

**KARPAGAM ACADEMY OF HIGHER EDUCATION**
(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore - 641 021, India

FACULTY OF ARTS, SCIENCE AND HUMANITIES (FASH)

Department of CS,CA & IT

I M.Sc CS**I SEMESTER****BATCH : 2019 - 2021****19CSP101****PYTHON PROGRAMMING****4H – 4C****Instruction Hours / week: L: 4 T: 0 P: 0 Marks: Int : 40 Ext : 60 Total: 100****SCOPE**

This programming language is versatile, robust and comprehensive programming language. It has true portability features and can be used across a multitude of platforms.

COURSE OBJECTIVES

- To learn how to design and program Python applications.
- To learn how to use indexing and slicing to access data in Python programs.
- To define the structure and components of a Python program.
- Master the principles of object-oriented programming and the interplay of algorithms and data structures in well-written modular code;
- Solve problems requiring the writing of well-documented programs in the Python language, including use of the logical constructs of that language;
- Demonstrate significant experience with the Python program development environment.

COURSE OUTCOME

After the course, students should be able to:

- Adequately use standard programming constructs: repetition, selection, functions, composition, modules, aggregated data (arrays, lists, etc.)
- Master an understanding of loops and decision statements and functions.
- Master an understanding of Python especially the object-oriented concepts.
- Master an understanding of the built-in objects ,List, tuple, set of Python

UNIT I

Python Basics: Introduction-features-Syntax and Statements- Variables and Assignments-Identifier- Operators .**Conditional and looping statement.** **Functions:** calling function-creating functions-Function arguments.

UNIT II

Numbers: Introduction- Integer-Floating Point-Complex numbers-Operators-Other numeric type. **Strings**-Strings and Operator-String only operator- Built-in-Functions-Built-in-Methods-String Features. **List** : Operators-Built-in-Functions-Built-in-Methods-Features of List

UNIT III

Tuple : Introduction- Operators and Built-in-Functions-Features. **Mapping and set type Dictionaries**-Operators-Built-in and Factory Functions-Built-in- Methods. **Set type:** Introduction- Operators-Built-in Function-Built-in Methods-

UNIT IV

Python Objects: Introduction-Standard Type- Built-in-type-Built-in functions. **Class:** Introduction- Class and Instance- Method calls. **File:** Objects- Built in Functions-Methods-Attributes- Command line Argument-File System-File Execution.

UNIT V

Exception and Tools: Why use it?- Exception roles-Exception in python-Try/finally statement. **Regular Expression:** Introduction-Special Symbols and characters-Regexes and Python- Examples of Regexes. **Network Programming:** Architecture- Socket-networking programming in python.

SUGGESTED READINGS

1. Chun, J Wesley. (2015). Core Python Programming (2nd ed) New Delhi: Pearson.
2. Wesley J Chun. Core python Application Programming. (3rd ed.)
3. Budd, T. (2013). Exploring Python (1st ed.). New Delhi: TMH.
4. Python Tutorial/Documentation www.python.org 2015.
5. Allen Downey., Jeffrey Elkner, & Chris Meyers. (2012). How to think like a computer scientist: learning with Python. Freely available online.

WEB SITES

1. www.programiz.com
2. www.guru99.com
3. <https://www.tutorialspoint.com>
4. <http://interactivepython.org/courselib/static/pythonds>.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python Basics: Introduction-features-Syntax and Statements- Variables and Assignments- Identifier- Operators .**Conditional and looping statement. Functions:** calling function-creating functions-Function arguments.

OVERVIEW OF PROGRAMMING:

PYTHON INTRODUCTION

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, makes 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 on special area such as web programming. That is why it is known as *multipurpose* 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.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

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 [official 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.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

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 History

- 
- Python laid its foundation in the late 1980s.
 - The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.
 - In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources.
 - In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
 - Python 2.0 added new features like: list comprehensions, garbage collection system.
 - On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.
 - *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
 - Python is influenced by following programming languages:
 - ABC language.

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

- Modula-3

Python Version

Python programming language is being updated regularly with new features and supports. There are lots of updations in python versions, started from 1994 to current release.

A list of python versions with its released date is given below.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT I

Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.6.4	December 19, 2017

Python Applications Area

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 specifying 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.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

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

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python Example

Python is easy to learn and code and can be execute with python interpreter. We can also use Python interactive shell to test python code immediately.

A simple hello world example is given below. Write below code in a file and save with **.py** extension. Python source file has **.py** extension.

hello.py

```
print("hello world by python!")
```

Execute this example by using following command.

Python3 hello.py

After executing, it produces the following output to the screen.

Output

```
hello world by python!
```

Python Example using Interactive Shell

Python interactive shell is used to test the code immediately and does not require to write and save code in file.

Python code is simple and easy to run. Here is a simple Python code that will print "Welcome to Python".

A simple python example is given below.

```
>>> a="Welcome To Python"
```

```
>>> print a
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Welcome To Python

```
>>>
```

Python 3.4 Example

In python 3.4 version, you need to add parenthesis () in a string code to print it.

```
>>> a=("Welcome To Python Example")
```

```
>>> print a
```

Welcome To Python Example

```
>>>
```

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.

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Assigning value to variable:

Value should be given on the right side of assignment operator(=) and variable on left side.

```
>>>counter =45  
print(counter)
```

Assigning a single value to several variables simultaneously:

```
>>> a=b=c=100
```

Assigning multiple values to multiple variables:

```
>>> a,b,c=2,4,"ram"
```

Variables do not need to be declared with any particular type and can even change type after they have been set.

Example

```
x = 4 # x is of type int  
x = "Sally" # x is now of type str  
print(x)
```

String variables can be declared either by using single or double quotes:

Example

```
x = "John"  
# is the same as  
x = 'John'
```

Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Output:

```
Orange
Banana
Cherry
```

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Output:

```
Orange
Orange
Orange
```

Python program to swap two variables

Variable swapping:

In computer programming, swapping two variables specifies the mutual exchange of values of the variables. It is generally done by using a temporary variable.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

For example:

```
data_item x := 1
data_item y := 0
swap (x, y)
```

After swapping:

```
data_item x := 0
data_item y := 1
```

See this example:

```
# Python swap program
x = input('Enter value of x: ')
y = input('Enter value of y: ')
# create a temporary variable and swap the values
temp = x
x = y
y = temp
print('The value of x after swapping: {}'.format(x))
print('The value of y after swapping: {}'.format(y))
```

Output:

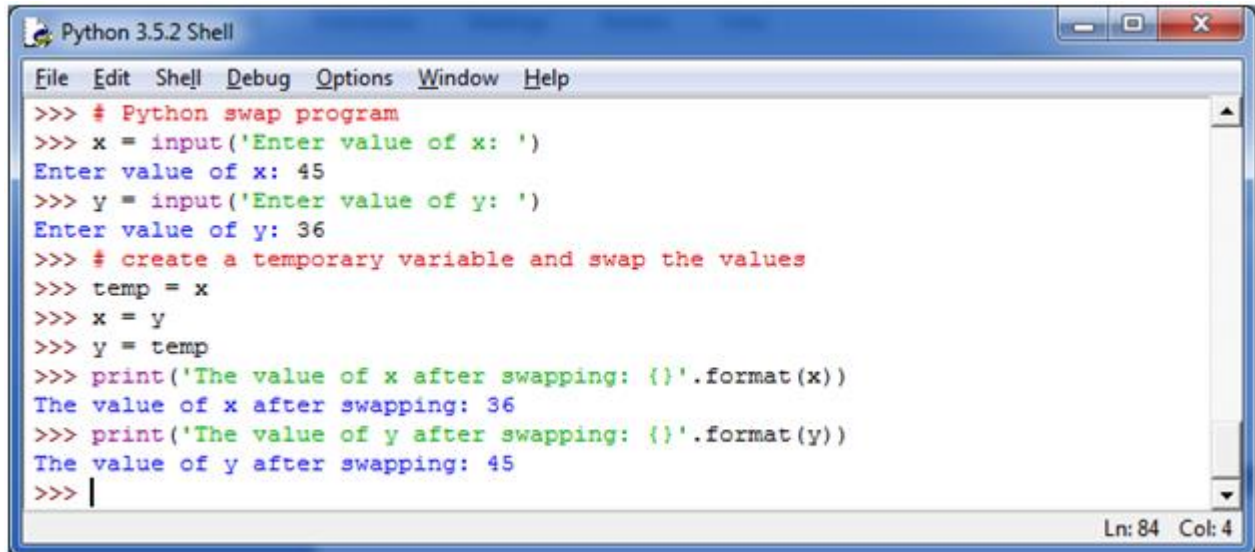
CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

A screenshot of a Python 3.5.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following code:

```
>>> # Python swap program
>>> x = input('Enter value of x: ')
Enter value of x: 45
>>> y = input('Enter value of y: ')
Enter value of y: 36
>>> # create a temporary variable and swap the values
>>> temp = x
>>> x = y
>>> y = temp
>>> print('The value of x after swapping: {}'.format(x))
The value of x after swapping: 36
>>> print('The value of y after swapping: {}'.format(y))
The value of y after swapping: 45
>>> |
```

The status bar at the bottom right shows 'Ln: 84 Col: 4'. A large, light gray watermark 'KAPAGAM' is visible diagonally across the page.

UNIT I

VALUES AND DATA TYPES

Value:

Value can be any letter ,number or string.

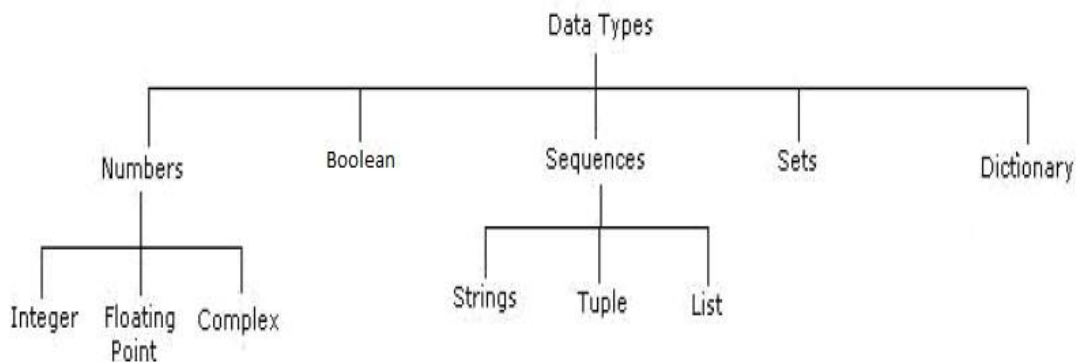
Eg, Values are 2, 42.0, and 'Hello, World!'. (These values belong to different datatypes.)

Data type:

Every value in Python has a data type.

It is a set of values, and the allowable operations on those values.

Python has four standard data types:



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
------	-------	------	-----	----

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

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

Identifiers

Identifiers are the names given to the fundamental building blocks in a program.

These can be variables ,class ,object ,functions , lists , dictionaries etc.

There are certain rules defined for naming i.e., Identifiers.

I. An identifier is a long sequence of characters and numbers.

II.No special character except underscore (_) can be used as an identifier.

III.Keyword should not be used as an identifier name.

IV.Python is case sensitive. So using case is significant.

V.First character of an identifier can be character, underscore (_) but not digit.

Python Operators

Operators are particular symbols that are used to perform operations on operands. It returns result that can be used in application.

Example $4 + 5 = 9$

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT I

Here 4 and 5 are Operands and (+) , (=) signs are the operators. This expression produces the output 9.

Types of Operators

Python supports the following operators

1. Arithmetic Operators.
2. Relational Operators.
3. Assignment Operators.
4. Logical Operators.
5. Membership Operators.
6. Identity Operators.
7. Bitwise Operators.

Arithmetic Operators

The following table contains the arithmetic operators that are used to perform arithmetic operations.

Operators	Description
//	Perform Floor division(gives integer value after division)
+	To perform addition
-	To perform subtraction
*	To perform multiplication

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

/	To perform division
%	To return remainder after division(Modulus)
**	Perform exponent(raise to power)

Example

```
>>> 10+20
```

```
30
```

```
>>> 20-10
```

```
10
```

```
>>> 10*2
```

```
20
```

```
>>> 10/2
```

```
5
```

```
>>> 10%3
```

```
1
```

```
>>> 2**3
```

```
8
```

```
>>> 10//3
```

```
3
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT I

>>>

Relational Operators

The following table contains the relational operators that are used to check relations.

Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
<>	Not equal to(similar to !=)

eg:

>>> 10<20

True

>>> 10>20

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

False

```
>>> 10<=10
```

True

```
>>> 20>=15
```

True

```
>>> 5==6
```

False

```
>>> 5!=6
```

True

```
>>> 10<>2
```

True

```
>>>
```

Assignment Operators

The following table contains the assignment operators that are used to assign values to the variables.

Operators	Description
=	Assignment
/=	Divide and Assign

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

+=	Add and assign
-=	Subtract and Assign
*=	Multiply and assign
%=	Modulus and assign
**=	Exponent and assign
//=	Floor division and assign

Example

```
>>> c=10
```

```
>>> c
```

```
10
```

```
>>> c+=5
```

```
>>> c
```

```
15
```

```
>>> c-=5
```

```
>>> c
```

```
10
```

```
>>> c*=2
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

```
>>> c
```

```
20
```

```
>>> c/=2
```

```
>>> c
```

```
10
```

```
>>> c%=3
```

```
>>> c
```

```
1
```

```
>>> c=5
```

```
>>> c**=2
```

```
>>> c
```

```
25
```

```
>>> c//=2
```

```
>>> c
```

```
12
```

```
>>>
```

Logical Operators

The following table contains the arithmetic operators that are used to perform arithmetic operations.

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Operators	Description
and	Logical AND(When both conditions are true output will be true)
or	Logical OR (If any one condition is true output will be true)
not	Logical NOT(Compliment the condition i.e., reverse)

Example

```
a=5>4 and 3>2
```

```
print a
```

```
b=5>4 or 3<2
```

```
print b
```

```
c=not(5>4)
```

```
print c
```

Output:

```
>>>
```

```
True
```

```
True
```

```
False
```

```
>>>
```


Conditional and looping statement

Control statements

Python If Statements

The Python if statement is a statement which is used to test specified condition. We can use if statement to perform conditional operations in our Python application. The if statement executes only when specified condition is **true**. We can pass any valid expression into the if parentheses.

There are various types of if statements in Python.

- if statement
- if-else statement
- nested if statement

Python If Statement Syntax

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

Python If Statement Example

```
a=10  
if a==10:  
    print "Welcome to javatpoint"
```

Output: Hello User

Python If Else Statements

The If statement is used to test specified condition and if the condition is true, if block executes, otherwise else block executes. The else statement executes when the if statement is false.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python If Else Syntax

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

Example-

```
year=2000
if year%4==0:
    print "Year is Leap"
else:
    print "Year is not Leap"
```

Output: Year is Leap

Python Nested If Else Statement

In python, we can use nested If Else to check multiple conditions. Python provides **elif** keyword to make nested If statement. This statement is like executing a if statement inside a else statement.

Python Nested If Else Syntax

```
If statement:
    Body
elif statement:
    Body
else:
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Body

Python Nested If Else Example

```
a=10
if a>=20:
    print "Condition is True"
else:
    if a>=15:
        print "Checking second value"
    else:
        print "All Conditions are false"
```

Output: All Conditions are false.

EXAMPLE:

Python Program to get a number num and check whether num is odd or even?

```
num1=int(input("Enter your number:"))
if(num1%2==0):
    print("{} is even".format(num1))
else:
    print("{} is odd".format(num1))
```

Looping

For Loop

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python **for loop** is used to iterate the elements of a collection in the order that they appear. This collection can be a sequence(list or string).

Python For Loop Syntax

```
for <variable> in <sequence>:
```

Python For Loop Simple Example

```
num=2  
for a in range (1,6):  
    print num * a
```

Output:

```
2  
4  
6  
8  
10
```

Python Example to Find Sum of 10 Numbers

```
sum=0  
for n in range(1,11):  
    sum+=n  
print sum
```

Output: 55

Python Nested For Loops

Loops defined within another Loop are called Nested Loops. Nested loops are used to iterate matrix elements or to perform complex computation. When an outer loop contains an inner loop in its body it is called Nested Looping.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python Nested For Loop Syntax

```
for <expression>:  
    for <expression>:  
        Body
```

Python Nested For Loop Example

```
for i in range(1,6):  
    for j in range (1,i+1):  
        print i,  
        print
```

Output:

```
>>>  
1  
2 2  
3 3 3  
4 4 4 4  
5 5 5 5 5  
>>>
```

Explanation:

For each value of Outer loop the whole inner loop is executed.

For each value of inner loop the Body is executed each time.

Python Nested Loop Example 2

```
for i in range (1,6):  
    for j in range (5,i-1,-1):  
        print "*",
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

`print`

Output:

```
>>>
* * * * *
* * * * *
* * *
* *
*
```

Python While Loop

In Python, while loop is used to execute number of statements or body till the specified condition is true. Once the condition is false, the control will come out of the loop.

Python While Loop Syntax

```
while <expression>:
```

```
    Body
```

Here, loop Body will execute till the expression passed is true. The Body may be a single statement or multiple statement.

Python While Loop Example 1

```
a=10
```

```
while a>0:
```

```
    print "Value of a is",a
```

```
    a=a-2
```

```
print "Loop is Completed"
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Output:

```
>>>
Value of a is 10
Value of a is 8
Value of a is 6
Value of a is 4
Value of a is 2
Loop is Completed
>>>
```

Python While Loop Example 2

```
n=153
sum=0
while n>0:
    r=n%10
    sum+=r
    n=n/10
print sum
```

Output:

```
>>>
9 >>>
```

Do While Loop

Python doesn't have do-while loop. But we can create a program like this.

The do while loop is used to check condition after executing the statement. It is like while loop but it is executed at least once.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

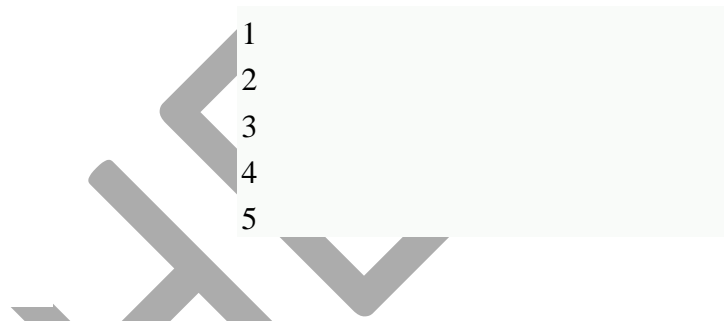
General Do While Loop Syntax

```
do {  
    //statement  
} while (condition);
```

Python Do While Loop Example

```
i = 1  
  
while True:  
    print(i)  
    i = i + 1  
    if(i > 5):  
        break
```

Output:



Python Break

Break statement is a jump statement which is used to transfer execution control. It breaks the current execution and in case of inner loop, inner loop terminates immediately.

When break statement is applied the control points to the line following the body of the loop, hence applying break statement makes the loop to terminate and controls goes to next line pointing after loop body.

Python Break Example 1

```
for i in [1,2,3,4,5]:  
    if i==4:
```

```
        print "Element found"  
        break  
    print i,
```


CLASS : I M.Sc CS
COURSE NAME : PYTHON PROGRAMMING

BATCH : 2019 - 2021
COURSE CODE : 19CSP101

UNIT I

Output:

1 2 3 Element found

>>>

>>>

Output:

Python Break Example 2

```
for letter in 'Python3':  
    if letter == 'o':  
        break  
    print (letter)
```

P

y

t

h

Python Continue Statement

Python Continue Statement is a jump statement which is used to skip execution of current iteration. After skipping, loop continues with next iteration. We can use continue statement with for as well as while loop in Python.

Python Continue Statement Example

```
a=0  
while a<=5:  
    a=a+1  
    if a%2==0:  
        continue  
    print a  
print "End of Loop"
```

>>>

1

3

5

End of Loop

>>>

Output:

Python Pass

In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty. It is same as the name refers to. It just makes the

control to pass by without executing any code. If we want to bypass any code pass statement can be used.

Python Pass Syntax

pass

Python Pass Example

```
for i in [1,2,3,4,5]:
    if i==3:
        pass
    print "Pass when value is",i
    print i,
```

Output:

```
>>>
1 2 Pass when value is 3
3 4 5
>>>
```

Python Functions

A Function is a self block of code which is used to organize the functional code. Function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability. Function is a subprogram that works on data and produces some output.

Types of Functions:

There are two types of Functions.

- a) Built-in Functions: Functions that are predefined and organized into a library. We have used many predefined functions in Python.
- b) User- Defined: Functions that are created by the programmer to meet the requirements.

Defining a Function

A Function defined in Python should follow the following format:

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

1) Keyword **def** is used to start and declare a function. Def specifies the starting of function block.

2) def is followed by function-name followed by parenthesis.

3) Parameters are passed inside the parenthesis. At the end a colon is marked.

Python Function Syntax

```
def <function_name>(parameters):  
    </function_name>
```

Example **def** sum(a,b):

4) Python code requires indentation (space) of code to keep it associate to the declared block.

5) The first statement of the function is optional. It is ?Documentation string? of function.

6) Following is the statement to be executed.

Syntax:

The diagram illustrates the syntax of a Python function definition. It shows the following structure:

```
def <function_name>([parameters]):  
    "function docstring"  
    Statement1  
    Statement2  
    ...  
    ....
```

Labels with arrows pointing to the corresponding parts of the code:

- def keyword** points to the `def` keyword.
- Function name** points to the `<function_name>` placeholder.
- Parameters** points to the `[parameters]` placeholder.
- Indentation** points to the four spaces at the beginning of the first line of the function body.

Watermarks for javatpoint.com and www.javatpoint.com are visible in the background.

Invoking a Python Function

To execute a function it needs to be called. This is called function calling.

Function Definition provides the information about function name, parameters and the definition what operation is to be performed. In order to execute the function definition, we need to call the function.

Python Function Syntax

```
<function_name>(parameters)  
</function_name>
```

Python Function Example

```
sum(a,b)
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Here, sum is the function and a, b are the parameters passed to the function definition.

Let's have a look over an example.

Python Function Example 2

```
#Providing Function Definition
def sum(x,y):
    "Going to add x and y"
    s=x+y
    print "Sum of two numbers is"
    print s
#Calling the sum Function
sum(10,20)
sum(20,30)
```

Output:

```
>>>
Sum of two numbers is
30
Sum of two numbers is
50
>>>
```

Python Function return Statement

return[expression] is used to return response to the caller function. We can use expression with the return keyword. send back the control to the caller with the expression. In case no expression is given after return it will return None. In other words return statement is used to exit the function definition.

Python Function return Example

```
def sum(a,b):
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

```
"Adding the two values"
print "Printing within Function"
print a+b
return a+b
def msg():
    print "Hello"
    return

total=sum(10,20)
print "Printing Outside: ",total
msg()
print "Rest of code"
```

Output:

```
>>>
Printing within Function
30
Printing outside: 30
Hello
Rest of code
>>>
```

Python Function Argument and Parameter

There can be two types of data passed in the function.

- 1) The First type of data is the data passed in the function call. This data is called 'arguments'.
- 2) The second type of data is the data received in the function definition. This data is called 'parameters'.

Arguments can be literals, variables and expressions. Parameters must be variable to hold incoming values. Alternatively, arguments can be called as actual parameters or actual arguments and parameters can be called as formal parameters or formal arguments.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python Function Example

```
def addition(x,y):  
    print x+y  
x=15  
addition(x ,10)  
addition(x,x)  
y=20  
addition(x,y)
```

Output:

```
>>>  
25  
30  
35  
>>>
```

Passing Parameters

Apart from matching the parameters, there are other ways of matching the parameters.

Python supports following types of formal argument:

- 1) Positional argument (Required argument).
- 2) Default argument.
- 3) Keyword argument (Named argument)

Positional/Required Arguments:

When the function call statement must match the number and order of arguments as defined in the function definition. It is Positional Argument matching.

Python Function Positional Argument Example

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

```
#Function definition of sum
def sum(a,b):
    "Function having two parameters"
    c=a+b
    print c
```

```
sum(10,20)
sum(20)
```

Output:

```
>>>
30
```

Traceback (most recent call last):

File "C:/Python27/su.py", line 8, in <module>

sum(20)

TypeError: sum() takes exactly 2 arguments (1 given)

```
>>>
```

```
</module>
```

Explanation:

1) In the first case, when sum() function is called passing two values i.e., 10 and 20 it matches with function definition parameter and hence 10 and 20 is assigned to a and b respectively. The sum is calculated and printed.

2) In the second case, when sum() function is called passing a single value i.e., 20 , it is passed to function definition. Function definition accepts two parameters whereas only one value is being passed, hence it will show an error.

Python Function Default Arguments

Default Argument is the argument which provides the default values to the parameters passed in the function definition, in case value is not provided in the function call default value is used.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Python Function Default Argument Example

```
#Function Definition
def msg(Id,Name,Age=21):
    "Printing the passed value"
    print Id
    print Name
    print Age
    return

#Function call
msg(Id=100,Name='Ravi',Age=20)
msg(Id=101,Name='Ratan')
```

Output:

```
>>>
100
Ravi
20
101
Ratan
21
>>>
```

Explanation:

- 1) In first case, when msg() function is called passing three different values i.e., 100 , Ravi and 20, these values will be assigned to respective parameters and thus respective values will be printed.
- 2) In second case, when msg() function is called passing two values i.e., 101 and Ratan, these values will be assigned to Id and Name respectively. No value is assigned for third argument via function call and hence it will retain its default value i.e, 21.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT I

Possible Questions:

2 Mark Questions

1. What is Python?
2. Define Namespace & mention its type.
3. How does a computer run a python program?
4. List the features of python.
5. What is necessary to execute a Python program?
6. What is a statement in a Python program?
7. What is interpreter?
8. Select and assign how an input operation was done in python.
9. Differentiate for loop and while loop.
10. Write the syntax for while loop with flowchart.

6 Mark Questions

1. Briefly discuss about the fundamental of Python.
2. Write a Python program to multiply two matrices.
3. Discuss the need and importance of function in python.
4. Explain conditional statements in detail with example
5. What are the different loop control statements available in python? Explain with suitable examples.
6. Briefly discuss about the types of decision making statement
7. Write a Python program using function to check given number is odd or even.



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS)

(BATCH 2019 - 2021)

I SEMESTER

PYTHON PROGRAMMING (19CSP101)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT I

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	Python is said to be easily	readable language	writable language	bug-able language	script-able language	readable language
2	Extensible programming language that can be extended through classes and programming interfaces is	Python	Perl	PHP	Ada	Python
3	Python was released publicly in	1941	1971	1981	1991	1991
4	What is the output when following code is executed ? print r"\nhello" The output is	a new line and hello	\nhello	the letter r and then hello	Error	\nhello
5	What is the output of the following code ? example = "snow world" example[3] = 's' print example	snow	snow world	Error	snos world	Error
6	Is Python case sensitive when dealing with identifiers?	yes	no	machine dependent	none	yes

7	What is the maximum possible length of an identifier?	31 characters	63 characters	79 characters	none of the mentioned	none of the mentioned
8	Which of the following is not a keyword?	eval	assert	nonlocal	pass	eval
9	All keywords in Python are in	lower case	UPPER CASE	Capitalized	None of the mentioned	None of the mentioned
10	Which of the following is true for variable names in Python?	unlimited length	all private members must have leading and trailing	underscore and ampersand are the only two special characters	none of the mentioned	unlimited length
11	Which of the following is an invalid statement?	abc = 1,000,000	a b c = 1000 2000 3000	a,b,c = 1000, 2000, 3000	a_b_c = 1,000,000	a b c = 1000 2000 3000
12	Which of the following cannot be a variable?	__init__	in	it	on	in
13	What is the output of print 0.1 + 0.2 == 0.3?	TRUE	FALSE	machine dependent	Error	FALSE
14	Which of the following is not a complex number?	k = 2 + 3j	k = complex(2, 3)	k = 2 + 3I	k = 2 + 3J	k = 2 + 3I
15	Which of the following data types is not supported in python?	number	list	string	slice	slice
16	Which of these is not a core data type?	Lists	Dictionary	Tuples	Class	Class
17	What data type is the object below ? L = [1, 23, 'hello', 1]	Lists	Dictionary	Tuples	Array	Lists

18	Which of the following function convert a string to a float in python?	int(x [,base])	long(x [,base])	float(x)	str(x)	float(x)
19	The sequence \n does what?	Makes a link	Prints a backslash followed by a n	Adds 5 spaces	Starts a new line	Starts a new line
20	Python is a _____ programming language	higher-level	lowe level	mid level	first level	higher-level
21	An _____ translates a source file into machine language as the program executes	complier	interpreter	editor	none	interpreter
22	A _____ translates a source file into an executable file	compiler	interpreter	editor	none	compiler
23	Messages can be printed in the output window by using Python's _____ function.	scan	edit	print	none	print
24	Python is a _____ language	case insensitive	case sensitive	character	none	case sensitive
25	The _____ function enables a Python program to display textual information to the user	scan	edit	print	input	print
26	Programs may use the _____ function to obtain information from the user	scan	edit	print	input	input
27	Python does not permit _____ to be used when expressing numeric literals	commas	quote	questionmark	arrow	commas
28	The statement a = b copies the value stored in _____	variable a into variable b	variable b into variable a	error	none	variable b into variable a
29	The _____ function accepts an optional prompt string.	scan	edit	print	input	input

30	The _____ function can be used to convert a string representing a numeric expression into its evaluated numeric value.	scan	eval	print	input	eval
31	Python was created by__	James Gosling	Bill Gates	Steve Jobs	Guido van Rossum	Guido van Rossum
32	To start Python from the command prompt, use the command _____.	execute python	run proram	pyhton	go pyhton	pyhton
33	To run python script file named t.py, use the command _____.	execute python t.py	run python t.py	python t.py	go python t.py	python t.py
34	A Python line comment begins with _____.	//	/*	#	@	#
35	A Python paragraph comment uses the style _____.	// comments //	/* comments */	" comments "	/# comments #/	" comments "
36	In Python, a syntax error is detected by the _____ at _____.	compiler/at compile time	interpreter/at runtime	compiler/at runtime	interpreter/at compile time	interpreter/at runtime
37	_____ is the code in natural language mixed with some program code.	Python program	A Python statement	Pseudocode	A flowchart diagram	Pseudocode
38	2 ** 3 evaluates to _____.	9	8	9.1	8.1	8
39	The following is NOT an example of a data type.	int	public	double	void	public
40	The _____ statement terminates the loop containing it.	break	continue	pass	stop	break
41	The _____ statement is used to skip the rest of the code inside a loop for the current iteration only	break	continue	pass	stop	continue

42	Which of the following keyword is a valid placeholder for body of the function ?	break	continue	pass	body	pass
----	--	-------	----------	------	------	------

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Numbers: Introduction- Integer-Floating Point-Complex numbers-Operators-Other numeric type. **Strings**-Strings and Operator-String only operator- Built-in-Functions-Built-in-Methods-String Features. **List** : Operators-Built-in-Functions-Built-in-Methods-Features of List

Numbers:

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Number objects are created when you assign a value to them.

For example

```
var1 = 1  
var2 = 10
```

You can also delete the reference to a number object by using the **del** statement. The syntax of the del statement is –

```
del var1[,var2[,var3[....,varN]]]
```

You can delete a single object or multiple objects by using the **del** statement. For example –

```
del var  
del var_a, var_b
```

Python supports four different numerical types –

- **int (signed integers)** – They are often called just integers or ints, are positive or negative whole numbers with no decimal point.
- **long (long integers)** – Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
- **float (floating point real values)** – Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 ($2.5e2 = 2.5 \times 10^2 = 250$).

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

- **complex (complex numbers)** – are of the form $a + bJ$, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a , and the imaginary part is b . Complex numbers are not used much in Python programming.

Examples

Here are some examples of numbers

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEL	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L .
- A complex number consists of an ordered pair of real floating point numbers denoted by $a + bj$, where a is the real part and b is the imaginary part of the complex number.

Number Type Conversion

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

- Type **int(x)** to convert x to a plain integer.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

- Type **long(x)** to convert x to a long integer.
- Type **float(x)** to convert x to a floating-point number.
- Type **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Add Two Numbers Provided by The User

```
# Store input numbers
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')
# Add two numbers
sum = float(num1) + float(num2)
# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Output

```
Enter first number: 1.5
Enter second number: 6.3
The sum of 1.5 and 6.3 is 7.8
```

Complex Number

A long time ago, mathematicians were absorbed by the following equation:

$$x^2 = -1$$

The reason for this is that any real number (positive or negative) multiplied by itself results in a positive number. How can you multiply any number with itself to get a negative number? No such real number exists. So in the eighteenth century, mathematicians invented something called an imaginary number I (or) j.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

Python complex() Function**Definition and Usage**

The `complex()` function returns a complex number by specifying a real number and an imaginary number.

Syntax

`complex(real, imaginary)`

Parameter Values

Parameter	Description
real	Required. A number representing the real part of the complex number. Default 0. The real number can also be a String, like this '3+5j', when this is the case, the second parameter should be omitted.
imaginary	Optional. A number representing the imaginary part of the complex number. Default 0.

Example

Convert the number 3 and imaginary number 5 into a complex number:

```
x = complex(3, 5)
```

More Examples

```
x = complex('3+5j')
```

```
print(x)
```

OUTPUT: (3+5j)

PYTHON STRINGS

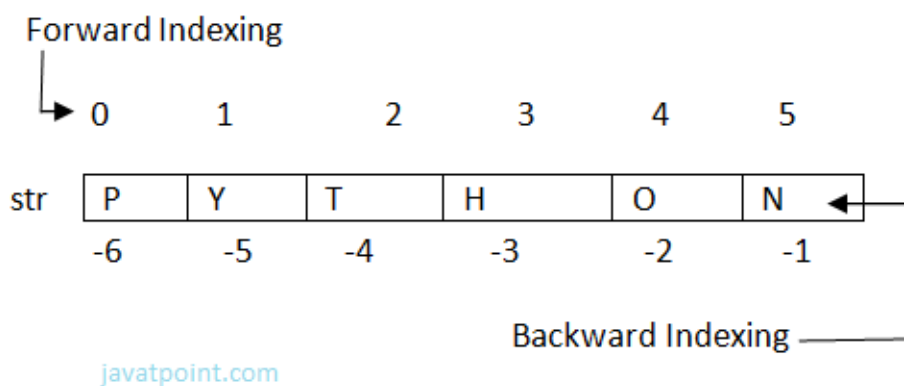
Python string is a built-in type text sequence. It is used to handle textual data in python. Python Strings are immutable sequences of Unicode points. Creating Strings are simplest and easy to use in Python.

We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

Accessing Python Strings

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions (forward and backward).
- Both forward as well as backward indexing are provided using Strings in Python.
 - Forward indexing starts with 0,1,2,3,....
 - Backward indexing starts with -1,-2,-3,-4,....

Example



`str[0]='P'=str[-6] , str[1]='Y' = str[-5] , str[2] = 'T' = str[-4] , str[3] = 'H' = str[-3]`

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

str[4] = 'O' = str[-2] , str[5] = 'N' = str[-1].

Python String Example

Here, we are creating a simple program to retrieve String in reverse as well as normal form.

```
name="Rajat"
length=len(name)
i=0
for n in range(-1,(-length-1),-1):
    print name[i],"\t",name[n]
    i+=1
```

**Output:**

```
>>>
R      t
a      a
j      j
a      a
t      R
>>>
```

Python Strings Operators

To perform operation on string, Python provides basically 3 types of Operators that are given below.

1. Basic Operators.
2. Membership Operators.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

3. Relational Operators.**Python String Basic Operators**

There are two types of basic operators in String "+" and "*".

String Concatenation Operator (+)

The concatenation operator (+) concatenates two Strings and creates a new String.

Python String Concatenation Example

```
>>> "ratan" + "jaiswal"
```

Output:

```
'ratanjaiswal'
```

```
>>>
```

Expression	Output
'10' + '20'	'1020'
"s" + "007"	's007'
'abcd123' + 'xyz4'	'abcd123xyz4'

Eg:

```
'abc' + 3
```

```
>>>
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

output:

```
Traceback (most recent call last):
  File "", line 1, in
    'abc' + 3
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Python String Replication Operator (*)

Replication operator uses two parameters for operation, One is the integer value and the other one is the String argument.

The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

Python String Replication Example

```
>>> 5*"Vimal"
```

Output:

```
'VimalVimalVimalVimalVimal'
```

Expression	Output
"soono"*2	'soonosoono'
3*'1'	'111'

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

'\$'*5	'\$\$\$\$\$'
--------	--------------

Python String Membership Operators

Membership Operators are already discussed in the Operators section. Let see with context of String.

There are two types of Membership operators

1) **in:**"in" operator returns true if a character or the entire substring is present in the specified string, otherwise false.

2) **not in:**"not in" operator returns true if a character or entire substring does not exist in the specified string, otherwise false.

Python String membership operator Example

```
>>> str1="javatpoint"
```

```
>>> str2='sssit'
```

```
>>> str3="seomount"
```

```
>>> str4='java'
```

```
>>> st5="it"
```

```
>>> str6="seo"
```

```
>>> str4 in str1
```

```
True
```

```
>>> str5 in str2
```

```
>>> st5 in str2
```


CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

True

```
>>> str6 in str3
```

True

```
>>> str4 not in str1
```

False

```
>>> str1 not in str4
```

True

Python Relational Operators

All the comparison (relational) operators i.e., (<,><=,>=,==,!=,<>) are also applicable for strings. The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).

Python Relational Operators Example

```
>>> "RAJAT"=="RAJAT"
```

True

```
>>> "afsha">='Afsha'
```

True

```
>>> "Z"<"z"
```

True

Explanation:

The ASCII value of a is 97, b is 98, c is 99 and so on. The ASCII value of A is 65,B is 66,C is 67 and so on. The comparison between strings are done on the basis on ASCII value.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

Python String Slice Notation

Python String slice can be defined as a substring which is the part of the string. Therefore further substring can be obtained from a string.

There can be many forms to slice a string, as string can be accessed or indexed from both the direction and hence string can also be sliced from both the directions.

Python String Slice Syntax

<string_name>[startIndex:endIndex],

<string_name>[:endIndex],

<string_name>[startIndex:]

Python String Slice Example 1

```
>>> str="Nikhil"
```

```
>>> str[0:6]
```

```
'Nikhil'
```

```
>>> str[0:3]
```

```
'Nik'
```

```
>>> str[2:5]
```

```
'khi'
```

```
>>> str[:6]
```

```
'Nikhil'
```

```
>>> str[3:]
```

```
'hil'
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

String slice can also be used with Concatenation operator to get whole string.

Python String Slice Example 2

```
>>> str="Mahesh"
```

```
>>> str[:6]+str[6:]
```

```
'Mahesh'
```

//here 6 is the length of the string.

Python String Functions and Methods

Python provides various predefined or built-in string functions. They are as follows:

capitalize()	It capitalizes the first character of the String.
count(string,begin,end)	It Counts number of times substring occurs in a String between begin and end index.
endswith(suffix ,begin=0,end=n)	It returns a Boolean value if the string terminates with given suffix between begin and end.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found and works same as find() method.
isalnum()	It returns True if characters in the string are alphanumeric

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

	i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.
isalpha()	It returns True when all the characters are alphabets and there is at least one character, otherwise False.
isdigit()	It returns True if all the characters are digit and there is at least one character, otherwise False.
islower()	It returns True if the characters of a string are in lower case, otherwise False.
isupper()	It returns False if characters of a string are in Upper case, otherwise False.
isspace()	It returns True if the characters of a string are whitespace, otherwise false.
len(string)	It returns the length of a string.
lower()	It converts all the characters of a string to Lower case.
upper()	It converts all the characters of a string to Upper Case.
startswith(str ,begin=0,end=n)	It returns a Boolean value if the string starts with given str between begin and end.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

swapcase()	It inverts case of all characters in a string.
lstrip()	It removes all leading whitespace of a string and can also be used to remove particular character from leading.
rstrip()	It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

Python String capitalize() Method Example

This method capitalizes the first character of the String.

```
>>> 'abc'.capitalize()
```

Output: 'Abc'

Python String count(string) Method Example

This method counts number of times substring occurs in a String between begin and end index.

```
msg = "welcome to sssit";  
substr1 = "o";  
print msg.count(substr1, 4, 16)  
substr2 = "t";  
print msg.count(substr2)
```

Output:

```
>>>
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
2
2
>>>
```

Python String endswith(string) Method Example

This method returns a Boolean value if the string terminates with given suffix between begin and end.

```
string1="Welcome to SSSIT";

substring1="SSSIT";

substring2="to";

substring3="of";

print string1.endswith(substring1);

print string1.endswith(substring2,2,16);

print string1.endswith(substring3,2,19);

print string1.endswith(substring3);
```

Output:

```
>>>
True
False
False
False
>>>
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Python String find(string) Method Example

This method returns the index value of the string where substring is found between begin index and end index.

```
str="Welcome to SSSIT";  
substr1="come";  
substr2="to";  
print str.find(substr1);  
print str.find(substr2);  
print str.find(substr1,3,10);  
print str.find(substr2,19);
```

Output:

```
>>>  
3  
8  
3  
-1  
>>>
```

Python String index() Method Example

This method returns the index value of the string where substring is found between begin index and end index.

```
str="Welcome to world of SSSIT";  
substr1="come";
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
substr2="of";

print str.index(substr1);

print str.index(substr2);

print str.index(substr1,3,10);

print str.index(substr2,19);
```

Output:

```
>>>
3
17
3
Traceback (most recent call last):
  File "C:/Python27/fin.py", line 7, in
    print str.index(substr2,19);
ValueError: substring not found
>>>
```

Python String isalnum() Method Example

This method returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.

```
str="Welcome to sssit";

print str.isalnum();

str1="Python47";

print str1.isalnum();
```

Output:

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
>>>
False
True
>>>
```

Python String isalpha() Method Example

It returns True when all the characters are alphabets and there is at least one character, otherwise False.

```
string1="HelloPython"; # Even space is not allowed
```

```
print string1.isalpha();
```

```
string2="This is Python2.7.4"
```

```
print string2.isalpha();
```

Output:

```
>>>
True
False
>>>
```

Python String isdigit() Method Example

This method returns True if all the characters are digit and there is at least one character, otherwise False.

```
string1="HelloPython";
```

```
print string1.isdigit();
```

```
string2="98564738"
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
print string2.isdigit();
```

Output:

```
>>>
False
True
>>>
```

Python String islower() Method Example

This method returns True if the characters of a string are in lower case, otherwise False.

```
string1="Hello Python";

print string1.islower();

string2="welcome to "

print string2.islower();
```

Output:

```
>>>
False
True
>>>
```

Python String isupper() Method Example

This method returns False if characters of a string are in Upper case, otherwise False.

```
string1="Hello Python";

print string1.isupper();
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
string2="WELCOME TO"
```

```
print string2.isupper();
```

Output:

```
>>>
False
True
>>>
```

Python String isspace() Method Example

This method returns True if the characters of a string are whitespace, otherwise false.

```
string1="  ";
```

```
print string1.isspace();
```

```
string2="WELCOME TO WORLD OF PYT"
```

```
print string2.isspace();
```

Output:

```
>>>
True
False
>>>
```

Python String len(string) Method Example

This method returns the length of a string.

```
string1="  ";
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
print len(string1);  
  
string2="WELCOME TO SSSIT"  
  
print len(string2);
```

Output:

```
>>>  
4  
16  
>>>
```

Python String lower() Method Example

It converts all the characters of a string to Lower case.

```
string1="Hello Python";  
  
print string1.lower();  
  
string2="WELCOME TO SSSIT"  
  
print string2.lower();
```

Output:

```
>>>  
hello python  
welcome to sssit  
>>>
```

Python String upper() Method Example

This method converts all the characters of a string to upper case.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
string1="Hello Python";  
  
print string1.upper();  
  
string2="welcome to SSSIT"  
  
print string2.upper();
```

Output:

```
>>>  
HELLO PYTHON  
WELCOME TO SSSIT  
>>>
```

Python String startswith(string) Method Example

This method returns a Boolean value if the string starts with given str between begin and end.

```
string1="Hello Python";  
  
print string1.startswith('Hello');  
  
string2="welcome to SSSIT"  
  
print string2.startswith('come',3,7);
```

Output:

```
>>>  
True  
True  
>>>
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Python String swapcase() Method Example

It inverts case of all characters in a string.

```
string1="Hello Python";  
  
print string1.swapcase();  
  
string2="welcome to SSSIT"  
  
print string2.swapcase();
```

Output:

```
>>>  
hELLO pYTHON  
WELCOME TO sssit  
>>>
```

Python String lstrip() Method Example

It removes all leading whitespace of a string and can also be used to remove particular character from leading.

```
string1="  Hello Python";  
  
print string1.lstrip();  
  
string2="@@@@@welcome to SSSIT"  
  
print string2.lstrip('@');
```

Output:

```
>>>  
Hello Python
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
welcome to world to SSSIT
```

```
>>>
```

Python String `rstrip()` Method Example

It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

```
string1=" Hello Python  ";
```

```
print string1.rstrip();
```

```
string2="@welcome to SSSIT!!!"
```

```
print string2.rstrip('!');
```



Output:

```
>>>
```

```
    Hello Python
```

```
@welcome to SSSIT
```

```
>>>
```

Python List

Python list is a data structure which is used to store various types of data.

In Python, lists are mutable i.e., Python will not create a new list if we modify an element of the list.

It works as a container that holds other objects in a given order. We can perform various operations like insertion and deletion on list.

A list can be composed by storing a sequence of different type of values separated by commas.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Python list is enclosed between square([]) brackets and elements are stored in the index basis with starting index 0.

Python List Example

```
data1=[1,2,3,4];
data2=['x','y','z'];
data3=[12.5,11.6];
data4=['raman','rahul'];
data5=[];
data6=['abhinav',10,56.4,'a'];
```

A list can be created by putting the value inside the square bracket and separated by comma.

Python List Syntax

```
<list_name>=[value1,value2,value3,...,valuen];
```

Syntax to Access Python List

```
<list_name>[index]
```

Python allows us to access value from the list by various ways.

Python Accessing List Elements Example

```
data1=[1,2,3,4];
data2=['x','y','z'];
print data1[0]
print data1[0:2]
print data2[-3:-1]
print data1[0:]
print data2[:2]
```


CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

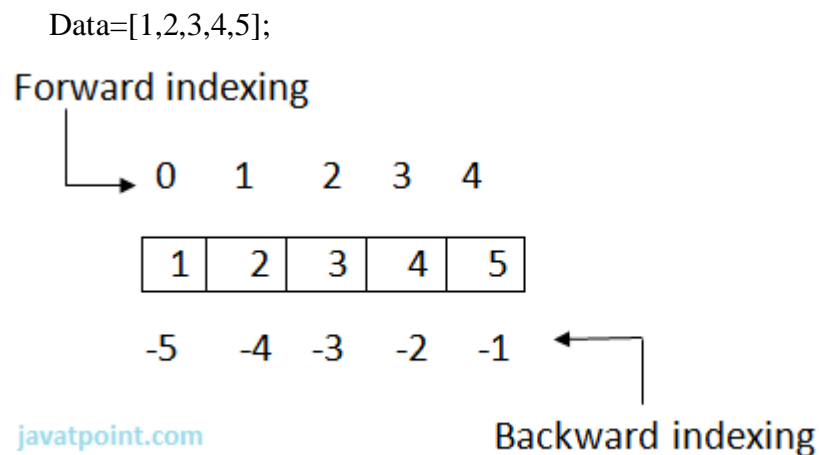
UNIT II

Output:

```
1
[1, 2]
['x', 'y']
[1, 2, 3, 4]
['x', 'y']
>>>
```

Elements in a Lists:

Following are the pictorial representation of a list. We can see that it allows to access elements from both end (forward and backward).



Data[0]=1=Data[-5] , Data[1]=2=Data[-4] , Data[2]=3=Data[-3] ,
=4=Data[-2] , Data[4]=5=Data[-1].

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Python List Operations

Apart from creating and accessing elements from the list, Python allows us to perform various other operations on the list. Some common operations are given below

a) Adding Python Lists

In Python, lists can be added by using the concatenation operator(+) to join two lists.

Add lists Example 1

```
list1=[10,20]
list2=[30,40]
list3=list1+list2
print list3
```

Output: [10, 20, 30, 40]

Add lists Example 2

```
list1=[10,20]
list1+30
print list1
```

Output:

Traceback (most recent call last):

```
File "C:/Python27/lis.py", line 2, in <module>
list1+30
```

b) Python Replicating lists

Replicating means repeating, It can be performed by using '*' operator by a specific number of time.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

Python list Replication Example

```
list1=[10,20]
print list1*1
```

Output:[10, 20]**c)Python List Slicing**

A subpart of a list can be retrieved on the basis of index. This subpart is known as list slice. This feature allows us to get sub-list of specified start and end index.

Python List Slicing Example

```
list1=[1,2,4,5,7]
print list1[0:2]
print list1[4]
list1[1]=9
print list1
```

Output:

```
[1, 2]
7
[1, 9, 4, 5, 7]
```

Python List Other Operations

Apart from above operations various other functions can also be performed on List such as Updating, Appending and Deleting elements from a List.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

Python Updating List

To update or change the value of particular index of a list, assign the value to that particular index of the List.

Python Updating List Example

```
data1=[5,10,15,20,25]
print "Values of list are: "
print data1
data1[2]="Multiple of 5"
print "Values of list are: "
print data1
```

Output:

```
Values of list are:
[5, 10, 15, 20, 25]
Values of list are:
[5, 10, 'Multiple of 5', 20, 25]
```

Appending Python List

Python provides, append() method which is used to append i.e., add an element at the end of the existing elements.

Python Append List Example

```
list1=[10,"rahul",'z']
print "Elements of List are: "
print list1
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
list1.append(10.45)
print "List after appending: "
print list1
```

Output:

```
Elements of List are:
[10, 'rahul', 'z']
List after appending:
[10, 'rahul', 'z', 10.45]
```

Deleting Elements

In Python, **del** statement can be used to delete an element from the list. It can also be used to delete all items from startIndex to endIndex.

Python delete List Example

```
list1=[10,'rahul',50.8,'a',20,30]
print list1
del list1[0]
print list1
del list1[0:3]
print list1
```

Output:

```
[10, 'rahul', 50.8, 'a', 20, 30]
['rahul', 50.8, 'a', 20, 30]
[20, 30]
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

Python lists Method

Python provides various Built-in functions and methods for Lists that we can apply on the list.

Following are the common list functions.

Function	Description
min(list)	It returns the minimum value from the list given.
max(list)	It returns the largest value from the given list.
len(list)	It returns number of elements in a list.
cmp(list1,list2)	It compares the two list.
list(sequence)	It takes sequence types and converts them to lists.

Python List min() method Example

This method is used to get min value from the list.

```
list1=[101,981,'abcd','xyz','m']
list2=['aman','shekhar',100.45,98.2]
print "Minimum value in List1: ",min(list1)
print "Minimum value in List2: ",min(list2)
```

Output:

```
Minimum value in List1: 101
Minimum value in List2: 98.2
```

Python List max() method Example

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

This method is used to get max value from the list.

```
list1=[101,981,'abcd','xyz','m']
list2=['aman','shekhar',100.45,98.2]
print "Maximum value in List : ",max(list1)
print "Maximum value in List : ",max(list2)
```

Output:

```
Maximum value in List : xyz
Maximum value in List : shekhar
```

Python List len() method Example

This method is used to get length of the the list.

```
list1=[101,981,'abcd','xyz','m']
list2=['aman','shekhar',100.45,98.2]
print "No. of elements in List1: ",len(list1)
print "No. of elements in List2: ",len(list2)
```

Output:

```
No. of elements in List1 : 5
No. of elements in List2 : 4
```

Python List cmp() method Example

Explanation: If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

- If numbers, perform comparison.
- If either element is a number, then the other element is returned.
- Otherwise, types are sorted alphabetically .

If we reached the end of one of the lists, the longer list is "larger." If both list are same it returns 0.

Python List cmp() method Example

```
list1=[101,981,'abcd','xyz','m']  
list2=['aman','shekhar',100.45,98.2]  
list3=[101,981,'abcd','xyz','m']  
print cmp(list1,list2)  
print cmp(list2,list1)  
print cmp(list3,list1)
```

Output:

```
-1  
1  
0
```

Python List list(sequence) method Example

This method is used to form a list from the given sequence of elements.

```
seq=(145,"abcd",'a')  
data=list(seq)  
print "List formed is : ",data
```


CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT II

Output:

List formed is : [145, 'abcd', 'a']

There are following built-in methods of List

Methods	Description
index(object)	It returns the index value of the object.
count(object)	It returns the number of times an object is repeated in list.
pop()/pop(index)	It returns the last object or the specified indexed object. It removes the popped object.
insert(index,object)	It inserts an object at the given index.
extend(sequence)	It adds the sequence to existing list.
remove(object)	It removes the object from the given List.
reverse()	It reverses the position of all the elements of a list.
sort()	It is used to sort the elements of the List.

Python List index() Method Example

```
data = [786,'abc','a',123.5]
print "Index of 123.5:", data.index(123.5)
print "Index of a is", data.index('a')
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Output:

Index of 123.5 : 3

Index of a is 2

Python List count(object) Method Example

```
data = [786,'abc','a',123.5,786,'rahul','b',786]
print "Number of times 123.5 occurred is", data.count(123.5)
print "Number of times 786 occurred is", data.count(786)
```

Output:

Number of times 123.5 occurred is 1

Number of times 786 occurred is 3

Python List pop()/pop(int) Method Example

```
data = [786,'abc','a',123.5,786]
print "Last element is", data.pop()
print "2nd position element:", data.pop(1)
print data
```

Output:

Last element is 786

2nd position element:abc

[786, 'a', 123.5]

Python List insert(index,object) Method Example

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

```
data=['abc',123,10.5,'a']  
data.insert(2,'hello')  
print data
```

Output:

```
['abc', 123, 'hello', 10.5, 'a']
```

Python List extend(sequence) Method Example

```
data1=['abc',123,10.5,'a']  
data2=['ram',541]  
data1.extend(data2)  
print data1  
print data2
```

Output:

```
['abc', 123, 10.5, 'a', 'ram', 541]  
['ram', 541]
```

Python List remove(object) Method Example

```
data1=['abc',123,10.5,'a','xyz']  
data2=['ram',541]  
print data1  
data1.remove('xyz')  
print data1  
print data2  
data2.remove('ram')  
print data2
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

Output:

```
['abc', 123, 10.5, 'a', 'xyz']
```

```
['abc', 123, 10.5, 'a']
```

```
['ram', 541]
```

```
[541]
```

Python List reverse() Method Example

```
list1=[10,20,30,40,50]
```

```
list1.reverse()
```

```
print list1
```

Output:

```
[50, 40, 30, 20, 10]
```

Python List sort() Method Example

```
list1=[10,50,13,'rahul','aakash']
```

```
list1.sort()
```

```
print list1
```

Output:

```
[10, 13, 50, 'aakash', 'rahul']
```

Possible Questions:

2 Mark Questions

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT II

1. Define string. How to get a string at run time?
2. Name the type of Boolean operators.
3. What is complex numbers?
4. Define Python list.
5. What are the list operations?
6. What are the different ways to create a list?
7. Illustrate negative indexing in list with an example.
8. Describe list slicing with examples.
9. Give the use of return () statement with a suitable example.

6 Mark Questions

1. What is Python list? Explain the basic list operations with suitable examples.
2. Discuss numbers and its methods in python.
3. Define string. How to get a string at run time? Explain with example.
4. Discuss numbers and its operators.
5. Describe the following
 - a) Creating the List
 - b) Accessing values in the Lists
 - c) Updating the Lists
 - d) Deleting the list Elements
6. Write a Python program to multiply two Matrices
7. Define methods in a string with an example program using at least five methods.
8. How to access characters of a string?



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS)

(BATCH 2019 - 2021)

I SEMESTER

PYTHON PROGRAMMING (19CSP101)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT II

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	The following is NOT an example of a data type.	int	public	double	void	public
2	Identifiers must contain at least ____character	one	two	three	four	one
3	A _____ word cannot be used as an identifier	variable	reserved	string	token	reserved
4	_____ are identifiers that have predefined meanings in Python	Keywords	string	variable	token	Keywords
5	A _____ operator performs an operation using one operand	binary	unary	trinary	none	unary
6	_____ expression, sometimes called a predicate	bitwise	assignemnt	Boolean	arithmetic	Boolean

7	In Python, a string literal is enclosed in _____.	parentheses	brackets	single-quotes	braces	single-quotes
8	Suppose x is a char variable with a value 'b'. What will be displayed by the statement <code>print(chr(ord(x) + 1))</code> ?	a	b	c	d	c
9	What is <code>chr(ord('B'))</code> ?	A	B	C	D	B
10	Which of the following statement prints <code>smith\exam1\test.txt</code> ?	<code>print("smith\exam1\test.txt")</code>	<code>print("smith\\exam1\\test.txt")</code>	<code>print("smith\"exam1\"test.txt")</code>	<code>print("smith"\exam1"\test.txt")</code>	<code>print("smith\\exam1\\test.txt")</code>
11	The Unicode of 'a' is 97. What is the Unicode for 'c'?	96	97	98	99	99
12	Suppose <code>s = "Welcome"</code> , what is <code>type(s)</code> ?	int	float	string	str	str
13	The <code>format</code> function returns _____.	an int	a float	a str	a chat	a str
14	Which of the following operators are right-associative.	=	*	+	—	=
15	Assume <code>x = 4</code> and <code>y = 5</code> , Which of the following is true?	<code>not (x == 4)</code>	<code>x != 4</code>	<code>x == 5</code>	<code>x != 5</code>	<code>x != 5</code>
16	Which operator is overloaded by the <code>or()</code> function?			//	/	

17	What is the output of the following program : i = 0 while i < 3: print i i++ print i+1	0 2 1 3 2 4	0 1 2 3 4 5	Error	1 0 2 4 3 5	Error
18	Which function overloads the >> operator?	more()	gt()	ge()	None of the above	None of the above
19	_____ creates a list.	list1 = list()	list1 = []	list1 = list([12, 4, 4])	list1 = [12, 4, 4]	All
20	Which of the following statements is used to create an empty set?	{ }	set()	[]	()	set()

21	What is the output of the following piece of code when executed in the python shell? <code>a={1,2,3}</code>	<code>{2,3}</code>	Error, duplicate item present in list	Error, no method called <code>intersection_update</code>	<code>{1,4,5}</code>	<code>{2,3}</code>
22	Which of the following lines of code will result in an error?	<code>s={abs}</code>	<code>s={4, 'abc', (1,2)}</code>	<code>s={2, 2.2, 3, 'xyz'}</code>	<code>s={san}</code>	<code>s={san}</code>
23	What is the output of the line of code shown below, if <code>s1= {1, 2, 3}</code> ? <code>s1.issubset(s1)</code>	TRUE	Error	No output	FALSE	TRUE
24	What is the output of the code shown below? <code>s=set([1, 2, 3])</code> <code>s.union([4, 5])</code> <code>s ([4, 5])</code>	<code>{1, 2, 3, 4, 5}</code> <code>{1, 2, 3, 4, 5}</code>	Error <code>{1, 2, 3, 4, 5}</code>	<code>{1, 2, 3, 4, 5}</code> Error	ErrorError	<code>{1, 2, 3, 4, 5}</code> Error
25	Which of the following function capitalizes first letter of string?	<code>shuffle(lst)</code>	<code>capitalize()</code>	<code>isalnum()</code>	<code>isdigit()</code>	<code>capitalize()</code>
26	Which of the following function checks in a string that all characters are digits?	<code>shuffle(lst)</code>	<code>capitalize()</code>	<code>isalnum()</code>	<code>isdigit()</code>	<code>isdigit()</code>
27	Which of the following function convert an integer to octal string in python?	<code>unichr(x)</code>	<code>ord(x)</code>	<code>hex()</code>	<code>oct(x)</code>	<code>oct(x)</code>
28	What is the name of data type for character in python ?	char	python do not have any data type for characters	charcter	chr	python do not have any data type for characters
29	In python 3 what does <code>//</code> operator do ?	Float division	Integer division	returns remainder	same as <code>a**b</code>	Integer division

30	What is "Programming is fun"[4: 6]?	ram	ra	r	pr	ra
31	What is "Programming is fun"[-1]?	pr	ram	ra	n	n
32	What is "Programming is fun"[1:1]?	pr	p	r	' '	' '
33	Given a string s = "Welcome", which of the following code is incorrect?	print(s[0])	print(s.lower())	s[1] = 'r'	print(s.strip())	s[1] = 'r'
34	Given a string s = "Programming is fun", what is s.find('ram')?	1	2	3	4	4
35	Given a string s = "Programming is fun", what is s.startswith('Program')?	0	1	TRUE	FALSE	TRUE
36	A function is:	An entity that receives inputs and outputs	A way of storing values	A sequence of characters enclosed by quotes	A kind of computer	An entity that receives inputs and outputs
37	The _____ operator is a string operator called the format operator	&	*	%	#	%
38	_____ function you can iterate through the sequence and retrieve the index position and its corresponding value at the same time.	enumerate()	enum()	e()	eindex()	enumerate()
39	_____ finds all the occurrences of match and return them as an iterator.	find()	finditer()	replace()	check()	finditer()
40	Invoking the _____ method converts raw byte data to a string.	encode()	decode()	convert()	toString()	decode()

41	The readlines() method returns a _____.	str	a list of lines	a list of single characters	a list of integers	a list of lines
42	The keyword _____ is required to define a class.	def	return	class	all	class
43	_____ terminates the process normally.	abort()	exit()	assert()	all	exit()
44	_____ is interpreted.	python	c++	ada	c	python
45	The _____ function immediately terminates the program.	sys.terminate()	sys.halt()	sys.exit()	sys.stop()	sys.exit()

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Tuple : Introduction- Operators and Built-in-Functions-Features. **Mapping and set type Dictionaries**-Operators-Built-in and Factory Functions-Built-in- Methods. **Set type:** Introduction- Operators-Built-in Function-Built-in Methods-

Python Tuple

A tuple is a sequence of immutable objects, therefore tuple cannot be changed. It can be used to collect different types of object.

The objects are enclosed within parenthesis and separated by comma.

Tuple is similar to list. Only the difference is that list is enclosed between square bracket, tuple between parenthesis and List has mutable objects whereas Tuple has immutable objects.

Python Tuple Example

```
>>> data=(10,20,'ram',56.8)
>>> data2="a",10,20.9
>>> data
(10, 20, 'ram', 56.8)
>>> data2
('a', 10, 20.9)
>>>
```

There can be an empty Tuple also which contains no object. Lets see an example of empty tuple.

Python Empty Tuple Example

```
tuple1=()
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Python Single Object Tuple Example

For a single valued tuple, there must be a comma at the end of the value.

```
Tuple1=(10,)
```

Python Tuple of Tuples Example

Tuples can also be nested, it means we can pass tuple as an element to create a new tuple. See, the following example in which we have created a tuple that contains tuples and the object.

```
tupl1='a','mahesh',10.56
tupl2=tupl1,(10,20,30)
print tupl1
print tupl2
```

Output:

```
>>>
('a', 'mahesh', 10.56)
(('a', 'mahesh', 10.56), (10, 20, 30))
>>>
```

Accessing Tuple

Accessing of tuple is pretty easy, we can access tuple in the same way as List. See, the following example.

Accessing Tuple Example

```
data1=(1,2,3,4)
data2=('x','y','z')
print data1[0]
print data1[0:2]
print data2[-3:-1]
```

```
print data1[0:]  
print data2[:2]
```

Output:

```
>>>  
1  
(1, 2)  
(x, 'y')  
(1, 2, 3, 4)  
(x, 'y')  
>>>
```

Elements in a Tuple

Data=(1,2,3,4,5,10,19,17)

Forward indexing

0 1 2 3 4 5 6 7

1	2	3	4	5	10	19	17
---	---	---	---	---	----	----	----

-8 -7 -6 -5 -4 -3 -2 -1

javatpoint.com

Backward index

Replicating Tuple Example

Replicating means repeating. It can be performed by using '*' operator by a specific number of time.

```
tuple1=(10,20,30);
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
tuple2=(40,50,60);  
print tuple1*2  
print tuple2*3
```

Output:

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

Python Tuple Slicing Example

A subpart of a tuple can be retrieved on the basis of index. This subpart is known as tuple slice.

```
data1=(1,2,4,5,7)  
print data1[0:2]  
print data1[4]  
print data1[: -1]  
print data1[-5:]  
print data1
```

Output:

```
>>>  
(1, 2)  
7  
(1, 2, 4, 5)  
(1, 2, 4, 5, 7)  
(1, 2, 4, 5, 7)  
>>>
```

Data[0]=1=Data[-8] , Data[1]=2=Data[-7] , Data[2]=3=Data[-6] ,

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT III

```
Data[3]=4=Data[-5] , Data[4]=5=Data[-4] , Data[5]=10=Data[-3],  
Data[6]=19=Data[-2],Data[7]=17=Data[-1]
```

Python Tuple Operations

Python allows us to perform various operations on the tuple. Following are the common tuple operations.

Adding Tuples Example

Tuple can be added by using the concatenation operator(+) to join two tuples.

```
data1=(1,2,3,4)  
data2=('x','y','z')  
data3=data1+data2  
print data1  
print data2  
print data3
```

Output:

```
>>>  
(1, 2, 3, 4)  
( 'x', 'y', 'z')  
(1, 2, 3, 4, 'x', 'y', 'z')  
>>>
```

Python Tuple other Operations**Updating elements in a List**

Elements of the Tuple cannot be updated. This is due to the fact that Tuples are immutable. Whereas the Tuple can be used to form a new Tuple.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Example

```
data=(10,20,30)
data[0]=100
print data
```

Output:

```
>>>
Traceback (most recent call last):
  File "C:/Python27/t.py", line 2, in
    data[0]=100
TypeError: 'tuple' object does not support item assignment
>>>
```

Creating Tuple from Existing Example

We can create a new tuple by assigning the existing tuple, see the following example.

```
data1=(10,20,30)
data2=(40,50,60)
data3=data1+data2
print data3
```

Output:

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

Python Tuple Deleting Example

Deleting individual element from a tuple is not supported. However the whole of the tuple can be deleted using the del statement.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT III

```
data=(10,20,'rahul',40.6,'z')
print data
del data    #will delete the tuple data
print data  #will show an error since tuple data is already deleted
```

Output:

```
>>>
(10, 20, 'rahul', 40.6, 'z')
Traceback (most recent call last):
  File "C:/Python27/t.py", line 4, in
    print data
NameError: name 'data' is not defined
>>>
```

Functions of Tuple

There are following in-built Type Functions

Function	Description
min(tuple)	It returns the minimum value from a tuple.
max(tuple)	It returns the maximum value from the tuple.
len(tuple)	It gives the length of a tuple
cmp(tuple1,tuple2)	It compares the two Tuples.
tuple(sequence)	It converts the sequence into tuple.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Python Tuple min(tuple) Method Example

This method is used to get min value from the sequence of tuple.

```
data=(10,20,'rahul',40.6,'z')
print min(data)
```

Output:

```
>>>
10
>>>
```

Python Tuple max(tuple) Method Example

This method is used to get max value from the sequence of tuple.

```
data=(10,20,'rahul',40.6,'z')
print max(data)
```

Output:

```
>>>
z
>>>
```

Python Tuple len(tuple) Method Example

This method is used to get length of the tuple.

```
data=(10,20,'rahul',40.6,'z')
print len(data)
```

Output:

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
>>>
5
>>>
```

Python Tuple cmp(tuple1,tuple2) Method Example

This method is used to compare tuples.

Explanation: If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers.

- If numbers, perform comparison.
- If either element is a number, then the other element is returned.
- Otherwise, types are sorted alphabetically .

If we reached the end of one of the lists, the longer list is "larger." If both list are same it returns 0.

```
data1=(10,20,'rahul',40.6,'z')
data2=(20,30,'sachin',50.2)
print cmp(data1,data2)
print cmp(data2,data1)
data3=(20,30,'sachin',50.2)
print cmp(data2,data3)
```

Output:

```
>>>
-1
1
0
>>>
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

5) tuple(sequence):

Eg:

```
dat=[10,20,30,40]
data=tuple(dat)
print data
```

Output:

```
>>>
(10, 20, 30, 40)
>>>
```

Why should we use Tuple? (Advantages of Tuple)

1. Processing of Tuples are faster than Lists.
2. It makes the data safe as Tuples are immutable and hence cannot be changed.
3. Tuples are used for String formatting.

Mapping and set type

Dictionaries

Dictionary is an unordered set of key and value pair. It is a container that contains data, enclosed within curly braces.

The pair i.e., key and value is known as item. The key passed in the item must be unique.

The key and the value is separated by a colon(:). This pair is known as item. Items are separated from each other by a comma(.). Different items are enclosed within a curly brace and this forms Dictionary.

Python Dictionary Example

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
data={ 100:'Ravi' ,101:'Vijay' ,102:'Rahul'}  
print data
```

Output:

```
>>>  
{100: 'Ravi', 101: 'Vijay', 102: 'Rahul'}  
>>>
```

Note:

Dictionary is mutable i.e., value can be updated.

Key must be unique and immutable. Value is accessed by key. Value can be updated while key cannot be changed.

Dictionary is known as Associative array since the Key works as Index and they are decided by the user.

Python Dictionary Example

```
plant={ }  
plant[1]='Ravi'  
plant[2]='Manoj'  
plant['name']='Hari'  
plant[4]='Om'  
print plant[2]  
print plant['name']  
print plant[1]  
print plant
```

Output:

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT III

```
>>>
Manoj
Hari
Ravi
{1: 'Ravi', 2: 'Manoj', 4: 'Om', 'name': 'Hari'}
>>>
```

Accessing Dictionary Values

Since Index is not defined, a Dictionary values can be accessed by their keys only. It means, to access dictionary elements we need to pass key, associated to the value.

Python Accessing Dictionary Element Syntax

```
<dictionary_name>[key]
</dictionary_name>
```

Accessing Elements Example

```
data1={'Id':100, 'Name':'Suresh', 'Profession':'Developer'}
data2={'Id':101, 'Name':'Ramesh', 'Profession':'Trainer'}
print "Id of 1st employer is",data1['Id']
print "Id of 2nd employer is",data2['Id']
print "Name of 1st employer:",data1['Name']
print "Profession of 2nd employer:",data2['Profession']
```

Output:

```
>>>
Id of 1st employer is 100
Id of 2nd employer is 101
Name of 1st employer is Suresh
Profession of 2nd employer is Trainer
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT III

```
>>>
```

Updating Python Dictionary Elements

The item i.e., key-value pair can be updated. Updating means new item can be added. The values can be modified.

Example

```
data1={'Id':100, 'Name':'Suresh', 'Profession':'Developer'}
data2={'Id':101, 'Name':'Ramesh', 'Profession':'Trainer'}
data1['Profession']='Manager'
data2['Salary']=20000
data1['Salary']=15000
print data1
print data2
```

Output:

```
>>>
{'Salary': 15000, 'Profession': 'Manager', 'Id': 100, 'Name': 'Suresh'}
{'Salary': 20000, 'Profession': 'Trainer', 'Id': 101, 'Name': 'Ramesh'}
>>>
```

Deleting Python Dictionary Elements Example

del statement is used for performing deletion operation.

An item can be deleted from a dictionary using the key only.

Delete Syntax

```
del <dictionary_name>[key]
</dictionary_name>
```


CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT III

Whole of the dictionary can also be deleted using the del statement.

Example

```
data={ 100:'Ram', 101:'Suraj', 102:'Alok'}  
del data[102]  
print data  
del data  
print data #will show an error since dictionary is deleted.
```

Output:

```
>>>  
{100: 'Ram', 101: 'Suraj'}  
  
Traceback (most recent call last):  
  File "C:/Python27/dict.py", line 5, in  
    print data  
NameError: name 'data' is not defined  
>>>
```

Python Dictionary Functions and Methods

Python Dictionary supports the following Functions

Python Dictionary Functions

Functions	Description
len(dictionary)	It returns number of items in a dictionary.
cmp(dictionary1,dictionary2)	It compares the two dictionaries.

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

str(dictionary)

It gives the string representation of a string.

Python Dictionary Methods

Methods	Description
keys()	It returns all the keys element of a dictionary.
values()	It returns all the values element of a dictionary.
items()	It returns all the items(key-value pair) of a dictionary.
update(dictionary2)	It is used to add items of dictionary2 to first dictionary.
clear()	It is used to remove all items of a dictionary. It returns an empty dictionary.
fromkeys(sequence,value1)/ fromkeys(sequence)	It is used to create a new dictionary from the sequence where sequence elements forms the key and all keys share the values ?value1?. In case value1 is not give, it set the values of keys to be none.
copy()	It returns an ordered copy of the data.
has_key(key)	It returns a boolean value. True in case if key is present in the dictionary ,else false.
get(key)	It returns the value of the given key. If key is not present it returns none.

Python Dictionary len(dictionary) Example

It returns length of the dictionary.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
data={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data  
print len(data)
```

Output:

```
>>>  
{100: 'Ram', 101: 'Suraj', 102: 'Alok'}  
3  
>>>
```

Python Dictionary cmp(dictionary1,dictionary2) Example

The comparison is done on the basis of key and value.

```
If, dictionary1 == dictionary2, returns 0.  
dictionary1 < dictionary2, returns -1.  
dictionary1 > dictionary2, returns 1.  
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
data2={103:'abc', 104:'xyz', 105:'mno'}  
data3={'Id':10, 'First':'Aman','Second':'Sharma'}  
data4={100:'Ram', 101:'Suraj', 102:'Alok'}  
print cmp(data1,data2)  
print cmp(data1,data4)  
print cmp(data3,data2)
```

Output:

```
>>>  
-1  
0  
1  
>>>
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Python Dictionary str(dictionary) Example

This method returns string formation of the value.

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print str(data1)
```

Output:

```
>>>  
{100: 'Ram', 101: 'Suraj', 102: 'Alok'}  
>>>
```

Python Dictionary keys() Method Example

This method returns all the keys element of a dictionary.

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data1.keys()
```

Output:

```
>>>  
[100, 101, 102]  
>>>
```

Python Dictionary values() Method Example

This method returns all the values element of a dictionary.

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data1.values()
```

Output:

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
>>>
['Ram', 'Suraj', 'Alok']
>>>
```

Python Dictionary items() Method Example

This method returns all the items(key-value pair) of a dictionary.

```
data1={ 100:'Ram', 101:'Suraj', 102:'Alok'}
print data1.items()
```

Output:

```
>>>
[(100, 'Ram'), (101, 'Suraj'), (102, 'Alok')]
>>>
```

Python Dictionary update(dictionary2) Method Example

This method is used to add items of dictionary2 to first dictionary.

```
data1={ 100:'Ram', 101:'Suraj', 102:'Alok'}
data2={ 103:'Sanjay'}
data1.update(data2)
print data1
print data2
```

Output:

```
>>>
{100: 'Ram', 101: 'Suraj', 102: 'Alok', 103: 'Sanjay'}
{103: 'Sanjay'}
>>>
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Python Dictionary clear() Method Example

It returns an ordered copy of the data.

```
data1={100:'Ram', 101:'Suraj', 102:'Alok'}  
print data1  
data1.clear()  
print data1
```

Output:

```
>>>  
{100: 'Ram', 101: 'Suraj', 102: 'Alok'}  
{}  
>>>
```

Python Dictionary fromkeys(sequence)/ fromkeys(seq,value) Method Example

This method is used to create a new dictionary from the sequence where sequence elements forms the key and all keys share the values ?value1?. In case value1 is not give, it set the values of keys to be none.

```
sequence=('Id' , 'Number' , 'Email')  
data={}  
data1={}  
data=data.fromkeys(sequence)  
print data  
data1=data1.fromkeys(sequence,100)  
print data1
```

Output:

```
>>>  
{'Email': None, 'Id': None, 'Number': None}
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
{'Email': 100, 'Id': 100, 'Number': 100}
>>>
```

Python Dictionary copy() Method Example

This method returns an ordered copy of the data.

```
data={'Id':100 , 'Name':'Aakash' , 'Age':23}
data1=data.copy()
print data1
```

Output:

```
>>>
{'Age': 23, 'Id': 100, 'Name': 'Aakash'}
>>>
```

Python Dictionary has_key(key) Method Example

It returns a boolean value. True in case if key is present in the dictionary, else false.

```
data={'Id':100 , 'Name':'Aakash' , 'Age':23}
print data.has_key('Age')
print data.has_key('Email')
```

Output:

```
>>>
True
False
>>>
```

Python Dictionary get(key) Method Example

This method returns the value of the given key. If key is not present it returns none.

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT III

```
data={'Id':100 , 'Name':'Aakash' , 'Age':23}  
print data.get('Age')  
print data.get('Email')
```

Output:

```
>>>  
23  
None  
>>>
```

Set type

Mathematically a set is a collection of items not in any particular order. A Python set is similar to this mathematical definition with below additional conditions.

The elements in the set cannot be duplicates.

The elements in the set are immutable(cannot be modified) but the set as a whole is mutable.

There is no index attached to any element in a python set. So they do not support any indexing or slicing operation.

Set Operations

The sets in python are typically used for mathematical operations like union, intersection, difference and complement etc. We can create a set, access it's elements and carry out these mathematical operations as shown below.

Creating a set

A set is created by using the set() function or placing all the elements within a pair of curly braces.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
```


CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
Months={"Jan","Feb","Mar"}
```

```
Dates={21,22,17}
```

```
print(Days)
```

```
print(Months)
```

```
print(Dates)
```

When the above code is executed, it produces the following result. Please note how the order of the elements has changed in the result.

```
set(['Wed', 'Sun', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
```

```
set(['Jan', 'Mar', 'Feb'])
```

```
set([17, 21, 22])
```

Accessing Values in a Set

We cannot access individual values in a set. We can only access all the elements together as shown above. But we can also get a list of individual elements by looping through the set.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
```

```
for d in Days:
```

```
    print(d)
```

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

```
Wed
```

```
Sun
```

```
Fri
```

```
Tue
```

```
Mon
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

Thu

Sat

Methods

Adding Items to a Set

We can add elements to a set by using add() method. Again as discussed there is no specific index attached to the newly added element.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])
```

```
Days.add("Sun")
```

```
print(Days)
```

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

```
set(['Wed', 'Sun', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
```

Removing Item from a Set

We can remove elements from a set by using discard() method. Again as discussed there is no specific index attached to the newly added element.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])
```

```
Days.discard("Sun")
```

```
print(Days)
```

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

```
set(['Wed', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
```

Union of Sets

The union operation on two sets produces a new set containing all the distinct elements from both the sets. In the below example the element “Wed” is present in both the sets.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
DaysA = set(["Mon", "Tue", "Wed"])
DaysB = set(["Wed", "Thu", "Fri", "Sat", "Sun"])
AllDays = DaysA|DaysB
print(AllDays)
```

When the above code is executed, it produces the following result. Please note the result has only one “wed”.

```
set(['Wed', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
```

Intersection of Sets

The intersection operation on two sets produces a new set containing only the common elements from both the sets. In the below example the element “Wed” is present in both the sets.

```
DaysA = set(["Mon", "Tue", "Wed"])
DaysB = set(["Wed", "Thu", "Fri", "Sat", "Sun"])
AllDays = DaysA & DaysB
print(AllDays)
```

When the above code is executed, it produces the following result. Please note the result has only one “wed”.

```
set(['Wed'])
```

Difference of Sets

The difference operation on two sets produces a new set containing only the elements from the first set and none from the second set. In the below example the element “Wed” is present in both the sets so it will not be found in the result set.

```
DaysA = set(["Mon", "Tue", "Wed"])
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

```
DaysB = set(["Wed", "Thu", "Fri", "Sat", "Sun"])
```

```
AllDays = DaysA - DaysB
```

```
print(AllDays)
```

When the above code is executed, it produces the following result. Please note the result has only one “wed”.

```
set(['Mon', 'Tue'])
```

Compare Sets

We can check if a given set is a subset or superset of another set. The result is True or False depending on the elements present in the sets.

```
DaysA = set(["Mon", "Tue", "Wed"])
```

```
DaysB = set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
```

```
SubsetRes = DaysA <= DaysB
```

```
SupersetRes = DaysB >= DaysA
```

```
print(SubsetRes)
```

```
print(SupersetRes)
```

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

True

True

Possible Questions:

2 Mark Questions

1. How to pass tuple as argument

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT III

2. How can you insert values in to dictionary
3. What is key value pair
4. Mention different data types can be used in key and value
5. What are the immutable data types available in python
6. What is the use of fromkeys() in dictionary.
7. Create tuple with single element
8. What is the use of map () function.
9. How can you distinguish between tuples and lists?

6 Mark Questions

1. Demonstrate tuple and its functions with example.
2. Generalize the uses of dictionaries and its methods.
3. What are the accessing elements in a tuple? Explain With suitable Programs.
4. What are the different ways to create a set? Explain its functions.
5. Discuss the various operation that can be performed on a tuple with an example program.
6. How can you access elements from the dictionary? Explain with example.
7. List out the methods that are available with set object in python. Explain it.
8. Define Python tuple. Classify the Python accessing Elements in a tuples.



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS)

(BATCH 2019 - 2021)

I SEMESTER

PYTHON PROGRAMMING (19CSP101)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT III

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	A ____ is an associative array of key-value pairs	dictionary	list	tuple	sequence	dictionary
2	A _____ is though similar to a list, but it's immutable.	dictionary	list	tuple	sequence	tuple
3	A ____, in Python, stores a sequence of objects in a defined order.	dictionary	list	tuple	sequence	list
4	What Will Be The Output Of The Following Code Snippet? a=[1,2,3,4,5,6,7,8,9] print(a[::-2])	[1,2]	[8,9]	[1,3,5,7,9]	[1,2,3]	[1,3,5,7,9]
5	_____ enclosed between square bracket	dictionary	list	tuple	sequence	list
6	_____ enclosed between parenthesis	dictionary	list	tuple	sequence	tuple

7	A subpart of a tuple can be retrieved on the basis of _____	index	semicolon	colon	comma	index
8	Elements of the _____ cannot be updated.	dictionary	list	tuple	sequence	tuple
9	Deleting individual element from a _____ is not supported	dictionary	list	tuple	sequence	tuple
10	_____converts the sequence into tuple.	tuple(sequence)	list(sequence)	map(sequence)	none	tuple(sequence)
11	Processing of Tuples are faster than _____.	dictionary	list	tuple	sequence	list
12	_____ are used for String formatting	dictionary	list	tuple	sequence	tuple
13	_____ is an unordered set of key and value pair	dictionary	list	tuple	sequence	dictionary
14	_____ enclosed within curly braces.	dictionary	list	tuple	sequence	dictionary
15	The key and the value is separated by a _____	index	semicolon	colon	comma	colon
16	_____ is mutable.i.e., value can be updated	dictionary	list	tuple	sequence	dictionary
17	_____ is known as Associative array since the Key works as Index and they are decided by the user	dictionary	list	tuple	sequence	dictionary

18	_____ statement is used for performing deletion operation.	del	delete	remove	erase	del
19	_____ returns all the key-value pair of a dictionary	values()	keys()	items()	clear()	items()
20	has_key(key) returns a _____ value	char	float	int	boolean	boolean
21	_____ used to create a new dictionary from the sequence	update(dictionary2)	fromkeys(sequence,value1)	has_key(key)	get(key)	fromkeys(sequence,value1)
22	_____ used to remove all items of a dictionary	erase()	delete()	remove()	clear()	clear()
23	The _____ objects are enclosed within parenthesis and separated by comma.	dictionary	list	tuple	sequence	tuple
24	_____ method returns the value of the given key. If key is not present it returns none	update(dictionary2)	fromkeys(sequence,value1)	has_key(key)	get(key)	get(key)
25	What Will Be The Output Of The Following Code Snippet? a=[1,2,3,4,5,6,7,8,9] print(a[::-2])	[1,2]	[8,9]	[1,3,5,7,9]	[1,2,3]	[1,3,5,7,9]
26	What Will Be The Output Of The Following Code Snippet? a=[1,2,3,4,5,6,7,8,9] a[::-2]=10,20,30,40,50,60 print(a)	ValueError: attempt to assign sequence of size 6 to extended slice	[10, 2, 20, 4, 30, 6, 40, 8, 50, 60]	[1, 2, 10, 20, 30, 40, 50, 60]	[1, 10, 3, 20, 5, 30, 7, 40, 9, 50, 60]	ValueError: attempt to assign sequence of size 6 to extended slice of size 5
27	What Will Be The Output Of The Following Code Snippet? a=[1,2,3,4,5] print(a[3:0:-1])	Syntax error	[4, 3, 2]	[4, 3]	[4, 3, 2, 1]	[4, 3, 2]

28	What Is The Correct Command To Shuffle The Following List? fruit=['apple', 'banana', 'papaya', 'cherry']	fruit.shuffle()	shuffle(fruit)	random.shuffle(fruit)	random.shuffleList(fruit)	random.shuffle(fruit)
29	What Will Be The Output Of The Following Code Snippet? a = {(1,2):1,(2,3):2} print(a[1,2])	Key Error	1	{(2,3):2}	{(1,2):1}	1
30	_____ as a mapping between a set of indices	dictionary	list	tuple	sequence	dictionary
31	The curly brackets, {}, represent an empty_____ .	list	tuple	dictionary	sequence	dictionary
32	_____ is a statistical term for a set of counters	bar chart	histogram	pie chart	line chart	histogram
33	What Will Be The Output Of The Following Code Snippet? counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100} print(counts.get('jan', 0))	0	1	42	100	100
34	<pre>>>>a=[5,6,7,8,9,4] a[1:]</pre>	[9,8,7,6,5,4]	[8, 7, 6, 5, 4]	[7, 6, 5, 4]	[7, 6, 5]	[8, 7, 6, 5, 4]
35	<pre>>>>a=[5,6,7,8,9,4] a[::-1]</pre>	[8, 7, 6, 5, 4]	[4, 5, 6, 7, 8, 9]	[9,8,7,6,5,4]	[7, 6, 5, 4]	[4, 5, 6, 7, 8, 9]
36	<pre>>>>a=[1,2,3,4,5] >>>b=[7,8,9] >>>a.extend(b) >>>print(a)</pre>	[9,8,7,6,5,4]	[1,2,3,4,5,7,8,9]	1, 2, 3, 4, 5, 6, 7, 8	[7, 6, 5, 4]	[1,2,3,4,5,7,8,9]

37	>>>[8, 7, 6, 5, 4, 3, 2, 1, 0] >>>a.pop(0)	[1,2,3,4,5,7,8,9]	[4, 5, 6, 7, 8, 9]	[9,8,7,6,5,4]	[8, 7, 6, 5, 4, 3, 2, 1,]	[8, 7, 6, 5, 4, 3, 2, 1,]
38	_____ is immutable so changes cannot be done on the elements of a tuple once it is assigned.	dictionary	list	tuple	sequence	tuple
39	>>>a={1: 'ONE', 2: 'two', 3: 'three'} >>>a.keys()	{1: 'ONE', 2: 'two', 3: 'three'}	dict_keys([1, 2, 3])	([(1, 'ONE'), (2, 'two'), (3, 'three')])	dict_keys([0, 2, 3])	dict_keys([1, 2, 3])
40	_____Heterogeneous	dictionary	list	tuple	sequence	tuple
41	In _____,Slicing can't be done	dictionary	list	tuple	sequence	dictionary

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

Python Objects: Introduction-Standard Type- Built-in-type-Built-in functions. **Class:** Introduction- Class and Instance- Method calls. **File:** Objects- Built in Functions-Methods- Attributes- Command line Argument-File System-File Execution.

Python Classes and 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)
```

Run example »

The **init** () Function

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

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.

Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute –

- `__dict__` – Dictionary containing the class's namespace.
- `__doc__` – Class documentation string or none, if undefined.
- `__name__` – Class name.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

- **__module__** – Module name in which the class is defined. This attribute is "**__main__**" in interactive mode.
- **__bases__** – A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

For the above class let us try to access all these attributes –

Object Methods

Objects can also contain methods. Methods in objects are functions that belongs to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

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

    def myfunc(self):
        print("Hello my name is " + self.name)
```

```
p1 = Person("John", 36)
p1.myfunc()
```

Note: The **self** parameter is a reference to the class itself, and is used to access variables that belongs to the class.

The self Parameter

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

The **self** parameter is a reference to the class itself, and is used to access variables that belongs to the class.

It does not have to be named **self**, you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age

    def myfunc(abc):
        print("Hello my name is " + abc.name)
```

```
p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects by using the **del** keyword:

Example

Delete the age property from the p1 object:

```
del p1.age
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

Delete Objects

You can delete objects by using the **del** keyword:

Example

Delete the p1 object:

```
del p1
```

Python Object

Python is an object oriented programming language. So, its main focus is on objects unlike procedure oriented programming languages which mainly focuses on functions.

In object oriented programming language, object is simply a collection of data (variables) and methods (functions) that act on those data.

Python Class

A class is a blueprint for the object. Let's understand it by an example:

Suppose a class is a prototype of a building. A building contains all the details about the floor, doors, windows, etc. we can make another buildings (as many as we want) based on these details. So building is a class and we can create many objects from a class.

An object is also called an instance of a class and the process of creating this object is known as instantiation.

Python classes contain all the standard features of Object Oriented Programming. A python class is a mixture of class mechanism of C++ and Modula-3.

Define a class in Python

In Python, a class is defined by using a keyword **class** like a function definition begins with the keyword **def**.

Syntax of a class definition:

```
class ClassName:
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT IV

<statement-1>

.

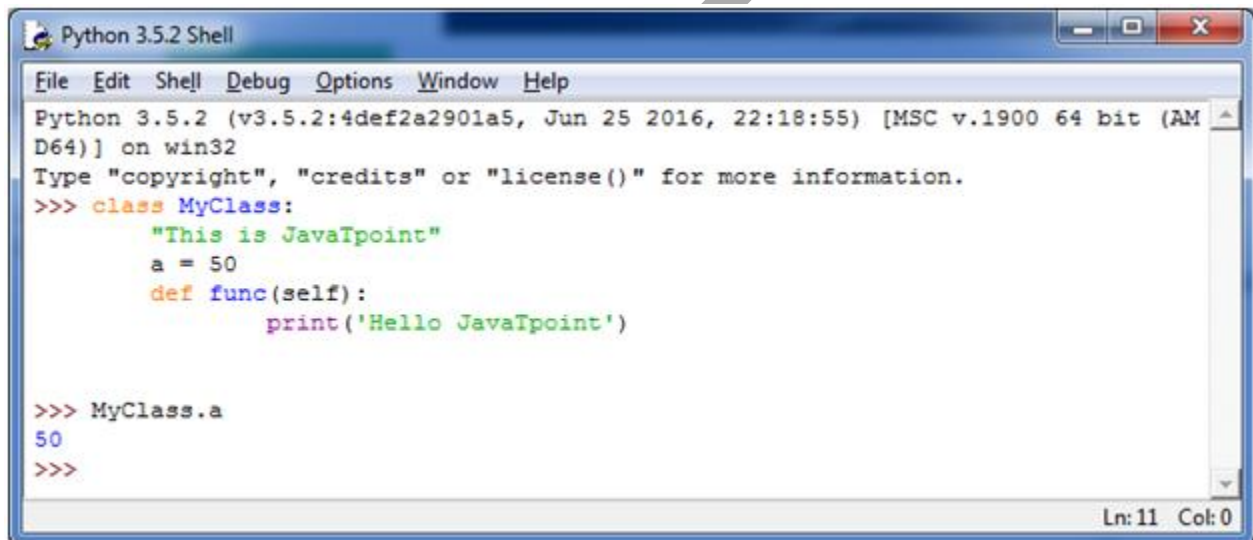
.

.

<statement-N>

A class creates a new local namespace to define its all attributes. These attributes may be data or functions.

See this example:

A screenshot of a Python 3.5.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code:

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> class MyClass:
    "This is JavaTpoint"
    a = 50
    def func(self):
        print('Hello JavaTpoint')

>>> MyClass.a
50
>>>
```

The status bar at the bottom right indicates 'Ln: 11 Col: 0'.

There are also some special attributes that begins with double underscore (__). For example: `__doc__` attribute. It is used to fetch the docstring of that class. When we define a class, a new class object is created with the same class name. This new class object provides a facility to access the different attributes as well as to instantiate new objects of that class.

See this example:

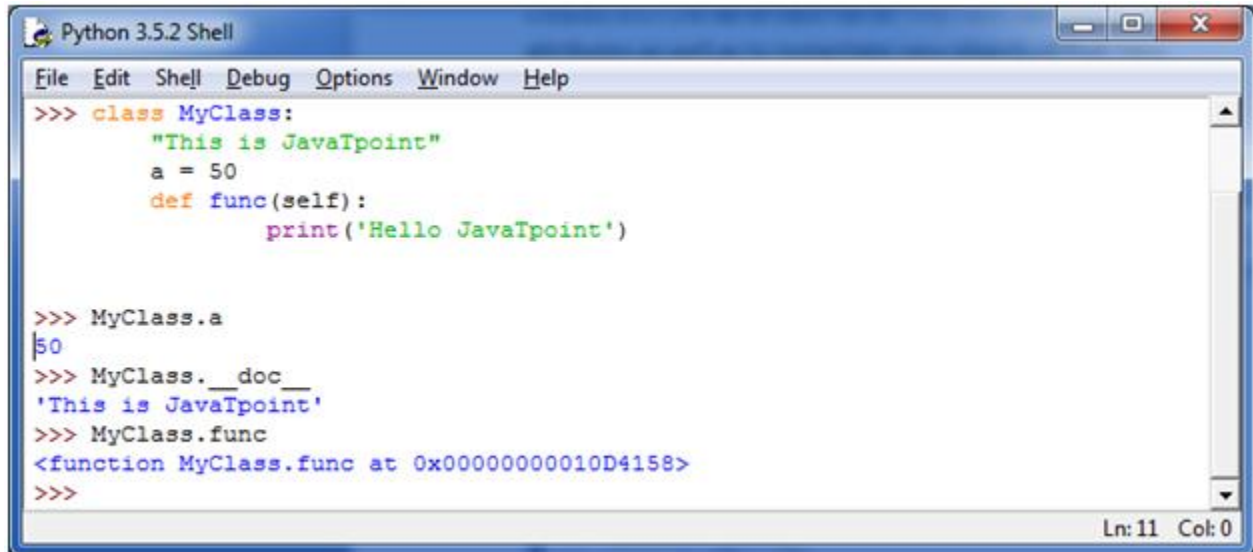
CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> class MyClass:
>>>     "This is JavaTpoint"
>>>     a = 50
>>>     def func(self):
>>>         print('Hello JavaTpoint')
>>>
>>> MyClass.a
50
>>> MyClass.__doc__
'This is JavaTpoint'
>>> MyClass.func
<function MyClass.func at 0x00000000010D4158>
>>>
```

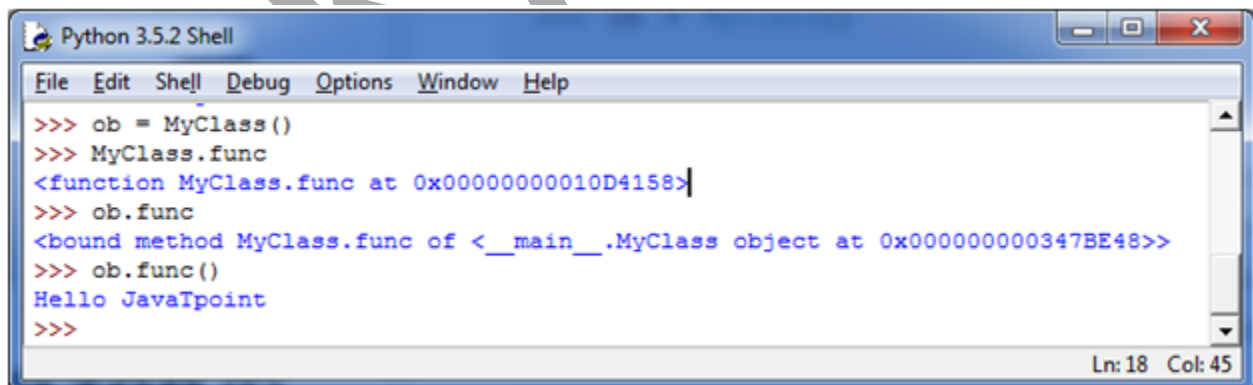
Ln: 11 Col: 0

Create an Object in Python

We can create new object instances of the classes. The procedure to create an object is similar to a function call.

Let's take an example to create a new instance object **ob**. We can access attributes of objects by using the object name prefix.

See this example:



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> ob = MyClass()
>>> MyClass.func
<function MyClass.func at 0x00000000010D4158>
>>> ob.func
<bound method MyClass.func of <__main__.MyClass object at 0x0000000000347BE48>>
>>> ob.func()
Hello JavaTpoint
>>>
```

Ln: 18 Col: 45

Here, attributes may be data or method. Method of an object is corresponding functions of that class. For example: MyClass.func is a function object and ob.func is a method object.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

Python Object Class Example

```
class Student:
    def __init__(self, rollno, name):
        self.rollno = rollno
        self.name = name
    def displayStudent(self):
        print "rollno : ", self.rollno, ", name: ", self.name
emp1 = Student(121, "Ajeet")
emp2 = Student(122, "Sonoo")
emp1.displayStudent()
emp2.displayStudent()
```

Output:

```
rollno : 121 , name: Ajeet
rollno : 122 , name: Sonoo
```

Python File Handling

Python provides the facility of working on Files. A File is an external storage on hard disk from where data can be stored and retrieved.

Operations on Files:

1) Opening a File: Before working with Files you have to open the File. To open a File, Python built in function open() is used. It returns an object of File which is used with other functions. Having opened the file now you can perform read, write, etc. operations on the File.

Syntax:

obj=open(filename , mode , buffer)

here,

filename: It is the name of the file which you want to access.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

mode: It specifies the mode in which File is to be opened. There are many types of mode. Mode depends on the operation to be performed on File. Default access mode is read.

2) Closing a File: Once you are finished with the operations on File at the end you need to close the file. It is done by the close() method. close() method is used to close a File.

Syntax:

```
fileobject.close()
```

3) Writing to a File: write() method is used to write a string into a file.

Syntax:

```
fileobject.write(string str)
```

4) Reading from a File: read() method is used to read data from the File.

Syntax:

```
fileobject.read(value)
```

here, value is the number of bytes to be read. In case, no value is given it reads till end of file is reached.

Program to read and write data from a file.

```
obj=open("abcd.txt","w")
obj.write("Welcome to the world of Python")
obj.close()
obj1=open("abcd.txt","r")
s=obj1.read()
print s
obj1.close()
obj2=open("abcd.txt","r")
s1=obj2.read(20)
print s1
obj2.close()
```

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

Output:

```
>>>
Welcome to the world of Python
Welcome to the world
>>>
```

Attributes of File:

There are following File attributes.

Attribute	Description
Name	Returns the name of the file.
Mode	Returns the mode in which file is being opened.
Closed	Returns Boolean value. True, in case if file is closed else false.

Example

```
obj = open("data.txt", "w")
print obj.name
print obj.mode
print obj.closed
```

Output:

```
>>>
data.txt
w
False
>>>
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT IV

Modes of File:

There are different modes of file in which it can be opened. They are mentioned in the following table.

A File can be opened in two modes:

- 1) Text Mode.
- 2) Binary Mode.

Mode	Description
R	It opens in Reading mode. It is default mode of File. Pointer is at beginning of the file.
rb	It opens in Reading mode for binary format. It is the default mode. Pointer is at beginning of file.
r+	Opens file for reading and writing. Pointer is at beginning of file.
rb+	Opens file for reading and writing in binary format. Pointer is at beginning of file.
W	Opens file in Writing mode. If file already exists, then overwrite the file else create a new file.
wb	Opens file in Writing mode in binary format. If file already exists, then overwrite the file else create a new file.
w+	Opens file for reading and writing. If file already exists, then overwrite the file else create a new file.
wb+	Opens file for reading and writing in binary format. If file already exists, then overwrite the file else create a new file.
a	Opens file in Appending mode. If file already exists, then append the data

KARPAGAM ACADEMY OF HIGHER EDUCATION

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

	at the end of existing file, else create a new file.
ab	Opens file in Appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.
a+	Opens file in reading and appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab+	Opens file in reading and appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.

Methods:

There are many methods related to File Handling. They are given in the following table:

There is a module "os" defined in Python that provides various functions which are used to perform various operations on Files. To use these functions 'os' needs to be imported.

Method	Description
rename()	It is used to rename a file. It takes two arguments, existing_file_name and new_file_name.
remove()	It is used to delete a file. It takes one argument. Pass the name of the file which is to be deleted as the argument of method.
mkdir()	It is used to create a directory. A directory contains the files. It takes one argument which is the name of the directory.
chdir()	It is used to change the current working directory. It takes one argument which is the name of the directory.
getcwd()	It gives the current working directory.
rmdir()	It is used to delete a directory. It takes one argument which is the name of the directory.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

tell()

It is used to get the exact position in the file.

1) rename():

Syntax:

```
os.rename(existing_file_name, new_file_name)
```

eg:

```
import os
os.rename('mno.txt', 'pqr.txt')
```

2) remove():

Syntax:

```
os.remove(file_name)
```

eg:

```
import os
os.remove('mno.txt')
```

3) mkdir()

Syntax:

```
os.mkdir("file_name")
```

eg:

```
import os
os.mkdir("new")
```

4) chdir()

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

Syntax:

```
os.chdir("file_name")
```

Example

```
import os  
os.chdir("new")
```

5) getcwd()

Syntax:

```
os.getcwd()
```

Example

```
import os  
print os.getcwd()
```

6) rmdir()

Syntax:

```
os.rmdir("directory_name")
```

Example

```
import os  
os.rmdir("new")
```

NOTE: In order to delete a directory, it should be empty. In case directory is not empty first delete the files.

Command line arguments

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT IV

In the **command line**, we can start a program with additional arguments. These **arguments** are passed **into the program**.

Python programs can start with command line arguments.

For example:

\$ python **program.py image.bmp**

where program.py and image.bmp is are arguments. (the program is Python)

How to use command line arguments in python?

We can use modules to get arguments.

Which modules can get command line arguments?

Module	Use	Python version
sys	All arguments in sys.argv (basic)	All
argparse	Build a command line interface	>= 2.3
docopt	Create command line interfaces	>= 2.5
fire	Automatically generating command line interfaces (CLIs)	All
optparse	Deprecated	< 2.7

Sys argv

You can get access to the command line parameters using the sys module. len(sys.argv) contains the number of arguments. To print all of the arguments simply execute str(sys.argv)

```
#!/usr/bin/python
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

```
import sys

print('Arguments:', len(sys.argv))
print('List:', str(sys.argv))
```

Example:

```
$ python3 example.py image.bmp color
Arguments: 3
List: ['example.py', 'image.bmp', 'color']
```

Storing command line arguments

You can store the arguments given at the start of the program in variables.
For example, an image loader program may start like this:

```
#!/usr/bin/python
#!/usr/bin/python

import sys

print('Arguments:', len(sys.argv))
print('List:', str(sys.argv))

if sys.argv < 2:
    print('To few arguments, please specify a filename')

filename = sys.argv[1]
print('Filename:', filename)
```

Another example:

```
('Arguments:', 2)
('List:', "['example.py', 'world.png']")
('Filename:', 'world.png')
```

CLASS : I M.Sc CS
COURSE NAME : PYTHON PROGRAMMING

BATCH : 2019 - 2021
COURSE CODE : 19CSP101

UNIT IV

Argparse

If you need more advanced parsing, you can use argparse.
You can define arguments like (-o, -s).

The example below parses parameters:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('-o', '--open-file', help='Description', required=False)
parser.add_argument('-s', '--save-file', help='Description', required=False)

args = parser.parse_args()

print(args.open_file)
print(args.save_file)
```

Docopt

Docopt can be used to create command line interfaces.

```
from docopt import docopt

if __name__ == '__main__':
    arguments = docopt(__doc__, version='Example 1.0')
    print(arguments)
```

Note: docopt is not tested with Python 3.6

Fire

Python Fire automatically generates a command line interface, you only need one line of code.
Unlike the other modules, this works instantly.

You don't need to define any arguments, all methods are linked by default.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

To install it type:

```
pip install fire
```

Then define or use a class:

```
import fire

class Program(object):
    def hello(self):
        print("Hello World")

    def openfile(self, filename):
        print("Opening file " + filename + ".txt")

if __name__ == '__main__':
    fire.Fire(Program)
```

You then have the options matching the class methods:

```
python example.py hello
python example.py openfile filename.txt
```

Possible Questions:

2 Mark Questions

1. Point out different modes of file opening
2. Discover the format operator available in files.
3. Define read and write file.
4. Define class and object
5. What are command line arguments?
6. Mention the built in objects in python

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT IV

6 Mark Questions

1. Illustrate Command line Argument with proper example.
2. Briefly discuss about Built-in-type objects in python.
3. Describe classes and its methods with example.
4. Identify the various methods used in file. Explain it.
5. What is an object? Describe built it objects with example.
6. Define file types. Write a Python program to demonstrate the file I/O operations.
7. Discover syntax and example for Built in Functions in file.
8. Define class. Explain classes and it methods with example.
9. Discuss with suitable examples i) Close a File. ii) Writing to a File.
10. Define object .Describe built in methods with example.



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS) (BATCH 2019 - 2021) I SEMESTER

PYTHON PROGRAMMING (19CSP101)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT III

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	What is setattr() used for?	To access the attribute of the object	To set an attribute	To check if an attribute exists or not	To delete an attribute	To set an attribute
2	What are the methods which begin and end with two underscore characters called?	Special methods	In-built methods	User-defined methods	Additional methods	Special methods
3	Which Of The Following Statements Can Be Used To Check, Whether An Object “Obj” Is An Instance Of Class A Or Not?	obj.isinstance(A)	A.isinstance(obj)	isinstance(obj, A)	isinstance(A, obj)	isinstance(obj, A)
4	s = "\t\tWelcome\n" print(s.strip())	\t\tWelcome\n	Welcome\n	\t\tWELCOME	Welcome	Welcome
5	Which Of The Following Statements Are Correct?	A reference variable is an object.	A reference variable refers to an object.	An object may contain other objects.	An object can contain the references to other objects.	B and D
6	Which Of The Following Represents A Distinctly Identifiable Entity In The Real World?	A class	An object	A method	A data field	An object

7	Which Of The Following Keywords Mark The Beginning Of The Class Definition?	def	return	class	All of the above.	class
8	which Of The Following Is Required To Create A New Instance Of The Class?	A constructor	A class	A value-returning method	A None method	A constructor
9	Which Of The Following Statements Is Most Accurate For The Declaration X = Circle()?	x contains an int value.	x contains an object of the Circle type.	x contains a reference to a Circle object.	you can assign an int value to x.	x contains a reference to a Circle object.
10	Which Of The Following Represents A Template, Blueprint, Or Contract That Defines Objects Of The Same Type?	A class	An object	A method	A data field	A class
11	What Relationship Correctly Fits For University And Professor?	association	composition	inheritance	All of the above	composition
12	What Relationship Is Appropriate For Fruit And Papaya?	association	composition	inheritance	All of the above	inheritance
13	_____are mutable	objects	inheritance	class	methods	objects
14	A ____ object is well-formed if the values of minute and second are between 0 and 60 and if hour is positive	date	circle	Time	constructor	Time
15	_____is a function that is associated with a particular class	class	method	object	string	method
16	The ____ method is a special method that gets invoked when an object is instantiated.	string	date	Time	init	init

17	>>> time = Time(9, 45) >>> time.print_time()	0:00:00	9:00:00	9:45:00	2:00:00	9:45:00
18	_____ is a named location on disk to store related information.	folder	file	record	disk	file
19	_____ is volatile	ram	rom	eprom	EEPROM	ram
20	_____ Symbol is used for Opening a file for exclusive creation.	'x'	'r'	'w'	'a'	'x'
21	_____ Symbol is used for Opening a file for updating	'x'	'w'	+	-	+
22	cwd stands for _____	current word dict	current working directory	common working directory	current working Drive	current working directory
23	A _____ path starts from the current directory	isolate	absolute	modified	relative	relative
24	an _____ path starts from the topmost directory in the file system	isolate	absolute	modified	relative	absolute
25	_____, Open in binary mode.	'x'	'r'	'w'	'b'	'b'
26	The ____ variable contains an array of strings consisting of each argument from the command line	args	argv	argc	argz	argv
27	_____ variable contains the number of arguments entered	args	argv	argc	argz	argc
28	the value for argc is simply the number of items in the _____ list	sys.argv	sys.argc	sys.argv	sys.argz	sys.argv

29	_____ is the list of command-line arguments	sys.argv	sys.argv	sys.argv	sys.argv	sys.argv
30	_____ is the number of command-line arguments	len(sys.argv)	len(sys.argv)	len(sys.argv)	len(sys.argv)	len(sys.argv)
31	_____Generate filenames in a directory tree	walk()	chroot()	release()	root()	walk()
32	_____Return last file access time	getctime()	getmtime()	getatime()	getbtime()	getatime()
33	_____Return last file modification time	getctime()	getmtime()	getatime()	getbtime()	getmtime()
34	An _____ is a data or functional element that belongs to another object.	method	attribute	class	object	attribute
35	_____ numbers have data attributes (real and imag)	complex	integer	float	char	complex
36	_____ keyword with which to create an instance of a class	insert	new	modify	delete	new
37	The _____ function immediately terminates the program.	sys.terminate()	sys.halt()	sys.exit()	sys.stop()	sys.exit()
38	_____,Unique identifier that differentiates an object from all others	identity	type	value	set	identity
39	_____,an object's type indicates what kind of values an object can hold	identity	type	value	set	type

40	_____,Data item that is represented by an object.	identity	type	value	set	value
----	---	----------	------	-------	-----	-------

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

Exception and Tools: Why use it?- Exception roles-Exception in python-Try/finally statement. **Regular Expression:** Introduction-Special Symbols and characters-Regexes and Python- Examples of Regexes. **Network Programming:** Architecture- Socket- networking programming in python.



Python Exception Handling

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.

Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Common Exceptions

- 
1. ZeroDivisionError: Occurs when a number is divided by zero.
 2. NameError: It occurs when a name is not found. It may be local or global.
 3. IndentationError: If incorrect indentation is given.
 4. IOError: It occurs when Input Output operation fails.
 5. EOFError: It occurs when end of the file is reached and yet operations are being performed.
- 

Exception Handling:

The suspicious code can be handled by using the try block. Enclose the code which raises an exception inside the try block. The try block is followed except statement. It is then further followed by statements which are executed during exception and in case if exception does not occur.

Syntax:

```
try:  
    malicious code
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

```
except Exception1:
    execute code
except Exception2:
    execute code
....
....
except ExceptionN:
    execute code
else:
    In case of no exception, execute the else block code.
```

Python Exception Handling Example

```
try:
    a=10/0
    print a
except ArithmeticError:
    print "This statement is raising an exception"
else:
    print "Welcome"
```

Output:

```
>>>
This statement is raising an exception
>>>
```

Explanation:

1. The malicious code (code having exception) is enclosed in the try block.
2. Try block is followed by except statement. There can be multiple except statement with a single try block.
3. Except statement specifies the exception which occurred. In case that exception is occurred, the corresponding statement will be executed.
4. At the last you can provide else statement. It is executed when no exception is occurred.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

Python Exception(Except with no Exception) Example

Except statement can also be used without specifying Exception.

Syntax:

```
try:
    code
except:
    code to be executed in case exception occurs.
else:
    code to be executed in case exception does not occur.
```

Example

```
try:
    a=10/0;
except:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

Output:

```
>>>
Arithmetic Exception
>>>
```

Declaring Multiple Exception in Python

Python allows us to declare multiple exceptions using the same except statement.

Syntax:

```
try:
    code
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

except Exception1,Exception2,Exception3,...,ExceptionN
execute this code **in** case any Exception of these occur.
else:
execute code **in** case no exception occurred.

Example

```
try:
    a=10/0;
except ArithmeticError,StandardError:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

Output:

```
>>>
Arithmetic Exception
>>>
```

Finally Block:

In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.

Syntax:

```
try:
    Code
finally:
    code which is must to be executed.
```

Example

```
try:
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

```
a=10/0;
print "Exception occurred"
finally:
    print "Code to be executed"
```

Output:

```
>>>
Code to be executed
Traceback (most recent call last):
  File "C:/Python27/noexception.py", line 2, in <module>
    a=10/0;
ZeroDivisionError: integer division or modulo by zero
>>>
```

In the above example finally block is executed. Since exception is not handled therefore exception occurred and execution is stopped.

Raise an Exception:

You can explicitly throw an exception in Python using `raise` statement. `raise` will cause an exception to occur and thus execution control will stop in case it is not handled.

Syntax:

```
raise Exception_class,<value>
```

Example

```
try:
    a=10
    print a
    raise NameError("Hello")
except NameError as e:
    print "An exception occurred"
    print e
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

Output:

```
>>>
10
An exception occurred
Hello
>>>
```

Explanation:

- i) To raise an exception, raise statement is used. It is followed by exception class name.
- ii) Exception can be provided with a value that can be given in the parenthesis. (here, Hello)
- iii) To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

Custom Exception:

Refer to this section after visiting Class and Object section:

Creating your own Exception class or User Defined Exceptions are known as Custom Exception.

Example

```
class ErrorInCode(Exception):
    def __init__(self, data):
        self.data = data
    def __str__(self):
        return repr(self.data)

try:
    raise ErrorInCode(2000)
except ErrorInCode as ae:
    print "Received error:", ae.data
```

Output:

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

```
>>>  
Received error : 2000  
>>>
```

Regular expressions

Regular expressions are a very useful technique for extracting information from text such as code, spreadsheets, documents or log-files. The first thing to keep in mind while implementing regular expression is that everything essentially needs to be a character & programmers write patterns to match a specific sequence of characters/strings.

Defining Regular expressions

Regular expressions are characters in special order that help programmers find other sequences of characters or strings or set of strings using specialized syntax held in a pattern. Python supports regular expressions through the standard Python library's 're' which is packed with every Python installation.

Here, we will be learning about the vital functions that are used to handle regular expressions. There are many characters having special meaning when they are used as regular expressions. This is mostly used in UNIX.

Raw Strings in Python

It is recommended to use raw-strings instead of regular strings. When programmers write regular expressions in Python, they begin raw strings with a special prefix 'r' and backslashes and special meta-characters in the string, that allows us to pass through them to regular-expression-engine directly.

match Function

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

This method is used to test whether a regular expression matches a specific string in Python. The re.match(). The function returns 'none' if the pattern doesn't match or includes additional information about which part of the string the match was found.

Syntax:

```
re.match (pattern, string, flags=0)
```

Here, all the parts are explained below:

- match(): is a method
- pattern: this is the regular expression that uses meta-characters to describe what strings can be matched.
- string: is used to search & match the pattern at the string's initiation.
- flags: programmers can identify different flags using bitwise operator '|' (OR)

Example:

```
import re#simple structure of re.match()

matchObject = re.match(pattern, input_str, flags=0)
```

A Program by USING re.match:

Example:

```
import re

list = [ "mouse", "cat", "dog", "no-match"]

# Loop starts here

for elements in list:

    m = re.match("(d\\w+) \\W(d\\w+)", element)
```

CLASS : I M.Sc CS**BATCH : 2019 - 2021****COURSE NAME : PYTHON PROGRAMMING****COURSE CODE : 19CSP101**

UNIT V

```
# Check for matching
```

```
if m:
```

```
    print(m.groups())
```

In the above example, the pattern uses meta-character to describe what strings it can match. Here '\w' means word-character & + (plus) symbol denotes one-or-more.

Most of the regular expressions' control technique comes to a role when "patterns" are used.

The re.match function returns a **match** object on success, **None** on failure. We use group(num) or groups() function of **match** object to get matched expression.

Sr.No.	Match Object Method & Description
1	group(num=0) This method returns entire match (or specific subgroup num)
2	groups() This method returns all matching subgroups in a tuple (empty if there weren't any)

search Function

It works in a different manner than that of a match. Though both of them uses pattern; but 'search' attempts this at all possible starting points in the string. It scans through the input string and tries to match at any location.

Syntax:

```
re.search( pattern, strings, flags=0)
```

CLASS : I M.Sc CS
COURSE NAME : PYTHON PROGRAMMING

BATCH : 2019 - 2021
COURSE CODE : 19CSP101

UNIT V

Program to show how it is used:

```
import re

value = "cyberdyne"

g = re.search("(dy.*)", value)

if g:

    print("search: ", g.group(1))

s = re.match("(vi.*)", value)

if s:

    print("match:", m.group(1))
```

Output:

dyne

split Function

The re.split() accepts a pattern that specifies the delimiter. Using this, we can match pattern & separate text data. 'split()' is also available directly on a string & handles no regular expression.

Program to show how to use split():

Example:

```
import re

value = "two 2 four 4 six 6"
```

CLASS : I M.Sc CS
COURSE NAME : PYTHON PROGRAMMING

BATCH : 2019 - 2021
COURSE CODE : 19CSP101

UNIT V

```
#separate those non-digit characters
```

```
res = re.split ("\D+", value)
```

```
# print the result
```

```
for elements in res :
```

```
    print (elements)
```

Output:

2

4

6

In the above program, \D+ represents one or more non-digit characters.

Network Programming

Python plays an essential role in network programming. The standard library of Python has full support for network protocols, encoding and decoding of data and other networking concepts and it is simpler to write network programs in Python than that of C++.

Here, we will learn about the essence of network programming concerning Python. But for this programmers must need to have basic knowledge of:

- Low-Level Programming using sockets
- Data encoding
- HTTP and web-programming
- High-Level client modules

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

- Basic networking terms and their concepts etc.

Python Network Services

There are two levels of network service access in Python. These are:

- Low-Level Access
- High-Level Access

In the first case, programmers can use and access the basic socket support for the operating system using Python's libraries, and programmers can implement both connection-less and connection-oriented protocols for programming.

Application level network protocols can also be accessed using high-level access provided by Python libraries. These protocols are HTTP, FTP, etc.

Defining Socket

A socket is the end-point in a flow of communication between two programs or communication channels operating over a network. They are created using a set of programming request called socket API (Application Programming Interface). Python's socket library offers classes for handling common transports as a generic interface.

Sockets use protocols for determining the connection type for port-to-port communication between client and server machines. The protocols are used for:

- Domain Name Servers (DNS)
- IP addressing
- E-mail
- FTP (File Transfer Protocol) etc...

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

Socket Program

Python has socket method that let programmers' set-up different types of socket virtually. The syntax for socket method is:

Syntax:

```
g = socket.socket (socket_family, type_of_socket, protocol=value)
```

For example, if we want to establish a TCP socket, we can write the following code snippet:

Example:

```
# imports everything from 'socket'

from socket import *

# use socket.socket() - function

tcp1=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here's another example to establish UDP socket. The code is:

```
udp1=socket.socket (socket.AF_INET, socket.SOCK_DGRAM)
```

After you defined the socket, you can use several methods to manage the connections. Some of the important server socket methods are:

- listen(): is used to establish and start TCP listener.
- bind(): is used to bind address (host-name, port number) to the socket.
- accept(): is used to TCP client connection until the connection arrives.
- connect(): is used to initiate TCP server connection.

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

- send(): is used to send TCP messages.
- recv(): is used to receive TCP messages.
- sendto(): is used to send UDP messages
- close(): is used to close a socket.

A Simple Network Program Using Python

Example:

```
import socket

T_PORT = 60

TCP_IP = '127.0.0.1'

BUF_SIZE = 30

# create a socket object name 'k'

k = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

k.bind((TCP_IP, T_PORT))

k.listen(1)

con, addr = k.accept()

print ('Connection Address is: ', addr)

while True :

    data = con.recv(BUF_SIZE)

if not data:
```


CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

```
break

print ("Received data", data)

con.send(data)

con.close()
```

Save the file with filename - tcpserver.py

This will open a web server at port 60. In the above program, everything you write in the client goes to the server.

Now a simple Python client script:

```
import socket

T_PORT = 5006

TCP_IP = '127.0.0.1'

BUF_SIZE = 1024

MSG = "Hello karl"

# create a socket object name 'k'

k = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

k.connect((TCP_IP, T_PORT))

k.send(MSG)

data = k.recv(BUF_SIZE)
```

CLASS : I M.Sc CS

BATCH : 2019 - 2021

COURSE NAME : PYTHON PROGRAMMING

COURSE CODE : 19CSP101

UNIT V

k.close

Sending messages back and forth using different basic protocols is simple and straightforward. It shows that programming takes a significant role in client-server architecture where the client makes data request to a server, and the server replies that machine.

Possible Questions:

2 Mark Questions

1. What is exception?
2. Mention the types of exception in python.
3. Define Socket.
4. Define networks.
5. Define Exception and its types
6. What is regular expression?
7. Write the syntax for re.match in RE.

6 Mark Questions

1. Describe in detail exception handling with sample program.
2. Explain network programming with neat sketch.
3. Summarize regular expression in python.
4. Demonstrate architecture of network programming in python
5. Define Exception and its types. Write a program to catch a Divide by zero exception.
6. Analyze regular expression using in python.
7. How to handle exception in python. Explain its types.
8. Difference between built in exceptions and handling exception with example.



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS)

(BATCH 2019 - 2021)

I SEMESTER

PYTHON PROGRAMMING (19CSP101)

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

UNIT V

S.NO	QUESTIONS	OPT 1	OPT 2	OPT 3	OPT 4	ANSWER
1	When will the else part of try-except-else be executed?	always	when an exception occurs	when no exception occurs	when an exception occurs in to except block	when no exception occurs
2	What is the output of the expression? round(4.5676,2)?	4.5	4.6	4.57	4.56	4.57
3	What is the output of the code shown below? def f(x): yield x+1 print("test") yield x+2 g=f(9)	Error	test	test1012	No output	No output
4	How many keyword arguments can be passed to a function in a single function call?	zero	one	zero or more	one or more	zero or more
5	Which module in the python standard library parses options received from the command line?	getopt	os	getarg	main	getopt

6	What is the type of sys.argv?	set	list	tuple	string	list
7	How are keyword arguments specified in the function heading?	one star followed by a valid identifier	one underscore followed by a valid identifier	two stars followed by a valid identifier	two underscores followed by a valid identifier	two stars followed by a valid identifier
8	What is the output of the code shown below? <code>def f1():</code>	Error	100	101	99	100
9	When is the finally block executed?	always	when an exception occurs	when no exception occurs	when an exception occurs in to except block	always
10	What is the output of the following code? <code>def foo(): try:</code>	1 2	1	2	none	1 2
11	Which of the following is not an exception handling keyword in Python?	try	except	accept	finally	accept
12	What is the output of the code shown below? <code>g = (i for i in range(5)) type(g)</code>	class <'loop'>	class <'iteration'>	class <'range'>	class <'generator'>	class <'generator'>
13	Python has a very powerful library called ____ expressions that handles many of these tasks quite elegantly.	regular	accept	irrugar	start	regular
14	_____ expressions are almost their own little programming language for searching and parsing strings	irrugar	accept	Regular	start	Regular

15	_____ method to extract all of the substrings which match a regular expression.	replaceall()	findall()	find()	replace()	findall()
16	Identify the type of error in the codes shown below. Print(“Good Morning”) print(“Good night)	Syntax, Syntax	Semantic, Syntax	Semantic, Semantic	Syntax, Semantic	Semantic, Syntax
17	Which of the following is not a standard exception in Python?	NameError	IOError	AssignmentError	ValueError	AssignmentError
18	An exception is:	an object	a special function	standard modul	a module	an object
19	_____ exceptions are raised as a result of an error in opening a particular file.	NameError	IOError	AssignmentError	ValueError	IOError
20	Which of the following blocks will be executed whether an exception is thrown or not?	except	else	finally	assert	finally
21	The _____ statement allows the programmer to force a specific exception to occur.	except	else	raise	assert	raise
22	What is the output of the code shown? def f(x): for i in range(5): yield i g=f(8) print(list(g))	[0, 1, 2, 3, 4]	[1, 2, 3, 4, 5, 6, 7, 8]	[1, 2, 3, 4, 5]	[0, 1, 2, 3, 4, 5, 6, 7]	[0, 1, 2, 3, 4]

23	What is the output of the code shown below? #generator def f(x): yield x+1 g=f(8) print(next(g))	8	9	7	ERROR	9
24	Which of the following is not an exception handling keyword in Python?	accept	finally	except	Try	accept
25	_____ statements allow one to detect and handle exceptions	TRy-finally	try-except	except_suite	try_suite	try-except
26	_____ statements allow only for detection and processing of any obligatory cleanup	except_suite	try_suite	TRy-finally	try-except	TRy-finally
27	The_____ built-in function has a primary purpose of converting any numeric type to a float	int()	double()	long()	float()	float()
28	_____Request termination of Python interpreter	SystemEnd	SystemStop	SystemFinish	SystemExit	SystemExit
29	_____ Iteration has no further values	SystemEnd	SystemStop	StopIteration	SystemExit	StopIteration
30	_____Base class for all standard built-in exceptions	StopError	StandardError	BulitinError	ComplieError	StandardError
31	it is used to choose from one of the different regular expressions, which are separated by the _____ symbol	star	pipe	comma	colon	pipe

32	_____ Match 0 or more occurrences of preceding Regular Expression	star	plus	minus	divide	star
33	_____ Match 1 or more occurrences of preceding RE	star	plus	minus	divide	plus
34	_____ Match end of string	\$!	#	%	\$
35	_____ Match start of string	\$	^	#	%	^
36	_____ Match any character (except NEWLINE)	\$	^	.	%	.
37	_____ will either return the entire match, or a specific subgroup,	split	splits()	group()	groups()	group()
38	_____ will simply return a tuple consisting of only/all the subgroups	split	groups()	group()	splits()	groups()
39	_____ servers are examples of hardware servers	web	printer	Software	hardware	printer
40	_____ servers also run on a piece of hardware but do not have dedicated peripheral devices as hardware servers	Software	hardware	web	printer	Software
41	One of the more common software servers today is the _____ server.	Software	hardware	Web	printer	Web