GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

ON

PYTHON PROGRAMMING

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with profe-ssional clients and the public.

HANDOUT ON PYTHON PROGRAMMING

Class & Sem.: I B.Tech – II Semester **Year**: 2019-20

Branch : CSE **Credits:** 3

==========

1. **Brief History and Scope of the Subject**

Python was first developed by *Guido van Rossum* in the late 80's and early 90's at the National Research Institute for Mathematics and Computer Science in the Netherlands.It has been derived from many languages such as ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell and other scripting languages.

There is really a good scope in Python in today's world ,In last few years Python leads among the programming languages due to some of the libraries used in the most demanding work in the world like Data Science, Machine Learning, Artificial Intelligence .By the help of Python you can do everything you want to do .But mainly due to data science and machine learning g python is on the top of demanding languages now a days .Apart from this you can create a webpage , game, Application … also by using python.

2. **Pre-Requisites**
   - Knowledge on Problem Solving Through Computer Programming.
3. **Course Objectives:**
   - To introduce Scripting Language.
   - To explore various problems solving approaches of computer science.
   - To develop a basic understanding of Python programming.
4. **Course Outcomes:**

Upon successful completion of the course, the students will be able to

CO1: Demonstrate the basic elements of Python.

CO2: Implement programs using Python Control Structures.

CO3: Design functions in Python to solve the problems.

CO4: Apply strings, lists and tuples in developing Python programs.

CO5: Implement programs with the help of Dictionaries to solve the problems.

CO6: Develop python programs by using files.

5. Program Outcomes:

Engineering Graduates will be able to:

a. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

b. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

c. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

d. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

e. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

f. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

g. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental

contexts, and demonstrate the knowledge of, and need for sustainable development.

h. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

i. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

j. Communication: Communicate effectively on complex engineering activities wit the engineering community and wit society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

k. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

l. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

6. Mapping of Course Outcomes with Program Outcomes:

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**
CO1	**H**											**H**
CO2	**H**		**H**	**M**	**L**							
CO3	**H**		**H**	**M**	**L**							
CO4	**H**		**H**	**M**	**L**							
CO5	**H**		**H**	**M**	**L**							
CO6	**H**		**H**	**M**	**L**							

7. Prescribed Text Books

1. Reema Thareja, “Python Programming – Using Problem Solving Approach ", Oxford University Press, 2014 Edition.

8. Reference Text Books

1. Wesley J. Chun, “Core Python Programming”, Second Edition, Prentice Hall.
2. Martin C. Brown, “Python: The Complete Reference”, 2001 Edition, Osborne/Tata McGraw Hill Publishing Company Limited.
3. Kenneth A. Lambert, ‘Fundamentals of Python – first programs”, 2012 Edition, CENGAGE publication.

9. URLs and Other E-Learning Resources

https://pythonprogramming.net/beginner-python-programming-tutorials/
https://www.tutorialspoint.com/python/
https://www.javatpoint.com/python-tutorial
https://www.learnpython.org/
https://www.programiz.com/python-programming

10. Digital Learning Materials:

http://nptel.ac.in/courses/106106145/5
http://freevideolectures.com/Course/2512/Python-Programming
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/lecture-videos/
http://onlinevideolecture.com/?subject=python+programming

11. Lecture Schedule / Lesson Plan

Topic	No.of Periods	
	Theory	**Tutorial**
UNIT –1: Basics of Python programming		
Features of Python	1	1
History of Python	1	
Literal Constants	2	
Data Types ,Variables	2	
Operators	2	1
Input operation	1	
Write a python program to print "Hello World!" on the screen.	2	
Write a Python program to find sum of two numbers.		
Write a Python program to compute distance between two points taking input from the user. (use Pythagorean Theorem).	2	
Total	9+4	2
UNIT – 2: Decision Control and Looping Statements		
Conditional Branching.	2	1
un-conditional Branching.	1	
Iterative statements.	2	
Nesting of decision control statements.	2	
Nesting of loops.	1	1
Write a python program to test whether a given number is even or odd.	1	
Write a Python Program to print out the decimal equivalents of 1/2, 1/3, 1/4, . . . ,1/10, using a for loop.	2	
Write a Python Program to print a countdown from the given number to zero. Using a while loop.		
Write a Python Program to find the sum of all the primes below hundred.	2	
Write a Python program to find the factorial of a given number		
Total	8+5	2
UNIT–III: Functions and Strings		
Functions-function definition	2	1
Function call , Function return statement		
Types of arguments	2	
Recursive functions, Modules	2	
Strings -Basic string operations	2	
String formatting operator	2	1

Built-in functions		
Write a function cumulative_product to compute cumulative product of a list of numbers	1	
Write function to compute gcd, lcm of two numbers. Each function shouldn't exceed one line.		
Find the sum of the even-valued terms in the Fibonacci sequence whose values do not exceed ten thousand.	2	
Write a program that accepts a string from a user and re-displays the same after removing vowels from it.		
Write a program to calculate the length of a string.	1	
Write a function to reverse a given string.		
Total	10+4	2
UNIT–IV: Tuples, Lists		
Tuples – creating, accessing values.	2	1
Updating, deleting elements in a tuple.		
Basic Tuple operations.	2	
Lists – accessing, updating values in Lists.	2	
Basic List operations.		
mutability of lists.	2	
Creating Python Lists and deleting some elements, creating and accessing Python tuple elements.	2	1
write a program to swap two values using Tuple assignments		
Write a program to sort a Tuple of values	2	
Write program that scans an email address and forms a tuple of user name and domain name.		
Write a program to print sum and average of the elements present in the list.		
Write a program that forms a list of first character of every word present in another list.	2	
Total	8+6	2
UNIT–V: Dictionaries		
Dictionaries – Creating a Dictionary.	2	1
Adding an item, Deleting items.	2	
Sorting items		
Looping over a dictionary	2	
Basic Dictionary operations	2	1
Built-in functions	2	
Write a program to count the number of characters in the string and store them in a dictionary.	2	

Write a program to sort keys of a dictionary.		
Write a program that prints maximum and minimum value in a dictionary.	2	
Total	10+4	2
UNIT–VI: File Handling		
File types, File path.	2	1
File operations-open, close	2	
File operations-read,write	2	
Types of arguments	2	1
Write a program to print each line of a file in reverse order	2	
Write a program to compute the number of characters, words and lines in a file.	1	
Write a program to copy contents of one file into another file.	1	
Total	8+4	2
Total No of Periods	**53+27**	**12**

UNIT–I

Objective:

To explore basic knowledge on Python language basics features.

Syllabus:

Features and History of Python, Literal Constants, Data Types, Variables, Operators, input operation.

Programs: Write a python program to 1. Print “Hello World!” on the screen.2. Find sum of two numbers. 3. Compute distance between two points taking input from the user. (use Pythagorean Theorem)

Learning Outcomes:

At the end of the unit student will be able to

- understand the features and history of python.
- develop algorithms and design logical flow charts for solving problems.
- describe python tokens.
- solve simple formula based problems on computer using Python.

LEARNING MATERIAL

1. Features and History of Python:

1.1Features of Python

- **Simple:** Reading a program written in Python feels almost like reading english. The main strength of Python which allows programmer to concentrate on the solution to the problem rather than language itself.
- **Easy to Learn:** Python program is clearly defined and easily readable. The structure of the program is simple. It uses few keywords and clearly defined syntax.
- **Versatile:** Python supports development of wide range of applications such as simple text processing, WWW browsers and games etc..
- **Free and Open Source:** It is a Open Source Software. So, anyone can freely distribute it, read the source code, edit it, and even use the code to write new (free) programs.
- **High-level Language:** While writing programs in Python we do not worry about the low-level details like managing memory used by the program.
- **Interactive:** Programs in Python work in interactive mode which allows interactive testing and debugging of pieces of code. Programmer can easily interact with the interpreter directly at the python prompt to write their programs.
- **Portable**: It is a portable language and hence the programs behave the same on wide variety of hardware platforms with different operating systems**.**
- **Object Oriented:** Python supports object-oriented as well as procedure-oriented style of programming .While object-oriented technique encapsulates data and functionality with in objects, Procedure oriented at other hand, builds programs around procedure or functions.

- **Interpreted:** Python is processed at runtime by interpreter. So, there is no need to compile a program before executing it. You can simply run the program. Basically python converts source program into intermediate form called byte code.
- **Dynamic and strongly typed language:** Python is strongly typed as the interpreter keeps track of all variables types. It's also very dynamic as it rarely uses what it knows to limit variable usage.
- **Extensible:** Since Python is an open source software, anyone can add low-level modules to the python interpreter. These modules enable programmers to add to or customize their tools to work more efficiently.
- **Embeddable:** Programmers can embed Python within their C, C++, COM, ActiveX, CORBA and Java Programs to give 'scripting 'capability for users.
- **Extensive Libraries:** Python has huge set of libraries that is easily portable across different platforms with different operating systems.
- **Easy maintenance:** Code Written in Python is easy to maintain.
- **Secure:** This Programming language is secure for tampering. Modules can be distributed to prevent altering of source code. Additionally, Security checks can be easily added to implement additional security features.
- **Robust:** Python Programmers cannot manipulate memory directly, errors are raised as exceptions that can be catch and handled by the program code. For every syntactical mistake, a simple and easy to interpret message is displayed. All these make python robust.
- **Multi-threaded:** Python supports executing more than one process of a program simultaneously with the help of Multi Threading.
- **Garbage Collection:** The Python run-time environment handles garbage collection of all python objects. For this, a reference counter is maintained to assure that no object that is currently in use is deleted.

1.2 History of Python.

- Python was first developed by ***Guido van Rossum*** in the late 80's and early 90's at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- It has been derived from many languages such as ABC, Modula-3, C, C++, Algol-68, SmallTalk, UNIX shell and other scripting languages.
- Since early 90's Python has been improved tremendously. Its version 1.0 was released in 1991, which introduced several new functional programming tools.
- While version 2.0included list comprehension was released in 2000 by the Be Open Python Labs team.
- Python 2.7 which is still used today will be supported till 2020.
- Currently Python 3.6.4 is already available. The newer versions have better features like flexible string representation e.t.c,
- Although Python is copyrighted, its source code is available under GNU General Public License (GPL) like that Perl.
- Python is currently maintained by a core development team at the institute which is directed by Guido Van Rossum.
- These days, from data to web development, Python has emerged as very powerful and popular language. It would be surprising to know that python is actually older than Java, R and JavaScript.

1.3 Applications of Python:

- **Embedded scripting language**: Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.
- **3D Software**: 3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.
- **Web development**: Python is an easily extensible language that provides good integration with database and other web standards.

- **GUI-based desktop applications***:* Simple syntax, modular architecture, rich text processing tools and the ability to work on multiple operating systems makes Python a preferred choice for developing desktop-based applications.
- **Image processing and graphic design applications***:* Python is used to make 2D imaging software such as Inkscape, GIMP, Paint Shop Pro and Scribus. It is also used to make 3D animation packages, like Blender, 3ds Max, Cinema 4D, Houdini, Light wave and Maya.
- **Scientific and Computational applications***:* Features like high speed, productivity and availability of tools, such as Scientific Python and Numeric Python, have made Python a preferred language to perform computation and processing of scientific data. 3D modeling software, such as FreeCAD, and finite element method software, like Abaqus, are coded in Python.
- **Games:** Python has various modules, libraries, and platforms that support development of games. Games like Civilization-IV, Disney's Toontown Online, Vega Strike, etc. are coded using Python.
- **Enterprise and Business applications:** Simple and reliable syntax, modules and libraries, extensibility, scalability together make Python a suitable coding language for customizing larger applications. For example, Reddit which was originally written in Common Lips, was rewritten in Python in 2005. A large part of Youtube code is also written in Python.
- **Operating Systems:** Python forms an integral part of Linux distributions.

1.4 Keyword in Python:

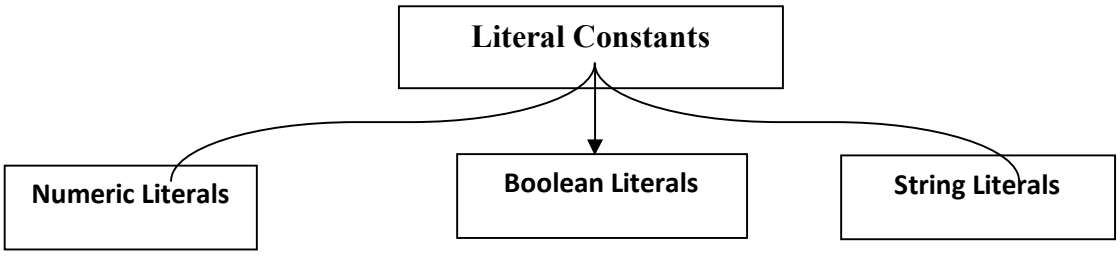
- Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier.
- Here's a list of all keywords in Python Programming.

- There are 33 keywords in Python 3.3. This number can vary slightly in course of time.
- All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

Keywords in Python Programming Language				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

2. Literal Constants

- In programming constants are referred to variables that cannot be changed.
- Generally Literal constants are classified in to three types.



2.1 Numeric Literals

- The value of a literal constant can be used directly in programs. For example, 7, 3.9, 'A', and "Hello" are literal constants.
- Numbers refers to a numeric value. You can use four types of numbers in Python program- **integers, long integers, floating point and complex numbers.**
- Numbers like 5 or other whole numbers are referred to as ***integers.*** Bigger whole numbers are called ***long integers***. For example, 535633629843L is a long integer.
- Numbers like are 3.23 and 91.5E-2 are termed as ***floating point numbers***.
- Numbers of a + bj form (like -3 + 7j) **are *complex numbers***.

>>> 50 + 40 - 35 55	>>> 12 * 10 120	>>> 96 / 12 8.0	>>> (-30 * 4) + 500 380

>>> 78//5 15	>>> 78 % 5 3	>>> 152.78 // 3.0 50.0	>>> 152.78 % 3.0 2.780000000000001

```
>>> 5**3
125
>>> 121**0.5
11.0
```

2.2 Boolean Literals

- A Literals Boolean type can have one of the two values- True or False.

Examples:

>>>Boolean_var = True >>>print(Boolean_var) True	>>> 20 == 30 False	>>>"Python" == "Python" True
>>> 20 != 20 False	>>>"Python"! = "Python3.4" True	>>>30 > 50 False
>>> 90 <= 90 True	>>>87 == 87.0 False	>>>87 > 87.0 False
>>>87 < 87.0 False	>>>87 >= 87.0 True	>>>87 <= 87.0 True

Programming Tip: <, > operators can also be used to compare strings lexicographically.

2.3 String Literals

- A *string* is a group of characters.
- *Using Single Quotes (')*: For example, a string can be written as 'HELLO'.
- *Using Double Quotes (")*: Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO".
- *Using Triple Quotes (''' ''')*: You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.

Examples:

```
>>> 'Hello'
'Hello'
```

```
>>> "HELLO"
'HELLO'
```

```
>>> '''HELLO'''
'HELLO'
```

2.3.1Unicode Strings

- Unicode is a standard way of writing international text. That is,if you want to write some text in your native language like hindi,then you need to have a Unicode-enable text editor.
- Python allows you to specify Unicode text by prefixing the string with a u or U.
- For Example: u"Sample Unicode string"

  Note :The 'U' prefix specifies that the file contains text written in language other than English

2.3.2Escape Sequences

- Some characters (like ", \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them.

Example:

```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

Escape Sequence	Purpose	Example	Output
\\	Prints Backslash	print("\\")	\
\'	Prints single-quote	print("\'")	'
\"	Prints double-quote	print("\"")	"
\a	Rings bell	print("\a")	Bell rings
\f	Prints form feed character	print("Hello\fWorld")	Hello World
\n	Prints newline character	print("Hello\nWorld")	Hello World
\t	Prints a tab	print("Hello\tWorld")	Hello World
\o	Prints octal value	print("\o56")	.
\x	Prints hex value	print("\x87")	+

2.3.2Raw Strings

- If you want to specify a string that should not handle any escape sequences and want to display exactly as specified then you need to specify that string as a *raw string*. A raw string is specified by prefixing r or R to the string.

Example:

```
>>> print(R "What\'s your name?")
What\'s your name?
```

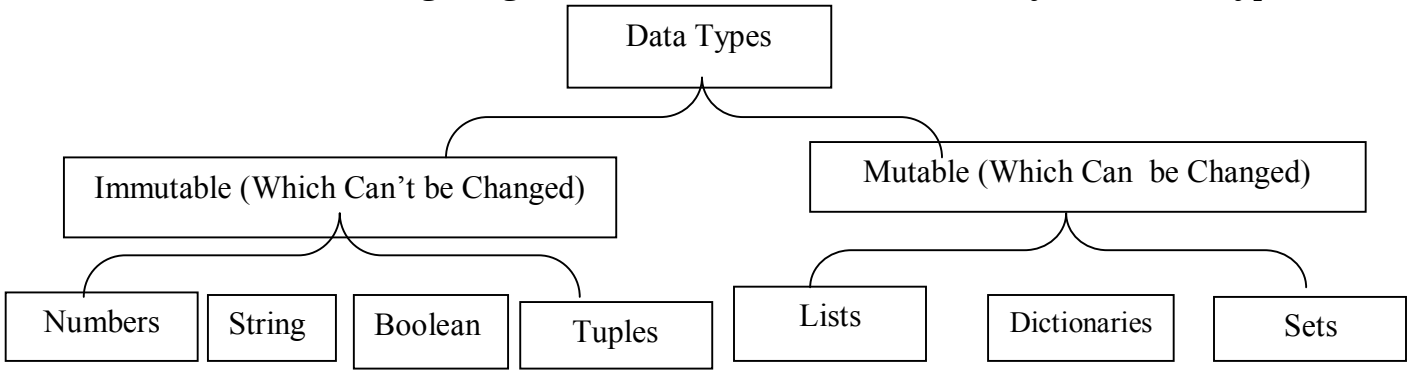
3.Data types

- The variables can hold values of different type called ***Data Type***.
- Data type is a set of values and the allowable operations on those values.
- Python has a great set of useful data types. Python's data types are built in the core of the language. They are easy to use and straightforward.
- Example a person age is stored in a number ,his name is made only with characters, and his address is made with mixture of numbers and characters.
- Python ha various standard data types to define the operations possible on them and storage method for each of them.
- Python supports the following five standard data types

1.Numbers
2.Strings
3.Lists
4.Tuple
5.Dictionary

Note: Python is pure object oriented programming language.it refers to everything as an object including numbers and strings.

- The Following Diagram shows the classification of Python Data Types.



3.1.1. Assigning or Initializing Values to Variables

- In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign (=). The operand on the left side of equal sign is the name of the variable and the operand on its right side is the value to be stored in that variable.
  Example: Program to display data of different types using variables and literal constants.

```
num = 7
amt = 123.45
code = 'A'
pi = 3.1415926536
population_of_India = 10000000000
msg = "Hi"

print("NUM = "+str(num))
print("\n AMT = "+ str(amt))
print("\n CODE = " + str(code))
print("\n POPULATION OF INDIA = " + str(population_of_India))
print("\n MESSAGE = "+str(msg))

OUTPUT
NUM = 7
AMT = 123.45
CODE = A
POPULATION OF INDIA = 10000000000
MESSAGE = Hi
```

- In Python , you can reassign variables as many times as you want to change the value stored in them. You may even store value of one data type in a statement and other data type in subsequent statement. This is possible because Python variables do not have specific types, i.e., we can assign integer to the variable, later we assign string to the same variable.
  Example: Program to reassign value to a variable

```
val = 'Hello'
print(val)
val = 100
print(val)
val=10.32
print(val)
```

Output

Hello

100

10.32

3.1.2Multiple Assignments

- Python allows programmers to assign single value to more than one variable simultaneously.
- For example

  >>>sum = flag = a = b = 0
- In the above statement, all four integer variables are assigned a value 0.You can also assign different values to multiple variables simultaneously as shown below
- For example

  >>>sum, a, b, mesg = 0, 3, 5, “Result”

  Here, variable sum,a and b are integers(numbers) and mesg is assigned “Result”.

Note: Removing a variable means that the reference from the name to the value has been deleted.However, deleted variables can be used again in the code if and only if you reassign them some value.

3.2 Boolean Type

A variable of Boolean type can have one of the two values- True or False.

Similar to other variables, the Boolean variables are also created while we assign a value to them or when we use a relational operator on them**.**

Examples:

>>>Boolean_var = True >>>print(Boolean_var) True	>>> 20 == 30 False	>>>"Python" == "Python" True
>>> 20 != 20 False	>>>"Python"! = "Python3.4" True	>>>30 > 50 False
>>> 90 <= 90 True	>>>87 == 87.0 False	>>>87 > 87.0 False
>>>87 < 87.0 False	>>>87 >= 87.0 True	>>>87 <= 87.0 True

Programming Tip: <, > operators can also be used to compare strings lexicographically.

3.3 Tuples

- A tuple is similar to the list as it also consists of a number of values separated by commas and enclosed within parentheses.
- The main difference between lists and tuples is that you can change the values in a list but not in a tuple. This means that while tuple is a read only data type, the list is not.

Examples:

```
Tup = ('a', 'bc', 78, 1.23)
Tup2 = ('d', 78)
print(Tup)
print(Tup[0])       # Prints first element of the Tuple
print(Tup[1:3])     # Prints elements starting from 2nd till 3rd
print(Tup[2:])      # Prints elements starting from 3rd element
print(Tup *2)       # Repeats the Tuple
print(Tup + Tup2)  # Concatenates two Tuples

OUTPUT
('a', 'bc', 78, 1.23)
a
('bc', 78)
(78, 1.23)
('a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23)
('a', 'bc', 78, 1.23, 'd', 78)
```

3.4 Lists

- Lists are the most versatile data type of Python language.
- A list consist of items separated by commas and enclosed within square brackets The values stored in a list are accessed using indexes.
- The index of the first element being 0 and n-1 as that of the last element, where n is the total number of elements in the list. Like strings, you can also use the slice, concatenation and repetition operations on lists.
- Example program to demonstrate operations on lists

```
list = ['a', 'bc', 78, 1.23]
list1 = ['d', 78]
print(list)
print(list[0])
print(list[1:3])
print(list[2:])
print(list * 2)
print(list + list1)
```

Output:

['a', 'bc', 78, 1.23]

a

['bc', 78]

[78, 1.23]

['a', 'bc', 78, 1.23, 'a', 'bc', 78, 1.23]

['a', 'bc', 78, 1.23, 'd', 78]

3.5 Dictionary

- Python's dictionaries stores data in key-value pairs.
- The key values are usually strings and value can be of any data type. The key value pairs are enclosed with curly braces ({ }).
- Each key value pair separated from the other using a colon (:). To access any value in the dictionary, you just need to specify its key in square braces ([]).Basically dictionaries are used for fast retrieval of data.

Example

```
Dict = {"Item" : "Chocolate", "Price" : 100}
print(Dict["Item"])
print(Dict["Pricc"])

OUTPUT
Chocolate
100
```

4. Variables and Identifiers

4.1 Variables

- Variable means its value can vary. You can store any piece of information in a variable.

- Variables are nothing but just parts of your computer's memory where information is stored. To identify a variable easily, each variable is given an appropriate name.

4.2Identifiers

Identifiers are names given to identify something. This something can be a variable, function, class, module or other object. For naming any identifier, there are some basic rules like:

- The first character of an identifier must be an underscore ('_') or a letter (upper or lowercase).
- The rest of the identifier name can be underscores ('_'), letters (upper or lowercase), or digits (0-9).
- Identifier names are case-sensitive. For example, myvar and myVar are not the same.
- Punctuation characters such as @, $, and % are not allowed within identifiers.
- *Examples of valid identifier names* are sum, __my_var, num1, r, var_20, First, etc.
- *Examples of invalid identifier names* are 1num, my-var, %check, Basic Sal, H#R&A, etc.,

5. Operators

- Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.
- For example:

```
>>> 2+3
5
```

Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.

- Python supports the following operators
    1. Arithmetic operators
    2. Comparison (Relational) operators
    3. Unary Operators
    4. Bitwise operators

5. Shift Operators
6. Logical Operators
7. Membership and Identity Operators
8. Assignment operators
9. Special operators

5.1Arithmetic Operators

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.
- This operator will work on two operands.
- Example: If a=100 and b=200 then look at the table below, to see the result of arithmetic operations.

Operator	Description	Example	Output
+	Addition: Adds the operands	>>> print(a + b)	300
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	>>> print(a – b)	-100
*	Multiplication: Multiplies the operands	>>> print(a * b)	20000
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	>>> print(b / a)	2.0
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	>>> print(b % a)	0

//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	>>> print(12//5)
		>>> print(12.0//5.0)
		>>> print(-19//5)
		>>> print(-20.0//3)
			2
			2.0
			
			-4
			-7.0

**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	>>> print(a**b)	100^{200}

5.2Comparision (Relational) Operators

- A Relational or Comparison operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0
- For Example assuming a=100 and b=2000,we can use the comparison operators on them as specified in the following table.

Operator	Description	Example	Output
==	Returns True if the two values are exactly equal.	>>> print(a == b)	False
!=	Returns True if the two values are not equal.	>>> print(a != b)	True
>	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	>>> print(a > b)	False
<	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	>>> print(a < b)	True
>=	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	>>> print(a >= b)	False
<=	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	>>> print(a <= b)	True

5.3Unary Operator

- Unary operators act on single operands. Python supports unary minus operator.
- Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand.
- When an operand is preceded by a minus sign, the unary operator negates its value.
- For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.

b = 10 a = -(b)

- The result of this expression, is a = -10, because variable b has a positive value. After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.

5.4 Bitwise Operators

- As the name suggests, bitwise operators perform operations at the bit level.

- These operators include bitwise AND, bitwise OR, bitwise XOR, and shift operators.
- Bitwise operators expect their operands to be of integers and treat them as a sequence of bits.
- The truth tables of these bitwise operators are given below.

A	B	A&B	A	B	A|B	A	B	A^B	A	!A
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

- Example: If a=60 and b=13 then look at the table below, to see the result of Bitwise operations.

Operator	**Description**	**Example**
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) =12 (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.

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

5.5 Shift Operators

- Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>).
- These operations are used to shift bits to the left or to the right. The syntax for a shift operation can be given as follows:

Examples:

```
operand op num
```

```
if we have x = 0001 1101, then
x << 1 gives result = 0011 1010
```

```
if we have x = 0001 1101, then
x << 4 gives result = 1010 0000
```

```
if we have x = 0001 1101, then
x >> 1 gives result = 0000 1110.
Similarly, if we have x = 0001 1101 then
x << 4 gives result = 0000 0001
```

5.6 Logical Operators

- Logical operators are used to simultaneously evaluate two conditions or expressions with relational operators.
- ***Logical AND (and)*** If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true.

  For example, If we have an expression (a>b) and (b>c), then the whole expression is true only if both expressions are true. That is, if b is greater than a and c.
- ***Logical OR (or)*** operator is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the

expressions of the logical operator is true, then the whole expression is true.

For example, If we have an expression (a>b) or (b>c), then the whole expression is true if either b is greater than a or b is greater than c.

- ***Logical NOT (not)*** operator takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a 1 if the expression produces a zero. In other words, it just reverses the value of the expression.

  For example, a = 10; b = not a; Now, the value of b = 0.

5.7 Membership and Identity Operators

5.7.1. Membership Operator

Python supports two types of membership operators–in and not in. These operators, test for membership in a sequence such as strings, lists, or tuples.

- **in Operator:** The operator returns true if a variable is found in the specified sequence and false otherwise. For example, a in nums returns 1, if a is a member of nums.
- **not in Operator:** The operator returns true if a variable is not found in the specified sequence and false otherwise. For example, a not in nums returns 1, if a is not a member of nums.

5.7.2. Identity Operators

- **is Operator:** Returns true if operands or values on both sides of the operator point to the same object and false otherwise. For example, if a is b returns 1, if id(a) is same as id(b).
- **is not Operator:** Returns true if operands or values on both sides of the operator does not point to the same object and false otherwise. For example, if a is not b returns 1, if id(a) is not same as id(b).

5.8 Assignment Operators

- Assignment operators are used in Python to assign values to variables.

- a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

Assignment operators in Python		
Operator	**Example**	**Equivatent to**
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

5.9 Operator Precedence

- The operator precedence in Python are listed in the following table. It is in descending order, upper group has higher precedence than the lower ones.

Operator precedence rule in Python	
Operators	**Meaning**
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparison, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

6. Input Operation

- To take input from the users, Python makes use of the input( ) function. The input( ) function prompts the user to provide some information on which the program can work and give the result.
- However, we must always remember that the input function takes user's input as a string.

Example:

```
name = input("What's your name?")
age = input("Enter your age : ")
print(name + ", you are " + age + " years old")
```

OUTPUT

```
What's your name? Goransh
Enter your age : 10
Goransh, you are 10 years old
```

7. Comments

- Comments are the non-executable statements in a program. They are just added to describe the statements in the program code.
- Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.
- In Python, a hash sign (#) that is not inside a string literal begins a comment. All characters following the # and up to the end of the line are part of the comment

Example:

```
# This is a comment
print("Hello")  # to display hello
# Program ends here
```

OUTPUT

```
Hello
```

\

- **Note:** For writing *Multi line comments*. Make sure to indent the leading ‘ ‘ ‘ appropriately to avoid an Indentation Error

```
‘ ‘ ‘
This is a multiline
comment.
‘ ‘ ‘
```

8. Indentation

- Whitespace at the beginning of the line is called *indentation*. These *whitespaces or the indentation* are very important in Python.
- In a Python program, the leading whitespace including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

Example:

```
age = 21
    print("You can vote") # Error! Tab at the start of the line
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2
    print("You can vote")
    ^
IndentationError: unexpected indent
```

9. Expressions

- An expression is any legal combination of symbols (like variables, constants and operators) that represents a value.
- In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value. *Operand* is the value on which operator is applied.
- Generally Expressions are divided into the following types
  1. *Constant Expressions*: One that involves only constants.
     Example: 8 + 9 – 2
  2. *Integral Expressions:* One that produces an integer result after evaluating the expression.
     Example:a = 10
  3. *Floating Point Expressions:* One that produces floating point results.
     Example: a * b / 2.0
  4. *Relational Expressions: One that returns either true or false value.*
     Example: c = a>b
  5. *Logical Expressions:* One that combines two or more relational expressions and returns a value as *True* or *False*.
     Example: a>b and y! = 0

6. *Bitwise Expressions:* One that manipulates data at bit level.
   Example: x = y&z
7. *Assignment Expressions:* One that assigns a value to a variable.
   Example: c = a + b or c = 10

   Example Program:

   Give the output for the following statements.(***April 2018 Regular***)

```
a = 20
b = 10
c = 15
d = 5
print ("a:%d b:%d c:%d d:%d" % (a,b,c,d ))
e = (a + b) * c / d
print ("Value of (a + b) * c / d is ", e)
e = ((a + b) * c) / d
print ("Value of ((a + b) * c) / d is ", e)
e = (a + b) * (c / d)
print ("Value of (a + b) * (c / d) is ", e)
e = a + (b * c) / d
print ("Value of a + (b * c) / d is ", e)
```

   Output:

```
a:20 b:10 c:15 d:5
Value of (a + b) * c / d is  90.0
Value of ((a + b) * c) / d is  90.0
Value of (a + b) * (c / d) is  90.0
Value of a + (b * c) / d is  50.0
```

10. Operations on Strings

- Like numbers we can also manipulate strings by performing operations on them.
- Basically there are three operations on strings
  1. String concatenation.
  2. String repetition or multiplication.

3.String slicing.

10.1String Concatenation

- Like numbers we can add two strings in Python. The process of combining two strings is called concatenation.
- Two strings whether created using single or double quote are concatenated in the same way.Look at the codes given in the following example.
- Example: Codes to demonstrate how easily two strings are concatenated.

```
>>>print("hello"  +  "-world")
```

output

hello-world

```
>>>print("hello"+'-world')
```

output

hello-world

```
>>print('hello'+'-world')
```

output

hello-world.

Note:we cannot add string to number which generates an error.

10.2 String Repetition or Multiplication

- You cannot add string and number but we can multiply string and number with the help of this string repetition.
- When an string is multiplied with an integer n, then the string is repeted n times.
- Thus, the * operator is also called as string repetition operator.
- The order of string and integer is not important.

Example

```
>>print("hello" * 5)
```

Output

hello hello hello hello hello

```
>>print(5 * "hello")
```

Output

hello hello hello hello hello

Note:We cannot multiply two strings and cannot multiply string with floating point number.

10.3 Slice Operations on Strings

- You can extract subsets of strings by using the slice operator ([ ] and [:]). You need to specify index or the range of index of characters to be extracted.
- The index of the first character is 0 and the index of the last character is n-1, where n is the number of characters in the string.
- If you want to extract characters starting from the end of the string, then you must specify the index as a negative number. For example, the index of the last character is -1.
- Program to perform slice operation on strings

```
#string operations
str = 'Python is easy !!!'
print(str)
print(str[0])
print(str[3:9])
print(str[4:])
print(str[-1])
print(str[:5])
print(str * 2)
print(str + 'Isn't gec')
```

Output

```
Python is easy !!!
P
hon is
on is easy
!
Pytho
Python is easy !!! Python is easy !!!
Python is easy !!! Isn't gec
```

11.Type Conversion

- In Python, it is just not possible to complete certain operations that involves different types of data.
- For example, it is not possible to perform "2" + 4 since one operand is an integer and the other is of string type.

```
>>>"20" + "30"                    >>> int("2") + int("3")
```

Output **Output**

'2030' **5**

- Another situation in which type conversion is must when we want to accept a non string value(integer or float) as an input.we know that input function returns string,so we must typecast the input to numbers to perform calculations on them.
- **Example1:**

```
x=input("Enter the first number")
y=input("Enter the second number")
print(x+y)
```

output

```
Enter the first number 6
Enter the second number 7
67
```

- **Example2:**

```
x=int(input("Enterthefirstnumber))
y=int(input("Enterthesecondnumber))
print(x+y)
```

Output

```
Enterthefirstnumber6
Enterthesecondnumber7
13
```

- Python provides various built-in functions to convert value from one data type to another datatype.The following are the functions return new object representing the coverted value. Some of them are given in the following table.

Function	Description
int(x)	Converts x to an integer
long(x)	Converts x to a long integer
float(x)	Converts x to a floating point number
str(x)	Converts x to a string
tuple(x)	Converts x to a tuple
list(x)	Converts x to a list
set(x)	Converts x to a set
ord(x)	Converts a single character to its integer value
oct(x)	Converts an integer to an octal string
hex(x)	Converts an integer to a hexadecimal string
chr(x)	Converts an integer to a character
unichr(x)	Converts an integer to a Unicode character
dict(x)	Creates a dictionary if x forms a (key-value) pair

12Type Casting vs Type Coercion

- we have done explicit conversion of a value from one data type to another. This is known as *type casting*.
- However, in most of the programming languages including Python, there is an implicit conversion of data types either during compilation or during run-time. This is also known *type coercion*.
- For example, in an expression that has integer and floating point numbers (like 21 + 2.1 gives 23.1), the compiler will automatically convert the integer into floating point number so that fractional part is not lost.

13 Limitations of Python:

- Parallel processing can be done in Python but not as elegantly as done in some other languages (like JavaScript and Go Lang).
- Being an interpreted language, Python is slow as compared to C/C++. Python is not a very good choice for those developing a high-graphic 3d game that takes up a lot of CPU.
- As compared to other languages, Python is evolving continuously and there is little substantial documentation available for the language.
- As of now, there are few users of Python as compared to those using C, C++ or Java.
- It lacks true multiprocessor support.
- It has very limited commercial support point.

- Python is slower than C or C++ when it comes to computation heavy tasks and desktop applications.
- It is difficult to pack up a big Python application into a single executable file. This makes it difficult to distribute Python to non-technical.

Procedure for Executing your First Python Program

Step1: Type the program in notepad and save your python program with the following extension

Filename.py

Step2: To Open Python Shell goto

Start->All Programs->Python 3.4->IDLE(Python GUI)

Step3: To Load your python file in the shell goto

Select->open->Select the path of the python file where it actually Store.

*Output:*A Separate window will be opened

Step4: For running your Python program click on run tab on the top of the separate window in the previous step and select run module or simply press F5.

1.simplemsg.py

Write a Python program to display the simple message "Hello World"

```
print("Hello World")
```

Output

```
Hello World
```

2.sumoftwonumbers.py

Write a Python program to add two numbers

```
num1=int(input("Enter first number"))
num2=int(input("Enter second number"))
res=num1+num2
print(str("The sum of two numbers is"))
print(str(num1)+"+"+str(num2)+"="+str(res))
```

Output:

```
Enter first number 10
Enter second number 20
```

The sum of two numbers is

10+20=30

3.distancebetweentwopoints.py

#Write a Python Program to find distance between two points using Pythagoras Theorem

```
x1=(int(input("Enter x1")))
x2=(int(input("Enter y1")))
y1=(int(input("Enter x2")))
y2=(int(input("Enter y2")))
distance=((x2-x1)**2+(y2-y1)**2)**0.5
print("Distance Between two points")
print(distance)
```

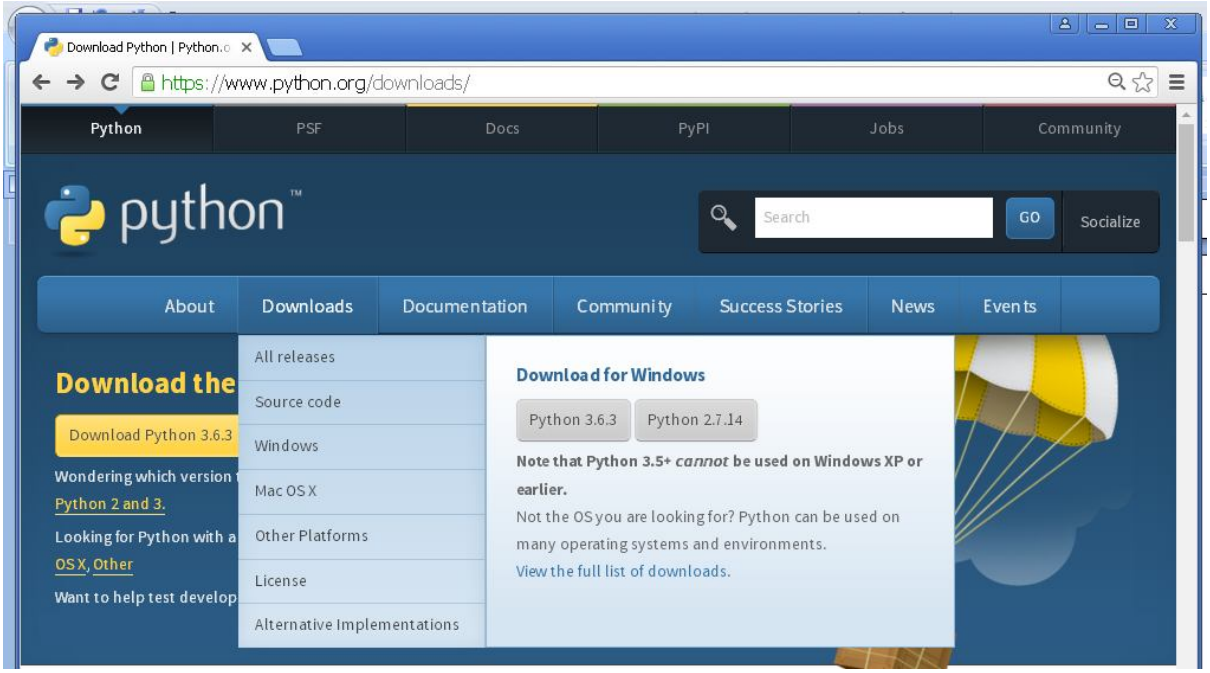
Output:

```
Enter x1   8
Enter y1   9
Enter x2   10
Enter y2  12
Distance Between two points
2.2360679775
```

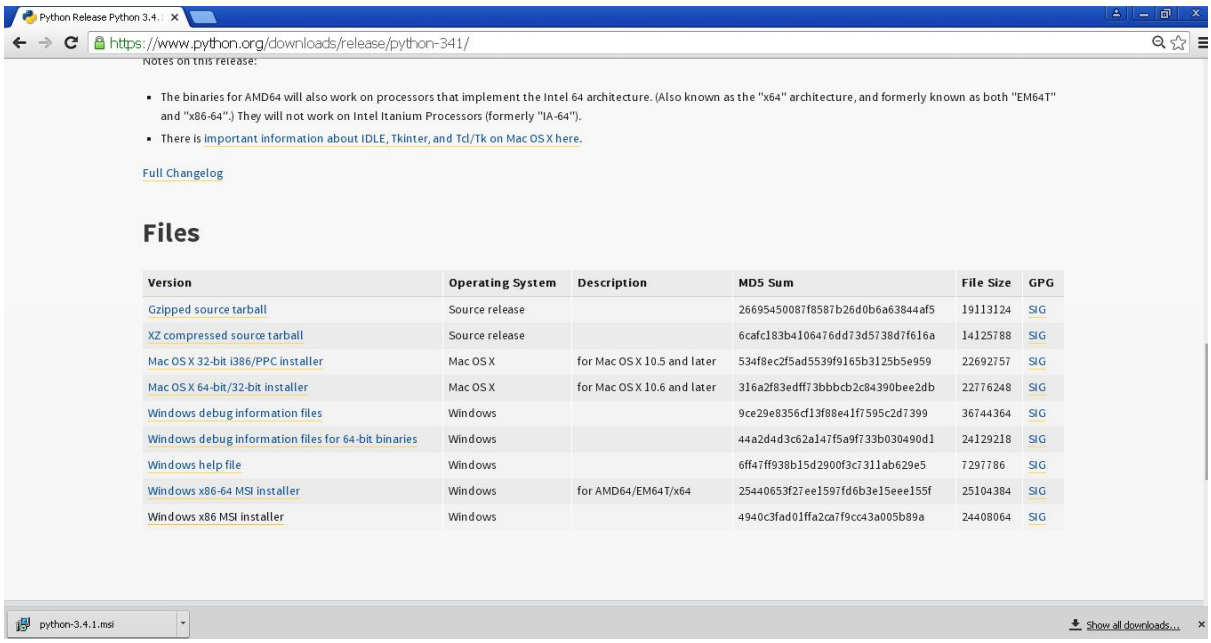
HOW TO INSTALL PYTHON

Step1: In order to use Python, it must first be installed on your computer. Follow these steps.

Go to the python website **www.python.org** and click on the 'Download' menu choice.



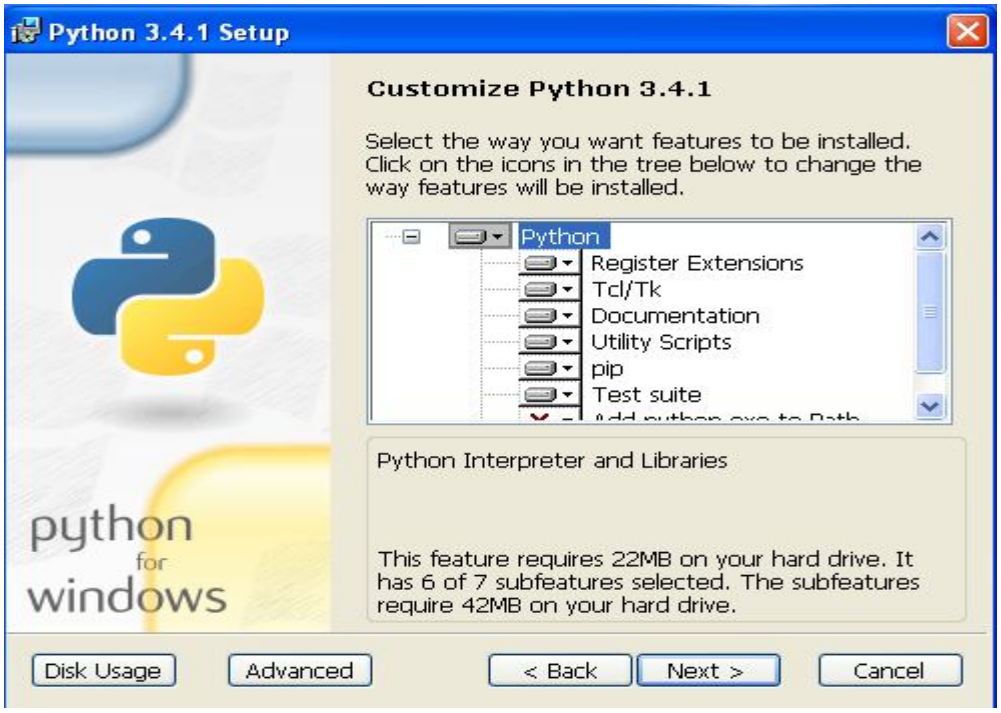
Step2: Next click on the Python 3.4.1 (note the version number may change) Windows Installer to download the installer. If you know you're running a 32-bit os, you can choose the **Windows x86 MSI installer**



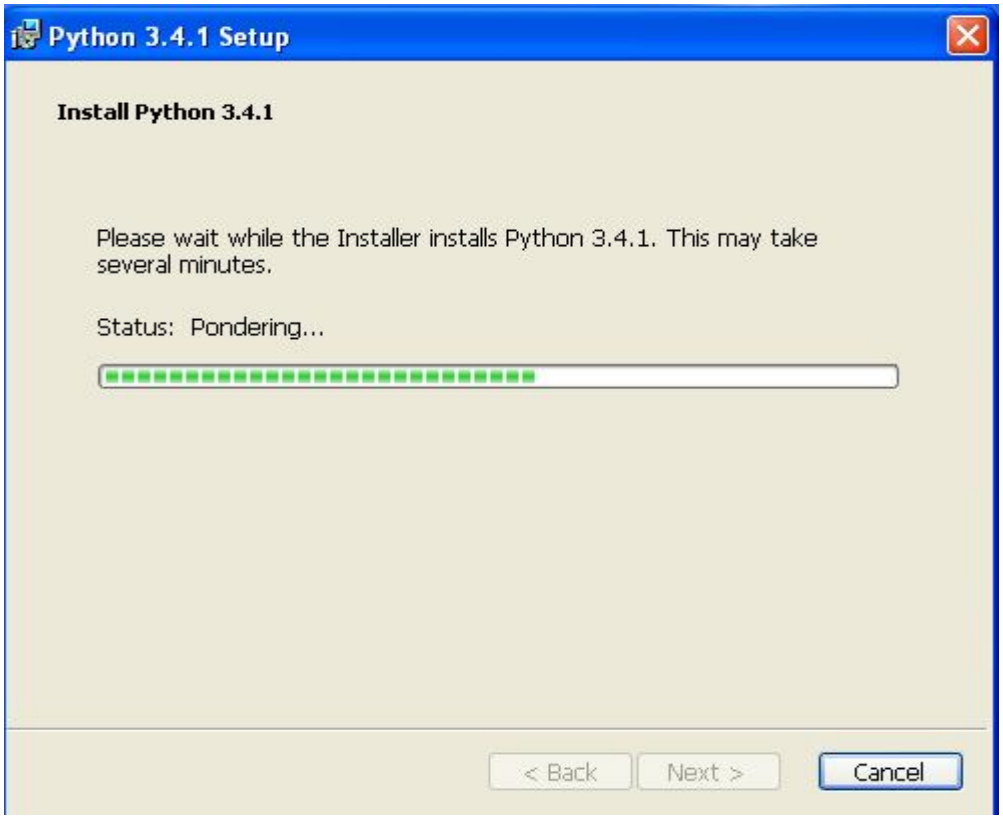
Step3: Once the installer starts, it will ask who to install the program for. Usually installing for all users is the best choice.

Step4: Next, it needs to know where to install the file. The default choice is fine.

Step5: You don't need to install the entire package, but we did.

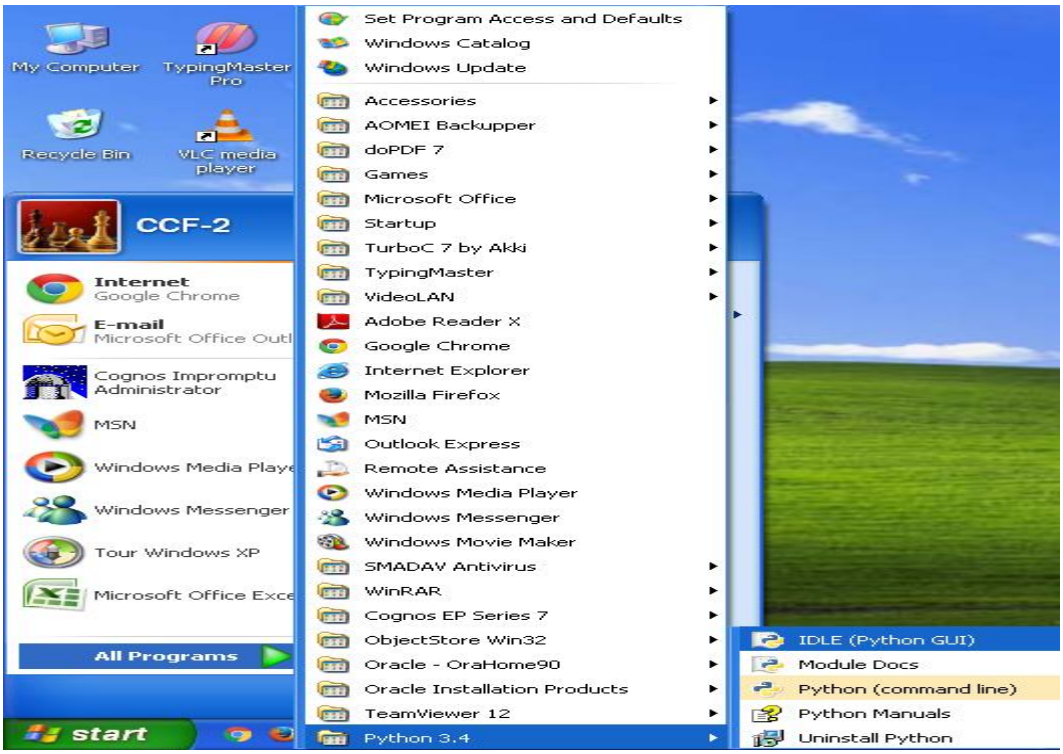


Step6: It will take a while to install.



Step7: Click 'Finish' to exit the installer

Step8: After installed, you should now have a Python menu choice. Start the program by choosing IDLE (Python GUI)



Assignment-Cum-Tutorial Questions

Objective Questions

SECTION-A

1. Literal is of the form a+bj is called_____
2. Identify the words which describes Python [ ]
   a)Interpreted b)simple c)reliable d)all of these
3. Python allows you to specify Unicode Text by prefixing the string with which character [ ]
   a)U b)R c)S d)A
4. Which of the following is a valid string literal [ ]
   a)"computer" b)'computer' c)'''computer''' d) all of these
5. Which of this is valid variable name in Python [ ]
   a) This is a variable b)This_is_a_variable c)This-is-a-variable d)^var
6. A Comments in python start with which symbol_____
7. All spaces and tabs with in a string are preserved in quotes [True/False]
8. Bitwise Operator can be applied on which datatype [ ]
   a)integer b)float c)string d)list
9. Identify valid assignment statements [ ]
   a)=b+1 b)a=a+1 c)a+b=10 d)a+1=1
10. _____operator perform logical negation on each bit of the operand.
11. What should be written in the blank to generate ZeroDivisionError in the case of (25+36)/(-8+_____)
12. Predict the output of the following program [ ]
    >>spam="eggs"
    >>print(spam*3)
    a)spamspamspam b)eggseggseggs c)"spamspamspam" d)spam*3
13. Which of the following returns true [ ]
    a)>>>9=9 and 1==1
    b)>>>3==5 and 7==3
    c)>>>7!=1 and 5==5
    d)>>>4<1 and 1>6
14. Identify the valid numeric literals in Python [ ]

a)5678 b)5,678 c)5678.0 d)0.5678 e)0.56+10

15. You can print string without using print function [True/False]
16. Predict the output of the following program [ ]

    >>>print (format(56.78901,'.3f'))

    a)56.789 b)5.6789 c)0.56789 d)56789
17. The following statement will produce ___lines of output [ ]

    >>print('Good\nMorning\nWorld\n---Bye')

    a)1 b)2 c)3 d)4
18. Identify the correct arithmetic expression in python [ ]

    a)11(12+13) b)(5*6)(7+8) c)4*(3-2) d)5***3
19. Which line of code produce error [ ]

    a)"one"+"2" b)'5'+6 c)3+4 d)"7"+'eight'
20. Predict the output of the following program [ ]

    >>>print(abs(10-20)*3)

    a)-30 b)30 c)-50 d)none of these

SECTION-B

Subjective Questions

1. Describe the features of Python
2. Differentiate between literals and variables in python. (***November 2018*** **Supplementary**)
3. What are literals? Explain with the help of suitable examples?
4. Explain the significance of Escape sequences with relevant examples
5. Write briefly about Data types in Python
6. Explain in detail about Membership and Identity Operators.
7. How can the ternary operator used in python? (***April 2018 Regular***)
8. Give the operator precedence in python. (***November 2018*** **Supplementary**)
9. Define Expression? Explain different types of Expressions supported by Python?
10. Differentiate string with slicing operator.
11. What is tuple? What are the different operations performed on tuple? Explain with an example? (***November 2018*** **Supplementary**)
12. Write briefly about Type Conversion process in Python.Write the meaning for the following. str(x), chr(x), float(x), ord(x) (***November 2018*** **Supplementary**)

13. Momentum is calculated as, $e=mc^2$, where m is the mass of the object and c is the velocity. Write a Python program that accepts object's mass (in kilograms) and velocity (in meters per second) and displays its momentum.

14. a)Write a Python Program to convert temperature in Celsius to Fahrenheit

    b) Write a Python Program to convert Fahrenheit to Celsius.

15. Write a Python program to calculate the area of triangle using Heron's formula

$$\textbf{\textit{Hint}}: \sqrt{s(s-a)(s-b)(s-c)}$$

16. Evaluate the following Expression

    a) True and False

    b) (100<0) and (100>20)

    c) not(true) and false

    d) not true and false or true

    e) not(100<0 or 100>20)

    f)100<0 and not 100>20

17. Give an appropriate boolean expression for the each of the following

    a)check if variable v is greater than or equal to 0,and less than 10

    b)check if variable v is less than 10 and greater than or equal to 0,or it is equal to 20.

    c)check if either the name 'cse' or 'it' appears in the list of names assigned to variable last_names.

    d)check if the name 'cse' appears and the name 'it'does not appear in the list of last name assigned to variable last_names.

18. Identify the datatype is best suitable to represent the following data values

    a)Number of days in the year

    b)The circumference of a rectangle

    c)Yours father salary

    d)Distance between moon and earth

    e)Name of your best friend

    f)Whether you go for the party

----------OOO----------

UNIT-II

Syllabus: Decision Control and Looping Statements

Conditional and un-conditional branching, iterative statements, nesting of decision Control Statements and loops.

Programs: Write a python program to

1. Test whether a given number is even or odd.
2. Print out the decimal equivalents of ½, 1/3 ,1/4.....1/10,using for a loop.
3. Print a count down from the given number to zero using a while loop.
4. Find the sum of all the primes below hundred.
5. Find the factorial of a given number.

Learning Outcomes:

At the end of the unit student will be able to

- understand the conditional and unconditional Statement.
- describe conditional and unconditional Statements, Decision Control Statements .
- adapt and combine standard algorithms to solve a given problem using decision control and looping statements.
- implement programs using python control Structure

Learning Material

Control Statements:

- A *control statement* is a statement that determines the control flow of a set of instructions, i.e., it decides the sequence in which the instructions in a program are to be executed.
- Types of Control Statements —
  - **Selection/Conditional Control:** To execute only a selected set of statements.
  - **Iterative Control:** To execute a set of statements repeatedly.

➢ **Un-conditional Control:**

1. Selection /Conditional Branching Statements:

- Python language supports different types of conditional branching statements which are as follows:
    - if Statement
    - if-else Statement
    - Nested if statement
    - if-elif-else statement.

1. 1 if Statement:

➢ An if statement is a selection control statement which is based on the value of a given Boolean Expression.

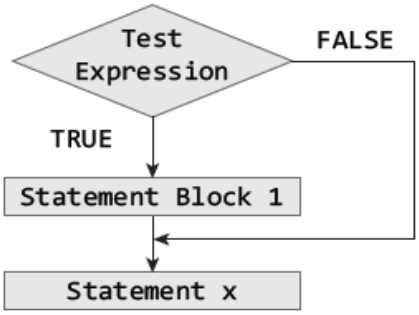
Syntax:

```
if test_expression:
    statement 1
    .....
    statement n

statement x
```

➢ if structure may include 1 or n statements enclosed within if block.

➢ First, test expression is evaluated. If the test expression is true, the statement of if block (statement 1 to n) are executed, otherwise these statements will be skipped and the execution will jump to statement x.

Flow chart:



Example:

```
x = 10        #Initialize the value of x
if(x>0):      #test the value of x
   x = x+1      #Increment the value of x if it is > 0
print(x)      #Print the value of x

OUTPUT
x = 11
```

1.2 if else Statement:

- The if .... else statement executes a group of statements when a test expression is true; otherwise, it will execute another group of statements.

Syntax:

if (test expression):

statement_block 1

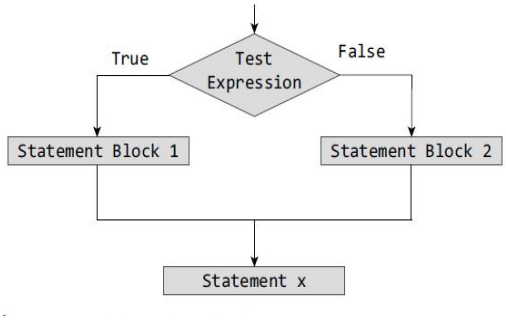
else:

statement_block 2

statement x

- If the condition is **true,** then it will execute statement block 1 and if the condition is **false** then it will execute statement block 2.

Flowchart:



Example: Write a program to determine whether a person is eligible to vote:

```
age = int(input("Enter the age : "))
if(age>=18):
   print("You are eligible to vote")
else:
        yrs = 18 - age
        print("You have to wait for another " + str(yrs) +" years to cast your vote")
```

OUTPUT

```
Enter the age : 10
You have to wait for another 8 years to cast your vote
```

1.3 Nested if Statements :

- A statement that contains other statements is called a compound statement.
- To perform more complex checks, if statements can be nested, that is, can be placed one inside the other.
- In such a case, the inner if statement is the statement part of the outer one.
- Nested if statements are used to check if more than one conditions are satisfied.
- if statements can be nested resulting in multi-way selection.

```
var = 100
if var < 200:
   print( "Expression value is less than 200")
   if var == 150:
      print ("Which is 150”)
   elif var == 100:
      print ("Which is 100")
   elif var == 50:
      print ("Which is 50")
   elif var < 50:
      print ("Expression value is less than 50")
else:
   print ("Could not find true expression")
```

print ("Good bye!")

Output:-

Expression value is less than 200

Which is 100

Good bye!

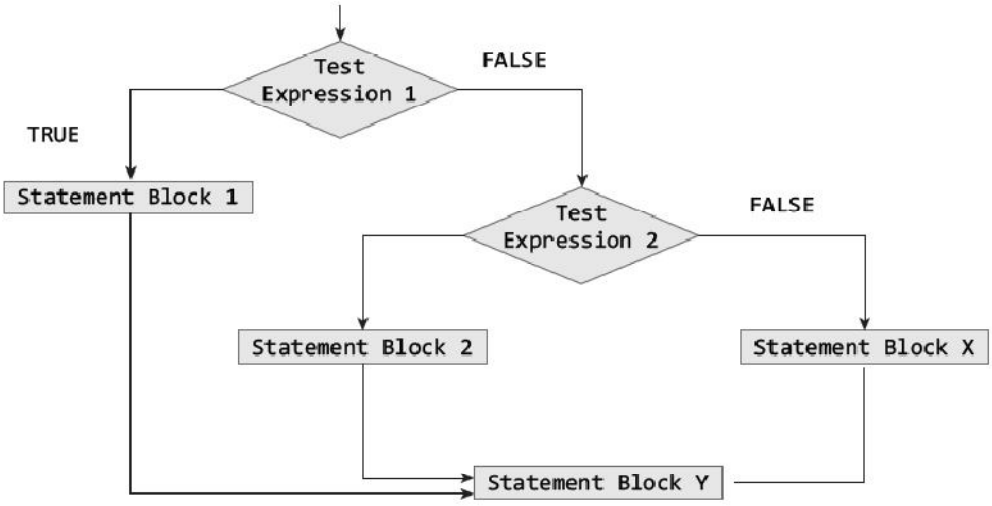
1.4 if-elif-else Statement :

- Python supports if-elif-else statements to test additional conditions apart from the initial test expression.
- The if-elif-else construct works in the same way as a usual if-else statement.
- If-elif-else construct is also known as nested-if construct.
- A series of if and elif statements have a final else block, which is executed if none of the if or elif expressions is True.

Syntax:

```
if (test expression 1):
        statement block1
elif (test expression 2):
        statement block2
. . . . . .
elif( test expression N):
        statement block N
else:
        statement block X
```

Flowchart:



Program: To test whether a number entered by the user is negative, positive, or zero

```
num = int(input("Enter any number : "))
if(num==0):
    print("The value is equal to zero")
elif(num>0):
    print("The number is positive")
else:
    print("The number is negative")
```

OUTPUT

```
Enter any number : -10
The number is negative
```

2. Looping Statements/Iterative Structure:

- ***Iterative statements*** are decision control statements that are used to repeat the execution of a list of statements.
- Python supports 2 types of iterative statements-*while* loop and *for* loop.

2.1 while Loop :

- The While loop provides a mechanism to repeat one or more statements while a particular condition is **TRUE.**

Syntax:

Statement x

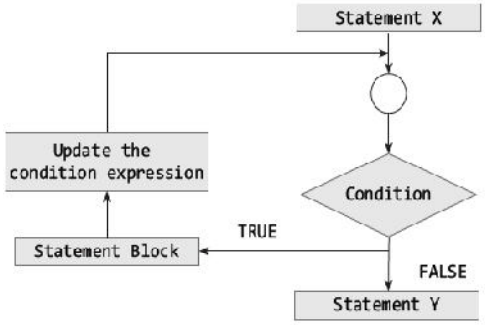
while (condition):

Statement block

Statement y

- In while loop, the condition is tested before any of the statements in the statement block is executed.
- If the condition is **TRUE,** only then the statements will be executed otherwise if the condition is **False,** the control will jump to statement y, that is the immediate statement outside the *while* loop block.

Flowchart:



Example: Program to print first 10 numbers using a while loop

```
i=0
while(i<=10):
        print(i, end=" ")
        i=i+1
```

Output: 0 1 2 3 4 5 6 7 8 9 10

2.2 for Loop:

- For loop provides a mechanism to repeat a task until a particular condition is True. It is usually known as a determinate or definite

loop because the programmer knows exactly how many times the loop will repeat.

- The for...in statement is a looping statement used in Python to iterate over a sequence of objects.

Syntax:

for **loop_control_var** in **sequence**:
    statement block

Flowchart:

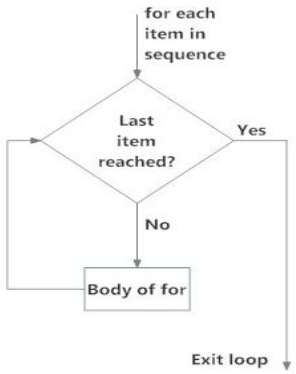


Fig: operation of for loop

2.2.1 for Loop and range() Function :

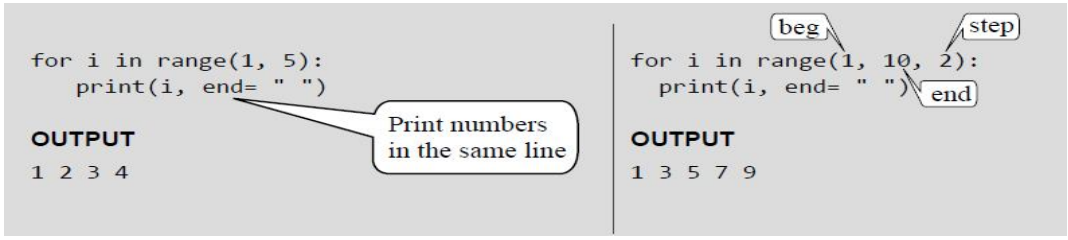
- The range( ) function is a built-in function in Python that is used to iterate over a sequence of numbers.

  Syntax:

  range(beg, end, [step])

- The range( ) produces a sequence of numbers starting with beg (inclusive) and ending with one less than the number end.
- The step argument is option (that is why it is placed in brackets). By default, every number in the range is incremented by 1 but we can specify a different increment using step. It can be both negative and positive, but not zero.

Example: Program to print first n numbers using the range() in a for



loop

If range( ) function is given a single argument, it produces an object with values from 0 to argument-1. **For example**: range(10) is equal to writing range(0, 10).

- If range( ) is called with two arguments, it produces values from the first to the second. **For example**, range(0, 10) gives 0-9.
- If range( ) has three arguments then the third argument specifies the interval of the sequence produced. In this case, the third argument must be an integer. **For example**, range(1, 20, 3) gives 1, 4, 7, 10, 13, 16, 19.

Example:

```
for i in range(10):
    print (i, end= ' ')

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

```
for i in range(1,15):
    print (i, end= ' ')

OUTPUT
1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
for i in range(1,20,3):
    print (i, end= ' ')

OUTPUT
1 4 7 10 13 16 19
```

1. Program that accepts an integer (n) and computes the value of n+nn+nnn. (Eg. If n=5, find 5+55+555).

```
n = int(input("Enter a number: "))
str_n = str(n)
sum = n
sum_str = str(n)
for i in range(1, 3):
    sum_str = sum_str + str_n
    sum = sum + int(sum_str)

print(sum)
```

2. Program that accepts a word from the user and reverse it

```
s = input("Enter a word: ")
str = ""
for i in s:
str = i + str
print("Reverse of", s, "is:", str)
```

2.3 Nested Loops :

- Python allows its users to have nested loops, that is, loops that can be placed inside other loops.
- Although this feature will work with any loop like while loop as well as for loop.
- A for loop can be used to control the number of times a particular set of statements will be executed.
- Another outer loop could be used to control the number of times that a whole loop is repeated.
- Loops should be properly indented to identify which statements are contained within each for statement.

Example:

1. Program to print the following pattern

```
**********
**********
**********
**********
**********
for i in range(5):
    print()
    for j in range(5):
        print("*",end=' ')
```

2. Program to display multiplication tables from 1 to 10

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i, '*', j, '=', i*j)
```

2.4 Condition-controlled and Counter-controlled Loops :

Attitude	Counter-controlled loop	Condition controlled loop
Number of execution	Used when number of times the loop has to be executed is known in advance.	Used when number of times the loop has to be executed is not known in advance.
Condition variable	In counter-controlled loops, we have a counter variable.	In condition-controlled loops, we use a sentinel variable.
Value and limitation of variable	The value of the counter variable and the condition for loop execution, both are strict.	The value of the counter variable and the condition for loop execution, both are strict.

Example

```
i = 0
while(i<=10):
    print(i, end = " ")
    i+=1
```

```
i = 1
while(i>0):
    print(i, end = " ")
    i+=1
    if(i==10):
        break
```

2.5 The Break Statement:

- The ***break*** statement is used to terminate the execution of the nearest enclosing loop in which it appears.
- The break statement is widely used with for loop and while loop.
- When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.

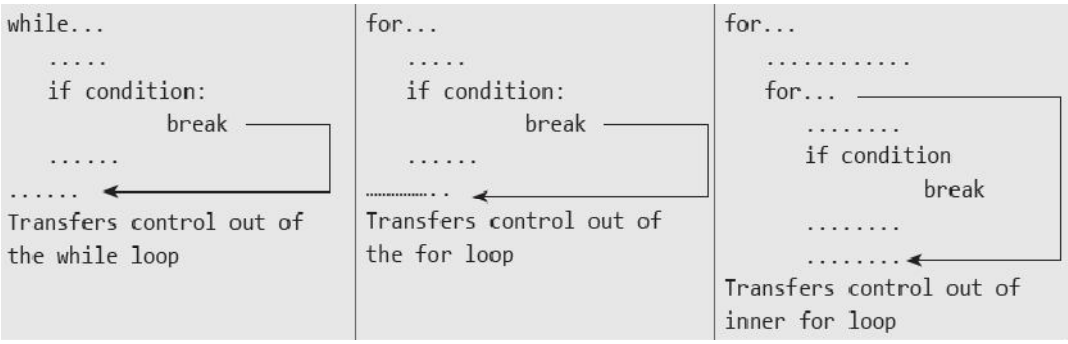
Syntax:

break

Example: Program to demonstrate the break statement

```
i = 1
while i <= 10:
        print(i, end=" ")
        if i==5:
                break
        i = i+1
print("\n Done")

OUTPUT
1 2 3 4 5
Done
```

- Above code is meant to print first 10 numbers using a ***while*** loop, but it will actually print only numbers from 0 to 4. As soon as i becomes equal to 5, the **break** statement is executed and the control jumps to the following while loop.
- Hence, the break statement is used to exit a loop from any point with in its body, by passing its normal termination expression. Below, Figure shows the transfer of control when the ***break*** statement is encountered.



2.6 The Continue Statement:

- Like the break statement, the continue statement can only appear in the body of a loop.
- When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

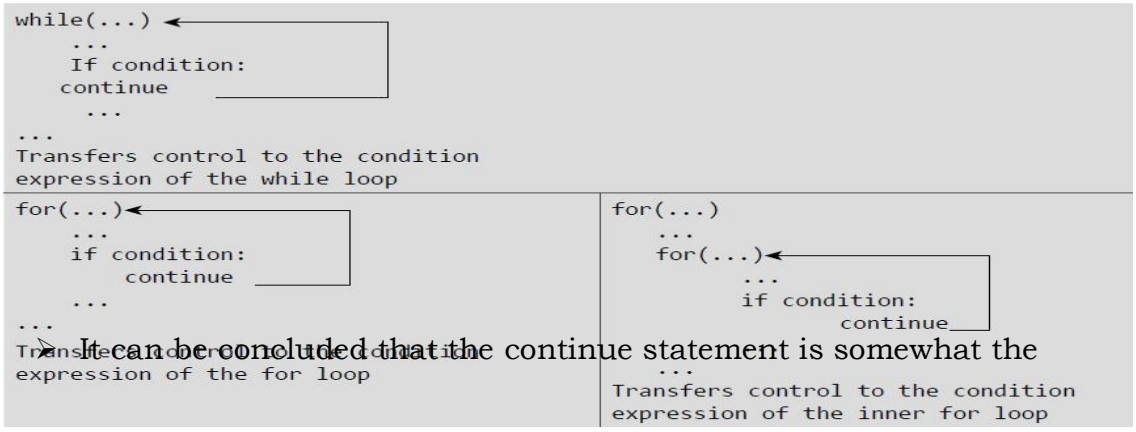
Syntax:

Continue

Example: Program to demonstrate continue statement

```
for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ")
print("\n Done")

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

- Note that the code is meant to print numbers from 0 to 10.But as soon as i becomes equal to 5, the continue statement is encountered, so rest of the statements in the loop are skipped. In the output, 5 is missing as continue caused early increment of i and skipping of statement that printed the value of i on screen.
- Below figure illustrates the use of ***continue*** statement in loops.



- It can be concluded that the continue statement is somewhat the
- opposite of the break statement. It forces the next iteration of the loop to take place, skipping any code in between itself and the test condition of the loop.
- The continue statement is usually used to restart a statement sequence when an error occurs.

2.7 The Pass Statement:

- Pass statement is used when a statement is required syntactically but no command or code has to be executed.
- It specified a null operation or simply No Operation (NOP) statement. Nothing happens when the pass statement is executed.
- The difference between a comment and pass statement is that while the interpreter ignores a comment entirely, pass is not ignored.
- Comment is not executed but pass statement is executed but nothing happens.
- Pass is a null statement.

Example:

1. Program to demonstrate **pass** statement

```
for letter in "HELLO":
        pass     #The statement is doing nothing
        print("Pass : ", letter)
print("Done")

OUTPUT
Pass :  H
Pass :  E
Pass :  L
Pass :  L
Pass :  O
Done
```

2.8 Difference between break, continue and pass

- The **break** statement terminates the execution of the nearest enclosing loop in which it appears.
- The **continue** statement skips the rest of the statements in the loop transfer the control un-conditionally to the loop-continuation portion of the nearest enclosing loop.
- The **pass** statement is a do-nothing statement in a loop. It is just added to make the loop syntactically correct. i.e, a pass statement is written as we can not have an empty body of the loop.

2.9 The Else Statement Used With Loops:

- In Python you can have the ***else*** statement associated with a loop statements.
- If the else statement is used with a ***for*** loop, the else statement is executed when the loop has completed iterating.
- But when used with the ***while*** loop, the else statement is executed when the condition becomes false.

Examples:

```
for letter in "HELLO":
        print(letter, end=" ")
else:
        print("Done")

OUTPUT
H E L L O Done
```

```
i = 1
while(i<0):
        print(i)
        i = i - 1
else:
        print(i, "is not negative so
loop did not execute")

OUTPUT
1 is not negative so loop did not execute
```

3. Programs:

3. 1 Write a python program to Test whether a given number is even or odd.

```
num = int(input("Enter a number: "))
if (num % 2==0):
        print(num, "is an even number.")
else:
        print(num, "is an odd number.")
```

Output:

Enter a number: 5

5 is an odd number.

3. 2 Write a python program to Print out the decimal equivalents of 1/1, 1/2, 1/3, 1/4.....1/10 using for loop.

```
i=1
for i in range(1,11):
        value=1.0/i
        print("1/", i, "=", value)
```

Output:

1/ 1 = 1.0

1/ 2 = 0.5

1/ 3 = 0.333333333333

1/ 4 = 0.25

1/ 5 = 0.2

1/ 6 = 0.166666666667

1/ 7 = 0.142857142857

1/ 8 = 0.125

1/ 9 = 0.111111111111

1/ 10 = 0.1

3. 3. Write a python program to Print a count down from the given number to zero using a while loop.

```
num=int(input("Enter a number: "))
print("count down from ", num, "to 0 :")
while (num >= 0):
    print(num)
    num = num - 1
```

Output:

Enter a number: 6

count down from 6 to 0:

6

5

4

3

2

1

0

3.4. Write a python program to Find the sum of all the primes below hundred.

```
sum=0
for j in range(1,100):
    for i in range(2,j):
        if (j% i) == 0:
```

```
                    break
            else:
                    sum=sum+j #where j is a prime number
print("Sum of prime numbers up to 100 is", sum)
```

Output:

Sum of prime numbers up to 100 is 1061

3. 5. Write a python program to find the factorial of a given number.

```
num=int(input("Enter a number: "))
fact=1
while (num>0):
    fact=fact*num
    num=num-1
print("Factorial of", num, "is",fact)
```

Output:

Enter a number: 6

Factorial of 6 is 720

UNIT-III

Objective: To learn use of functions and strings in developing programs in Python language.

Syllabus:

Functions and Strings

Functions-function definition, call, return statement, Types of arguments Recursive functions, modules.

Strings -Basic string operations, String formatting operator, Built-in functions.

Programs:

1. Write a function cumulative_product to compute cumulative product of a list of numbers.
2. Write function to compute gcd, lcm of two numbers. Each function shouldn't exceed one line.
3. Find the sum of the even-valued terms in the Fibonacci sequence whose values do not exceed ten thousand.
4. Write a program that accepts a string from a user and re-displays the same after removing vowels from it.
5. Write a program to calculate the length of a string.
6. Write a function to reverse a given string.

Learning Outcomes:

At the end of the unit student will be able to

- understand need for functions, variable scope and lifetime.
- identify use of modules
- use various operators in concatenating, appending, and multiplying strings.
- develop programs using built-in string methods and functions.

Learning Material

Functions

- A function is a block of organized and reusable program code that performs a single, specific, and well-defined task.
- Python enables its programmers to break up a program into functions, each of which can be written more or less independently of the others. Therefore, the code of one function is completely insulated from the codes of the other functions.

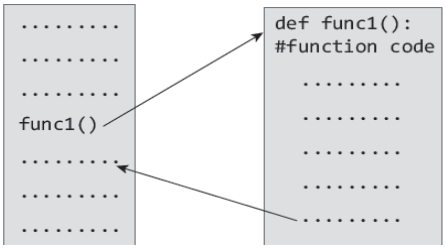


Figure 1: Calling a function

- In figure 1 which explains how a function func1() is called to perform a well-defined task. As soon as func1() is called, the program control is passed to the first statement in the function. All the statements in the function are executed and then the program control is passed to the statement following the one that called the function.

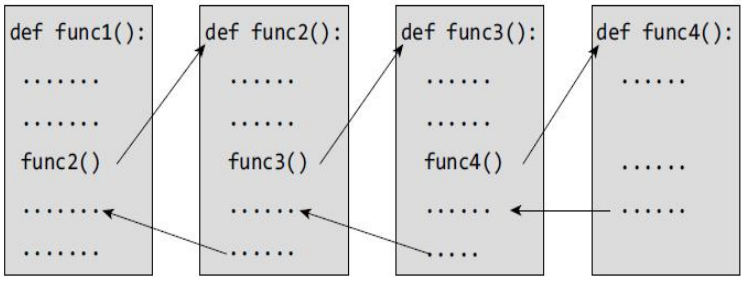


Figure 2: Function calling another function

- In figure 2 func1() calls function named func2(). Therefore, func1() is known as the *calling function* and func2() is known as the *called function.* The moment the compiler encounters a function call, instead

of executing the next statement in the calling function, the control jumps to the statements that are a part of the called function. After called function is executed, the control is returned back to the calling program.

- It is not necessary that the func1() can call only one function, it can call as many functions as it wants and as many times as it wants. For example, a function call placed within for loop or while loop may call the same function multiple times until the condition holds true.

Need for Functions:

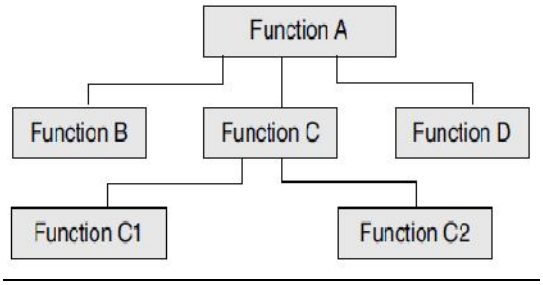


Figure 3: Top-down approach of solving a problem

- Each function to be written and tested separately.
- Understanding, coding and testing multiple separate functions are far easier than doing the same for one huge function.
- When a big program is broken into comparatively smaller functions, then different programmers working on that project can divide the workload by writing different functions.
- All the libraries in Python contain pre-defined and pre-tested functions which the programmers are free to use directly in their programs, without worrying about their code details. This speed up program development.
- Like Python libraries, programmers can also make their own functions and use them from different points in the main program or any other

program that needs its functionalities. So *code reuse* is one of the most prominent reasons to use functions.

Function Declaration and Definition:

• A function, f that uses another function g, is known as the *calling function* and g is known as the *called function*.

• The inputs that the function takes are known as *arguments/parameters.*

• When a called function returns some result back to the calling function, it is said to return that result.

• The calling function may or may not pass parameters to the called function. If the called function accepts arguments, the calling function will pass parameters, else not.

• *Function declaration* is a declaration statement that identifies a function with its name, a list of arguments that it accepts and the type of data it returns.

• *Function definition* consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

Function Definition

There are two basic types of functions

1. built-in functions eg: dir(), len(), abs() etc.,

2. user defined functions.

- Function blocks starts with the keyword def.
- The keyword is followed by the function name and parentheses (( )).
- After the parentheses a colon (:) is placed.
- Parameters or arguments that the function accept are placed within parentheses.

- The first statement of a function can be an optional statement - the *docstring*describe what the function does.
- The code block within the function is properly indented to form the block code.
- A function may have a return[expression] statement. That is, the return statement is optional.
- You can assign the function name to a variable. Doing this will allow you to call same function using the name of that variable.

```
def diff(x,y):          # function to subtract two numbers
        return x-y
a = 20
b = 10
operation = diff        # function name assigned to a variable
print(operation(a,b))  # function called using variable name
```

OUTPUT

```
10
```

Figure 4: Program that subtracts two numbers using a function.

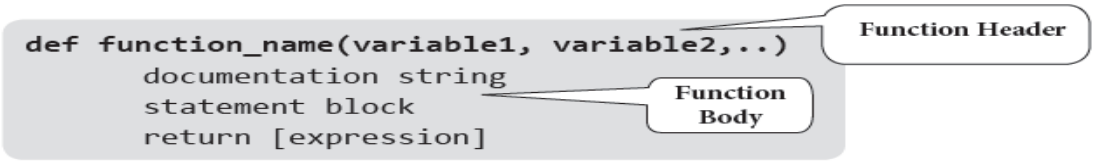


Figure 5: The syntax of a function definition.

Function Call

- Defining a function means specifying its name, parameters that are expected, and the set of instructions.
- The function call statement invokes the function. When a function is invoked the program control jumps to the called function to execute the statements that are a part of that function. Once the called function is executed, the program control passes back to the calling function.

Function Parameters

- A function can take parameters which are nothing but some values that are passed to it so that the function can manipulate them to produce the desired result. These parameters are normal variables with a small difference that the values of these variables are defined (initialized) when we call the function and are then passed to the function.
- Function name and the number and type of arguments in the function call must be same as that given in the function definition.
- If the data type of the argument passed does not matches with that expected in function then an error is generated.

```
def func():
        for i in range(4):
                print("Hello World")
func()      #function call
```

OUTPUT

```
Hello World
Hello World
Hello World
Hello World
```

Figure 6: a function that displays string repeatedly.

```
def func(i):              # function definition header accepts a variable with name i
print("Hello World", i)
j = 10
func(j)                   # Function is called using variable j
```

OUTPUT

```
Hello World 10
```

Figure 7: Program to demonstrate mismatch of name of function parameters and arguments.

Note: Names of variables in function call and header of function definition may vary.

```
def func(i):
        print("Hello World", i)
func(5+2*3)

OUTPUT
Hello World 11
```

Figure 8: Arguments may be passed in the form of expressions to the called function.

```
def total(a,b):     # function accepting parameters
    result = a+b
    print("Sum of ", a, " and ", b, " = ", result)

a = int(input("Enter the first number : "))
b = int(input("Enter the second number : "))
total(a,b) #function call with two arguments

OUTPUT
Enter the first number : 10
Enter the second number : 20
Sum of 10 and 20 = 30
```

Figure 9: Program to add two integers using functions

Variable scope and lifetime:

In python, you cannot just access any variable from any part of your program. Some of the variables may not even exist for the entire duration of the program. In which part of the program you can access a variable and in which parts of the program a variable exits depends on how the variable has been declared. Therefore, we need to understand these two things:

1. Scope of the variable: Part of the program in which a variable is accessible is called its *scope.*

2. Lifetime of the variable: Duration for which the variable exits it's called its *lifetime.*

Local and Global variables:

A variable which is defined within a function is *local* to that function. A local variable can be accessed from the point of its definition until the end of the function in which it is defined. It exists as long as the function is executing. Function parameters behave like local variables in the function. Moreover, whenever we use the assignment operator (=) inside a function, a new local variable is created.

Global variables are those variables which are defined in the main body of the program file. They are visible throughout the program file. As a good programming habit, you must try to avoid the use of global variables because they may get altered by mistake and then result in erroneous output.

```
num1 = 10    # global variable
print("Global variable num1 = ", num1)
def func(num2):           # num2 is function parameter
        print("In Function - Local Variable num2 = ",num2)
        num3 = 30         #num3 is a local variable
        print("In Function - Local Variable num3 = ",num3)
func(20)       #20 is passed as an argument to the function
print("num1 again = ", num1)        #global variable is being accessed
#Error- local variable can't be used outside the function in which it is defined
print("num3 outside function = ", num3)
```

OUTPUT

```
Global variable num1 = 10
In Function - Local Variable num2 = 20
In Function - Local Variable num3 = 30
num1 again =  10
num3 outside function =
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 12, in <module>
    print("num3 outside function = ", num3)
NameError: name 'num3' is not defined
```

Programming Tip: Variables can only be used after the point of their declaration

Figure 10: lists the differences between global and local variables.

Comparison between global and local variables

Global variables	Local variables
They are defined in the main body of the program file.	They are defined within a function and is*local*to that function.

They can be accessed throughout the program life.	They can be accessed from the point of its definition until the end of the block in which it is defined.
Global variables are accessible to all functions in the program.	They are not related in any way to other variables with the same names used outside the function.

Using the Global Statement

To define a variable defined inside a function as global, you must use the global statement. This declares the local or the inner variable of the function to have module scope.

Key points to remember:

You can have a variable with the same name as that of a global variable in the program. In such a case a new local variable of that name is created which is different from the global variable.

```
var = "Good"
def show():
        global var1
        var1 = "Morning"
        print("In Function var is - ", var)
show()
print("Outside function, var1 is - ", var1)      #accessible as it is global
variable
print("var is - ", var)
```

OUTPUT

```
In Function var is -  Good
Outside function, var1 is -  Morning
var is -  Good
```

Programming Tip: All variables have the scope of the block.

Figure 11: Program to demonstrate the use of global statement.

Resolution of names

Scope defines the visibility of a name within a block. If a local variable is defined in a block, its scope is that particular block. If it is defined in a function, then its scope is all blocks within that function.

When a variable name is used in a code block, it is resolved using the nearest enclosing scope. If no variable of that name is found, then a NameError is raised. In the code given below, str is a global string because it has been defined before calling the function.

```
def func():
        print(str)
str = "Hello World !!!"
func()

OUTPUT
Hello World !!!
```

Figure 12: Program that demonstrates using a variable defined in global namespace.

The Return Statement

The syntax of return statement is,

return [expression]

The expression is written in brackets because it is optional. If the expression is present, it is evaluated and the resultant value is returned to the calling function. However, if no expression is specified then the function will return none.

The return statement is used for two things.

• Return a value to the caller

• To end and exit a function and go back to its caller

```
def cube(x):
   return (x*x*x)
num = 10
result = cube(num)
print('Cube of ', num, ' = ', result)
```

OUTPUT

```
Cube of 10 = 1000
```

Figure 13: Program to write another function which returns an integer to the caller.

More on defining functions:

In this section we will discuss some more ways of defining a function.

1. Required arguments

2. Keyword arguments

3. Default arguments

4. Variable-length arguments

Required Arguments

In the *required arguments*, the arguments are passed to a function in correct positional order. Also, the number of arguments in the function call should exactly match with the number of arguments specified in the function definition

Example:

```
def display():
   print "Hello"
display("Hi")
```

OUTPUT

```
TypeError: display() takes
no arguments (1 given)
```

```
def display(str):
   print str
display()
```

OUTPUT

```
TypeError: display() takes
exactly 1 argument (0
given)
```

```
def display(str):
   print str
str ="Hello"
display(str)
```

OUTPUT

```
Hello
```

Keyword Arguments

When we call a function with some values, the values are assigned to the arguments based on their position. Python also allow functions to be called using keyword arguments in which the order (or position) of the arguments can be changed. The values are not assigned to arguments according to their position but based on their name (or keyword).

Keyword arguments are beneficial in two cases.

• First, if you skip arguments.

• Second, if in the function call you change the order of parameters.

Example:

```
def display(str, int_x, float_y):
   print("The string is : ",str)
   print("The integer value is : ", int_x)
   print("The floating point value is : ", float_y)
display(float_y = 56789.045, str = "Hello", int_x = 1234)
```

OUTPUT

```
The string is:  Hello
The integer value is:  1234
The floating point value is:  56789.045
```

Default Arguments

Python allows users to specify function arguments that can have default values. This means that a function can be called with fewer arguments than it is defined to have. That is, if the function accepts three parameters, but function call provides only two arguments, then the third parameter will be assigned the default (already specified) value. The default value to an argument is provided by using the assignment operator (=). Users can specify adefault value for one or more arguments.

Example:

```
def display(name, course = "BTech"):
    print("Name : " + name)
    print("Course : ", course)
display(course = "BCA", name = "Arav") # Keyword Arguments
display(name = "Reyansh")              # Default Argument for course
```

OUTPUT

```
Name : Arav
Course :  BCA
Name : Reyansh
Course :  BTech
```

Variable-length Arguments

In some situations, it is not known in advance how many arguments will be passed to a function. In such cases, Python allows programmers to make function calls with arbitrary (or any) number of arguments.

When we use arbitrary arguments or variable length arguments, then the function definition use an asterisk (*) before the parameter name. The syntax for a function using variable arguments can be given as,

```
def functionname([arg1, arg2,.... ] *var_args_tuple ):
   function statements
   return [expression]
```

Example:

```
def func(name, *fav_subjects):
    print("\n", name, " likes to read ")
    for subject in fav_subjects:
        print(subject)
func("Goransh", "Mathematics", "Android Programming")
func("Richa", "C", "Data Structures", "Design and Analysis of Algorithms")
func("Krish")
```

OUTPUT

```
Goransh  likes to read  Mathematics Android Programming
Richa likes to read  C Data Structures Design and Analysis of Algorithms
Krish  likes to read
```

Lambda Functions or Anonymous Functions

Lambda or anonymous functions are so called because they are not declared as other functions using the def keyword. Rather, they are created using the lambda keyword. Lambda functions are throw-away functions, i.e. they are just needed where they have been created and can be used anywhere a function is required. The lambda feature was added to Python due to the demand from LISP programmers.

Lambda functions contain only a single line. Its syntax can be given as,

```
lambda arguments: expression
```

Example

```
sum = lambda x, y: x + y
print("Sum = ", sum(3, 5))
```

OUTPUT

```
Sum = 8
```

Documentation Strings

Docstrings (documentation strings) serve the same purpose as that of comments, as they are designed to explain code. However, they are more specific and have a proper syntax.

```
def functionname(parameters):
   "function_docstring"
   function statements
   return [expression]
```

Example:

```
def func():
    """The program just prints a message.
    It will display Hello World !!!   """
    print("Hello World !!!")
print(func.__doc__)
```

OUTPUT

```
The program just prints a message.
    It will display Hello World !!!
```

Recursive Functions

A recursive function is defined as a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself. Every recursive solution has two major cases, which are as follows:

• *base case*, in which the problem is simple enough to be solved directly without making any further calls to the same function.

• *recursive case,* in which first the problem at hand is divided into simpler sub parts.

Recursion utilized divide and conquer technique of problem solving.

Example:

```
def fact(n):
      if(n==1 or n==0):
            return 1
      else:
            return n*fact(n-1)
n = int(input("Enter the value of n : "))
print("The factorial of",n,"is",fact(n))
```

OUTPUT

```
Enter the value of n : 6
The factorial of 6 is 720
```

Recursion vs Iteration:

Recursion is more of a top-down approach to problem solving in while the original problem is divided into smaller sub-problems.

Iteration follows a bottom-up approach that begins with what is known and then constructing the solution step-by-step.

Pros The benefits of using a recursive program are:

- Recursive solutions often tend to be shorter and simpler than non-recursive ones.
- Code is clearer and easier to use.
- Recursion uses the original formula to solve a problem.
- It follows a divide and conquer technique to solve problems.
- In some instances, recursion may be more efficient.

Cons The limitations of using a recursive program are:

- For some programmers and readers, recursion is difficult concept.
- Recursion is implemented using system stack. If the stack space on the system is limited, recursion to a deeper level will be difficult to implement.

- Aborting a recursive process in midstream is slow and sometimes nasty.
- Using a recursive function takes more memory and time to execute as compared to its non-recursive counterpart.
- It is difficult to find bugs, particularly when using global variables.

Conclusion: The advantages of recursion pays off for the extra overhead involved in terms of time and space required.

Modules

- We have seen that functions help us to reuse a particular piece of code. Module goes a step ahead. It allows you to reuse one or more functions in your programs, even in the programs in which those functions have not been defined.
- Putting simply, module is a file with a.py extension that has definitions of all functions and variables that you would like to use even in other programs. The program in which you want to use functions or variables defined in the module will simply import that particular module (or .py file).
- Modules are pre-written pieces of code that are used to perform common tasks like generating random numbers, performing mathematical operations, etc.
- The basic way to use a module is to add import module_name as the first line of your program and then writing module_name.var to access functions and values with the name var in the module.

The from...import Statement

A module may contain definition for many variables and functions. When you import a module, you can use any variable or function defined in that module. But if you want to use only selected variables or functions, then you can use the from...import statement. For example, in the aforementioned program you are using only the path

variable in the sys module, so you could have better written from sys import path.

Example:

```
from math import pi
print("PI = ", + pi)
```

OUTPUT

```
3.141592653589793
```

To import more than one item from a module, use a comma separated list. For example, to import the value of pi and sqrt() from the math module you can write,

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

Making your own Modules

Every Python program is a module, that is, every file that you save as .py extension is a module.

- Modules should be placed in the same directory as that of the program in which it is imported. It can also be stored in one of the directories listed in sys.path.

First write these lines in a file and save the file as MyModule.py

```
def display():      #function definition
   print("Hello")
   print("Name of called module is : ", __name__)

str = "Welcome to the world of Python !!!     #variable definition
```

Then, open another file (main.py) and write the lines of code given below.

```
import MyModule
print("MyModule str = ", MyModule.str)      #using variable defined in MyModule
MyModule.display()                          #using function defined in MyModule
print("Name of calling module is : ", __name__)
```

When you run this code, you will get the following output.

```
MyModule str -  Welcome to the world of Python !!!
Hello
Name of called module is :  MyModule
Name of calling module is :  __main__
```

The dir() function

dir() is a built-in function that lists the identifiers defined in a module. These identifiers may include functions, classes and variables. If no name is specified, the dir() will return the list of names defined in the current module.

Example: demonstrate the use of dir() function.

```
def print_var(x):
    print(x)
x = 10
print_var(x)
print(dir())
```

OUTPUT

```
10
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'print_var', 'x']
```

The Python Module:

- We have seen that a *Python module* is a file that contains some definitions and statements. When a Python file is executed directly, it is considered the main module of a program.
- Main modules are given the special name __main__ and provide the basis for a complete Python program.
- The main module may import any number of other modules which may in turn import other modules. But the main module of a Python program cannot be imported into other modules.

Modules and Namespaces

A namespace is a container that provides a named context for identifiers. Two identifiers with the same name in the same scope will lead to a name clash. In simple terms, Python does not allow programmers to have two different identifiers with the same name. However, in some situations we need to have same name identifiers. To cater to such situations, namespaces is the keyword. Namespaces enable programs to avoid potential

name clashes by associating each identifier with the namespace from which it originates.

Example:

```
# module1
def repeat_x(x):
    return x*2

# module2
def repeat_x(x):
    return x**2

import module1
import module2
result = repeat_x(10)     # ambiguous reference for identifier repeat_x
```

Local, Global, and Built-in Namespaces

During a program's execution, there are three main namespaces that are referenced- the built-in namespace, the global namespace, and the local namespace. The built-in namespace, as the name suggests contains names of all the built-in functions, constants, etc that are already defined in Python. The global namespace contains identifiers of the currently executing module and the local namespace has identifiers defined in the currently executing function (if any).

When the Python interpreter sees an identifier, it first searches the local namespace, then the global namespace, and finally the built-in namespace. Therefore, if two identifiers with same name are defined in more than one of these namespaces, it becomes masked.

Example: Program to demonstrate name clashes in different namespaces.

```
def max(numbers):      # global namespace
    print("USER DEFINED FUNCTION MAX.....")
    large = -1       # local namespace
    for i in numbers:
        if i>large:
            large = i
    return large
```

```
numbers = [9,-1,4,2,/]
print(max(numbers))
print("Sum of these numbers = ", sum(numbers))  #built in namespace
```

OUTPUT

```
USER DEFINED FUNCTION MAX.....
9
Sum of these numbers =  21
```

Module Private Variables

- In Python, all identifiers defined in a module are public by default. This means that all identifiers are accessible by any other module that imports it. But, if you want some variables or functions in a module to be privately used within the module, but not to be accessed from outside it, then you need to declare those identifiers as private.
- In Python identifiers whose name starts with two underscores (__) are known as private identifiers. These identifiers can be used only within the module. In no way, they can be accessed from outside the module.
- Therefore, when the module is imported using the import * form modulename, all the identifiers of a module's namespace is imported except the private ones (ones beginning with double underscores). Thus, private identifiers become inaccessible from within the importing module.

Advantages of Modules:

- Python modules provide all the benefits of modular software design. These modules provide services and functionality that can be reused in other programs.

- Even the standard library of Python contains a set of modules. It allows you to logically organize the code so that it becomes easier to understand and use.

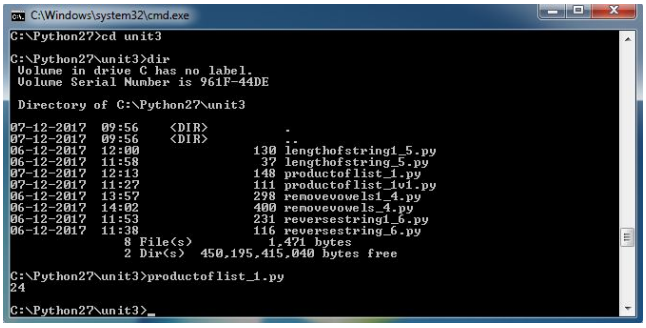
Programs:

1. Write a function cumulative product to compute cumulative product of a list of numbers.

Program:

```
def cumulative_product():
    list=[1,2,3,4]
    prod=1
    for i in list:
        prod=prod*i
    print prod
cumulative_product()
```

Output:

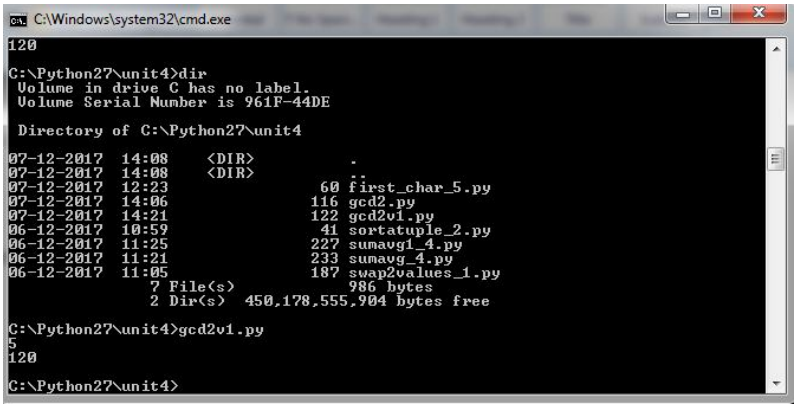


2. Write function to compute gcd, lcm of two numbers. Each function shouldn't exceed one line.

Program:

```
from fractions import gcd
print gcd(5,25)
def lcm():
    a=60
    b=40
    print (a * b) // gcd(a, b)
lcm()
```

output:

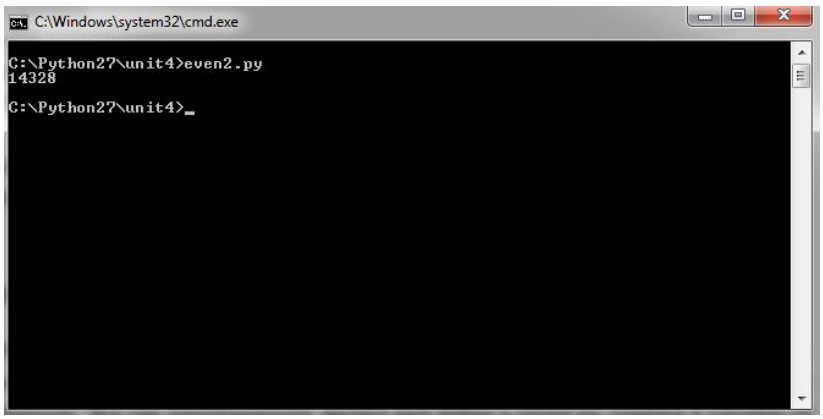


3. Find the sum of the even-valued terms in the Fibonacci sequence whose values do not exceed ten thousand.

program:

```
i=0
j=1
sum=0
while(i<10000):
        i=i+j
        j=i-j
        if(i%2==0):
           sum+=i
print sum
```

output:



Strings

- Python treats strings as contiguous series of characters delimited by single, double or even triple quotes. Python has a built-in string class named "str" that has many useful features. We can simultaneously declare and define a string by creating a variable of string type. This can be done in several ways which are as follows:
- name = "India" graduate = 'N' country = name nationality = str("Indian")

- Indexing: Individual characters in a string are accessed using the subscript ([ ]) operator. The expression in brackets is called an index. The index specifies a member of an ordered set and in this case it specifies the character we want to access from the given set of characters in the string.
- The index of the first character is 0 and that of the last character is n-1 where n is the number of characters in the string. If you try to exceed the bounds (below 0 or above n-1), then an error is raised.

- **Traversing a String:** A string can be traversed by accessing character(s) from one index to another. For example, the following program uses indexing to traverse a string from first character to the last.

Example:

```
message = "Hello!"
index = 0
for i in message:
print("message[", index, "] = ", i)
index += 1
```

OUTPUT

```
message[ 0 ] =  H
message[ 1 ] =  e
message[ 2 ] =  l
message[ 3 ] =  l
message[ 4 ] =  o
message[ 5 ] =  !
```

Concatenating, Appending and Multiplying Strings

Example: Program to concatenate two strings using + operator

```
str1 = "Hello "
str2 = "World"
str3 = str1 + str2
print("The concatenated string is : ", str3)
```

OUTPUT

```
The concatenated string is :  Hello World
```

Example: Program to repeat a string using * operator

```
str = "Hello"
print(str * 3)
```

OUTPUT

```
Hello Hello Hello
```

Example: Program to append a string using += operator

```
str = "Hello, "
name = input("\n Enter your name : ")
str += name
str += ". Welcome to Python Programming."
print(str)
```

OUTPUT

```
Enter your name : Arnav
Hello, Arnav. Welcome to Python Programming.
```

Strings are Immutable

Python strings are immutable which means that once created they cannot be changed. Whenever you try to modify an existing string variable, a new string is created.

Example:

```
str1 = "Hello"
print("Str1 is : ", str1)
print("ID of str1 is : ", id(str1))
str2 = "World"
print("Str2 is : ", str2)
```

```
print("ID of str1 is : ", id(str2))
str1 += str2
print("Str1 after concatenation is : ", str1)
print("ID of str1 is : ", id(str1))
str3 = str1
print("str3 = ", str3)
print("ID of str3 is : ", id(str3))
```

OUTPUT

```
Str1 is :  Hello
ID of str1 is :  45093344
Str2 is :  World
ID of str1 is :  45093312
Str1 after concatenation is :  HelloWorld
ID of str1 is :  43861792
str3 =  HelloWorld
ID of str3 is :  43861792
```

String Formatting Operator

- The % operator takes a format string on the left (that has %d, %s, etc) and the corresponding values in a tuple on the right. The format operator, % allow users to construct strings, replacing parts of the strings with the data stored in variables. The syntax for the string formatting operation is:
- **"<Format>" % (<Values>)**

  Example:

```
name = "Aarish"
age = 8
print("Name = %s and Age = %d" %(name, age))
print("Name = %s and Age = %d" %("Anika", 6))
```

OUTPUT

```
Name = Aarish and Age = 8
Name = Anika and Age = 6
```

Built-in String Methods and Functions

Function	Usage	Example
capitalize()	This function is used to capitalize first letter of the string.	str = "hello" print(str.capitalize()) **OUTPUT** Hello
center(width, fillchar)	Returns a string with the original string centered to a total of width columns and filled with fillchar in columns that do not have characters.	str = "hello" print(str.center(10, '*')) **OUTPUT** **hello***
find(str, beg, end)	Checks if str is present in string. If found it returns the position at which str occurs in string, otherwise returns -1. You can either set beg = 0 and end equal to the length of the message to search entire string or use any other value to search a part of it.	message = "She is my best friend" print(message. find("my",0, len (message))) **OUTPUT** 7
index(str, beg, end)	Same as find but raises an exception if str is not found.	message = "She is my best friend" print(message.index("mine", 0, len(message))) **OUTPUT** ValueError: substring not found
rfind(str, beg, end)	Same as find but starts searching from the end.	str = "Is this your bag?" print(str.rfind("is", 0, len(str))) **OUTPUT** 5
rindex(str, beg, end)	Same as rindex but start searching from the end and raises an exception if str is not found.	str = "Is this your bag?" print(str.rindex("you", 0, len(str))) **OUTPUT** 8
count(str, beg, end)	Counts number of times str occurs in a string. You can specify beg as 0 and end as the length of the message to search the entire string or use any other value to just search a part of the string.	str = "he" message = "helloworldhellohello" print(message. count (str,0, len (message))) **OUTPUT** 3
endswith (suffix, beg, end)	Checks if string ends with suffix; returns True if so and False otherwise. You can either set beg = 0 and end equal to the length of the message to search entire string or use any other value to search a part of it.	message = "She is my best friend" print(message.endswith("end", 0,len(message))) **OUTPUT** True

`isalnum()`	Returns True if string has at least 1 character and every character is either a number or an alphabet and False otherwise.	`message = "JamesBond007"`
		`print(message.isalnum())`
		OUTPUT
		`True`

`isalpha()`	Returns True if string has at least 1 character and every character is an alphabet and False otherwise.	`message = "JamesBond007"`
		`print(message.isalpha())`
		OUTPUT
		`False`

`isdigit()`	Returns True if string contains only digits and False otherwise.	`message = "007"`
		`print(message.isdigit())`
		OUTPUT
		`True`

`islower()`	Returns True if string has at least 1 character and every character is a lowercase alphabet and False otherwise.	`message = "Hello"`
		`print(message.islower())`
		OUTPUT
		`False`

`isspace()`	Returns True if string contains only whitespace characters and False otherwise.	`message = "  "`
		`print(message.isspace())`
		OUTPUT
		`True`

`isupper()`	Returns True if string has at least 1 character and every character is an upper case alphabet and False otherwise.	`message = "HELLO"`
		`print(message.isupper())`
		OUTPUT
		`True`

`len(string)`	Returns the length of the string.	`str = "Hello"`
		`print(len(str))`
		OUTPUT
		`5`

`ljust(width[, fillchar])`	Returns a string left-justified to a total of `width` columns. Columns without characters are padded with the character specified in the `fillchar` argument.	`str = "Hello"`
		`print(str.ljust(10, '*'))`
		OUTPUT
		`Hello*****`

`rjust(width[, fillchar])`	Returns a string right-justified to a total of `width` columns. Columns without characters are padded with the character specified in the `fillchar` argument.	`str = "Hello"`
		`print(str.rjust(10, '*'))`
		OUTPUT
		`*****Hello`

Programs:

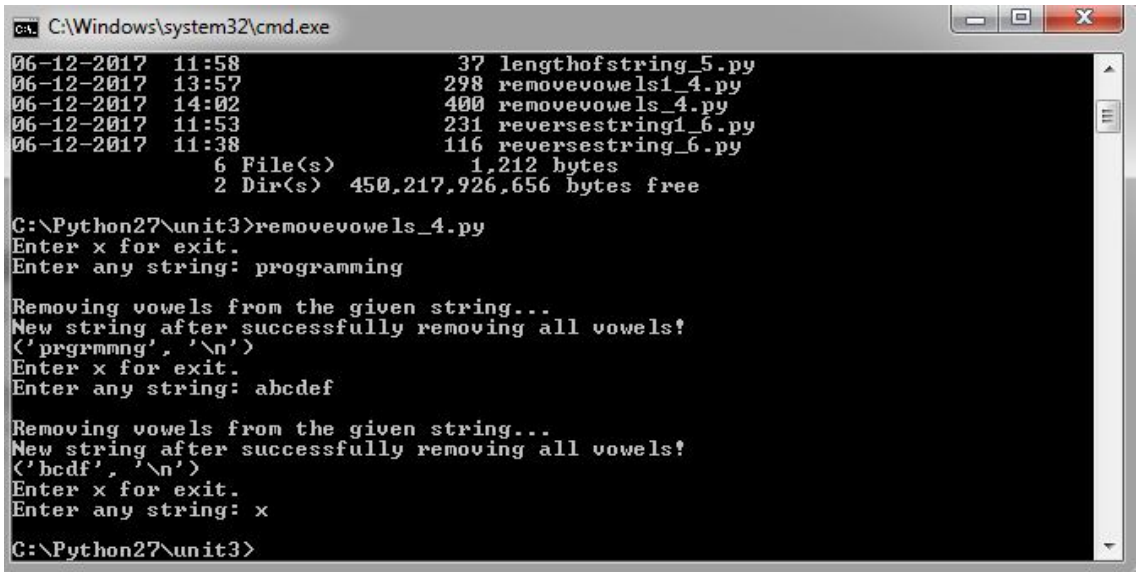
1. Write a program that accepts a string from a user and re-displays the same after removing vowels from it.

Program:

```
while True:
    print('Enter x for exit.')
    string = raw_input('Enter any string: ')
    if string == 'x':
        break
    else:
        newstr = string
```

```
print("\nRemoving vowels from the given string...")
vowels = ('a', 'e', 'i', 'o', 'u')
for x in string.lower():
        if x in vowels:
                newstr = newstr.replace(x,"")
print("New string after successfully removing all vowels!")
print(newstr,"\n")
```

output:

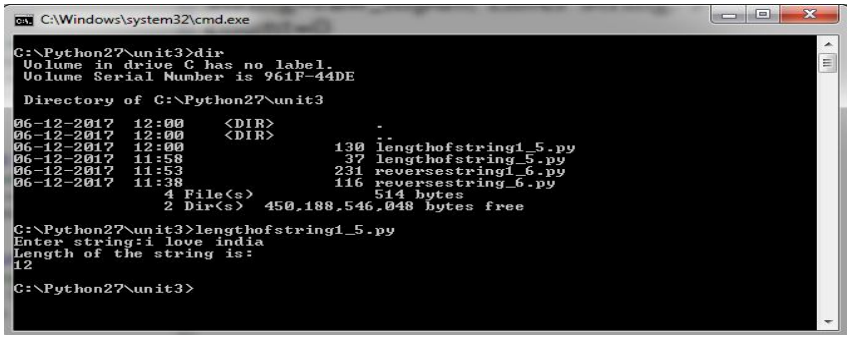


2. Write a program to calculate the length of a string.

Program:

```
string=raw_input("Enter string:")

count=0

for i in string:

    count=count+1

print("Length of the string is:")

print(count)
```

output:

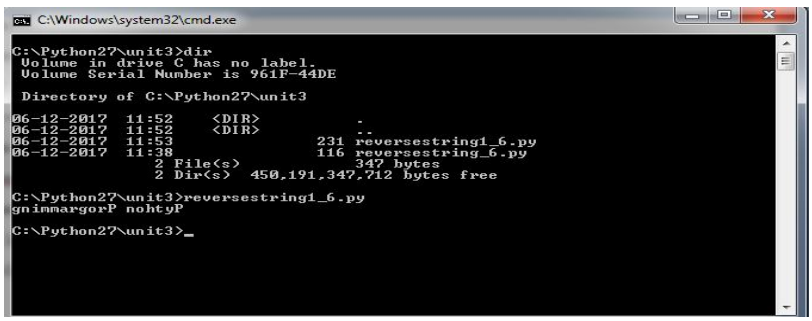


3. Write a function to reverse a given string.

Program:

```
def reverse(text):
    lst = []
    count = 1
    for i in range(0,len(text)):
        lst.append(text[len(text)-count])
        count += 1
    lst = ''.join(lst)
    return lst
print reverse('Python Programming')
```

output:



UNIT-III

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. User-defined functions are created by using the _____ keyword.

2. The _____ is used to uniquely identify the function.

3. The return statement is optional [Yes/No]

4. DRY principle makes the code [ ]

a) Reusable b) Loop forever c)Bad and repetitive d) Complex

5. _____ of a variable determines the part of the program in which it is accessible [ ]

a) Scopeb) Lifetime c) Data Type d) Value

6. Arbitrary arguments have which symbol in the function definition before the parameter name? [ ]

a) & b) # c) % d) *

7. _____dir()____ is built-in function that lists the identifiers defined in a module.

8. Arguments may be passed in the form of expressions to the called function [yes/ No]

9. In Python a string is appended to another string by using which operator? [ ]

a) + b)* c)[] d)+=

10. Which error is generated when a character in a string variable is modified? [ ]

a) IndexError b) NameError c) TypeError d)BoundError

11. The code will print how many numbers? [ ]

```
def display(x):
fori in range(x):
print(i)
return
display(10)
```

a) 0 b) 1 c) 9 d) 10

12. How many times will the print() execute in the code given below? [ ]

```
def display():
    print('a')
    print('b')
    return
print('c')
print('d')
```

a) 1 b) 2 c) 3 d)4

13. What is the output of this code? [ ]

```
import random as r
print(random.randomint(1,10))
```

a) An error occurs b) 1 c) 10 d) any random value.

14. Identify the correct way of calling a function named display() that prints Hello on the screen.

a) print(display) b) displayHello [ ]

c) result = display() d) displayHello()

15. Find the error in following Python code. [ ]

```
Def func():
```

Print(“Hello world”)

a) Hello world b) “Hello world” c) no function call d) none of the above

16. Find the output of the following Python code. [ ]

```
deffunc(var):
var+=1
var *=2
print(var)
func(9)
print(var)
```

a) 20 20 b) 20 c) 9 d) 20'var' is not defined

17. Find the output in following Python code. [ ]

```
Def func():
    global x
    print(“x=”,x)
    x=100
    print(‘x is now = ‘,x)
x=10
func()
print(‘x =’,x)
```

a) 100 10 100 b) 100 10 10

c) NameError: name 'x' is not defined d) Error 100 10

18. Find the output in following Python code. [ ]

```
def display(name, deptt, sal):
    print(“Name:”, name)
    print(“Department: “, deptt)
```

```
        print("Salary: ", sal)
    display (sal = 100000, name="Tavisha", deptt = "sales")
    display (deptt = "HR", name="Dev", sal = 50000)
```

a) Name: Tavisha
   Department: sales
   Salary: 100000
   Name: Dev
   Department: HR
   Salary: 50000

b) Name: Tavisha
   Department: sales
   Salary: 100000
   Department: HR
   Name: Dev
   Salary: 50000

c) Name: Tavisha
   Department: sales
   Salary: 100000
   Sequence Error:

d) Indentation Error:

19. "Cool" become "COOL", which two functions must have been applied?

a) strip() and upper() b) strip() and lower() [ ]

c) strip() and capitalize() d) lstrip() and rstrip()

20. Find the error in following Python code. [ ]

```
str = "Hello world"
str[6] = 'w'
print(str)
```

a) Hello world

b) 'str' object does not support item assignment

c) in line 2 use double quotes

d) Hello wworld

SECTION-B

Subjective Questions

1. Define function and give its advantages.
2. Differentiate between local and global variables.
3. What are modules? How do you use them in your programs?
4. Write short notes on

a) Keyword arguments

b) Default arguments

5. What are docstrings?
6. Write short note on format operator.
7. With the help of an example, explain how we can create string variables in Python.
8. What are user-defined functions? Explain with the help of example.
9. Briefly describe String formatting operator with an example.
10. List out Advantages and disadvantages of Recursion.
11. Write a python program to find the factorial of a given number using recursion.
12. Write any 5 Built-in string methods and functions usage and example.

Unit-4

Tuples and Lists

Objective:

To familiarize concepts of tuples and lists in Python programming.

Outcome:

Apply lists and tuples in developing Python programs.

Syllabus

Tuples – creating, accessing values, updating, deleting elements in a tuple, Basic Tuple operations.
Lists – accessing, updating values in Lists, Basic List operations, mutability of lists.

Programs: Write a python program to
1. swap two values using Tuple assignments.
2. sort a Tuple of values.
3. scans an email address and forms a tuple of user name and domain name.
4. print sum and average of the elements present in the list.
5. forms a list of first character of every word present in another list.

Learning Outcomes:

At the end of the unit student will be able to

1. demonstrate creating, accessing elements in a tuples and lists.
2. describe updating and deleting elements in a tuple.
3. apply various operations on lists and tuple.
4. develop programs using lists and tuples.

Learning Material

Tuple Definition:

1. A tuple is a sequence of immutable objects. That is, you can change the value of one or more items in a list; you cannot change the values in a tuple.

2. Tuples use parenthesis to define its elements. Whereas lists use square brackets.

Creating a Tuple:

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

Syntax: Tup1=(val1,val2,….)

Where val (or values) can be an integer, a floating number, a character, or a string.

Examples:

1) Tup1=( ) #creates an empty tuple.
   print(Tup1)

   output:
   note: no ouput will be displayed.

2) Tup1=(5) #creates a tuple with single element
   print(Tup1)

   Output:

   5

3) Tup1=(1,2,3,4,5) #creates a tuple of integers
   print (Tup1)
   Tup2=('a','b','c','d') #creates a tuple of characters
   print(Tup2)
   Tup3=("abc","def","ghi") #creates a tuple of strings
   print(Tup3)

```
Tup4=(1.2,2.3,3.4,4.5,5.6)    #creates a tuple of floating point numbers
print(Tup4)
Tup5=(1,"abc",2.3,'d')  #creates a tuple of mixed values
print(Tup5)
```

Output:

```
1,2,3,4,5
'a','b','c','d'
'abc','def','ghi'
1.2,2.3,3.4,4.5,5.6
1,'abc',2.3,'d'
```

4) A Tuple with parenthesis

```
print('a',"bcd",2,4.6)
```

Output:

```
A bcd2 4.6
```

5) Default Tuple without parenthesis

```
a,b=10,20
print(a,b)
```

Output:

```
10 20
```

Accessing values of tuples:

- Like strings and lists tuples indices also starts with 0.
- The operations performed are slice, concatenate etc.,
- To access values in tuple, slice operation is used along with the index.

Example :

1) Tup1=(1,2,3,4,5,6,7,8,9,10)

```
print("Tup[3:6]=",Tup1[3:6])

print("Tup[:8]=",Tup1[:4])
```

print("Tup[4:]=",Tup1[4:])

print("Tup[:]=",Tup1[:])

Output:

Tup[3:6]=(4,5,6)

Tup[:8]=(1,2,3,4)

Tup[4:]=5,6,7,8,9,10)

Tup[:]=(1,2,3,4,5,6,7,8,9,10)

The tuple values can be accessed using square brackets:

2) Tuple =(1,2,3,4,5.5,'str')

Input:

1.print tuple

2.print tuple[5]

3.print tuple[1:5]

Output:

1.1,2,3,4,5.5,'str'

2.'str'

3.2,3,4,5.5

Updating tuples:

As we all know tuples are immutable objects so we cannot update the values but we can just extract the values from a tuple to form another tuple.

Example:

1) Tup1=(1,2,3,4,5)
   Tup2=(6,7,8,9,10)
   Tup3=Tup1+Tup2

print(Tup3)

Output:
(1,2,3,4,5,6,7,8,9,10)

2) Tup1=(1,2,3,4,5)

Tup2=('sree','vidya','ram')

Tup3=Tup1+Tup2

print Tup3

Output:

(1,2,3,4,5,'sree','vidya','ram')

Deleting elements of a tuple:

1.Deleting a single element in a tuple is not possible as we know tuple is a immutable object.

Hence there is another option to delete a single element of a tuple i.e..,you can create a new tuple that has all elements in your tuple except the ones you don't want.

Example:

1) Tup1=(1,2,3,4,5)

del Tup1[3]

print Tup1

Output:

```
Traceback (most recent call last):
File "test.py", line 9, in <module>
```

del Tup1[3]

Type error: 'tuple' object doesn't support item deletion

2) however, you can always delete the entire tuple by using del statement.

Tup1=(1,2,3,4,5)

```
del Tup1

print Tup1
```

Output:

```
Traceback (most recent call last):
File "test.py", line 9, in <module>
print Tup1;
NameError: name 'Tup1' is not defined
```

Key Note: Note that exception is raised because you are now trying to print a tuple that has already been deleted.

Basic tuple operations:

Like strings and lists,you can also perform operations like concatenation, repetition,etc. on tuples. The only difference is that a new tuple should be created when a change is required in an existing tuple.

Operation	Expression	Output
Length	len((1,2,3,4,5,6))	6
Concatenation	(1,2,3)+(4,5,6)	(1,2,3,4,5,6)
Repetition	('Good..')*3	'Good ..Good..Good'
Membership	5 in (1,2,3,4,5,6,7,8,9)	True
Iteration	for i in (1,2,3,4,5,6,7,8,9,10): print(i,end=' ')	1,2,3,4,5,6,7,8,910
Comparision(Use >,<,==)	Tup1=(1,2,3,4,5) Tup2=(1,2,3,4,5) print(Tup1>Tup2)	False
Maximum	max(1,0,3,8,2,9)	9

Minimum	min(1,0,3,8,2,9)	0
Convert to tuple(converts a sequence into a tuple)	tuple("Hello") tuple([1,2,3,4,5])	('H','e','l','l','o') (1,2,3,4,5)
Sorting(The sorted( ) function takes elements in a tuple and returns a new sorted list (does not sort the tuple itself)).	t=(4,67,9) sorted(t)	[4, 9, 67]

1) Length of the tuple:

Ex:

Input:

Tup1= (1,2,3,4,5)

print len (Tup1)

Output:

5

2) Concatenation:

Ex:

Input:

Tup1=(1,2,3,4)

Tup2=(5,6,7)

print tup1+tup2

Output:

(1,2,3,4,5,6,7)

3) Repetition:

Ex:

Input:

Tuple1=(‘my’)

print tuple1*3

Output:

(‘my’,’my’,’my’)

4) Membership:

Ex:

Input:

Tuple1=(1,2,3,4,6,7)

5) Iteration:

Ex:

Input:

For i in (1,2,3,4,5,6,7,8,9,10):

print (i,end=’ ‘)

Output:

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

6) Comparison:

Ex:

Input:

Tup1 = (1,2,3,4,5)

Tup2 =(6,7,8,9,10)

print(Tup1<tup2)

Output:

True

7) Maximum:

Ex:

Input:

Max(1,2,6,5,4)

Output:

6

8) Minimum:

Ex:

Input:

Min(1,2,3,4,5)

Output:

1

9) Convert to tuple:

Ex:

Input:

Tuple("vidya")

Output:

('v','i','d','y','a')

LISTS

The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth. There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership.

Creating a List:

Creating a list is as simple as putting different comma-separated values between square brackets. Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Syntax:

List_variable = [val1,val2,…]

Example:

1) list_A =[1,2,3,4]
   print(list_A)

Output

[1,2,3,4]

2) list_C=[‘Good”,”Going”]
   print(list_C)

Output

[‘Good’,’Going’]

Accessing values in lists:

- Similar to strings, lists can be sliced and concatenated.
- To access values in lists, square brackets are used to slice along with index or indices to get value stored at that index.
- **syntax**

  s=list[start:stop:step]

For Example:

Seq=List[::2] # get every other element, starting with index 0.

Seq=List[1::2] # get every other element, starting with index 1.

Example 1:

```
num_list=[1,2,3,4,5,6,7,8,9,10]
print("num_list is:",num_list)
print("first elemnent in the list is",num_list[0])
print("num_list[2:5]=",num_list[2:5])
print("num_list[::2]=",num_list[::2])
print("num_list[1::3]=",num_list[1::3])
```

Output:

```
num_list is: [1,2,3,4,5,6,7,8,9,10]
first elemnent in the list is 1
num_list[2:5]= [3,4,5]
num_list[::2]= [1,3,5,7,9]
num_list[1::3]= [2,5,8]
```

Updating values in the lists:

- once created, one or more elements of a list can be easily updated by giving the slice on the left-hand side of the assignment operator.
- You can also append new values in the list and remove existing values from the list using the append( ) method and del statement respectively.

Example:

1)
```
num_list= [1,2,3,4,5,6,7,8,9,10]
print("list is:",num_list)
num_list[5]=100
print("List after updation is:",num_list)
```

```
num_list.append(200)
print("List after appending a value is: ",num_list)
del num_list[3]
print("List after deleting a value is:",num_list)
```

Output:

```
list is: [1,2,3,4,5,6,7,8,9,10]
List after updation is: [1,2,3,4,5,100,7,8,9,10]
List after appending a value is: [1,2,3,4,5,100,7,8,9,10,200]
List after deleting a value is: [1,2,3,5,100,7,8,9,10,200]
```

Basic list operations:

Operation	Description	Example	Output
len	Returns length of list	len([1,2,3,4,5,6,7,8,9,10])	10
concatenation	Joins two lists	[1,2,3,4,5]+[6,7,8,9,10]	[1,2,3,4,5,6,7,8,9,10]
repetition	Repeats elements in the lists	"Hello","World"*2	['Hello','World','Hello','World']
in	Checks if the value is present in the list	'a' in['a','e','i','o','u']	True
not in	Checks if the value is not present in the list	3 not in [0,2,4,6,8]	True
max	Returns maximum value in the list	num_list=[6,3,7,0,1,2,4,9] print(max(num_list))	9
min	Returns minimum value in the list	num_list=[6,3,7,0,1,2,4,9] print(min(num_list))	0
sum	Adds the values in the list that has numbers	num_list=[1,2,3,4,5,6,7,8,9,10] print("SUM=",sum(num_li	SUM=55

		st))	
all	Returns True if all elements of the list are true(or if the list is empty)	num_list=[0,1,2,3] print(all(num_list))	False
any	Returns True if any element of the list is true. if the list is empty return false	num_list=[6,3,7,0,1,2,4,9] print(any(num_list))	True
list	Converts iterable(tuple,string,set,dictionary)	list1=list("HELLO") print(list1)	['H','E','L','L','O']
sorted	Returns a new sorted list. The original list not sorted	list1=[3,4,1,2,7,8] list2=sorted(list1) print(list2)	[1,2,3,4,7,8]

Mutability of lists:

- Unlike strings, lists are **mutable**.
- This means we can change an item in a list by accessing it directly as part of the assignment statement.
- Using the indexing operator (square brackets) on the left side of an assignment, we can update one of the list items.

Example:

```
fruit = ["banana", "apple", "cherry"]

print(fruit)

fruit[0] = "pear"

fruit[-1] = "orange"

print(fruit)
```

Output:

```
['banana', 'apple', 'cherry']
['pear', 'apple', 'orange']
```

Functional Programming:

Functional Programming decomposes a problem into a set of functions. The map(),filter(), and reduce() functions.

1) map( ) Function:
   The map() function applies a particular function to every element of a list.

Syntax:

map(function,sequence)

After applying the specified function in the sequence, the map( ) function returns the modified list.

Ex: Program that adds 2 to every value in the list.

```
def add_2(x):
    x+=2
    return x
num_list=[1,2,3,4,5,6,7]
print("original list is:",num_list)
new_list=list(map(add_2,num_list))
print("modified list is:",new_list)
```

output:

original list is: [1,2,3,4,5,6,7]

modified list is:[3,4,5,6,7,8,9]

2) reduce ( ):

The reduce( ) function with syntax as given below returns a single value generated by calling the function on the first two items of the sequence, then on the result and the next item and so on.

Syntax: reduce(function,sequence)

Ex: Program to calculate the sum of values in a list using the reduce( ) function.

```
import functools          # functools is a module that contains the function reduce( )
def add(x,y):
    return x+y
num_list=[1,2,3,4,5]
print("sum of values in list=")
print(functools.reduce(add,num_list))
```

Output:

sum of values in list= 15

3) filter( ) function:

It constructs a list from those elements of the list for which a function returns True.

Syntax:

filter(function,sequence)

As per the syntax filter( ) function returns a sequence that contains items from the sequence for which the function is True. If sequence is a string, Unicode, or a tuple, then the result will be the same type;

Ex: Program to create a list of numbers divisible by 2 or 4 using list comprehension.

```
def check(x):
    if(x%2==0 or x%4 ==0):
        return 1
```

```
#call check( ) for every value between 2 to 21
evens=list(filter(check,range(2,22))
print(evens)
```

Output:

[2,4,6,8,10,12,14,16,18,20]

programs:

1. Write a program to swap two values using Tuple assignments.

Program:

```
(val1,val2,val3)=(1,2,3)
(tup1,tup2,tup3)=(4,5,6)
(a,b,c)=(val1,val2,val3)
(val1,val2,val3)=(tup1,tup2,tup3)
(tup1,tup2,tup3)=(a,b,c)
print (val1,val2,val3)
print (tup1,tup2,tup3)
```

Output:

(4, 5, 6)

(1, 2, 3)

2. Write a program to sort a Tuple of values.

Program:

```
tup=[5,1,40,8,6,2,1]
print(sorted(tup))
```

Output:

[1, 1, 2, 5, 6, 8, 40]

3. Write program that scans an email address and forms a tuple of user name and domain name.

Program:

```
addr =input('Enter email address:')
(uname, domain) = addr.split('@')
print('Username:',uname)
print('domain name:',domain)
```

Output:

```
Enter email address:gec@gmail.com
Username: gec
domain name: gmail.com
```

4. Write a program to print sum and average of the elements present in the list.

Program:

```
lst = [ ]

num = int(input('How many numbers: '))

for n in range(num):

    numbers = int(input('Enter number '))

    lst.append(numbers)

print("Sum of elements in given list is :", sum(lst))

avg=sum(lst)/num

print(avg)
```

Output:

```
How many numbers: 7

Enter number 1

Enter number 2

Enter number 3
```

Enter number 4

Enter number 5

Enter number 6

Enter number 7

Sum of elements in given list is : 28

4

5. Write a program that forms a list of first character of every word present in another list.

Program:

```
b= [ ]
l= ["gudlavalleru","engineering","college"]
for item in l:
    b.append(item[0])
print(b)
```

Output: [‘g’,’e’,’c’]

Assignment-Cum-Tutorial Questions

A) *Objective Questions*

1. If list=[1,2,3,4,5] then the list[5] will result in---------- [ ]
   (a) 4 (b) 3 (c) 2 (d) Index Error
2. If List=[1,2,3,4,5] and rewrite List[3]=List[1], then what will be the List[3] [ ]
   (a) 1 (b) 3 (c) 2 (d) 4
3. In lists index value starts from Zero. [True/False] [ ]
4. print len((1,2,3,4,5,6)) is [ ]
   (a) 5 (b) 6 (c) 21 (d) 7
5. Tuple is immutable and list is mutable. [True/False]
6. It is possible to add, edit, and delete elements from a list. [True/False]
7. list=['a','b','c','d','e'] output for print list[2:5] = ['c', 'd', 'e'].
8. tuple=('abcd',23,2.4,1)
   print tuple[:3] what is the output? [ ]
   a) ('abcd',23,2.4) b) (1) c) (23,2.4,1) d) ('abcd',23,2.4,1)
9. what is the output of print tuple[2:] if tuple=('abcd',786,2.23,1,2) [ ]
   a) (cd,786,2.23,1,2) b) (2.23,1,2) c) (786,2.23,1,2) d) (1,2)
10. Suppose t = (1, 2, 4, 3), which of the following is incorrect? [ ]
    a) print(t[3]) b) t[3] = 45
    c) print(max(t)) d) print(len(t))

11. What is the output of the program: [ ]

```
for fruit in ['apple','banana','mango']:
print("I like",fruit)
```

a) [apple',’banana',’mango’] b) I like ‘apple’ c) I like apple d) I like
I like ‘banana’ I like banana I like
I like ‘mango’ I like mango I like

12. What is the output of the program [ ]

```
my_list = ['p','r','o','b','l','e','m']
print('p' in my_list)
print('a' in my_list)
print('c' not in my_list)
```

a) True
False
True

b)True
True
False

c)False
True
False

d) False
True
True

13. What is the output of the program [ ]

```
my_tuple = ('p','e','r','m','i','t')
print(my_tuple[-1])
print(my_tuple[-6])
```

a) t
p

b) t
t

c) p
p

d) t
NULL

14. What is the output of the program [ ]

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
print(my_tuple[1:4])
print(my_tuple[:-7])
print(my_tuple[7:])
print(my_tuple[:])
```

a) ('r', 'o', 'g')
('p', 'r')
('i', 'z')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

b) (‘p’,’r’,’o’)
(‘r’,’p’)
(‘z’,’’i’)
('p','r','o','g','r','a','m','i','z')

c)('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
('r', 'o', 'g')

d) ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
('i', 'z')

('p', 'r') ('p', 'r')

('i', 'z') ('r', 'o', 'g')

15. What is the output of the program [ ]

```
print((1, 2, 3) + (4, 5, 6))
print(("Repeat",) * 3)
```

a) (1, 2, 3, 4, 5, 6) b) (‘Repeat’,Repeat’,’Repeat’)

('Repeat', 'Repeat', 'Repeat') (1,2,3,4,5,6)

c)(1,2,3)+(4,5,6) d) (“Repeat”,)*3

“Repeat” (1,2,3)+(4,5,6)

16. What is the output of the program [ ]

```
my_tuple = ('a','p','p','l','e',)
print(my_tuple.count('p'))
print(my_tuple.index('l'))
```

a) 2 b) 2 c) 3 d) 3

3 2 2 3

17. What is the output of the program [ ]

```
pow2 = [2 ** x for x in range(10)]
print(pow2)
```

a) [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

b) [512,256,128,64,32,16,8,4,2,1]

c) [1,2,3,4,5,6,7,8,9,10]

d) [1,3,5,7,9]

18. What is the output of the program [ ]

```
my_list = ['p','r','o','b','e']
print(my_list[-1])
print(my_list[-5])
```

a) e b) e c) p d) e

NULL p e e

19. What is the output of the program [ ]

```
odd=[1,3,5]
Print(odd+[9,7,5])
Print(["re"]*3)
```

a) [1,3,5,9,7,5] ["re","re","re"]

b) [1,3,5,9,7] ["re","re","re"]

c) (odd+[9,7,5]) (["re"]*3)

d) [1,3,5] (["re"]*3)

20. What is the output of the program [ ]

```
odd = [1, 9]
odd.insert(1,3)
print(odd)
odd[2:2] = [5, 7]
print(odd)
```

a) [1, 3, 9] [1, 3, 5, 7, 9]

b) [1,3,5,7,9] [1,3,5,7,9]

c) [1,9,3] [1,9,3,5,7]

d) [1,9,1,3] [1,9,1,3,5,7]

B) Subjective Questions

1. What is negative index in list and tuple? [April-2018]
2. What is tuple? What are the different operations performed on tuple? Explain with an example? [NOV-2018]
3. Illustrate the ways of creating the tuple and the tuple assignment with suitable programs. [April-2018]
4. Summarize basic List operations with examples. [NOV-2018]
5. How can you access and update values in a list?
6. Explain mutability of lists?
7. Write a set of commands that covers at least five tuple functions and five list functions?
8. Write a program to find sum of all even numbers in a list?

9. Write a program that reverses a list using a loop?
10. Write a program to find whether a particular element is present in the list?
11. Write a program that finds the sum of all the numbers in a list using a while loop?
12. Write a program that forms a List of first character of every word present in another List. [NOV-2018]
13. Write a program that creates a list[‘a’,’b’,’c’], then create a tuple from that list.
14. Write a program that converts a list of characters into their corresponding ASCII values using map( ) function.
15. Write a program using filter function to list cubes of numbers from 1-10.
16. Write a code snippet in Python to Access Elements of a Tuple. [NOV-2018]
17. Write code snippets in Python for modifying and deleting Elements of Tuple. [NOV-2018]
18. “Tuples are immutable”. Explain with examples. [April-2018]

Unit-V

Dictionaries

Objective:

To familiarize concepts of dictionaries in Python programming.

Outcome:

Apply dictionaries in developing Python programs.

Syllabus:

Dictionaries-creating a Dictionary, adding an item, deleting items, sorting items, looping over a dictionary, basic dictionary operations, built in functions.

Lab Programs:

1. Write a program to count number of characters in the string and store them in dictionary.
2. Write a program to sort keys of Dictionary.
3. Write a program that prints maximum and minimum value in a dictionary.

Learning Outcomes:

At the end of the unit student will be able to

1. Understand creating, accessing elements in dictionaries.
2. Describe updating and deleting elements in dictionaries.
3. Discuss various dictionaries operations.
4. Implement programs using dictionaries.

Learning Material:

- **Dictionary:**
  - It is a data structure in which we store values as a pair of key and value.
  - Each key is separated from its value by a colon (:), and consecutive items are separated by commas.
  - The entire items in a dictionary are enclosed in curly brackets ({}).

Syntax:

dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3}

If there are many keys and values in dictionaries, then we can also write just one key-value pair on a line to make the code easier to read and understand. This is shown below.

dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3 , ….}

- Keys in the dictionary must be unique and be of any immutable data type (like Strings, numbers, or tuples), there is no strict requirement for uniqueness and type of values.
- Values of a key can be of any type.
- Dictionaries are not Sequences, rather they are mappings.
- **Mappings** are collections of objects that are store objects by key instead of by relative position.

Creating a Dictionary:

- The Syntax to create an empty dictionary can be given as:

  Dictionary_variable= { }

- The Syntax to create a dictionary with key-value pair is:

  Dictionary_variable= {key1:val1, key2:val2……}

- **A** dictionary can be also created by specifying key-value pairs separated by a colon in curly brackets as shown below.
- Note that one key value pair is separated from the other using a comma.

Example:

```
d= {'roll_no':'18/001','Name:':'Arav','Course':'B.tech'}
print(d)
```

Output: {'Name:': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}

Accessing Values:

- In Dictionary, through key accessing values,
- **Example**:

```
d={'Name': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}
print('d[Name]:',d['Name'])
print('d[course]:',d['Course'])
print('d[roll_no]:',d['roll_no'])
```

output:

```
d[Name]: Arav
d[course]: B.tech
d[roll_no]: 18/001
```

➢ Adding and Modifying an Item in a Dictionary:

- To add a new entry or a key-value pair in a dictionary, just specify the key-value pair as you had done for the existing pairs.

Syntax: ***dictionary_ variable[key]= val***

Example:

1. Program to add a new item in the dictionary

```
d={'Name': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}
d['marks']=99                    #new entry
print('d[Name]:',d['Name'])
print('d[course]:',d['Course'])
print('d[roll_no]:',d['roll_no'])
print('d[marks]:',d['marks'])
```

Output:

```
d[Name]: Arav
d[course]: B.tech
d[roll_no]: 18/001
d[marks]: 99
```

Modifying an Entry:

- To modify an entry, just overwrite the existing value as shown in the following example:

1. **program to modify an item in the dictionary**

```
d={'Name': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}
d['marks']=99                    #new entry
print('d[Name]:',d['Name'])
print('d[course]:',d['Course'])
print('d[roll_no]:',d['roll_no'])
print('d[marks]:',d['marks'])
d[‘Course’]=’BCA’                #Updated entry
print('d[course]:',d['Course'])
```

Output:

```
d[Name]: Arav
d[course]: B.tech
d[roll_no]: 18/001
d[marks]: 99
d[course]: BCA
```

➢ Deleting Items :

- You can delete one or more items using the del keyword.
- To delete or remove all the items in just one statement, use the **clear ()** function.

- Finally, to remove an entire dictionary from the memory, we can gain use the **del** statement as **del Dict_name**.
- The syntax to use the del statement can be given as,

  del dictionary_variable[key]

Example:

1. Program to demonstrate the use of **del** statement and **clear()** function

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Name is : ", Dict.pop('Name'))    # returns Name)
print("Dictionary after popping Name is : ", Dict)
print("Marks is :", Dict.pop('Marks', -1))   # returns default value
print("Dictionary after popping Marks is : ", Dict)
print("Randomly popping any item : ",Dict.popitem())
print("Dictionary after random popping is : ", Dict)
print("Aggregate is :", Dict.pop('Aggr'))    # generates error
print("Dictionary after popping Aggregate is : ", Dict)
```

OUTPUT

```
Name is :  Arav
Dictionary after popping Name is :  {'Course': 'BTech', 'Roll_No': '16/001'}
Marks is : -1
Dictionary after popping Marks is :  {'Course': 'BTech', 'Roll_No': '16/001'}
Randomly popping any item :  ('Course', 'BTech')
Dictionary after random popping is :  {'Roll_No': '16/001'}
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 8, in <module>
    print("Aggregate is :", Dict.pop('Aggr'))
KeyError: 'Aggr'
```

- Keys must have unique values.
- Not even a single key can be duplicated in a dictionary. If you try to add a duplicate key, then the last assignment is retained.
- In a dictionary, keys should be strictly of a type that is immutable. This means that a key can be of strings, numbers, or tuple type but it cannot be a list which is mutable.
- In case you try to make your key of mutable type, then a **Type error** will be granted.
- Tuples can be used as keys only if they contain immutable objects like strings, numbers, or other tuples.
- If a tuple used as key contains any mutable object either directly or indirectly, then an error is generated.
- The **in** keyword can be used to check whether a single key is present in the dictionary.

➢ Sorting Items in a Dictionary:

- The **keys()** method of dictionary returns a list of all the keys used in the dictionary in a arbitrary order.
- The **sorted()** function is used to sort the keys as shown below:

Example:

1. Program to sort keys of a dictionary

```
d={'roll_no':653,'name':'python','course':'b.tech'}
print(sorted(d.keys()))
```

output:

['course', 'name', 'roll_no']

➢ Looping Over a Dictionary:

You can loop over a dictionary to access only values, only keys,and both using the **for loop** as shown the code given below:

1. **Program to access**

```
d={'roll_no':653,'name':'python','course':'b.tech'}
print("KEYS:",end=' ')
for key in d:
        print(key,end=' ')
print("\n VALUES:",end=' ')
for val in d.values():
        print(val,end=' ')
print("\n Dictionary:",end=' ')
for key,val in d.items():
        print(key,val,end=';')
```

output:

KEYS: course name roll_no
VALUES: b.tech python 653
Dictionary: course b.tech;name python;roll_no 653;

➢ Built-in Dictionary Functions and Methods:

Operation	Description	Example	Output
`len(Dict)`	Returns the length of dictionary. That is, number of items (key-value pairs)	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print(len(Dict1))`	`3`
`str(Dict)`	Returns a string representation of the dictionary	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print(str(Dict1))`	`{'Name': 'Arav', 'Roll_No': '16/001', 'Course': 'BTech'}`

`Dict.clear()`	Deletes all entries in the dictionary	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}`
		`Dict1.clear()`
		`print(Dict1)`
			`{}`

`Dict.copy()`	Returns a shallow copy of the dictionary, i.e., the dictionary returned will not have a duplicate copy of `Dict` but will have the same reference	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}`
		`Dict2 = Dict1.copy()`
		`print("Dict2 : ", Dict2)`
		`Dict2['Name'] = 'Saesha'`
		`print("Dict1 after modification : ", Dict1)`
			`Dict2 : {'Course': 'BTech', 'Name': 'Arav', 'Roll_No': '16/001'}`
			`Dict1 after modification: {'Course': 'BTech',`

		`print("Dict2 after modification : ",Dict2)`	`'Name': 'Arav', 'Roll_No': '16/001'}`
			`Dict2 after modification:`
			`{'Course': 'BTech', 'Name': 'Saesha', 'Roll_No': '16/001'}`

`Dict.fromkeys(seq[,val])`	Create a new dictionary with keys from seq and values set to val. If no val is specified then, None is assigned as default value	`Subjects = ['CSA', 'C++', 'DS', 'OS']`
		`Marks = dict.fromkeys(Subjects,-1)`
		`print(Marks)`
			`{'OS': -1, 'DS': -1, 'CSA': -1, 'C++': -1}`

`Dict.get(key)`	Returns the value for the key passed as argument. If the key is not present in dictionary, it will return the default value. If no default value is specified then it will return None	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print( Dict1.get('Name'))`	`Arav`
`Dict.has_key(key)`	Returns True if the key is present in the dictionary and False otherwise	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print('Marks' in Dict1)`	`False`
`Dict.items()`	Returns a list of tuples (key-value pair)	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print(Dict1.items())`	`[('Course', 'BTech'), ('Name', 'Arav'), ('Roll_No', '16/001')]`
`Dict.keys()`	Returns a list of keys in the dictionary	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print(Dict1.keys())`	`['Course', 'Name', 'Roll_No']`

`Dict.setdefault(key, value)`	Sets a default value for a key that is not present in the dictionary	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}`
		`Dict1.setdefault('Marks',0)`
		`print(Dict1['Name'], "has got marks = ", Dict1.get('Marks'))`
			`Arav has got marks = 0`

`Dict1.update(Dict2)`	Adds the key-value pairs of Dict2 to the key-value pairs of Dict1	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}`
		`Dict2 = {'Marks' : 90, 'Grade' : 'O'}`
		`Dict1.update(Dict2)`
		`print(Dict1)`
			`{'Grade':'O', 'Course': 'BTech', 'Name': 'Arav', 'Roll_No': '16/001', 'Marks': 90}`

`Dict.values()`	Returns a list of values in dictionary	`Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}` `print(Dict1.values())`	`['BTech', 'Arav', '16/001']`

`Dict.iteritems()`	Used to iterate through items in the dictionary	`Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}`
		`for i,j in Dict.iteritems():`
		`  print(i, j)`
			`Course BTech`
			`Name Arav`
			`Roll_No 16/001`

`in and not in`	Checks whether a given key is present in dictionary or not	`Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}`
		`print('Name' in Dict)`
		`print('Marks' in Dict)`
			`True`
			`False`

➢ Nested Dictionaries :

Dictionary with in another dictionary is called Nested dictionary.

Example:

```
students={'cse1':{'c':90,'ds':89,'python':98},
        'cse2':{'c':90,'ds':99,'python':98},
        'cse3':{'c':99,'ds':99,'python':98}}
for key,value in students.items():
            print(key,value)
```

Output:

```
cse3 {'python': 98, 'c': 99, 'ds': 99}
cse2 {'python': 98, 'c': 90, 'ds': 99}
cse1 {'python': 98, 'c': 90, 'ds': 89}
```

➢ Difference between a List and a Dictionary:

- First, a list is an ordered set of items. But, a dictionary is a data structure that is used for matching one item (key) with another (value).
- Second, in lists, you can use indexing to access a particular item. But, these indexes should be a number. In dictionaries, you can use any type (immutable) of value as an index. For example, when we write **Dict['Name'],** Name acts as an index but it is not a number but a string.
- Third, lists are used to look up a value whereas a dictionary is used to take one value and look up another value. For this reason, dictionary is also known as a ***lookup table***.
- Fourth, the key-value pair may not be displayed in the order in which it was specified while defining the dictionary. This is because Python uses complex algorithms (called **hashing**) to provide fast access to the items stored in the dictionary. This also makes dictionary preferable to use over a list of tuples.

➢ String Formatting with Dictionaries:

Python also allows you to use string formatting feature with dictionaries. So you can use %s, %d, %f, etc. to represent string, integer, floating point number, or any other data.

Example:

Program that uses string formatting feature to print the key-value pairs stored in the dictionary.

```
d={"cse":98,"ece":99,"eee":90}
for key,value in d.items():
   print("%s branch:%d"%(key,value))
```

output:

```
ece branch:99
cse branch:98
eee branch:90
```

Lab Programs:

5a.Write a program to count the number of characters in the string and store them in dictionary

```
n=int(input("Enter the number"))
i=0;
dict1={}
while(i<n):
        str1=input("Enter the string")
         length=len(str1)
         dict1[str1]=length
        i=i+1;
print('Entered dictionary elements are')
print(dict1)
```

Output

```
Enter the number2
Enter the stringhari
Enter the stringchennai
Entered dictionary elements are
{'hari': 4, 'chennai': 7}
```

5b.Write a program to sort keys in a Dictionary

```
Dict1={'Course':'B.Tech','Rollno':'565','Address':'GDV'}
for key in sorted(Dict1):
   print("%s%s"%(key,Dict1[key]))
```

Output

```
AddressGDV
CourseB.Tech
Rollno565
```

5c.Write a Program that prints maximum and minimum value in a dictionary

```
Dict1={'Course':'B.Tech','Rollno':'565','Address':'GDV'}
print('Minimum value is ',min(Dict1.values()))
print('Maximum value is',max(Dict1.values()))
```

Output

```
>>>
Minimum value is  565
Maximum value is GDV
```

Assignment-Cum-Tutorial Questions

I) Objective Questions

1) Which of these about a dictionary is false? [ ]

a) The values of a dictionary can be accessed using keys
b) The keys of a dictionary can be accessed using values
c) Dictionaries aren't ordered
d) Dictionaries are mutable

2) Which of the following statements create a dictionary? [ ]

a) d = {}
b) d = {"john":40, "peter":45}
c) d = {40:"john", 45:"peter"}
d) All of the mentioned

3) Which of the following is not a declaration of the dictionary? [ ]

a) {1: 'A', 2: 'B'}
b) dict ([[1,"A"],[2,"B"]])
c) {1,"A",2,"B"}
d) { }

4) What is the output of the following code? [ ]

```
A = {1:"A",2:"B",3:"C"}
for i,j in a.items():
    print(i,j,end=" ")
```

a) 1 A 2 B 3 C b) 1 2 3 c) A B C d) 1:"A" 2:"B" 3:"C"

5) Which of the following isn't true about dictionary keys? [ ]

a) More than one key isn't allowed
b) Keys must be immutable
c) Keys must be integers
d) When duplicate keys encountered, the last assignment wins

6) Suppose d = {"john":40, "peter":45}, to delete the entry for "john" what command do we use [ ]

a) d.delete("john":40)
b) d.delete("john")
c) del d["john"].
d) del d("john":40)

7) Suppose d = {“john”:40, “peter”:45}, what happens when we try to retrieve a value using the expression d[“susan”]? **[ ]**

a) Since “susan” is not a value in the dictionary, Python raises a KeyError exception
b) It is executed fine and no exception is raised, and it returns None
c) Since “susan” is not a key in the dictionary, Python raises a KeyError exception
d) Since “susan” is not a key in the set, Python raises a syntax error

8) What gets printed? [ ]

```
foo = {1:'1', 2:'2', 3:'3'}
del foo[1]
foo[1] = '10'
del foo[2]
print(len(foo))
```

a) 1 b) 2 c) 3 d) 4 e) An Exception is thrown

9) If Dict = {1:2, 3:4, 4:11, 5:6, 7:8}, then **print(Dict(Dict[3]))** will print ? [ ]
a) 2 b) 8 c) 11 d) 6

10) Which Data type does not support indexing? [ ]
a) List b) Tuple c) Dictionary d) Set

11) Which function is used to delete all entries in the dictionary _____?

12) Which methods will return all the keys and Values in a Dictionary _____?

13) What are the Data types supported for Key in Dictionary Data type _____?

14) Fill in the blanks to create a Dictionary.

```
Dict = dict(1 ___ “abc” __ 2__”hai”)
Dict1=__1:”abc”,2:”hai”__
```

15) Find the output of the below program?

```
D={"India":"Delhi", "Nepal":"Kathmandu", "USA":"DC"}
del D["Nepal"]
for key,val in D.items():
    print(key)
```

II) Subjective Questions

1) Explain the importance of Dictionary data type in python? **(Nov-2018)**

2) List-out various operations can be performed on Dictionary Data type? **(Nov-2018)**

3) List-out the Built-in functions and methods of Dictionary Data type in python? **(Nov-2018)**

4) Write a Python program to check if all dictionaries in a list are empty or not. **(Nov-2018)**

5) How to delete items from a dictionary? Explain with an example.**(April-2008)**

6) Write a Python script to sort (ascending and descending) a dictionary by value.

7) Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).
   Sample:
   Dictionary (n = 5):
   Expected Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

8) Write a Python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys.
   Sample Dictionary:
   {1: 1, 2: 5, 3: 9, 4: 15, 5: 25, 6: 36, 7: 49, 80: 64, 9: 80, 90: 100, 11: 121, 12: 144, 13: 169, 14: 200, 105: 225}

9) Write a Python program to map two lists into a dictionary.

10) Write a python program to check if all dictionaries in a list are empty or not?

11) Write a Python program to combine two dictionary adding values for common keys.
    d1 = {'a': 100, 'b': 200, 'c':300}
    d2 = {'a': 300, 'b': 200,'d':400}
    Sample output: {'a': 400, 'b': 400,'d': 400, 'c': 300}

12) Write a Python program to create and display all combinations of letters, selecting each letter from a different key in a dictionary

Sample data: {'1':['a','b'], '2':['c','d']}

Expected Output:

ac
ad
bc
bd

13) Write a Python program to get the top three items in a shop.

Sample data: {'item1': 45.50, 'item2':35, 'item3': 41.30, 'item4':55, 'item5': 24}

Expected Output:

item4: 55
item1: 45.5
item3: 41.3

UNIT-VI

File Handling

Syllabus:

File types, File path, File operations-open, close, read, write, Types of arguments.

Programs: Write a python program to

1. Write a program to print each line of a file in reverse order.
2. Write a program to compute the number of characters, words and lines in a file.
3. Write a program to copy contents of one file into another file.

Learning Outcomes:

At the end of the unit student will be able to

- Understand various File Types.
- Describe File Paths like Absolute and Relative.
- Understand various operations on files.
- Implement programs on Files.

Learning Material

1. Introduction to Files:

- When a program is being executed,its data is stored in **RAM**.Though RAM can be accessed faster by the CPU,it is also **volatile,**which means when the program ends, or the computer shuts down, all the data is lost.If you want to use the data in future,then you need to store this data on a permanent or non-volatile storage media such as hard disk, USB drive and DVD e.t.c.,
- **A file** is a collection of data stored on a secondary storage device like **hard disk**.
- A file is basically used because real-life applications involve large amounts of data and in such situations the console oriented I/O operations pose two major problems:
- ***First***, it becomes cumbersome and time consuming to handle huge amount of data through terminals.
- ***Second***, when doing I/O using terminal, the entire data is lost when either the program is terminated or computer is turned off. Therefore, it becomes necessary to store data on a permanent storage (the disks) and read whenever necessary, without destroying the data.

2. File Types

- Like C and C++,Python also supports two types of files

1.ASCII Text Files
2.Binary Files

2.1ASCII Text Files

- A ***text file*** is a stream of characters that can be sequentially processed by a computer in forward direction. For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.
- Because text files can process characters, they can only read or write data one character at a time. In Python, a text stream is treated as a special kind of file.
- Depending on the requirements of the operating system and on the operation that has to be performed (read/write operation) on the file, the newline characters may be converted to or from carriage-return/linefeed combinations.

- Besides this, other character conversions may also be done to satisfy the storage requirements of the operating system. However, these conversions occur transparently to process a text file. In a text file, each line contains zero or more characters and ends with one or more characters.
- Another important thing is that when a text file is used, there are actually two representations of data- internal or external. For example, an integer value will be represented as a number that occupies 2 or 4 bytes of memory internally but externally the integer value will be represented as a string of characters representing its decimal or hexadecimal value**.**

  Note: *In a text file, each line of data ends with a newline character. Each file ends with a special character called end-of-file (EOF) Marker.*

2.2 Binary Files

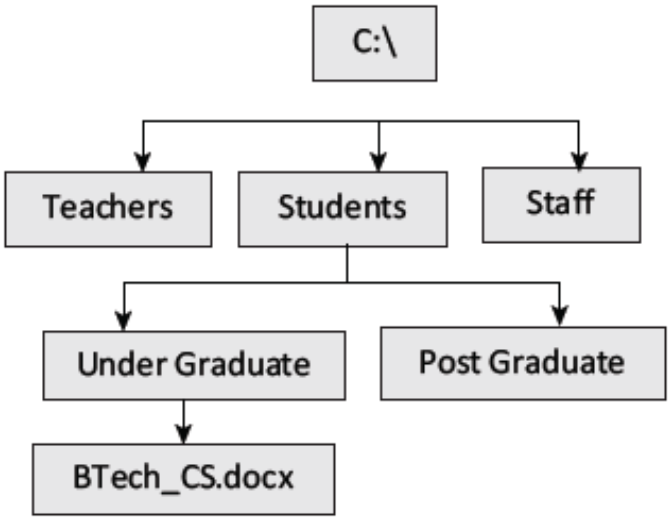
- A ***binary file*** is a file which may contain any type of data, encoded in binary form for computer storage and processing purposes. It includes files such as word processing documents, PDFs, images, spreadsheets, videos, zip files and other executable programs.
- Like a text file, a binary file is a collection of bytes. A binary file is also referred to as a character stream with following two essential differences.
- A binary file does not require any special processing of the data and each byte of data is transferred to or from the disk unprocessed.
- Python places no constructs on the file, and it may be read from, or written to, in any manner the programmer wants.
- While **text files** can be processed **sequentially**, **binary files**, on the other hand, can be either processed **sequentially or randomly** depending on the needs of the application.

Note*:Binary files store data in the internal representation format.Therefore, an integer value will be stored in binary form as 2 byte value.The same format is used to store data in memory as well as in files.Like Text files,Binary files also ends with an EOF Marker*

3.File Path:

- Files that we use are stored on a storage medium like the hard disk in such a way that they can be easily retrieved as and when required.
- Every file is identified by its path that begins from the root node or the root folder. In Windows, C:\ (also known as C drive) is the root folder but you can also have a path that starts from other drives like D:\, E:\, etc. The file path is also known as ***pathname***.
- In order to access a file on a particular disk we have two paths.

1. Absolute Path
2. Relative Path



- While an ***absolute path*** always contains the root and the complete directory list to specify the exact location the file.
  *Example:*To access BTech_CS.docx,The absolute path is
  C:\Students\Under Graduate\BTech_CS.docx

- ***Relative path*** needs to be combined with another path in order to access a file. It starts with respect to the current working directory and therefore lacks the leading slashes.
  Example: Suppose you are working on current directory Under Graduate in order to access BTech_CS.docx,The Relative path is
  Under Graduate\BTech_CS.docx

4.File Operations

- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Python has many in-built functions and methods to manipulate files.These
- Hence, in Python, a file operation takes place in the following order.
    1. **Open a file**
    2. **Read or write (perform operation)**
    3. **Close the file**

4.1 Opening A File

- Before reading from or writing to a file, you must first open it using Python's built-in open() function. This function creates a file object, which will be used to invoke methods associated with it.
- The Syntax of open() is:

    fileObj = open(file_name [, access_mode])

    Where file_name is a string value that specifies name of the file that you want to access. *access_mode* indicates the mode in which the file has to be opened, i.e., read, write, append, etc.
- **Example**:**Write a Program to print the details of file object**

```
file = open("File1.txt", "rb")
print(file)

OUTPUT
<open file 'File1.txt', mode 'rb' at 0x02A850D0>
```

Note: Access mode is an optional parameter and the default file access mode is read(r).

4.1.1.Access modes

- Python supports the following access modes for opening a file those are

Mode	Purpose
r	This is the default mode of opening a file which opens the file for reading only. The file pointer is placed at the beginning of the file.
rb	This mode opens a file for reading only in binary format. The file pointer is placed at the beginning of the file.
r+	This mode opens a file for both reading and writing. The file pointer is placed at the beginning of the file.
rb+	This mode opens the file for both reading and writing in binary format. The file pointer is placed at the beginning of the file.
w	This mode opens the file for writing only. When a file is opened in w mode, two things can happen. If the file does not exist, a new file is created for writing. If the file already exists and has some data stored in it, the contents are overwritten.
wb	Opens a file in binary format for writing only. When a file is opened in this mode, two things can happen. If the file does not exist, a new file is created for writing. If the file already exists and has some data stored in it, the contents are overwritten.
w+	Opens a file for both writing and reading. When a file is opened in this mode, two things can happen. If the file does not exist, a new file is created for reading as well as writing. If the file already exists and has some data stored in it, the contents are overwritten.
wb+	Opens a file in binary format for both reading and writing. When a file is opened in this mode, two things can happen. If the file does not exist, a new file is created for reading as well as writing. If the file already exists and has some data stored in it, the contents are overwritten.
a	Opens a file for appending. The file pointer is placed at the end of the file if the file exists. If the file does not exist, it creates a new file for writing.
ab	Opens a file in binary format for appending. The file pointer is at the end of the file if the file exists. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both reading and appending. The file pointer is placed at the end of the file if the file exists. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file in binary format for both reading and appending. The file pointer is placed at the end of the file if the file exists. If the file does not exist, a new file is created for reading and writing.

4.1.2The File Object Attributes

- Once a file is successfully opened, a *file* object is returned. Using this file object, you can easily access different type of information related to that file. This information can be obtained by reading values of specific attributes of the file.
- The Following table shows list attributes related to file object.

Attribute	Information Obtained
fileObj.closed	Returns true if the file is closed and false otherwise
fileObj.mode	Returns access mode with which file has been opened
fileObj.name	Returns name of the file

- **Example: Program to open a file and print its attribute values.**

```
file = open("File1.txt", "wb")
print("Name of the file: ", file.name)
print("File is closed.", file.closed)
```

```
print("File has been opened in ", file.mode, "mode")
```

OUTPUT

```
Name of the file:  File1.txt
File is closed. False
File has been opened in wb mode
```

4.2 Closing A File

- The *close() method* is used to close the file object. Once a file object is closed, you cannot further read from or write into the file associated with the file object.
- While closing the file object the close() flushes any unwritten information. Although, Python automatically closes a file when the reference object of a file is reassigned to another file, but as a good programming habit you should always explicitly use the close() method to close a file.
- The syntax of close() is

fileObj.close()

- The close() method frees up any system resources such as file descriptors, file locks, etc. that are associated with the file.
- Once the file is closed using the close() method, any attempt to use the file object will result in an error.

Example2: Write a Python program to assess if a file is closed or not..
(April 2018 Regular)

```
file = open('File1.txt','wb')

print('Name of the file :',file.name)

print('File is closed:',file.closed)

print('File is now being closed')
```

```
file.close()

print('File is closed',file.closed)

print(file.read())
```

Output:

```
Name of the file : File1.txt

File is closed: False

File is now being closed

File is closed True

Traceback (most recent call last):

  File "D:/Python/sample.py", line 7, in <module>

    print(file.read())

io.UnsupportedOperation: read
```

4.3 Writing A File

- The *write() method* is used to write a string to an already opened file. Of course this string may include numbers, special characters or other symbols.
- While writing data to a file, you must remember that the write() method does not add a newline character ('\n') to the end of the string.
- The syntax of write() method is:

fileObj.write(string)

Example:Program that writes a message in the file,data.txt

```
file=open('data.txt','w')

file.write('hello cse we are learning python programming')

file.close()

print('file writing successful')
```

data.txt

hello cse we are learning python programming

Output:

```
file writing successful
```

4.3.1 writeline() method:

- The *writelines() method* is used to write a list of strings.

Example:Program to write to afile using the writelines() method

```
file=open('data.txt','w')
lines=['hello','cse','hope to enjoy','learning','python programming']
file.writelines(lines)
file.close()
print('file writing successful')
```

data.txt

hellocsehope to enjoylearningpython programming

Output:

file writing successful

4.3.2 append() method:

- Once you have stored some data in a file,you can always open that file again to write more data or append data to it.
- To append a file, you must open it using 'a' or 'ab' mode depending on whether it is text file or binary file.
- Note that if you open a file with 'w' or 'wb' mode and then start writing data into it, then the existing contents would be overwritten.

Example:Program to append data to an already existing file

```
file=open('data.txt','a')
file.write('\nHave a nice day')
file.close()
print('Data appended successful')
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

Data appended successful

4.4 Reading A File

- The *read() method* is used to read a string from an already opened file.As said before, the string can include, alphabets, numbers, characters or other symbols.
- The syntax of read() method is

fileObj.read([count])

Where In the above syntax, count is an optional parameter which if passed to the read() method specifies the number of bytes to be read from the opened file.

- The read() method starts reading from the beginning of the file and if *count* is missing or has a negative value then, it reads the entire contents of the file (i.e., till the end of file).

Example1:Program to print the first 8 characters of the file data.txt

```
file=open('data.txt','r')
print(file.read(8))
file.close()
```

data.txt

```
hellocsehope to enjoylearningpython programming
Have a nice day
```

Output:

```
hellocse
```

Example2:Program to display the content of file using for loop

```
file=open('data.txt','r')
for line in file:
    print(line)
file.close()
```

data.txt

```
hellocsehope to enjoylearningpython programming
Have a nice day
```

Output:

```
hellocsehope to enjoylearningpython programming

Have a nice day
```

Note:read() methods returns a newline as '\n'

4.4.1readline() Method

- It is used to read single line from the file.
- This method returns an empty string when end of the file has been reached.

ExampleProgram to demonstrate the usage of readline() function

```
file=open('data.txt','r')
print('firtsline:',file.readline())
print('second line:',file.readline())
print('third line:',file.readline())
file.close()
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

firtsline: hellocsehope to enjoylearningpython programming

second line: Have a nice day

third line:

4.4.2 readlines() Method

- readlines() Method is used to read all the lines in the file.

Example:Program to demonstrate readlines() function

```
file=open('data.txt','r')
print(file.readlines())
file.close()
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

['hellocsehope to enjoylearningpython programming\n', 'Have a nice day']

4.4.3 list() Method

- list() method is also used to display entire contents of the file.you need to pass the file object as an argument to the list() method.

Example: Program to display the contents of the file data.txt using the list() method

```
file=open('data.txt','r')
print(list(file))
file.close()
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

['hellocsehope to enjoylearningpython programming\n', 'Have a nice day']

4.4.4Opening a file using with keyword:

- It is good programming habit to use the ***with*** keyword when working with file objects.
- This has the advantage that the file is properly closed after it is used even if an error occurs during read or write operation or even when you forget to explicitly close the file.

```
with open("file1.txt", "rb") as file:
    for line in file:
        print(line)
print("Let's check if the file is
closed : ", file.close())
```

OUTPUT

```
Hello World
Welcome to the world of Python
Programming.
Let's check if the file is closed :  True
```

```
file = open("file1.txt", "rb")
for line in file:
       print(line)
print("Let's check if the file is
closed : ", file.close())
```

OUTPUT

```
Hello World
Welcome to the world of Python
Programming.
Let's check if the file is closed : False
```

Note: When you open a file for reading, or writing, the file is searched in the current directory. If the file exists somewhere else then you need to specify the path of the file.

4.4.5 Splitting Words:

- Python allows you to read line(s) from a file and splits the line (treated as a string) based on a character. By default, this character is space but you can even specify any other character to split words in the string.

Example: Program to split the line into series of words and use space to perform the split operation

```
with open('data.txt','r') as file:
```

data.txt

hellocsehope to enjoylearningpython programming

```
line=file.readline()
words=line.split()
print(words)
```

Output:

['hellocsehope', 'to', 'enjoylearningpython', 'programming']

4.5 Some Other Useful File Methods:

- The following are some of additional methods which will work on files

Method	Description	Example

`fileno()`	Returns the file number of the file (which is an integer descriptor)	`file = open("File1.txt", "w")`
		`print(file.fileno())`
		OUTPUT
		`3`

`flush()`	Flushes the write buffer of the file stream	`file = open("File1.txt", "w")` `file.flush()`

`isatty()`	Returns True if the file stream is interactive and False otherwise	`file = open("File1.txt", "w")`
		`file.write("Hello")`
		`print(file.isatty())`
		OUTPUT
		`False`

`readline(n)`	Reads and returns one line from file. n is optional. If n is specified then atmost n bytes are read	`file = open("Try.py", "r")`
		`print(file.readline(10))`
		OUTPUT
		`file = ope`

`truncate(n)`	Resizes the file to n bytes	`file = open("File.txt", "w")`
		`file.write("Welcome to the world of programming....")`
		`file.truncate(5)`
		`file = open("File.txt", "r")`
		`print(file.read())`
		OUTPUT
		`Welco`

`rstrip()`	Strips off whitespaces including newline characters from the right side of the string read from the file.	`file = open("File.txt")`
		`line = file.readline()`
		`print(line.rstrip())`
		OUTPUT
		`Greetings to All !!!`

4.6 File Positions:

- With every file, the file management system associates a pointer often known as ***file pointer*** that facilitates the movement across the file for reading and/ or writing data.
- The file pointer specifies a location from where the current read or write operation is initiated. Once the read/write operation is completed, the pointer is automatically updated.
- Python has various methods that tells or sets the position of the file pointer.
- For example, the **tell()** method tells the current position within the file at which the next read or write operation will occur. It is specified as number of bytes from the beginning of the file.
- When you just open a file for reading, the file pointer is positioned at location 0, which is the beginning of the file.
- The syntax for seek() function is

seek(offset[, from])

- The offset argument indicates the number of bytes to be moved and the from argument specifies the reference position from where the bytes are to be moved.

Example: Program that tells and sets the position of file pointer

File1.txt
Hello All,
Hope you are enjoying learning python

```
file = open("File1.txt", "rb")
print("Position of file pointer before reading is : ", file.tell())
print(file.read(10))
print("Position of file pointer after reading is : ", file.tell())
print("Setting 3 bytes from the current position of file pointer")
file.seek(3,1)
print(file.read())
file.close()
```

OUTPUT

```
Position of file pointer before reading is : 0
Hello All,
Position of file pointer after reading is : 10
Setting 3 bytes from the current position of file pointer
pe you are enjoying learning Python
```

4.7 Renaming and Deleting Files:

- The **os** module in Python has various methods that can be used to perform file-processing operations like renaming and deleting files. To use the methods defined in the **os** module, you should first import it in your program then call any related functions.
- The **rename()** Method: The ***rename()*** method takes two arguments, the current filename and the new filename.

Its syntax is: os.rename(old_file_name, new_file_name)

- The **remove()** Method: This method can be used to delete file(s). The method takes a filename (name of the file to be deleted) as an argument and deletes that file.

Its syntax is: os.remove(file_name)

Example: Program to rename file 'File1.txt' to 'student.txt'

```
import os
os.rename("File1.txt", "Students.txt")
print("File Renamed")

OUTPUT
File Renamed
```

Example: Program to delete a file named File1.txt

```
import os
os.remove("File1.txt")
print("File Deleted")

OUTPUT
File Deleted
```

5.Types of Arguments

5.1 Command line Arguments:

- The Python sys module provides access to any command-line arguments via the ***sys.argv***. This serves two purposes −
    - ***sys.argv*** is the list of command-line arguments.

- ***len(sys.argv)*** is the number of command-line arguments.

➢ Here ***sys.argv[0]*** is the program ie. script name.

Example1:Write a Python program to demonstrate the usage of Command Line Arguments

sample11.py

```
#!/usr/bin/python

import sys

print ('Number of arguments:', len(sys.argv), 'arguments.')

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

Output:

```
C:\Python34>sample11.py 1 2 3
Number of arguments: 4 arguments.
Argument List: ['C:\\Python34\\sample11.py', '1', '2', '3']
```

Example2: Write a Python program to copy the content of one file to another using command line arguments.

sample12.py

```
#!/usr/bin/python

import sys

print ('Number of arguments:', len(sys.argv), 'arguments.')

with open(str(sys.argv[1])) as f:

    with open((sys.argv[2]), "w") as f1:

        for line in f:

            f1.write(line)

print('File Copied Success')
```

input.txt
Hello hi
How are you

output.txt
Hello hi
How are you

```
C:\Python34>sample12.py input.txt output.txt
Number of arguments: 3 arguments.
File Copied Success
```

6. File Handling Programs

1.Write a program to print each line of a file in reverse order

Program:

```
with open('input.txt','r') as fp:
    for line in fp:
        print (line[::-1])
```

input.txt
Hello hi
How are you

Output:

```
ih olleH
uoy era woH
```

2. Write a program to compute the number of characters, words and lines in a file

Program:

```
fname = "data.txt"
num_lines = 0
num_words = 0
num_chars = 0
with open(fname, 'r') as f:
        for line in f:
            words = line.split()
            num_lines += 1
            num_words += len(words)
            num_chars += len(line)
print('The no of lines in a given file is',num_lines)
print('The no of words in a given file is',num_words)
print('The no of chars in a given file is',num_chars)
```

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

Output:

```
The no of lines in a given file is 2
The no of words in a given file is 8
The no of chars in a given file is 63
```

3. Write a program to copy contents of one file to another file

Program:

```
with open("data.txt") as f:
    with open("out.txt", "w") as f1:
        for line in f:
      f1.write(line)
```

Output:

data.txt

hellocsehope to enjoylearningpython programming

Have a nice day

out.txt

hellocsehope to enjoylearningpython programming

Have a nice day

4. Write a Python Program to count number of Vowels and Number of Consonants in a given file. (November 2018 Supplementary)

Program:

```
infile = open("data.txt", "r")
vowels = set("AEIOUaeiou")
cons = set("bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ")
countV = 0
countC = 0
for c in infile.read():
    if c in vowels:
        countV += 1
    elif c in cons:
        countC += 1
print("The no of Vowels are",countV)
print("The no of Consonants are",countC)
```

Assignment-Cum-Tutorial Questions

A. Objective Questions

1. Identify The right way to close a file [ ]
   a)File.close() b)close(file) c)close("file") d)File.closed
2. _____ is an example of volatile memory
3. A file is stored in _____ memory [ ]
   a)primary b)secondary c)cache d)volatile
4. What will happen when a file is opened in write mode and then immediately closed. [ ]
   a)Filecontentsaredeleted
   b) Nothing Happens
   c) A Blank Line is written to the file
   d)an error occurs
5. The default access mode of the file is _____
6. If a file opened in 'w' mode does not exist, then [ ]
   a) nothing will happen
   b) File will be created
   c) Data will be written to a afile that has a name similar to the specified name
   d) Error will be generated
7. Identify the delimiter in the Solaris file system [ ]
   a)/ b)\ c): d)|
8. By default a new file is created in which directory [ ]
   a)root b)current working c)Python directory d)D Drive
9. which method is used to read a single line from the file [ ]
   a)read() b)readline() c)readlines() d)reads()
10. When you open a file for appending that does not exist, then a new file is created [True/False]
11. Identify the correct way to write "Welcome to Python" in a file [ ]
    a)write(file,"Welcome to python")
    b)write("Welcome to Python",file)
    c)file.write("Welcome to Python")
    d)"Welcome to Python".write(file)
12. If the file.txt has 10 lines written in it,what will the result? [ ]
    len(open('file.txt').readlines())
    a)1 b)0 c)10 d)2
13. Identify the sub folder in the path [ ]
    C:\Students\UnderGraduates\B.Tech_CS.docx

a)C: b) Students c)B.Tech_CS.docx d) UnderGraduates

14. Which method returns a string that includes everything specified in the path? [ ]

a)os.path.dirname(path)
b)os.path.basename(path)
c)os.path.relpath()
d)os.path.abs()

15. if count is missing or has a negative value in the read() method then, no contents are read from the file. [True/False]

16. os.path.abs() method accepts a file path as an argument and returns True if the path is an absolute path and False otherwise [True/False]

17. How many characters would be printed by this code (One character is one byte)_____

```
file=open("file.txt","r")
    for i in range(100):
        print(file.read(10))
file.close()
```

18. Fill in the blank to open a file,read its content and prints its length

```
file=_____("file.txt","r")
text=file._____()
print(_____(text))
file.close()
```

19. Predict the output of the following program [ ]

```
f = None
for i in range (5):
   with open("data.txt", "w") as f:
      if i > 2
         break
print(f.closed)
```

a)True b) False c) None d) Error

20. Predict the output of the following program [ ]

```
with open("hello.txt", "w") as f:
f.write("Hello World how are you today")
with open('hello.txt', 'r') as f:
data = f.readlines()
for line in data:
    words = line.split()
    print (words)
f.close()
```

a. Runtime Error
b. Hello World how are you today
c. [‘Hello’, ‘World’, ‘how’, ‘are’, ‘you’, ‘today’]
d. Hello

B. Descriptive Questions

1. Define file. Explain about the importance of files in Python.
2. Define path. Distinguish between absolute and relative path with an example.
3. Discuss briefly about various types of file.
4. Write in detail about various modes of file.
5. Give an overview of File positions.
6. Explain different file operations with suitable programming examples.
   (April 2018 Regular and November 2018 Supplementary)
7. What is the purpose of opening a file using with keyword.
8. Write a Python program to count number of vowels and consonants in a given text file
9. Write a Python program that reads data from a file and Calculates the percentage of vowels and consonants in the file
10. Write a Python program that copies one file to another in such away that all comment lines are skipped and not copied in the destination file
11. Write a python program to find no of lines, words and characters in a given text file
12. Write a Python program to combine each line from first file with the corresponding line in second file.
13. Write a program that accepts file name as an input from the user. Open the file and count number of times a character appears in the file.
14. Write a program that tells and sets the position of the file pointer.

----------OOO----------