LAB MANUAL

PYTHON PROGRAMMING LAB

INDEX OF EXPERIMENTS

Week	Theory/ Practical (Group-I/ II)		Topic Covered Date and Remarks		
	Practical Day	Topics/ Programs	Date	HOD	Director-Principal
1st		Introduction to python programming			
2nd	1	To compute the GCD of two numbers using python			
3rd	2	To find square root of number			
4th	3	To find exponential using python programming			
5th	4	To find the maximum of list of numbers			
6th	5	To find a value using linear search in python program			
7th	6	To find a value using binary search in python program			
8th	7	To sort list of elements using selection sort			
9th	8	Write a program to find prime number			
10th	9	To multiply matrices using python			
11th	10	To sort list of elements using insertion sort			
12th	11	To sort list of elements using merge sort			
13th	12	To find word and lines in command line arguments			

Mandatory instructions for students:

1. Students should report to the concerned labs as per the given timetable.
2. Students should make an entry in the log book whenever they enter the labs during practical or for their own personal work.
3. When the experiment is completed, students should shut down the computers and make the counter entry in the logbook.
4. Any damage to the lab computers will be viewed seriously.
5. Students should not leave the lab without concerned faculty’s permission.

Assignment: 1

Aim:- Introduction to Python Programming.

Theory:- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Prerequisites

You should have a basic understanding of Computer Programming terminologies. A basic understanding of any of the programming languages is a plus.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

First Python Program:

1. **Open notepad and type following program**

   Print ("Hello World")

2. **Save above program with name.py**
3. **Open command prompt and change path to python program location**
4. **Type "*python name.py" (without quotes) to* run the program.**

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Python Variables: Declare, Concatenate, Global & Local

What is a Variable in Python?

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

```
a=100
print a
```

Python 1 Example

```
# Declare a variable and initialize it
f = 0
print f
# re-declaring the variable works
f = 'guru99'
print f
```

Python 2 Example

```
# Declare a variable and initialize it
f = 0
print(f)
# re-declaring the variable works
f = 'guru99'
print(f)
```

List of some different variable types

```
x = 123                 # integer
x = 123L                # long integer
x = 3.14                # double float
x = "hello"                     # string
x = [0,1,2]                     # list
x = (0,1,2)                     # tuple
  x = open('hello.py', 'r')     # file
```

Constants

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once place inside the bag.

Assigning value to a constant in Python

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example 3: Declaring and assigning value to a constant

Create a constant.py

```python
PI = 3.14
GRAVITY = 9.8
```

Create a main.py

```python
import constant

print(constant.PI)
print(constant.GRAVITY)
```

When you run the program, the output will be:

```
3.14
9.8
```

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a look on all operators one by one.

Python Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = 10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are	9//2 = 4 and

	removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −	9.0//2.0 = 4.0, 11//3 = -4, -11.0//3 = -4.0

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then −

[Show Example]

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.

<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then −

[Show Example]

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a

*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent
		to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13;

Now in binary format they will be as follows − a = 0011 1100 b = 0000 1101

----

a&b = 0000 1100

a|b = 0011 1101 a^b

= 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

[Show Example]

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.

<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

[Show Example]

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Used to reverse the logical state of its operand.

Python Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

[Show Example]

Operator	Description	Example
In	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in
		a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below −

[Show Example]

Operator	Description	Example

Is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is
		not equal to id(y).

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

[Show Example]

Sr.No.	Operator & Description
1	****** Exponentiation (raise to the power)
2	**~ + -** Complement, unary plus and minus (method names for the last two are +@ and -@)
3	*** / % //** Multiply, divide, modulo and floor division
4	**+ -** Addition and subtraction
5	**>> <<** Right and left bitwise shift
6	**&** Bitwise 'AND'
7	**^ |** Bitwise exclusive `OR' and regular `OR'

8	**<= < > >=** Comparison operators
9	**<> == !=** Equality operators
10	**= %= /= //= -= += *= **=** Assignment operators
11	**Is, is not** Identity operators
12	**In, not in** Membership operators
13	**not, or, and** Logical operators

Exercise:

1. **WAP to add two numbers in python**
2. **WAP to declare variables and display types of respective variables**
3. **WAP to demonstrate type casting in python**
4. **WAP to demonstrate Logical operators**

Assignment 2

Aim: To study strings in Python

Theory:

String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

print("Hello") print('Hello')

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.""" print(a)
```

Or three single quotes: **Example**

```
a           = '''Lorem ipsum dolor
sit amet, consectetur adipiscing
elit, sed do eiusmod tempor
incididunt ut labore et dolore
magna aliqua.''' print(a)
```

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Example

Substring. Get the characters from position 2 to position 5 (not included):

```
b          = "Hello, World!"
print(b[2:5])
```

In-bulit Functions in Python:

1. **Strip():**Removes all leading whitespace in string.
2. **len(string)**:Returns the length of the string
3. **upper()**:Converts lowercase letters in string to uppercase.
4. **Lower(): Vice versa**
5. **split(str="", num=string.count(str))**:Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
6. **replace(old, new [, max])**:Replaces all occurrences of old in string with new or at most max occurrences if max given.
7. **find(str, beg=0 end=len(string))**:Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise

Exercise:

1. WAP to Calculate length of string
2. WAP to make string from 1st two and last two characters from given string.
3. WAP to concatenate two strings

Assignment: 3

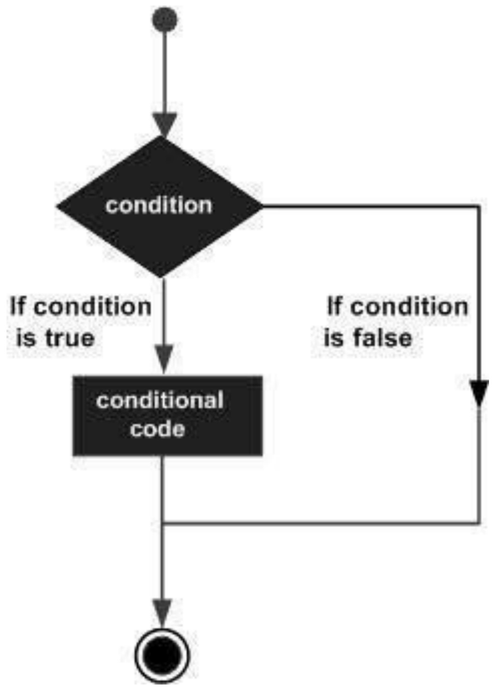
Aim: To study conditional statements in python

Theory:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages –



Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

Python programming language provides following types of decision making statements. Click the following links to check their detail.

Sr.No.	Statement & Description
1	**if statements** An **if statement** consists of a boolean expression followed by one or more statements.
2	**if...else statements** An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
3	**nested if statements** You can use one **if** or **else if** statement inside another **if** or **else if**statement(s).

Let us go through each decision making briefly −

Single Statement Suites

If the suite of an **if** clause consists only of a single line, it may go on the same line as the header statement.

Here is an example of a **one-line if** clause −

```
var = 100 if ( var == 100 ) : print "Value of

expression is 100" print "Good bye!"
```

When the above code is executed, it produces the following result −

```
Value of expression is 100 Good
bye!
```

Exercise:

1. WAP to find out greatest of 3 numbers
2. WAP to find whether given number is odd or even
3. Write a C program to check whether a character is uppercase or lowercasealphabet.
4. WAP to find whether given input is number or character

Assignment: 4

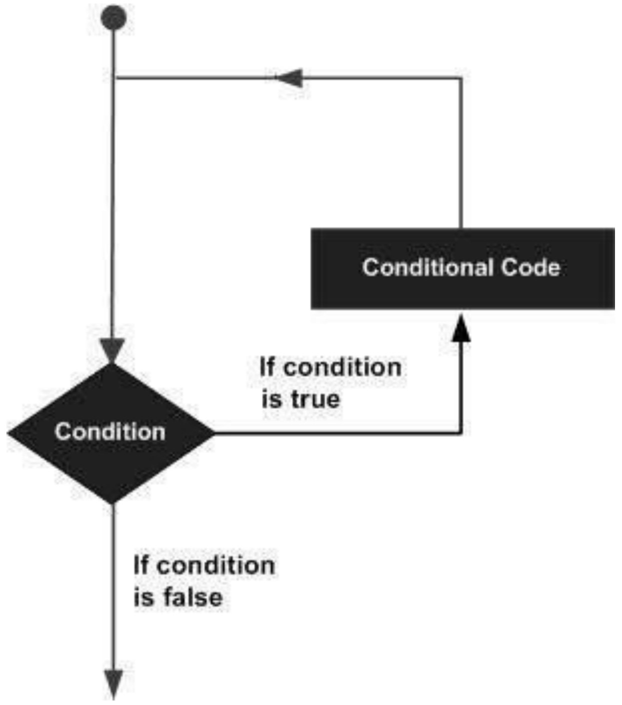
Aim: To study Loops in Python

Theory:

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −



Python programming language provides following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	**while loop** Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	**for loop** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	**nested loops** You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail. Let us go through the loop control statements briefly

Sr.No.	Control Statement & Description
1	**break statement** Terminates the loop statement and transfers execution to the statementimmediately following the loop.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest itscondition prior to reiterating.
3	**pass statement** The pass statement in Python is used when a statement is required syntacticallybut you do not want any command or code to execute.

Exercise:

1. WAP to display even numbers from 1-10
2. WAP to add odd numbers from 1-10
3. Write a Python program to get the Fibonacci series between 0 to 50.
4. Write a Python program to remove the characters which have odd index valuesof a given string.

Assignment 5

Aim: To study python arrays, list, tuples, set, dictionary

Theory:

What is an Array?

An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford" car2
= "Volvo" car3 =
"BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the *index number*.

Example

Get the value of the first array item:

```
x = cars[0]
```

Example

Modify the value of the first array item:

```
cars[0] = "Toyota"
```

The Length of an Array

Use the len() method to return the length of an array (the number of elements in an array).

Example

Return the number of elements in the cars array:

```
x = len(cars)
```

Note: The length of an array is always one more than the highest array index.

Looping Array Elements

You can use the for in loop to loop through all the elements of an array.

Example

Print each item in the cars array:

```
for x in cars:
  print(x)
```

Adding Array Elements

You can use the append() method to add an element to an array.

Example

Add one more element to the cars array:

```
cars.append("Honda")
```

Removing Array Elements

You can use the pop() method to remove an element from the array.

Example

Delete the second element of the cars array:

```
cars.pop(1)
```

You can also use the remove() method to remove an element from the array.

Example

Delete the element that has the value "Volvo":

```
cars.remove("Volvo")
```

Note: The remove() method only removes the first occurrence of the specified value.

Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	**Description**
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python List:

The list is a most versatile datatype available in Python which can be written as a list of commaseparated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ]; list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];

print  "list1[0]:  ",  list1[0]

print "list2[1:5]: ", list2[1:5]
```

When the above code is executed, it produces the following result −

```
list1[0]: physics list2[1:5]:
[2, 3, 4, 5]
```

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method. For example −

```
list = ['physics', 'chemistry', 1997, 2000];

print "Value available at index 2 : " print

list[2] list[2] = 2001; print "New value

available at index 2 : " print list[2]
```

Note − append() method is discussed in subsequent section. When

the above code is executed, it produces the following result −

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

Delete List Elements

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000]; print list1
```

```
del list1[2]; print "After deleting value

at index 2 : " print list1
```

When the above code is executed, it produces following result −

```
['physics', 'chemistry', 1997, 2000] After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

Note − remove() method is discussed in subsequent section.

Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.

Assuming following input −

```
L = ['spam', 'Spam', 'SPAM!']
```

Python Expression	Results	Description
L[2]	SPAM!	Offsets start at zero
L[-2]	Spam	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in List Functions & Methods

Python includes the following list functions −

Sr.No.	Function with Description
1	**cmp(list1, list2)** Compares elements of both lists.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

Python includes following list methods

Sr.No.	Methods with Description
1	**list.append(obj)**
	Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Python Tuple:

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 ); tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5, 6, 7 ); print

"tup1[0]: ", tup1[0]; print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result −

```
tup1[0]: physics tup2[1:5]:
[2, 3, 4, 5]
```

Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56); tup2

= ('abc', 'xyz');


# Following action is not valid for tuples

# tup1[0] = 100;


# So let's create a new tuple as follows
tup3 = tup1 + tup2; print

tup3;
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example −

```
tup = ('physics', 'chemistry', 1997, 2000); print tup; del tup; print "After

deleting tup : "; print tup;
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000) After
deleting tup :
Traceback (most recent call last): File
"test.py", line 9, in <module> print
tup;
NameError: name 'tup' is not defined
```

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −

```
L = ('spam', 'Spam', 'SPAM!')
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero

L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

No Enclosing Delimiters

Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., default to tuples, as indicated in these short examples –

```
print 'abc', -4.24e93, 18+6.6j, 'xyz';

x, y = 1, 2; print "Value of x , y : ",

x,y;
```

When the above code is executed, it produces the following result –

```
abc -4.24e+93 (18+6.6j) xyz Value
of x , y : 1 2
```

Built-in Tuple Functions

Python includes the following tuple functions –

Sr.No.	Function with Description
1	**cmp(tuple1, tuple2)** Compares elements of both tuples.
2	**len(tuple)** Gives the total length of the tuple.

3	**max(tuple)** Returns item from the tuple with max value.
4	**min(tuple)** Returns item from the tuple with min value.
5	**tuple(seq)** Converts a list into tuple.

Python Sets:

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Note: Sets are unordered, so the items will appear in a random order.

Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using thein keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Example

Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add Items

To add one item to a set use the add() method.

To add more than one item to a set use the update() method.

Example

Add an item to a set, using the add() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Example

Add multiple items to a set, using the update() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

Get the Length of a Set

To determine how many items a set has, use the len() method.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset)
```

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

Example

Remove "banana" by using the remove() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Note: If the item to remove does not exist, remove() will raise an error.

Example

Remove "banana" by using the discard() method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

Note: If the item to remove does not exist, discard() will **NOT** raise an error.

You can also use thepop(), method to remove an item, but this method will remove item.
the *last* ts are unordered, so you will not know what item that gets

removed.

The return value of the pop() method is the removed item.

Example

Remove the last item by using the pop() method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x) print(thisset)
```

Note: Sets are *unordered*, so when using the pop() method, you will not know which item that gets removed.

Example

The clear() method empties the set:

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Example

The del keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

Example

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Exercise:

1. Write a Python script to sort (ascending and descending) a dictionary by value
2. Write a Python script to check if a given key already exists in a dictionary.
3. Write a Python script to merge two Python dictionaries
4. Write a Python program to add an item in a tuple.
5. Write a Python program to create a tuple with different data types
6. Write a Python program to sum all the items in a list
7. Write a Python program to get the largest number from a list.
8. Write a Python program to add member(s) in a set.
9. Write a Python program to reverse the order of the items in the array
10. Write a Python program to create an array of 5 integers and display the array items. Accessindividual element through indexes.

Assignment:6

Aim: To study functions in python

Theory:

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

Example

```
my_function():  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():  print("Hello from a function") my_function()
```

Parameters

Information can be passed to functions as parameter. Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.The following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
def my_function(fname):  print(fname + " Refsnes")

my_function("Emil") my_function("Tobias")
my_function("Linus")
```

Default Parameter

Value

The following example shows how to use a default parameter value. If we call the function without parameter, it uses the default value:

Example

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India") my_function()
my_function("Brazil")
```

Passing a List as a Parameter

You can send any data types of parameter to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function. E.g. if you send a List as a parameter, it will still be a List when it reaches the function:

Example

```
def my_function(food): for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Return Values

To let a function return a value, use the return statement:

Example

```
def my_function(x):
  return 5 * x

print(my_function(3)) print(my_function(5))
print(my_function(9))
```

Recursion

Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematicallyelegant approach to programming.

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements-1) every time we recurse. The recursion ends when

the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Example

Recursion Example

```
def tri_recursion(k):
 if(k>0):
   result = k+tri_recursion(k-1)
   print(result)
else:     result
= 0
 return result

print("\n\nRecursion Example Results") tri_recursion(6)
```

Exercise:

1. Write a Python function to find the Max of three numbers.
2. Write a Python program to reverse a string.
   Sample String : "1234abcd"
   Expected Output : "dcba4321"
3. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

Assignment: 7

Aim: To study mysql commands

Theory: Mysql has two types of languages

a. **Data definition language:**

   i. **Create:** Used to create database and tables
      Syn: create database database_name;
      Ex: create database jnec;
      Syn: create table table_name(column_name1 type,column_name2 type);
      Ex: create table syit(id int, name char(10));

   ii. **Drop:** Used to delete database and table
      Syn: drop database/table database_name/table_name;
      Ex: drop database/table jnec/syit;

   iii. **Alter:** Used to add, delete or change data type of column of table. **Syn:** alter table table_name add column column_name type; alter table table_name drop column column name; alter table table_name alter column column name type; **ex:** alter table syit add column address varchar(10); alter table syit drop address;
      alter table syit alter column name varchar(10);

   iv. **Truncate:** Used to delete contents of table but it will preserve structure of table
      Syn: truncate table table_name ;
      Ex: truncate table syit;

b. **Data manipulation language:**

   i. **Select:** used to fetch rows from table
      Syn: select * from table_name;(fetches data from all columns) Select column_name from syit;(fetches data from respective column)
      Ex: select * from syit;
      Select name from syit;

   ii. **Insert:** used to insert data into table
      Syn: insert into table_name values('value1', value2); (if value is text then include in single quote, if it is number then don't use single quote)
      Ex: insert into syit values(1,'amol');

iii. **Update:** used to update value from particular column based on some contion
Syn: update table_name set column= value where column=value;
Ex: update syit set id=4 where name='amol'; **iv. Delete:** used to delete row from table
Syn: delete from table_name where col=value;
Ex: delete from syit where name='amol';

Exercise:

1. Create database called mgm.
2. Create table institutes in mgm with columns id, college_name, principal, contact.
3. Insert 5 entries in table.
4. Add column address in institute
5. Drop column contact
6. Modify type of college_name from char to varchar
7. Update contact of Dr H H shinde to 9028885925
8. Delete college of engineering from institutes
9. Truncate institutes
10. Drop institutes and mgm

Assignment: 8

Aim: To connect python with mysql with pymysql

Theory:

What is PyMySQL ?

PyMySQL is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and contains a pure-Python MySQL client library. The goal of PyMySQL is to be a drop-in replacement for MySQLdb.

How do I Install PyMySQL?

Before proceeding further, you make sure you have PyMySQL installed on your machine. Just type the following in your Python script and execute it −

```
import pymysql
```

If it produces the following result, then it means MySQLdb module is not installed −

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    Import pymysql
ImportError: No module named pymysql
```

The last stable release is available on PyPI and can be installed with pip −

```
pip install pymysql
```

Alternatively (e.g. if pip is not available), a tarball can be downloaded from GitHub and installed with Setuptools as follows −

```
$ # X.X is the desired pymysql version (e.g. 0.5 or 0.6).

$ curl -L https://github.com/PyMySQL/PyMySQL/tarball/pymysql-X.X | tar xz

$ cd PyMySQL*

$ python setup.py install

$ # The folder PyMySQL* can be safely removed now.
```

Note − Make sure you have root privilege to install the above module.

Database Connection

Before connecting to a MySQL database, make sure of the following points −

- You have created a database TESTDB.
- You have created a table EMPLOYEE in TESTDB.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.
- Python module PyMySQL is installed properly on your machine.
- You have gone through MySQL tutorial to understand MySQL Basics.

Example

Following is an example of connecting with MySQL database "TESTDB" −

```
import pymysql


# Open database connection db =

pymysql.connect("localhost","testuser","test123","TESTDB" )


# prepare a cursor object using cursor() method cursor

= db.cursor()


# execute SQL query using execute() method.

cursor.execute("SELECT VERSION()")


# Fetch a single row using fetchone() method.
data = cursor.fetchone() print ("Database
version : %s " % data)


# disconnect from server db.close()
```

While running this script, it produces the following result.

```
Database version : 5.5.20-log
```

If a connection is established with the datasource, then a Connection Object is returned and saved into **db** for further use, otherwise **db** is set to None. Next, **db** object is used to create a **cursor** object, which in turn is used to execute SQL queries. Finally, before coming out, it ensures that the database connection is closed and resources are released.

Creating Database Table

Once a database connection is established, we are ready to create tables or records into the database tables using **execute** method of the created cursor.

Example

Let us create a Database table EMPLOYEE −

```
import pymysql



# Open database connection db =

pymysql.connect("localhost","testuser","test123","TESTDB" )



# prepare a cursor object using cursor() method cursor

= db.cursor()



# Drop table if it already exist using execute() method. cursor.execute("DROP

TABLE IF EXISTS EMPLOYEE")



# Create table as per requirement sql
= """CREATE TABLE EMPLOYEE (
```

```
  FIRST_NAME CHAR(20) NOT NULL,

  LAST_NAME  CHAR(20),

  AGE INT,
  SEX CHAR(1),

  INCOME FLOAT )"""


cursor.execute(sql)


# disconnect from server db.close()
```

INSERT Operation

The INSERT Operation is required when you want to create your records into a database table.

Example

The following example, executes SQL *INSERT* statement to create a record in the EMPLOYEE table –

```
import pymysql


# Open database connection db =

pymysql.connect("localhost","testuser","test123","TESTDB" )


# prepare a cursor object using cursor() method cursor

= db.cursor()


# Prepare SQL query to INSERT a record into the database.
```

```
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,    LAST_NAME, AGE, SEX, INCOME)

   VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:

   # Execute the SQL command    cursor.execute(sql)

   # Commit your changes in the database    db.commit() except:

   # Rollback in case there is any error    db.rollback()


# disconnect from server db.close()
```

Example

The following code segment is another form of execution where you can pass parameters directly –

```
.................................
user_id = "test123"
password = "password"

con.execute('insert into Login values("%s", "%s")' % \            (user_id, password))
.................................
```

READ Operation

READ Operation on any database means to fetch some useful information from the database.

Once the database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch a single record or **fetchall()** method to fetch multiple values from a database table.

- **fetchone()** − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall()** − It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount** − This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Example

The following procedure queries all the records from EMPLOYEE table having salary more than 1000 −

```
import pymysql


# Open database connection db =

pymysql.connect("localhost","testuser","test123","TESTDB"  )


# prepare a cursor object using cursor() method cursor

= db.cursor()


# Prepare SQL query to INSERT a record into the database.

sql = "SELECT * FROM EMPLOYEE \

WHERE INCOME > '%d'" % (1000)

try:

  # Execute the SQL command

cursor.execute(sql)

  # Fetch all the rows in a list of lists.

  results = cursor.fetchall()
```

```
   for row in results:
fname = row[0]
lname = row[1]
age = row[2]       sex
= row[3]       income
= row[4]
      # Now print fetched result       print ("fname = %s,lname = %s,age =
%d,sex = %s,income = %d" % \
         (fname, lname, age, sex, income )) except:
   print ("Error: unable to fetch data")


# disconnect from server db.close()
```

Output

This will produce the following result −

```
fname = Mac, lname = Mohan, age = 20, sex = M, income = 2000
```

Update Operation

Exercise:

1. **Create python application which will accept name and age and store it into mysql**
2. **Create python application which will display all names and ages from mysql table**
3. **Write python application to delete all entries with ages 28**

Assignment: 9

Aim: To study file handling in python

Theory:

File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w " - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt") The code above is the same as: f =

open("demofile.txt", "rt")

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Open a File on the Server
Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt This file is for testing purposes. Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

Example

```
f = open("demofile.txt", "r") print(f.read())
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Read Lines

You can return one line by using the readline() method:

Example

Read one line of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
```

By calling readline() two times, you can read the two first lines:

Example

Read two lines of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

Example

Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

It is a good practice to always close the file when you are done with it.

Example

Close the file when you are finish with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Write to an Existing File

To write to an existing file, you must add a parameter to the `open()` function:

`"a"` - Append - will append to the end of the file "w" - Write - will overwrite any existing content

Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r") print(f.read())
```

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r") print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Exercise:

1. Write a Python program to read an entire text file
2. Write a Python program to append text to a file and display the text.
3. Write a Python program to read last n lines of a file

Assignment:10

Aim: To study classes in python

Theory:

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

Create Object

Now we can use the class named myClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.
All classes have a function called ___init_(), which is always executed when the class is being initiated.

Use the __init_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init_(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name) print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Exercise:

1. Write a Python class to reverse a string word by word.
Input string : 'hello .py'
Expected Output : '.py hello'