functions

The built-in python dictionary methods along with the description are given below.

SN	Function	Description
1	len(dict)	It is used to calculate the length of the dictionary.
2	str(dict)	It converts the dictionary into the printable string representation.
3	type(variable)	It is used to print the type of the passed variable.

<u>Built-in Dictionary Methods</u>

Python dictionary len() Method

- Python dictionary method **len()** gives the total length of the dictionary.
- This would be equal to the number of items in the dictionary.

Example

The following example shows the usage of len() method.

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Length :" len (dict))
```

When we run above program, it produces following result –

```
Length : 2
```

Python dictionary fromkeys() Method

Python dictionary method **fromkeys()** creates a new dictionary with keys from *seq* and *values* set to value.

Example

The following example shows the usage of fromkeys() method.

```
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print (dict)
dict = dict.fromkeys(seq, 10)
print (dict)
```

When we run above program, it produces following result –

```
New Dictionary : {'age': None, 'name': None, 'sex': None}
New Dictionary : {'age': 10, 'name': 10, 'sex': 10}
```

Python Dictionary clear() Method

The clear() method removes all items from the dictionary.

Examples:

```
d = {1: "geeks", 2: "for"}
      d.clear()
Output : d = {}
```

Python Dictionary copy()

They copy() method returns a shallow copy of the dictionary.

```
original = {1:'one', 2:'two'}
new = original.copy()
print('Orignal: ', original)
print('New: ', new)
```

Run

When you run the program, the output will be:

```
Orignal:  {1: 'one', 2: 'two'}
New:  {1: 'one', 2: 'two'}
```

Difference in Using copy() method, and = Operator to Copy Dictionaries

When copy() method is used, a new dictionary is created which is filled with a copy of the references from the original dictionary.
When = operator is used, a new reference to the original dictionary is created.

Example 2: Using = Operator to Copy Dictionaries

```
original = {1:'one', 2:'two'}
new = original
# removing all elements from the list
new.clear()
print('new: ', new)
print('original: ', original)
```

Run

When you run the program, the output will be:

```
new:  {}
original:  {}
```

Here, when the *new* dictionary is cleared, the *original* dictionary is also cleared.

Example 3: Using copy() to Copy Dictionaries

```
original = {1:'one', 2:'two'}
new = original.copy()
# removing all elements from the list
new.clear()
print('new: ', new)
print('original: ', original)
```

Run

When you run the program, the output will be:

```
new:  {}
original:  {1: 'one', 2: 'two'}
```

Here, when the *new* dictionary is cleared, the *original* dictionary remains unchanged.

Python Dictionary items() Method

- Python item() method returns a new view of the dictionary.
- This view is collection of key value tuples.
- This method does not take any parameter.
- Returns empty view if the dictionary is empty.

Let's see example of items() method to understand it's functionality.

This is a simple example which returns all the items present in the dictionary.

```
# Creating a dictionary
student = {'name':'rohan', 'course':'B.Tech', 'email':'rohan@abc.com'}
# Calling function
items = student.items()
# Displaying result
print(items)
```

Output:

```
dict_items([('name', 'rohan'), ('course', 'B.Tech'), ('email', 'rohan@abc.com')])
```

Python Dictionary keys() Method

- Python keys() method is used to fetch all the keys from the dictionary.
- It returns a list of keys and an empty list if the dictionary is empty.
- This method does not take any parameter.

Let's first see a simple example to fetch keys from the dictionary.

Python Dictionary keys() Method Example1

```
# Creating a dictionary
product = {'name':'laptop','brand':'hp','price':80000}
# Calling method
p = product.keys()
print(p)
```

Output:

```
dict_keys(['name', 'brand', 'price'])
```

Python Dictionary keys() Method Example 2

```
# Creating a dictionary
product = {'name':'laptop','brand':'hp','price':80000}
# Iterating using key and value
for p in product.keys():
    if p == 'price' and product[p] > 50000:
        print("product price is too high",)
```

Output:

```
product price is too high
```

Python Dictionary keys() Method Example 3

We can use this method into our python program. Here, we are using it into a program to check our inventory status.

```
# Creating a dictionary
inventory = {'apples': 25, 'bananas': 220, 'oranges': 525, 'pears': 217}
# Calling method
for akey in inventory.keys():
    if akey == 'bananas' and inventory[akey] > 200:
        print("We have sufficient inventory for the ", akey)
```

Output:

```
We have sufficient inventory for the bananas
```

Python Dictionary setdefault() Method

- Python setdefault() method is used to set default value to the key.
- It returns value, if the key is present.
- Otherwise it insert key with the default value.
- Default value for the key is None.

Python Dictionary setdefault() Method Example

If key is not present but default value is set, it returns default value. See an example.

```
# Creating a dictionary
coursefee = {'B,Tech': 400000, 'BA':2500, 'B.COM':50000}
# Calling function
p = coursefee.setdefault('BCA',100000) # Returns it's value
# Displaying result
print("default",p)
print(coursefee)
```

Output:

```
default 100000
{'B,Tech': 400000, 'BA': 2500, 'B.COM': 50000, 'BCA': 100000}
```

Python Dictionary update() Method

- Python update() method updates the dictionary with the key and value pairs.
- It inserts key/value if it is not present.
- It updates key/value if it is already present in the dictionary.
- It also allows an iterable of key/value pairs to update the dictionary. like: update(a=10,b=20) etc.

Python Dictionary update() Method Example 1

It is a simple example to update the dictionary by passing key/value pair. This method updates the dictionary. See an example below.

```
# Creating a dictionary
einventory = {'Fan': 200, 'Bulb':150, 'Led':1000}
print("Inventory:",einventory)
# Calling Method
```

```
einventory.update({'cooler':50})
print("Updated inventory:",einventory)
```

Output:

```
Inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000}
Updated inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50}
```

Python Dictionary update() Method Example 2

If element (key/value) pair is already presents in the dictionary, it will overwrite it. See an example below.

```
# Creating a dictionary
einventory = {'Fan': 200, 'Bulb':150, 'Led':1000,'cooler':50}
print("Inventory:",einventory)
# Calling Method
einventory.update({'cooler':50})
print("Updated inventory:",einventory)
einventory.update({'cooler':150})
print("Updated inventory:",einventory)
```

Output:

```
Inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50}
Updated inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50}
Updated inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 150}
```

Python Dictionary update() Method Example 3

The update() method also allows iterable key/value pairs as parameter. See, the example below two values are passed to the dictionary and it is updated.

```
# Python dictionary update() Method
# Creating a dictionary
einventory = {'Fan': 200, 'Bulb':150, 'Led':1000}
print("Inventory:",einventory)
# Calling Method
einventory.update(cooler=50,switches=1000)
print("Updated inventory:",einventory)
```

Output:

```
Inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000}
Updated inventory: {'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50, 'switches': 1000}
```

Python Dictionary values() Method

- Python values() method is used to collect all the values from a dictionary.
- It does not take any parameter and returns a dictionary of values.
- It returns an empty dictionary if the dictionary has no value.

Python Dictionary values() Method Example 1

It is a simple example which returns all the values from the dictionary. An example is given below.

```
# Creating a dictionary
einventory = {'Fan': 200, 'Bulb':150, 'Led':1000}
# Calling function
stock = einventory.values()
```

```
# Displaying result
print("Stock available",einventory)
```
Output:
```
Stock available {'Fan': 200, 'Bulb': 150, 'Led': 1000}
```

Python Dictionary values() Method Example 3

This example uses various methods to get information about dictionary such as length, keys etc.

```
# Creating a dictionary
einventory = {'Fan': 200, 'Bulb':150, 'Led':1000}
# Applying functions
length = len(einventory)
print("Total number of values:",length)
keys = einventory.keys()
print("All the Keys:",keys)
item = einventory.items()
print("Items:",einventory)
p = einventory.popitem()
print("Deleted items:",p)
stock = einventory.values()
print("Stock available",einventory)
```

Output:
```
Total number of values: 3
All the Keys: dict_keys(['Fan', 'Bulb', 'Led'])
Items: {'Fan': 200, 'Bulb': 150, 'Led': 1000}
Deleted items: ('Led', 1000)
Stock available {'Fan': 200, 'Bulb': 150}
```

Python Dictionary popitem() Method

- Python popitem() method removes an element from the dictionary.
- It removes arbitrary element and return its value.
- If the dictionary is empty, it returns an error KeyError.

Python Dictionary popitem() Method Example 1

Let's first see a simple example to remove an element using popitem() method.

```
# Creating a dictionary
inventory = {'shirts': 25, 'paints': 220, 'shocks': 525, 'tshirts': 217}
# Displaying result
print(inventory)
p = inventory.popitem()
print("Removed",p)
print(inventory)
```

Output:

```
{'shirts': 25, 'paints': 220, 'shocks': 525, 'tshirts': 217}
Removed ('tshirts', 217)
{'shirts': 25, 'paints': 220, 'shocks': 525}
```

Python Dictionary popitem() Method Example 2

In this example, we are removing and updating the dictionary to understand the functioning of this method.

```
# Creating a dictionary
inventory = {'shirts': 25, 'paints': 220, 'shocks': 525, 'tshirts': 217}
# Displaying result
print(inventory)
p = inventory.popitem()
print("Removed",p)
print(inventory)
inventory.update({'pajama':117})
print(inventory)
```

Output:

```
{'shirts': 25, 'paints': 220, 'shocks': 525, 'tshirts': 217}
Removed ('tshirts', 217)
{'shirts': 25, 'paints': 220, 'shocks': 525}
{'shirts': 25, 'paints': 220, 'shocks': 525, 'pajama': 117}
```

Python Dictionary pop() Method

- Python pop() method removes an element from the dictionary.
- It removes the element which is associated to the specified key.
- If specified key is present in the dictionary, it remove and return its value.
- If the specified key is not present, it throws an error KeyError.

Python Dictionary pop () Method Example 1

A simple example to pop an element from the dictionary. It returns popped value

```
# Creating a dictionary
inventory = {'shirts': 25, 'paints': 220, 'shock': 525, 'tshirts': 217}
# Calling method
element = inventory. Pop('shirts')
# Displaying result
print(element)
print(Inventory)
```

Output:

```
25
{'tshirts': 217, 'paints': 220, 'shock': 525}
```

Chapter 13: Python Set

- The set in python can be defined as the unordered collection of various items enclosed within the curly braces.
- The elements of the set can not be duplicate.
- The elements of the python set must be immutable.
- Unlike other collections in python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index.
- However, we can print them all together or we can get the list of elements by looping through the set.

Creating a set

- The set can be created by enclosing the comma separated items with the curly braces.
- Python also provides the set method which can be used to create the set by the passed sequence.

Example 1: using curly braces

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}

print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Output:

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}
<class 'set'>
looping through the set elements ...
Friday
Tuesday
Monday
Saturday
Thursday
Sunday
Wednesday
```

Example 2: using set() method

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
print(i)
```

Output:

```
{'Friday', 'Wednesday', 'Thursday', 'Saturday', 'Monday', 'Tuesday', 'Sunday'}
<class 'set'>
looping through the set elements ...
```

```
Friday
Wednesday
Thursday
Saturday
Monday
Tuesday
Sunday
```

Python Set operations

Adding items to the set

Python provides the add() method which can be used to add some particular item to the set. Consider the following example.

Example:

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nAdding other months to the set...");
Months.add("July");
Months.add("August");
print("\nPrinting the modified set...");
print(Months)
print("\nlooping through the set elements ... ")
for i in Months:
    print(i)
```

Output:

```
printing the original set ...
{'February', 'May', 'April', 'March', 'June', 'January'}
Adding other months to the set...
Printing the modified set...
{'February', 'July', 'May', 'April', 'March', 'August', 'June', 'January'}
looping through the set elements ...
February
July
May
April
March
August
June
January
```

To add more than one item in the set, Python provides the update() method.

Consider the following example.

Example

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nupdating the original set ... ")
Months.update(["July","August","September","October"]);
print("\nprinting the modified set ... ")
print(Months);
```

Output:

```
printing the original set ...
{'January', 'February', 'April', 'May', 'June', 'March'}
updating the original set ...
printing the modified set ...
{'January', 'February', 'April', 'August', 'October', 'May', 'June', 'July', 'September',
'March'}
```

Removing items from the set

Python provides discard() method which can be used to remove the items from the set.

Consider the following example.

Example

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving some months from the set...");
Months.discard("January");
Months.discard("May");
print("\nPrinting the modified set...");
print(Months)
print("\nlooping through the set elements ... ")
for i in Months:
    print(i)
```

Output:

```
printing the original set ...
{'February', 'January', 'March', 'April', 'June', 'May'}
Removing some months from the set...
Printing the modified set...
{'February', 'March', 'April', 'June'}
looping through the set elements ...
February
March
April
June
```

Python also provide the remove() method to remove the items from the set.

Consider the following example to remove the items using remove() method.

Example

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving some months from the set...");
Months.remove("January");
Months.remove("May");
print("\nPrinting the modified set...");
print(Months)
```

Output:

```
printing the original set ...
{'February', 'June', 'April', 'May', 'January', 'March'}
Removing some months from the set...
Printing the modified set...
{'February', 'June', 'April', 'March'}
```

- **We can also use the pop() method to remove the item.**
- **However, this method will always remove the last item**.

Consider the following example to remove the last item from the set.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving some months from the set...");
Months.pop();
Months.pop();
print("\nPrinting the modified set...");
print(Months)
```

Output:

```
printing the original set ...
{'June', 'January', 'May', 'April', 'February', 'March'}

Removing some months from the set...
Printing the modified set...
{'May', 'April', 'February', 'March'}
```

Python provides the clear() method to remove all the items from the set.

Consider the following example**.**

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving all the items from the set...");
```

```
5.    Months.clear()
6.    print("\nPrinting the modified set...")
7.    print(Months)
```
Output:

```
printing the original set ...
{'January', 'May', 'June', 'April', 'March', 'February'}
Removing all the items from the set...
Printing the modified set...
set()
```

Difference between discard() and remove()

Despite the fact that discard() and remove() method both perform the same task, There is one main difference between discard() and remove().

- **If the key to be deleted from the set using discard() doesn't exist in the set, the python will not give the error. The program maintains its control flow.**
- **On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the python will give the error.**

Consider the following example.

Example

```
1. Months = set(["January","February", "March", "April", "May", "June"])
2. print("\nprinting the original set ... ")
3. print(Months)
4. print("\nRemoving items through discard() method...");
5. Months.discard("Feb"); #will not give an error the key feb is not available in the set  print(
   "\nprinting the modified set...")
6. print(Months)
7. print("\nRemoving items through remove() method...");
8. Months.remove("Jan") #will give an error as the key jan is not available in the set.
9. print("\nPrinting the modified set...")
10. print(Months)
```
Output:

```
printing the original set ...
{'March', 'January', 'April', 'June', 'February', 'May'}
Removing items through discard() method...
printing the modified set...
{'March', 'January', 'April', 'June', 'February', 'May'}
Removing items through remove() method...
Traceback (most recent call last):
  File "set.py", line 9, in
    Months.remove("Jan")
KeyError: 'Jan'
```

Union of two Sets

- The union of two sets are calculated by using the or (|) operator.
- The union of the two sets contains the all the items that are present in both the sets.

Consider the following example to calculate the union of two sets.

Example 1 : using union | operator

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2) #printing the union of the sets
```

Output:

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

Python also provides the **union()** method which can also be used to calculate the union of two sets.

Consider the following example.

Example 2: using union() method

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1.union(Days2)) #printing the union of the sets
```

Output:

```
{'Friday', 'Monday', 'Tuesday', 'Thursday', 'Wednesday', 'Sunday', 'Saturday'}
```

Intersection of two sets

- The & (intersection) operator is used to calculate the intersection of the two sets in python.
- The intersection of the two sets are given as the set of the elements that common in both sets.

Consider the following example.

Example 1: using & operator

```
set1 = {"Ayush","John", "David", "Martin"}
set2 = {"Steve","Milan","David", "Martin"}
print(set1&set2) #prints the intersection of the two sets
```

Output:

```
{'Martin', 'David'}
```

Example 2: using intersection() method

```
set1 = {"Ayush","John", "David", "Martin"}
set2 = {"Steave","Milan","David", "Martin"}
print(set1.intersection(set2)) #prints the intersection of the two sets
```

Output:

```
{'Martin', 'David'}
```

The intersection_update() method

- The intersection_update() method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
- The Intersection_update() method is different from intersection() method since it modifies the original set by removing the unwanted items, on the other hand, intersection() method returns a new set.

Consider the following example.

```
a = {"ayush", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle"}

a.intersection_update(b, c)
print(a)
```

Output:

```
{'castle'}
```

Difference of two sets

- The difference of two sets can be calculated by using the subtraction (-) operator.
- The resulting set will be obtained by removing all the elements from set 1 that are present in set 2.

Consider the following example.

Example 1 : using subtraction (-) operator

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1-Days2) #{"Wednesday", "Thursday" will be printed}
```

Output:

```
{'Thursday', 'Wednesday'}
```

Example 2 : using difference() method

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1.difference(Days2)) # prints the difference of the two sets Days1 and Days2
```

Output: `{'Thursday', 'Wednesday'}`

Set comparisons

- Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is subset, superset, or equivalent to other set.
- The boolean true or false is returned depending upon the items present inside the sets.

Consider the following example.

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday"}
Days3 = {"Monday", "Tuesday", "Friday"}
#Days1 is the superset of Days2 hence it will print true.
print (Days1>Days2)
#prints false since Days1 is not the subset of Days2
print (Days1<Days2)
#prints false since Days2 and Days3 are not equivalent
print (Days2 == Days3)
```

Output:

```
True
False
False
```

FrozenSets

- The frozen sets are the immutable form of the normal sets,
- i.e., the items of the frozen set can not be changed and therefore it can be used as a key in dictionary.
- The elements of the frozen set can not be changed after the creation.
- We cannot change or append the content of the frozen sets by using the methods like add() or remove().
- The frozenset() method is used to create the frozenset object.

Consider the following example to create the frozen set.

```
1.      Frozenset = frozenset([1,2,3,4,5])
2.      print(type(Frozenset))
3.      print("\nprinting the content of frozen set...")
4.      for i in Frozenset:
5.          print(i);
6.      Frozenset.add(6) #gives an error since we cannot change the content of Frozenset a
fter creation
```

Output:

```
<class 'frozenset'>
printing the content of frozen set...
1
2
3
4
5
Traceback (most recent call last):
  File "set.py", line 6, in <module>
    Frozenset.add(6) #gives an error since we can change the content of Frozenset after
creation
AttributeError: 'frozenset' object has no attribute 'add'
```

Frozenset for the dictionary

If we pass the dictionary as the sequence inside the frozenset() method, it will take only the keys from the dictionary and returns a frozenset that contains the key of the dictionary as its elements.

Consider the following example.

```
1. dictionary = {"Name":"John", "Country":"USA", "ID":101}
2. print(type(Dictionary))
3. Frozenset = frozenset(Dictionary); #Frozenset will contain the keys of the dictionary
   print(type(Frozenset))
4. for i in Frozenset:
5. print(i)
```

```
Output:<class 'dict'>
<class 'frozenset'>
Name
Country
ID
```