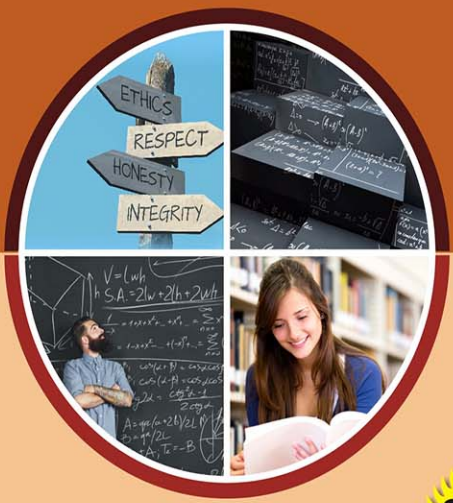
Quantum Series®
ENGINEERING

QUANTUM *Series*

Semester - 3 & 4 Common to All Branches

Python Programming



- Topic-wise coverage of entire syllabus in Question-Answer form.
- Short Questions (2 Marks)

Session 2019-20 Odd & Even Semester

Includes solution of following AKTU Question Papers

- 2018-19

QUANTUM SERIES

For

B.Tech Students of Second Year
of All Engineering Colleges Affiliated to
Dr. A.P.J. Abdul Kalam Technical University,
Uttar Pradesh, Lucknow
(Formerly Uttar Pradesh Technical University)

Python Programming

By

Aditya Kumar **Chetan Singhal**

QUANTUM PAGE PVT. LTD.
Ghaziabad ■ New Delhi

PUBLISHED BY : **Apram Singh**
Quantum Publications®
(A Unit of Quantum Page Pvt. Ltd.)
Plot No. 59/2/7, Site - 4, Industrial Area,
Sahibabad, Ghaziabad-201 010

Phone : 0120 - 4160479

Email : pagequantum@gmail.com **Website: www.quantumpage.co.in**

Delhi Office : 1/6590, East Rohtas Nagar, Shahdara, Delhi-110032

Python Programming (Common to All Branches: Sem-3 & 4)
1st Edition : *2019-20*

Price: Rs. 65/- only

Printed Version : e-Book.

CONTENTS

KNC 302 / KNC 402 : PYTHON PROGRAMMING

UNIT-1 : INTRODUCTION AND BASICS (1–1 T to 1–28 T)

Introduction: The Programming Cycle for Python , Python IDE, Interacting with Python Programs , Elements of Python, Type Conversion.

Basics: Expressions, Assignment Statement, Arithmetic Operators, Operator Precedence, Boolean Expression.

UNIT-2 : CONDITIONALS AND LOOPS (2–1 T to 2–23 T)

Conditionals: Conditional statement in Python (if-else statement, its working and execution), Nested-if statement and Elif statement in Python, Expression Evaluation & Float Representation.

Loops: Purpose and working of loops, While loop including its working, For Loop, Nested Loops, Break and Continue.

UNIT-3 : FUNCTION AND STRINGS (3–1 T to 3–31 T)

Function: Parts of A Function, Execution of A Function, Keyword and Default Arguments, Scope Rules.

Strings : Length of the string and perform Concatenation and Repeat operations in it, Indexing and Slicing of Strings.

Python Data Structure : Tuples, Unpacking Sequences, Lists, Mutable Sequences, List Comprehension, Sets, Dictionaries.

Higher Order Functions: Treat functions as first class Objects, Lambda Expressions.

UNIT-4 : SIEVE OF ERATOSTHENES & FILE I/O (4–1 T to 4–32 T)

Sieve of Eratosthenes: generate prime numbers with the help of an algorithm given by the Greek Mathematician named Eratosthenes, whose algorithm is known as Sieve of Eratosthenes.

File I/O : File input and output operations in Python Programming.

Exceptions and Assertions Modules : Introduction, Importing Modules.

Abstract Data Types : Abstract data types and ADT interface in Python Programming.

Classes : Class definition and other operations in the classes, Special Methods (such as _init_, _str_, comparison methods and Arithmetic methods etc.), Class Example, Inheritance, Inheritance and OOP.

UNIT-5 : ITERATORS & RECURSION (5–1 T to 5–18 T)

Iterators & Recursion: Recursive Fibonacci, Tower Of Hanoi.

Search : Simple Search and Estimating Search Time, Binary Search and Estimating Binary Search Time.

Sorting & Merging : Selection Sort, Merge List, Merge Sort, Higher Order Sort.

SHORT QUESTIONS (SQ–1 T to SQ–22 T)

SOLVED PAPERS (2018-19) (SP–1 T to SP–3 T)

Quantum Series®
ENGINEERING

QUANTUM *Series*

Related titles in Quantum Series

For Semester - 3 & 4
(Common to All Branches)

- Computer Science and Engineering
- Information Technology
- Mechanical Engineering
- Civil Engineering
- Electronics and Allied Branches
- Electrical and Electronics Engineering

A comprehensive book to get the big picture without spending hours over lengthy text books.

Quantum Series is the complete one-stop solution for engineering student looking for a simple yet effective guidance system for core engineering subject. Based on the needs of students and catering to the requirements of the syllabi, this series uniquely addresses the way in which concepts are tested through university examinations. The easy to comprehend question answer form adhered to by the books in this series is suitable and recommended for student. The students are able to effortlessly grasp the concepts and ideas discussed in their course books with the help of this series. The solved question papers of previous years act as a additional advantage for students to comprehend the paper pattern, and thus anticipate and prepare for examinations accordingly.

The coherent manner in which the books in this series present new ideas and concepts to students makes this series play an essential role in the preparation for university examinations. The detailed and comprehensive discussions, easy to understand examples, objective questions and ample exercises, all aid the students to understand everything in an all-inclusive manner.

- Topic-wise coverage in Question-Answer form.
- Clears course fundamentals.
- Includes solved University Questions.
- The perfect assistance for scoring good marks.
- Good for brush up before exams.
- Ideal for self-study.

Quantum Publications®
(A Unit of Quantum Page Pvt. Ltd.)
Plot No. 59/2/7, Site-4, Industrial Area, Sahibabad,
Ghaziabad, 201010, (U.P) Phone: 0120-4160479
E-mail: pagequantum@gmail.com Web: www.quantumpage.co.in
Find us on: **facebook.com/quantumseriesofficial**

Quantum Page™

1
UNIT

Introduction and Basics

CONTENTS

Part-1 : Introduction : The Programming Cycle for Python, Python IDE, Interacting with Python Programs ..... **1–2T to 1–6T**

Part-2 : Elements of Python, Type Conversion ..... **1–6T to 1–12T**

Part-3 : Basics : Expressions, Assignment Statement ..... **1–12T to 1–13T**

Part-4 : Arithmetic Operators ..... **1–13T to 1–18T**

Part-5 : Operator Precedence, Boolean Expression ..... **1–18T to 1–28T**

PART- 1

Introduction : The Programming Cycle for Python, Python IDE, Interacting with Python Programs.

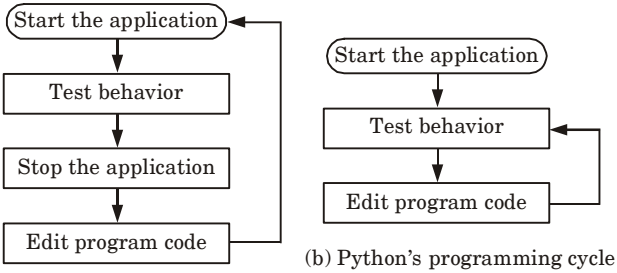
Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. Explain the programming cycle for Python.

Answer

1. Python's programming cycle is dramatically shorter than that of traditional programming cycle.
2. In Python, there are no compile or link steps.
3. Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.
4. In cases where dynamic module reloading can be used, it is even possible to change and reload parts of a running program without stopping it at all.
5. Fig. 1.1.1 shows Python's impact on the programming cycle.



(a) Python's programming cycle (b) Python's programming cycle with module reloading

Fig. 1.1.1.

6. Since Python is interpreted, there is a rapid turnaround after program changes. And because Python's parser is embedded in Python-based systems, it is easy to modify programs at runtime.

Que 1.2. What is IDE ? Discuss some Python IDE.

Answer

1. IDE is a software package that consists of several tools for developing and testing the software.
2. An IDE helps the developer by automating the process.
3. IDEs integrate many tools that are designed for SDLC.
4. IDEs were introduced to diminish the coding and typing errors.
5. Some of the Python IDEs are :

   a. **PyCharm :** PyCharm assists the developers to be more productive and provides smart suggestions. It saves time by taking care of routine tasks, hence increases productivity.

   Features of PyCharm :

   i. It has smart code navigation, good code editor, a function for quick refactoring.

   ii. The integrated activities with PyCharm are profiling, testing, debugging, remote development, and deployments.

   iii. PyCharm supports Python web development frameworks, Angular JS, JavaScript, CSS, HTML and live editing functions.

   b. **Spyder :** Spyder is widely used for data science works. It is mostly used to create a secure and scientific environment for Python. Spyder Python uses PyQt (Python plug-in) which a developer can add as an extension.

   Features of Spyder :

   i. It has good syntax highlighting and auto code completion features.

   ii. Spyder Python explores and edits variables directly from GUI.

   iii. It performs very well in multi-language editor.

   c. **PyDev :** It is an external plug-in for Eclipse and is very popular as Python interpreter.

   Features of PyDev :

   i. PyDev has strong parameters like refactoring, debugging, type hinting, code analysis, and code coverage function.

   ii. PyDev supports tokens browser, PyLint integration, interactive console, remote debugger, Unittest integration, etc.

   d. **IDLE :** IDLE is a basic IDE mainly used by beginner level developer.

   i. IDLE Python is a cross-platform IDE, hence it increases the flexibility for users.

   ii. It is developed only in Python in collaboration with Tkinter GUI toolkit.

iii. The feature of multi-window text editor in IDLE has some great functions like smart indentation, call tips, Python colorizing, and undo option.

iv. It also comes with a strong debugger along with continuous breakpoints, local spaces, and global view.

v. It supports browsers, editable configurations, and dialog boxes.

e. Visual studio : It enables development for various platforms and has its own marketplace for extensions.

Features of visual studio :

i. It supports Python coding in visual studio, debugging, and other activities.

ii. It has both paid and free versions in the market with great features.

Que 1.3. Discuss interaction with Python program with example.

Answer

1. The Python program that we have installed will by default act as an interpreter.
2. An interpreter takes text commands and runs them as we enter text.
3. After Python opens, it will show some contextual information like this :

   Python 3.5.0 (default, dec 20 2019, 11 : 28 : 25)

   [GCC 5.2.0] on Linux

   Type "help", "copyright", "credits" or "license" for more information

   >>>
4. The prompt >>> defines that we are now in an interactive Python interpreter session, also called the Python shell.
5. Now enter some code for Python to run such as print("Hello World!") and press the Enter key.
6. The interpreter's response should appear on the next line like this :

   >>> print ("Hello World!")

   Hello World!
7. After showing the results, Python will bring you back to the interactive prompt, where we could enter another command or code.
8. Python program communicates its result to user using print statement.

Que 1.4. What is Python ? How Python is interpreted ? What are the tools that help to find bugs or perform static analysis ? What are Python decorators ? **AKTU 2019-20, Marks 10**

Answer

Python : Python is a high-level, interpreted, interactive and object-oriented scripting language. It is a highly readable language. Unlike other programming languages, Python provides an interactive mode similar to that of a calculator.

Interpretation of Python :

1. An interpreter is a kind of program that executes other programs.
2. When we write Python programs, it converts source code written by the developer into intermediate language which is again translated into the machine language that is executed.
3. The python code we write is compiled into python bytecode, which creates file with extension .pyc .
4. The bytecode compilation happened internally and almost completely hidden from developer.
5. Compilation is simply a translation step, and byte code is a lower-level, and platform-independent, representation of source code.
6. Each of the source statements is translated into a group of bytecode instructions. This bytecode translation is performed to speed execution. Bytecode can be run much quicker than the original source code statements.
7. The .pyc file, created in compilation step, is then executed by appropriate virtual machines.
8. The Virtual Machine iterates through bytecode instructions, one by one, to carry out their operations.
9. The Virtual Machine is the runtime engine of Python and it is always present as part of the Python system, and is the component that actually runs the Python scripts.
10. It is the last step of Python interpreter.

Following tools are the static analysis tools that help to find bugs in python :

1. **Pychecker :** Pychecker is an open source tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.
2. **Pylint :**
   a. Pylint is highly configurable and it acts like special programs to control warnings and errors, it is an extensive configuration file
   b. It is an open source tool for static code analysis and it looks for programming errors and is used for coding standard.
   c. It also integrates with Python IDEs such as Pycharm, Spyder, Eclipse, and Jupyter.

Python decorators :

1. Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class.

2. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.
3. In decorators, functions are taken as the argument into another function and then called inside the wrapper function.
4. **Syntax :**

```
@gfg_decorator
def hello_decorator():
    print("Gfg")
```

5. gfg_decorator is a callable function, will add some code on the top of some another callable function, hello_decorator function and return the wrapper function.

PART-2

Elements of Python, Type Conversion.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.5. **What do you mean by comments in Python ?**

Answer

Comments :

1. Python allows us to add comments in the code.
2. Comments are used by the programmer to explain the piece of code to be understood by other programmer in a simple language. Every programming language makes use of some character for commenting.
3. Python uses the hash character (#) for comments. Putting # before a text ensures that the text will not be parsed by the interpreter.
4. Comments do not affect the programming part and the Python interpreter does not display any error message for comments.

   For example : Commenting using hash mark (#)

```
>>> 8 + 9                    # addition
17                           # Output
>>>
```

   In this example, 'addition' is written with a hash mark. Hence the interpreter understands it as a comment and does not display any more messages.

Que 1.6. **Explain identifiers and keywords with example.**

Answer

1. A Python identifier is the name given to a variable, function, class, module or other object.

2. An identifier can begin with an alphabet (A – Z or a – z), or an underscore (_) and can include any number of letters, digits, or underscores and spaces are not allowed.
3. Python will not accept @, $ and % as identifiers.
4. Python is a case-sensitive language. Thus, Hello and hello both are different identifiers. In python, a class name will always start with a capital letter.

Table 1.6.1. Examples of Identifiers

Valid	Invalid
MyName	My Name (Space is not allowed)
My_Name	3dfig (cannot start with a digit)
Your_Name	Your#Name (Only alphabetic character, Underscore (_) and numeric are allowed)

Reserved keywords : Python has a list of reserved words known as keywords. Every keyword has a specific purpose and use.

Some of the reserved keywords in Python :

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

Que 1.7. Define variable. Also discuss variable initialization.

Answer

Variables :

1. A variable holds a value that may change.
2. The process of writing the variable name is called declaring the variable.
3. In Python, variables do not need to be declared explicitly in order to reserve memory spaces as in other programming languages like C, Java, etc.
4. When we initialize the variable in Python, Python Interpreter automatically does the declaration process.

Initializing a variable :

1. The general format of assignment statement is as follows :

Variable = Expression

2. The equal sign (=) is known as assignment operator.
3. An expression is any value, text or arithmetic expression where as variable is the name of variable.
4. The value of the expression will be stored in the variable.

Example of initializing a variable :

```
>>>year=2016
>>> name='Albert'
```

The two given statements reserve two memory spaces with variable names year and name. 2016 and Albert, are stored in these memory spaces.

Que 1.8. What do you mean by data types ? Explain numeric and string data type with example.

Answer

Data types :

i. The data stored in the memory can be of many types. For example, a person's name is stored as an alphabetic value and his address is stored as an alphanumeric value.

ii. Python has six basic data types which are as follows :

1. Numeric
2. String
3. List
4. Tuple
5. Dictionary
6. Boolean

Numeric :

1. Numeric data can be broadly divided into integers and real numbers (*i.e.*, fractional numbers). Integers can be positive or negative.
2. The real numbers or fractional numbers are called, floating point numbers in programming languages. Such floating point numbers contain a decimal and a fractional part.

For example :

```
>>> num1 = 2   # integer number
>>>num2 = 2.5   # real number (float)
>>>num1
2              # Output
>>>num2
2.5            # Output
>>>
```

String :

1. Single quotes or double quotes are used to represent strings.
2. A string in Python can be a series or a sequence of alphabets, numerals and special characters.

For example :

```
>>> sample_string = "Hello"    # store string value
>>> sample_string    # display string value
'Hello'              # Output
```

Que 1.9. Discuss list and tuple data types in detail.

Answer

List :

1. A list can contain the same type of items.
2. Alternatively, a list can also contain different types of items.
3. A list is an ordered and indexable sequence.
4. To declare a list in Python, we need to separate the items using commas and enclose them within square brackets ([ ]).
5. Operations such as concatenation, repetition and sub-list are done on list using plus (+), asterisk (*) and slicing (:) operator.

For example :

```
>>>first = [1, "two", 3.0, "four" ]    # 1st list
>>>second = ["five", 6]    # 2nd list
>>>first    # display 1st list
[1, 'two', 3.0, 'four']    # Output
```

Tuple :

1. A tuple is also used to store sequence of items.
2. Like a list, a tuple consists of items separated by commas.
3. Tuples are enclosed within parentheses rather than within square brackets.

For example :

```
>>>third = (7, "eight", 9, 10.0)
>>>third
(7, 'eight', 9, 10.0)    # Output
```

Que 1.10. Explain dictionary and Boolean data type.

Answer

Dictionary :

1. A Python dictionary is an unordered collection of key-value pairs.

2. When we have the large amount of data, the dictionary data type is used.
3. Keys and values can be of any type in a dictionary.
4. Items in dictionary are enclosed in the curly-braces {} and separated by the comma (,).
5. A colon (:) is used to separate key from value. A key inside the square bracket [ ] is used for accessing the dictionary items.

For example :

```
>>> dict1 = {1:"first line", "second" : 2}    # declare dictionary
>>> dict1[3] = "third line"    # add  new item
>>> dict1 # display dictionary
    {1 : 'first line', 'second' : 2, 3: 'third line'}    # Output
```

Boolean :

1. In a programming language, mostly data is stored in the form of alphanumeric but sometimes we need to store the data in the form of 'Yes' or 'No'.
2. In terms of programming language, Yes is similar to True and No is similar to False.
3. This True and False data is known as Boolean data and the data types which stores this Boolean data are known as Boolean data types.

For example :

```
>>> a = True
>>> type (a)
<type 'bool'>
```

Que 1.11. What do you mean by type conversion ?

Answer

1. The process of converting one data type into another data type is known as type conversion.
2. There are mainly two types of type conversion methods in Python :
   - **a. Implicit type conversion :**
     - i. When the data type conversion takes place during compilation or during the run time, then it called an implicit data type conversion.
     - ii. Python handles the implicit data type conversion, so we do not have to explicitly convert the data type into another data type.

For example :

```
a = 5
b = 5.5
sum = a + b
print (sum)
print (type (sum)) # type() is used to display the datatype of a variable
```

Output :

```
10.5
<class 'float'>
```

iii. In the given example, we have taken two variables of integer and float data types and added them.

iv. Further, we have declared another variable named 'sum' and stored the result of the addition in it.

v. When we checked the data type of the sum variable, we can see that the data type of the sum variable has been automatically converted into the float data type by the Python compiler. This is called implicit type conversion.

b. Explicit type conversion:

i Explicit type conversion is also known as type casting.

ii. Explicit type conversion takes place when the programmer clearly and explicitly defines the variables in the program.

For example :

```
# adding string and integer data types using explicit type conversion
a = 100
b = "200"
result1 = a + b
b = int(b)
result2 = a + b
print (result2)
```

Output :

```
Traceback (most recent call last):
File "", line 1, in
TypeError : unsupported operand type (s) for +: 'int' and 'str'
300
```

iii. In the given example, the variable *a* is of the number data type and variable *b* is of the string data type.

iv. When we try to add these two integers and store the value in a variable named result1, a TypeError occurs. So, in order to perform this operation, we have to use explicit type casting.

v. We have converted the variable *b* into integer type and then added variable *a* and *b*. The sum is stored in the variable named result2, and when printed it displays 300 as output.

PART-3

Basics : Expressions, Assignment Statement.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.12. **What is expression ?**

Answer

1. An expression is a combination of symbols that evaluates to a value.
2. An expression is a combination of variables, operators, values sub-expressions and a reserve keyword.
3. Whenever we type an expression in the command line, the interpreter evaluates it and produces the result.
4. Expressions that evaluate to a numeric type are called arithmetic expressions.
5. A sub-expression is any expression that is part of a larger expression. Sub-expressions are denoted by the use of parentheses.

   For example : 4 + (3 * *k*)

   An expression can also consist of a single literal or variable. Thus, 4, 3, and *k* are each expression.

   This expression has two sub-expressions, 4 and (3 * *k*). Sub-expression (3 * *k*) itself has two sub-expressions, 3 and *k*.

Que 1.13. **What do you mean by assignment statement?**

Answer

1. Assignment statements are used to create new variables, assign values, and change values.
2. **Syntax of assignment statement :**

   # LHS <=> RHS

   Variable = Expression

3. There are three types of assignment statements :
   i. Value-based expression on RHS
   ii. Current variable on RHS
   iii. Operation on RHS

Que 1.14. Discuss types of assignment statements with examples.

Answer

Types of assignment statement :

1. **Value-based expression on RHS :** In this type, Python allocates a new memory location to the newly assigned variable.
   For example :
   test1 = "Hello"
   id(test1)
   Output :
   2524751071304

2. **Current variable on RHS :** In this type, Python does not allocate a new memory location.
   For example :
   current_var = "It's Quantum Series"
   print(id(current_var))
   new_var = current_var
   print(id(new_var))
   Output :
   24751106240
   2524751106240

3. **Operation on RHS :** In this type, we have an operation on the RHS of the statement, which is the defining factor of the type of our statement.
   For example :
   test1 = 7 * 2 / 10
   type(test1)
   Output :
   float

PART-4

Arithmetic Operators.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.15. Define the term operator.

Answer

1. An operator is a symbol that represents an operation that may be performed on one or more operands.
2. Operators are constructs used to modify the values of operands.
3. Operators that take one operand are called unary operators.
4. Operators that take two operands are called binary operators.
5. Based on functionality operators are categories into following seven types :
   i. Arithmetic operators.
   ii. Assignment operators.
   iii. Bitwise operators.
   iv. Comparison operators.
   v. Identity operators.
   vi. Logical operators.
   vii. Membership operators.

Que 1.16. Discuss arithmetic and comparison operator with example.

Answer

Arithmetic operators : These operators are used to perform arithmetic operation such as addition, subtraction, multiplication and division.

Table 1.16.1. List of arithmetic operators.

Operator	Description	Example
+	Addition operator to add two operands.	10 + 20 = 30
–	Subtraction operator to subtract two operands.	10 – 20 = –10
×	Multiplication operator to multiply two operands.	10 × 20 = 200
/	Division operator to divide left hand by right hand operator.	5 / 2 = 2.5
**	Exponential operator to calculate power.	5 ** 2 = 25
%	Modulus operator to find remainder.	5 % 2 = 1
//	Floor division operator to find the quotient and remove the fractional part.	5 // 2 = 2

Comparison operators : These operators are used to compare values. It is also called relational operators. The result of these operator is always a Boolean value *i.e.*, true or false.

Table 1.16.2. List of comparison operators.

Operator	Description	Example
= =	Operator to check whether two operand are equal.	10 = = 20, false
! = or <>	Operator to check whether two operand are not equal.	10 ! = 20, true
>	Operator to check whether first operand is greater than second operand.	10 > 20, false
<	Operator to check whether first operand is smaller than second operand.	10 < 20, true
> =	Operator to check whether first operand is greater than or equal to second operand.	10 > = 20, false
< =	Operator to check whether first operand is smaller than or equal to second operand.	10 < = 20, true

Que 1.17. **Explain assignment operator with example.**

Answer

Assignment operators : This operator is used to store right side operands in the left side operand.

Table 1.17.1. List of assignment operators.

Operator	Description	Example
=	Store right side operand in left side operand.	$a = b + c$
+ =	Add right side operand to left side operand and store the result in left side operand	$a + = b$ or $a = a + b$
– =	Subtract right side operand to left side operand and store the result in left side operand	$a - = b$ or $a = a - b$
* =	Multiply right side operand with left side operand and store the result in left side operand	$a * = b$ or $a = a * b$
/ =	Divide left side operand by right side operand and store the result in left side operand	$a / = b$ or $a = a / b$
% =	Find the modulus and store the remainder in left side operand	$a \% = b$ or $a = a \% b$
* * =	Find the exponential and store the result in left side operand	$a ** = b$ or $a = a ** b$
/ / =	Find the floor division and store the result in left side operand	$a // = b$ or $a = a // b$

Que 1.18. **Define bitwise operator with example.**

Answer

Bitwise operators : These operators perform bit level operation on operands. Let us take two operand $x = 10$ and $y = 12$. In binary format this can be written as $x = 1010$ and $y = 1100$.

Table 1.18.1. List of bitwise operators.

Operator	Description	Example
& Bitwise AND	This operator performs AND operation between operands. Operator copies bit if it exists in both operands	x & y results 1000
| Bitwise OR	This operator performs OR operation between operands. Operator copies bit if it exists in either operand	x | y results 1 110
^ Bitwise XOR	This operator performs XOR operation between operands. Operator copies bit if it exists only in one operand.	x ^ y results 0110
~ bitwise inverse	This operator is a unary operator used to inverse the bits of operand.	~ x results 0101
<< left shift	This operator is used to shift the bits towards left	x << 2 results 101000
<<right shift	This operator is used to shift the bits towards right	x >> 2 results 0010

Que 1.19. **Discuss logical and identity operator with example.**

Answer

Logical operators : These operators are used to check two or more conditions. The resultant of this operator is always a Boolean value. Here x and y are two operands that store either true of false Boolean values.

Table 1.19.1. List of logical operators.

Operator	Description	Example
and logical AND	This operator perform AND operation between operands. When both operands are true, the resultant become true.	*x* and *y* results false
or logical OR	This operator perform OR operation between operands. When any operand is true, the resultant become true.	*x* and *y* results true
not logical NOT	This operator is used to reverse the operand state.	not *x* result false

Identity operators : These operator are used to check whether both operands are same or not. Suppose *x* stores a value 20 and *y* stores a value 40. Then *x* is *y* returns false and *x* not is *y* return true.

Table 1.19.2. List of identity operators.

Operator	Description	Example
is	Return true, if the operands are same. Return false, if the operands are not same.	*x* is *y*, results false
not is	Return false, if the operands are same. Return true, if the operands are not same.	*x* not is *y*, results true

Que 1.20. Explain membership operator with example.

Answer

Membership operators :

1. These operators are used to check an item or an element that is part of a string, a list or a tuple.
2. A membership operator reduces the effort of searching an element in the list.
3. Suppose *x* stores a value 20 and *y* is the list containing items 10, 20, 30, and 40. Then *x* is a part of the list *y* because the value 20 is in the list *y*.

Table 1.20.1. List of Membership operators.

Operator	Description	Example
in	Return true, if item is in list or in sequence. Return false, if item is not in list or in sequence.	x in y, results true
not in	Return false, if item is in list or in sequence. Return true, if item is not in list or in sequence.	x not in y, results true

PART-5

Operator Precedence, Boolean Expression.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.21. What do you mean by operator precedence ?

Answer

1. When an expression has two or more operator, we need to identify the correct sequence to evaluate these operators. This is because result of the expression changes depending on the precedence.

   For example : Consider a mathematical expression :

   10 + 5 / 5

   When the given expression is evaluated left to right, the final answer becomes 3.

2. However, if the expression is evaluated right to left, the final answer becomes 11. This shows that changing the sequence in which the operators are evaluated in the given expression also changes the solution.

3. Precedence is the condition that specifies the importance of each operator relative to the other.

Table 1.21.1. Operator precedence from lower precedence to higher.

Operator	Description
NOT, OR AND	Logical operators
in , not in	Membership operator
is, not is	Identity operator
=, %=, /=, //=, –=, +=, *=, **==	Assignment operators.
<>, ==, !=	Equality comparison operator
<=, <, >, >=	Comparison operators
^, |	Bitwise XOR and OR operator
&	Bitwise AND operator
<<, >>	Bitwise left shift and right shift
+, –	Addition and subtraction
*, /, %, ??	Multiplication, Division, Modulus and floor division
**	Exponential operator

Que 1.22. Explain operator associativity.

Answer

Associativity :

1. Associativity decides the order in which the operators with same precedence are executed.
2. There are two types of associativity :
   a. **Left to right :** In left to right associativity, the operators of same precedence are executed from the left side first.
   b. **Right to left :** In right to left associativity, the operators of same precedence are executed from the right side first.
3. Most of the operators in Python have left to right associativity.
4. Left to right associative operators are multiplication, floor division, etc. and ** operator is right to left associative.
5. When two operators have the same precedence then operators are evaluated from left to right direction.

For example :

```
>>> 3 * 4 / / 6
2   # Output
>>> 3 * (4 / / 6)
```

```
0   # Output
>>> 3 * * 4 * * 2   # 3 ^ 16
43046721   # Output
>>> (3 * * 4) * * 2   # 81 ^ 2
6561   # Output
```

Que 1.23. **What do you mean by Boolean expression ?**

OR

Write short notes with example : The programming cycle for Python, elements of Python, type conversion in Python, operator precedence, and Boolean expression. **AKTU 2019-20, Marks 10**

Answer

Programming cycle for Python : Refer Q. 1.1, Page 1–2T, Unit-1.
Elements of Python : Refer Q. 1.8, Page 1–8T, Refer Q. 1.9, Page 1–9T, and Refer Q. 1.10, Page 1–9T; Unit-1.
Type conversion in Python : Refer Q. 1.11, Page 1–10T, Unit-1.
Operator precedence : Refer Q. 1.21, Page 1–18T, Unit-1.
Boolean expression : A boolean expression may have only one of two values : True or False.
For example : In the given example comparison operator (==) is used which compares two operands and prints true if they are equal otherwise print false :

```
>>> 5 == 5
True   # Output
>>> 5 == 6
False   # Output
```

Que 1.24. **How memory is managed in Python? Explain PEP 8. Write a Python program to print even length words in a string.**

AKTU 2019-20, Marks 10

Answer

Memory management :

1. Memory management in Python involves a private heap containing all Python objects and data structures.
2. The management of this private heap is ensured internally by the Python memory manager.
3. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.
4. At the lowest level, a raw memory allocator ensures that there is enough room in the private heap for storing all Python-related data by interacting with the memory manager of the operating system.

5. On top of the raw memory allocator, several object-specific allocators operate on the same heap and implement distinct memory management policies adapted to the peculiarities of every object type.
6. For example, integer objects are managed differently within the heap than strings, tuples or dictionaries because integers imply different storage requirements and speed/space tradeoffs.
7. Python memory manager thus delegates some of the work to the object-specific allocators, but ensures that the latter operate within the bounds of the private heap.

PEP 8 :

1. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment.
2. The PEP should provide a concise technical specification of the feature.
3. PEP is actually an acronym that stands for Python Enhancement Proposal.
4. PEP 8 is Python's style guide. It is a set of rules for how to format the Python code to maximize its readability.
5. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment.

Program to print even length words in a string :

```
def printWords(s) :
# split the string
    s = s.split(' ')
    # iterate in words of string
    for word in s:
    # if length is even
    if len(word)%2==0:
        print(word)
# Driver Code
s = "i am muskan"
printWords(s)
```

Output :

```
am
muskan
```

Que 1.25. **What will be the output after the following statements ?**

```
x = 6
y = 3
print(x / y)
```

Answer

2.0

Que 1.26. **What will be the output after the following statements ?**

```
x = 8
y = 2
print(x // y)
```

Answer

4

Que 1.27. **What will be the output after the following statements ?**

```
x = 5
y = 4
print(x % y)
```

Answer

1

Que 1.28. **What will be the output after the following statements ?**

```
x = 3
y = 2
x += y
print(x)
```

Answer

5

Que 1.29. **What will be the output after the following statements ?**

```
x = 5
y = 7
x *= y
print(x)
```

Answer

35

Que 1.30. **What will be the output after the following statements ?**

```
x = 30
```

```
y = 7
x %= y
print(x)
```

Answer

2

Que 1.31. **What will be the output after the following statements ?**

```
x = 3
y = 7
print(x == y)
```

Answer

False

Que 1.32. **What will be the output after the following statements ?**

```
x = 8
y = 6
print(x != y)
```

Answer

True

Que 1.33. **What will be the output after the following statements ?**

```
x = 83
y = 57
print(x > y)
```

Answer

True

Que 1.34. **What will be the output after the following statements ?**

```
x = 72
y = 64
print(x < y)
```

Answer

False

Que 1.35. **What will be the output after the following statements ?**

```
x = True
```

y = False
print(x and y)

Answer

False

Que 1.36. **What will be the output after the following statements ?**
x = True
y = False
print(x or y)

Answer

True

Que 1.37. **What will be the output after the following statements ?**
x = True
y = False
print(not x)

Answer

False

Que 1.38. **What will be the output after the following statements ?**
x = True
y = False
print(not y)

Answer

True

Que 1.39. **What will be the output after the following statements ?**
x = 20
y = 40
z = y if (y > x) else x
print(z)

Answer

40

Que 1.40. **What will be the output after the following statements ?**
x = 50

```
y = 10
z = y if (y > x) else x
print(z)
```

Answer

50

Que 1.41. **What will be the output after the following statements ?**

```
x = 65
y = 53
z = y if (x % 2 == 0) else x
print(z)
```

Answer

65

Que 1.42. **What will be the output after the following statements ?**

```
x = 46
y = 98
z = y if (y % 2 == 0) else x
print(z)
```

Answer

98

Que 1.43. **What will be the output after the following statements ?**

```
x = 2 * 4 + 7
print(x)
```

Answer

15

Que 1.44. **What will be the output after the following statements ?**

```
x = 7 * (4 + 5)
print(x)
```

Answer

63

Que 1.45. **What will be the output after the following statements ?**

```
x = '24' + '16'
```

print(x)

Answer

2416

Que 1.46. **What will be the output after the following statements ?**

```
x = 15 + 35
print(x)
```

Answer

50

Que 1.47. **What will be the data type of *x* after the following statement if input entered is 18 ?**

```
x = input('Enter a number:')
```

Answer

String

Que 1.48. **What will be the data type of *y* after the following statements if input entered is 50 ?**

```
x = input('Enter a number:')
y = int(x)
```

Answer

Integer

Que 1.49. **What will be the data type of *y* after the following statements ?**

```
x = 71
y = float(x)
```

Answer

Float

Que 1.50. **What will be the data type of *y* after the following statements ?**

```
x = 48
y = str(x)
```

Answer

String

Que 1.51. **What will be the output after the following statements ?**

```
x = y = z = 8
print(y)
```

Answer

8

Que 1.52. **What will be the value of *x*, *y* and *z* after the following statement ?**

```
x = y = z = 300
```

Answer

All three will have the value of 300

Que 1.53. **What will be the value of *x*, *y* and *z* after the following statement ?**

```
x, y, z = 3, 4, 5
```

Answer

x will have the value of 3, *y* will have the value 4 and *z* will have the value of 5.

Que 1.54. **In the order of precedence, which of the operation will be completed last in the following statement ?**

```
3 * 6 + 5 – 4 / 2
```

Answer

Subtraction

Que 1.55. **What will be the order of precedence of operations in the following statement ?**

```
10 * 4 – 1 + 8 / 5
```

Answer

Multiplication, Division, Addition, Subtraction

Que 1.56. **What will be the data type of *x* after the following statement if input entered is 64 ?**

```
x = float(input('Enter a number:'))
```

Answer

Float

Que 1.57. **What will be the output after the following statements ?**

```
a = 27 / 3 % 2 * 4**2
print(a)
```

Answer

16.0

Que 1.58. **What will be the output after the following statements ?**

```
a = 3 / 3 + 4 * 7 – 3 * 3
print(a)
```

Answer

20.0

2 UNIT

Conditionals and Loops

CONTENTS

Part-1 : Conditional Statement in Python (if-else statement, its working and execution) ..... **2–2T to 2–4T**

Part-2 : Nested-if Statement and Elif Statement in Python ..... **2–4T to 2–9T**

Part-3 : Expression Evaluation and Float Representation ..... **2–9T to 2–11T**

Part-4 : Purpose and Working of Loops, While Loop Including its Working ..... **2–11T to 2–14T**

Part-5 : For Loop, Nested Loop, Break and Continue Statements ..... **2–14T to 2–23T**

PART- 1

Conditional Statement in Python (if-else Statement, its Working and Execution).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. What are conditional statements in Python ?

Answer

1. Conditional statements help in making a decision based on certain conditions.
2. These conditions are specified by a set of conditional statements having Boolean expressions which are evaluated to true or false.
3. Conditional statements are also known as decision-making statements.
4. Python supports conditional execution using if-else statements.
5. In Python, we use different types of conditional statements:
   a. If statement
   b. If-else statement
   c. Nested-if statement
   d. Elif statement

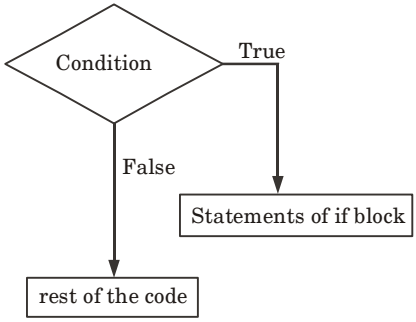
Que 2.2. Explain if statement with the help of an example.

Answer

1. An if statement consists of a Boolean expression followed by one or more statements.
2. With an if clause, a condition is provided; if the condition is true then the block of statement written in the if clause will be executed, otherwise not.
3. **Syntax :**

```
If (Boolean expression) : Block of code #Set of statements to execute if
                                        the condition is true
```

4. Flow chart :



For example :

```
var = 100
if ( var == 100 ) : print "value of expression is 100"
print "Good bye !"
```

Output :

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

Que 2.3. What do you mean by if-else statement ? Explain with the help of example.

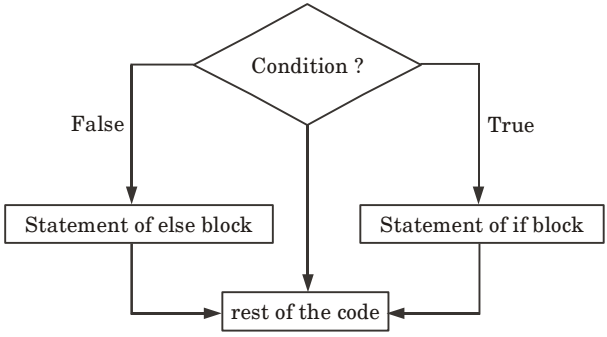
Answer

1. An if statement can be followed by an optional else statement, which executes when the Boolean expression is False.
2. The else condition is used when we have to judge one statement on the basis of other.
3. **Syntax :**

```
If (Boolean expression): Block of code  #Set of statements to execute if
                                          condition is true
else : Block of code #Set of statements to execute if condition is false
```

4. **Working and execution :**
   a. The condition will be evaluated to a Boolean expression (true or false).
   b. If the condition is true then the statements or program present inside the if block will be executed
   c. If the condition is false then the statements or program present inside else block will be executed.

5. Flow chart :



For example :

```
num = 5
if (num > 10) :
    print ("Number is greater than 10")
else :
    print ("Number is less than 10")
print ("This statement will always be executed")
```

Output :

Number is less than 10.

PART-2

Nested-if Statement and Elif Statement in Python.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.4. Discuss the Nested-if statement with the help of example.

Answer

1. Nested-if statements are nested inside other if statements. That is, a nested-if statement is the body of another if statement.
2. We use nested if statements when we need to check secondary conditions only if the fist condition executes as true.
3. **Syntax :**

```
if test expression 1 :
# executes when condition 1 is true
 body of if statement
if test expression 2 :
```

```
# executes when condition 2 is true
Body of nested-if
else :
body of nested-if :
else :
body of if-else statement
```

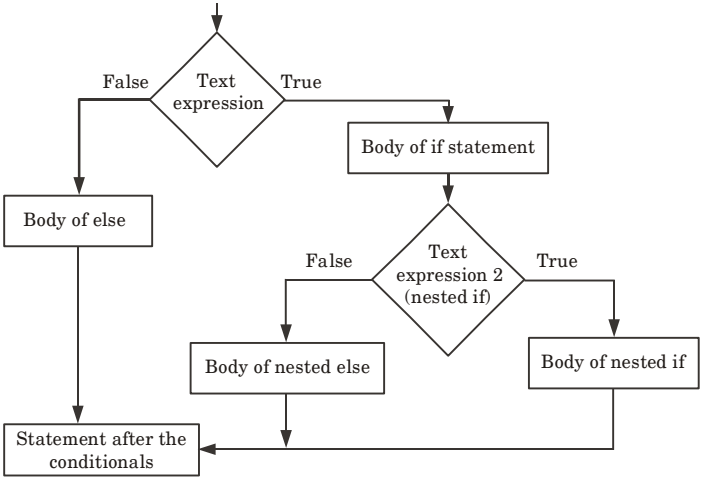
For example :

```
a = 20
if (a == 20) :
# First if statement
    if (a < 25) :
        print ("a is smaller than 25")
    else :
        print ("a is greater than 25")
else :
        print ("a is not equal to 20")
```

Output :

a is smaller than 25

4. Flow chart :



Que 2.5. **Explain elif statement in Python.**

OR

Explain all the conditional statement in Python using small code example. **AKTU 2019-20, Marks 10**

Answer

Different types of conditional statement are :

1. If statement : Refer Q. 2.2, Page 2–2T, Unit-2.

2. If else statement : Refer Q. 2.3, Page 2–3T, Unit-2.

3. Nested-if statement : Refer Q. 2.4, Page 2–4T, Unit-2.

4. Elif statement :

a. Elif stands for else if in Python.

b. We use elif statements when we need to check multiple conditions only if the given if condition executes as false.

c. Working and execution :

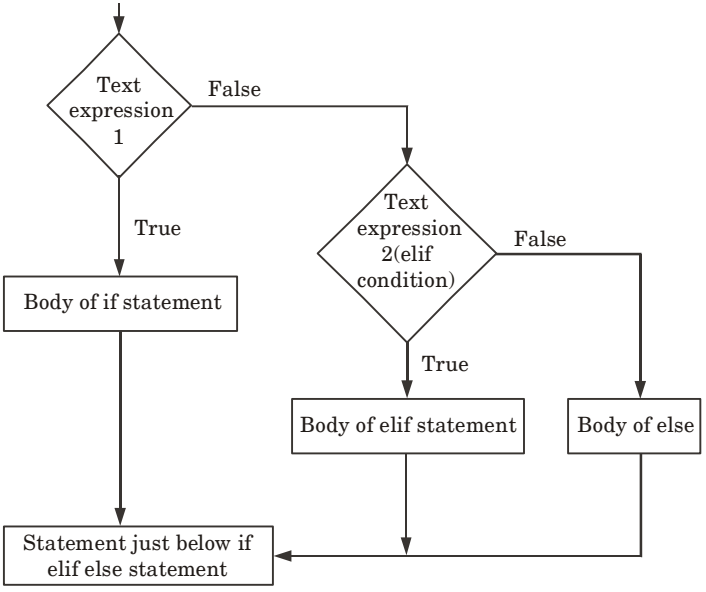
i. If the first if condition is true, the program will execute the body of the if statement. Otherwise, the program will go to the elif block (else if in Python) which basically checks for another if statement.

ii. Again, if the condition is true, the program will execute the body of the elif statement, and if the condition is found to be false, the program will go to the next else block and execute the body of the else block.

d. Syntax :

```
if test expression :
Body of if
elif test expression :
Body of elif
else :
Body of else
```

e. Flow chart :



For example :

```
a = 50
if (a == 29) :
        print ("value of variable a is 20")
    elif (a == 30) :
        print ("value of variable a is 30")
    elif (a == 40) :
        print ("value of variable a is 40")
else :
    print ("value of variable a is greater than 40")
```

Output :

value of variable a is greater than 40

Que 2.6. **Write a program to find whether a number is even or odd.**

Answer

```
>>> number = int (input ("Enter an integer number :"))
>>> if (number % 2) == 0 :
    print "Number is even"
else :
    print "Number is odd"
Enter an integer number : 6
Number is even                                  #Output
```

Que 2.7. **Write a program to check the largest among the given three numbers.**

Answer

```
>>> x = int (input ("Enter the first number :"))
Enter the first number : 14
>>> y = int (input ("Enter the second number :"))
Enter the second number : 21
>>> z = int (input ("Enter the third number :"))
Enter the third number : 10
>>> if (x > y) and (x > z) :
    1 = x
elif (y > x) and (y > z) :
    1 = y
else :
    1 = z
```

```
>>> print "The largest among the three is ", 1
The largest among the three is 21                    #Output
```

Que 2.8. **Write a Python program to check if the input year is a leap year or not.**

Answer

```
>>> year = int (input ("Enter year :"))
>>> if (year % 4) == 0 :
        if (year % 100) == 0 :
            if (year % 400) == 0 :
                print (year, 'is leap year')
            else :
                print (year, 'is not leap year')
        else :
            print (year, 'is leap year')
    else :
        print (year, 'is leap year')
```

Output :

Enter a year : 2016

2016 is leap year

Output :

Enter a year : 1985

1985 is not leap year

Que 2.9. **Write a Python program to display the Fibonacci sequence for *n* terms.**

Answer

```
>>> number = int (input ("Enter the value for x (where x > 2) ?"))
# first the terms
>>> x1 = 0
>>> x2 = 1
>>> count = 2
# check if the number of terms is valid
>>> if numbers < = 0 :
    print ("Please enter positive integer")
elif numbers == 1 :
    print ("Fibonacci sequence is :")
```

```
    print (x1)
else :
    print ("Fibonacci sequence is :")
    print (x1, ",", x2)
    while count < numbers :
        xth = x1 + x2
        print (xth)
        # update values
        x1 = x2
        x2 = xth
        count + = 1
```

Output :

Enter the value for n (where n > 2) ? 10

Fibonacci sequence :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

PART-3

Expression Evaluation and Float Representation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.10. **What do you mean by expression evaluation ?**

Answer

1. In Python actions are performed in two forms :
   a. Expression evaluation,
   b. Statement execution.
2. The key difference between these two forms is that expression evaluation returns a value whereas statement execution does not return any value.
3. A Python program contains one or more statements. A statement contains zero or more expressions.
4. Python executes a statement by evaluating its expressions to values one by one.

5. Python evaluates an expression by evaluating the sub-expressions and substituting their values.

   For example :

   ```
   >>> program = "Hello Python"
   >>> program
   'Hello Python'                    #Output
   >>> print program
   Hello Python                      #Output
   ```

6. An expression is not always a mathematical expression in Python. A value by itself is a simple expression, and so is a variable.
7. In the given example, we assigned a value "Hello Python" to the variable program. Now, when we type only program, we get the output 'Hello Python'. This is the term we typed when we assigned a value to the variable. When we use a print statement with program it gives the value of the variable *i.e.*, the value after removing quotes.

Que 2.11. Discuss float representation in Python.

Answer

1. Floating point representations vary from machine to machine.
2. The float type in Python represents the floating-point number.
3. Float is used to represent real numbers and is written with a decimal point dividing the integer and fractional parts.
4. For example: 97.98, 32.3 + e18, – 32.54e100 all are floating point numbers.
5. Python float values are represented as 64-bit double-precision values.
6. The maximum value any floating-point number can be is approx 1.8×10^{308}.
7. Any number greater than this will be indicated by the string inf in Python.
8. Floating-point numbers are represented in computer hardware as base 2 (binary) fractions.
9. For example, the decimal fraction 0.125 has value 1/10 + 2/100 + 5/1000, and in the same way the binary fraction 0.001 has value 0/2 + 0/4 + 1/8.

   For example : # Python code to demonstrate float values.

   ```
   Print(1.7e308)
   # greater than 1.8 * 10^308
   # will print 'inf'
   print(1.82e308)
   ```

   Output :

   ```
   1.7e+308
   inf
   ```

Que 2.12. **Explain expression evaluation and float representation with example. Write a Python program for how to check if a given number is Fibonacci number.** **AKTU 2019-20, Marks 10**

Answer

Expression evaluation : Refer Q. 2.10, Page 2–9T, Unit-2.

Float representation : Refer Q. 2.11, Page 2–10T, Unit-2.

Program :

```
import math
# A utility function that returns true if x is perfect square
def isPerfectSquare(x):
    s = int(math.sqrt(x))
    return s*s == x
# Returns true if n is a Fibonacci number, else false
def isFibonacci(n):
    return isPerfectSquare(5*n*n + 4) or isPerfectSquare(5*n*n – 4)
# A utility function to test above functions
for i in range(1,6):
    if (isFibonacci(i) == True):
        print i,“is a Fibonacci Number”
    else:
        print i,“is a not Fibonacci Number”
```

Output :

```
1 is a Fibonacci Number
2 is a Fibonacci Number
3 is a Fibonacci Number
4 is a not Fibonacci Number
5 is a Fibonacci Number
```

PART-4

Purpose and Working of Loops, While Loop Including its Working.

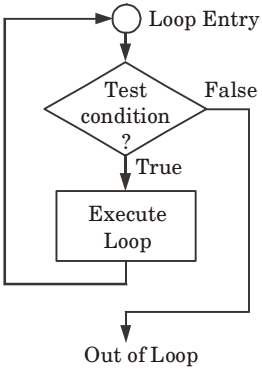
Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.13. **Define loop. Also, discuss the purpose and working of loops.**

Answer

1. A loop is a programming structure that repeats a sequence of instructions until a specific condition is met.
2. A loop statement allows us to execute a statement or group of statements multiple times.
3. Python programming language provides following types of loops to handle looping requirements:
   a. For
   b. While
   c. Nested
4. **Purpose :** The purpose of loops is to repeat the same, or similar, code a number of times. This number of times could be specified to a certain number, or the number of times could be dictated by a certain condition being met.
5. **Working :** Consider the flow chart for a loop execution :



   a. In the flow chart if the test condition is true, then the loop is executed, and if it is false then the execution breaks out of the loop.
   b. After the loop is successfully executed the execution again starts from the loop entry and again checks for the test condition, and this keeps on repeating until the condition is false.

Que 2.14. Discuss while loop in brief.

Answer

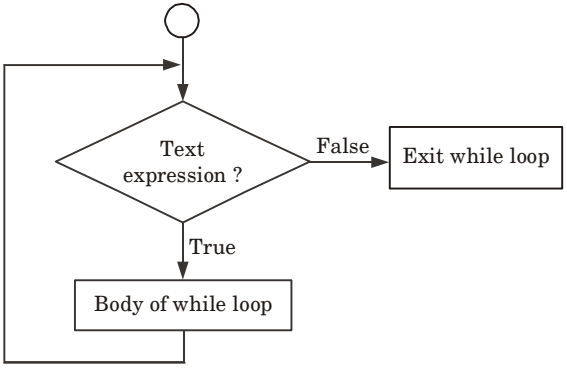
1. While loop statements in Python are used to repeatedly execute a certain statement as long as the condition provided in the while loop statement is true.

2. While loops let the program control to iterate over a block of code.

3. Syntax :

```
while test_expression:
    body of while
```

4. Flow chart :



5. Working :

a. The program first evaluates the while loop condition.

b. If it is true, then the program enters the loop and executes the body of the while loop.

c. It continues to execute the body of the while loop as long as the condition is true.

d. When it is false, the program comes out of the loop and stops repeating the body of the while loop.

For example :

```
>>>count = 0
while (count < 9) :
    print 'The count is :', count
    count = count + 1
```

Output :

```
The count is : 0
The count is : 1
The count is : 2
The count is : 3
The count is : 4
The count is : 5
```

The count is : 6
The count is : 7
The count is : 8

PART-5

For Loop, Nested Loop, Break and Continue Statements.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.15. Explain for loop with the help of example.

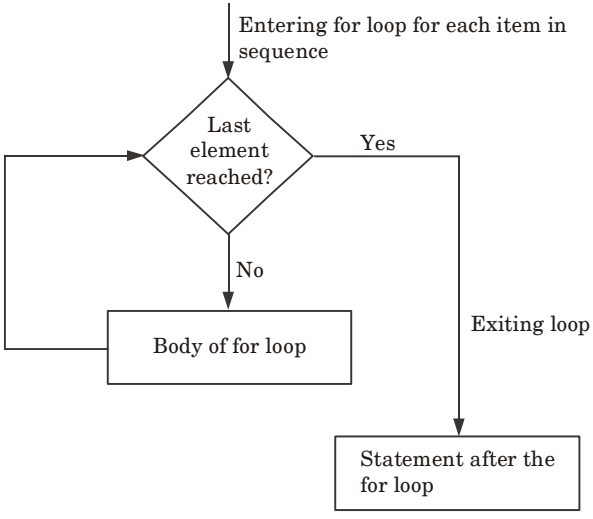
Answer

1. For loop in python is used to execute a block of statements or code several times until the given condition becomes false.
2. We use for loop when we know the number of times to iterate.
3. **Syntax :**

   for variable in sequence:

   Body of for loop

4. **Flow chart :**



For example :

```
i = 1
for i in range(1, 8) :
    print 2*i
```

Output :

2
4
6
8
10
12
14

Que 2.16. What is nested loop ? Explain.

Answer

1. Loop defined within another loop is known as nested loops.
2. Nested loops are the loops that are nested inside an existing loop, that is, nested loops are the body of another loop.
3. **Syntax :**

```
for condition1 :
    for condition2 :
        Body of for loop
```

4. **For example :**

```
for i in range(1,9,2) :
    for j in range(i) :
        print( i, end = ' ')
    print()
```

Output :

1
3 3 3
5 5 5 5 5
7 7 7 7 7 7 7

Que 2.17. What is break statement in Python ?

Answer

1. The break keyword terminates the loop and transfers the control to the end of the loop.
2. While loops, for loops can also be prematurely terminated using the break statement.
3. The break statement exits from the loop and transfers the execution from the loop to the statement that is immediately following the loop.

For example :

```
>>> count = 2
>>> while True :
        print count
        count = count + 2
        if count > = 12 :
            break    # breaks the loop
```

Output :

```
2
4
6
8
10
```

Que 2.18. Explain continue statement with example.

Answer

1. The continue statement causes execution to immediately continue at the start of the loop, it skips the execution of the remaining body part of the loop.
2. The continue keyword terminates the ongoing iteration and transfers the control to the top of the loop and the loop condition is evaluated again. If the condition is true, then the next iteration takes place.
3. Just as with while loops, the continue statement can also be used in Python for loops

For example :

```
>>> for i in range (1, 10) :
        if i % 2! = 0 :
            continue # if condition becomes true, it skips the print part
            print i
```

Output :

2

4

6

8

Que 2.19. **Explain the purpose and working of loops. Discuss break and continue with example. Write a Python program to convert time from 12 hour to 24-hour format.** **AKTU 2019-20, Marks 10**

Answer

Purpose and working of loops : Refer Q. 2.13, Page 2–11T, Unit-2.
Break statement : Refer Q. 2.17, Page 2–15T, Unit-2.
Continue statement : Refer Q. 2.18, Page 2–16T, Unit-2.
Program :

```
# Function to convert the date format
def convert24(str1):
# Checking if last two elements of time # is AM and first two elements are
12
    if str1[– 2:] == “AM” and str1[:2] == “12”:
    return “00” + str1[2:– 2]
# remove the AM
    elif str1[-2:] == “AM”:
        return str1[:– 2]
# Checking if last two elements of time is PM and first two elements are 12
    elif str1[– 2:] == “PM” and str1[:2] == “12”:
        return str1[:– 2]
    else:
# add 12 to hours and remove PM
        return str(int(str1[:2]) + 12) + str1[2:8]
# Driver Code
print(convert24(“08:05:45 PM”))
```

Que 2.20. **Write a program to demonstrate while loop with else.**

Answer

```
>>> count = 0
>>> while count < 3 :
        print (“Inside the while loop”)
        print (count)
        counter = count + 1
else :
```

```
        print ("Inside the else statement")
```

Output :

```
Inside the while loop
0
Inside the while loop
1
Inside the while loop
2
Inside the else
```

Que 2.21. Write a python program to print the numbers for a user provided range.

Answer

```
# Pythons program to print the prime numbers for a user provided range
# input range is provided from the user
>>> low = int(input("Enter lower range : "))
>>> up = int (input("Enter upper range : "))
>>> for n in range (low, up + 1) :
if n > 1 :
    for i in range (2, n) :
        if (n % i) == 0 :
        break
    else :
        print (n)
```

Output :

```
Enter lower range : 100
Enter upper range :
103
107
109
113
127
131
137
139
149
151
```

157
163
167
173

Que 2.22. **Differentiate between for and while loop.**

Answer

Properties	For	While
Format	Initialization, condition checking, iteration statement are written at the the top of the loop.	Only initialization and condition checking is done at top of the loop.
Use	The 'for' loop is used only when we already knew the number of iterations.	The 'while' loop is used only when the number of iteration are not exactly known.
Condition	If the condition is not given in 'for' loop, then loop iterates infinite times.	If the condition is not given in 'while' loop, it provides compilation error.
Initialization	In 'for' loop the initialization once done is never repeated.	In while loop if initialization is done during condition checking, then initialization is done each time the loop iterate.

Que 2.23. **What will be the output after the following statements ?**

```
x, y = 0, 1
while y < 10:
    print(y, end=' ')
    x, y = y, x + y
```

Answer

1 1 2 3 5 8

Que 2.24. **What will be the output after the following statements ?**

```
x = 1
while x < 4:
    x += 1
    y = 1
    while y < 3:
        print(y, end=' ')
```

```
        y += 1
```

Answer

1 2 1 2 1 2

Que 2.25. **What will be the output after the following statements ?**

```
x = 70
if x <= 30 or x >= 100:
    print('true')
elif x <= 50 and x == 50:
    print('not true')
elif x >= 150 or x <= 75:
    print('false')
else:
    print('not false')
```

Answer

false

Que 2.26. **What will be the output after the following statements ?**

```
x = 40
y = 25
if x + y >= 100:
    print('true')
elif x + y == 50:
    print('not true')
elif x + y <= 90:
    print('false')
else:
    print('not false')
```

Answer

false

Que 2.27. **What will be the output after the following statements ?**

```
x, y = 2, 5
while y – x < 5:
    print(x*y, end=' ')
    x += 3
    y += 4
```

Answer

10 45

Que 2.28. **What will be the output after the following statements ?**

```
for i in range(1, 25, 5):
```

```
    print(i, end=' ')
```

Answer

1 6 11 16 21

Que 2.29. **What will be the output after the following statements ?**

```
x = ['P', 'y', 't', 'h', 'o', 'n']
for i in x:
    print(i, end=' ')
```

Answer

Python

Que 2.30. **What will be the output after the following statements ?**

```
x = ['P', 'y', 't', 'h', 'o', 'n']
for i in enumerate(x):
    print(i, end=' ')
```

Answer

(0, 'P')(1, 'y')(2, 't')(3, 'h')(4, 'o')(5, 'n')

Que 2.31. **What will be the output after the following statements ?**

```
for i in range(1, 5):
    if i == 3:
        continue
    print(i, end=' ')
```

Answer

1 2 4

Que 2.32. **What will be the output after the following statements ?**

```
x = 15
if x > 15:
    print(0)
elif x == 15:
    print(1)
else:
    print(2)
```

Answer

1

Que 2.33. **What will be the output after the following statements ?**

```
x = 5
if x > 15:
```

```
    print('yes')
elif x == 15:
    print('equal')
else:
    print('no')
```

Answer

no

Que 2.34. **What will be the output after the following statements ?**

```
x = 50
if x > 10 and x < 15:
    print('true')
elif x > 15 and x < 25:
    print('not true')
elif x > 25 and x < 35:
    print('false')
else:
    print('not false')
```

Answer

not false

Que 2.35. **What will be the output after the following statements ?**

```
x = 25
if x > 10 and x < 15:
    print('true')
elif x > 15 and x < 25:
    print('not true')
elif x > 25 and x < 35:
    print('false')
else:
    print('not false')
```

Answer

not false

Que 2.36. **What will be the output after the following statements ?**

```
x = 15
if x > 10 and x <= 15:
    print('true')
elif x > 15 and x < 25:
```

```
    print('not true')
elif x > 25 and x < 35:
    print('false')
else:
    print('not false')
```

Answer

true

3
UNIT

Functions and Strings

CONTENTS

Part-1 : Function : Parts of a Function, Execution of a Function, Keyword and Default Arguments, Scope Rules ..... **3–2T to 3–7T**

Part-2 : Strings : Length of the String and Perform Concatenation and Repeat Operations in It, Indexing and Slicing of Strings ..... **3–8T to 3–10T**

Part-3 : Pythons Data Structure : Tuples, Unpacking Sequences, Lists ..... **3–11T to 3–19T**

Part-4 : Mutable Sequences, List Comprehension, Sets and Dictionaries ..... **3–19T to 3–26T**

Part-5 : Higher Order Functions : Treat Functions as First Class Objects, Lambda Expressions ..... **3–26T to 3–31T**

PART- 1

Function : Parts of a Function, Execution of a Function, Keyword and Default Arguments, Scope Rules.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. Define function and write its advantages.

Answer

1. Functions are self-contained programs that perform some particular tasks.
2. Once a function is created by the programmer for a specific task, this function can be called anytime to perform that task.
3. Each function is given a name, using which we call it. A function may or may not return a value.
4. There are many built-in functions provided by Python such as dir (), len ( ), abs ( ), etc.
5. Users can also build their own functions, which are called user-defined functions.

Advantages of using functions :

1. They reduce duplication of code in a program.
2. They break the large complex problems into small parts.
3. They help in improving the clarity of code (*i.e.*, make the code easy to understand).
4. A piece of code can be reused as many times as we want with the help of functions.

Que 3.2. How to define and call function in Python ? Explain different parts of a function.

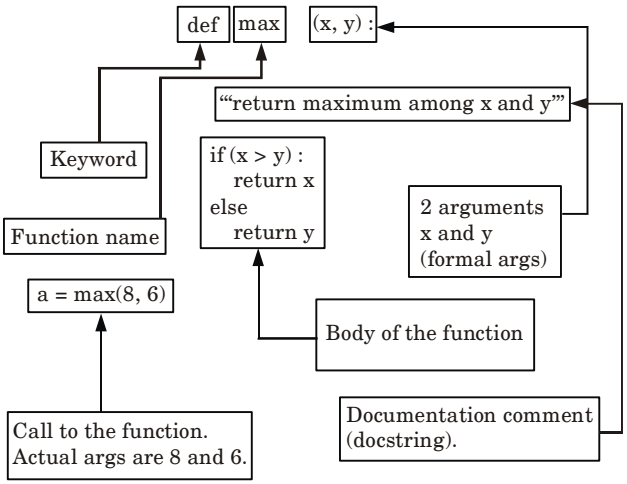
Answer

Function is defined by "def" keyword following by function name and parentheses.

Syntax of function definition : def function_name ( ) :

Syntax of functional call : function_name ( )

For example :



1. **Keyword :** The keyword 'def' is used to define a function header.
2. **Function name :** We define the function name for identification or to uniquely identify the function. In the given example, the function name is max. Function naming follows the same rules of writing identifiers in Python.
3. A colon (:) to mark the end of function header.
4. **Arguments :** Arguments are the values passed to the functions between parentheses. In the given example, two arguments are used, x and y. These are called formal arguments.
5. **Body of the function :** The body processes the arguments to do something useful. In the given example, body of the function is intended w.r.t. the def keyword.
6. **Documentation comment (docstring) :** A documentation string (docstring) to describe what the function does. In the given example, "return maximum among x and y" is the docstring.
7. An optional return statement to return a value from the function.
8. **Function call :** To execute a function, we have to call it. In the given example, a = max (8, 6) is calling function with 8 and 6 as arguments.

Que 3.3. **Discuss the execution of a function with the help of example.**

Answer

Let consider an example to understand the execution of function :

Step 1 : When function without "return" :

a. For example, we want the square of 4, and it should give answer "16" when the code is executed.

b. Which it gives when we simply use "print x*x" code, but when we call function "print square" it gives "None" as an output.

c. This is because when we call the function, recursion does not happen and leads to end of the function.

d. Python returns "None" for failing off the end of the function.

For example :

```
def square(x) :
    print(x*x)
print(square(4))
```

Output :

```
16
None
```

Step 2 : When we retrieve the output using "return" command :

When we use the "return" function and execute the code, it will give the output "16".

For example :

```
def square(x) :
    return(x*x)
print(square(4))
```

Output :

```
16
```

Step 3 : When function is treated as an object :

a. Functions in Python are themselves an object, and an object has some value.

b. When we run the command "print square" it returns the value of the object.

c. Since we have not passed any argument, we do not have any specific function to run hence it returns a default value (0x021B2D30) which is the location of the object.

For example :

```
def square(x) :
    return(x*x)
```

print(square)

Output :

<function square at 0x021B2D30>

Que 3.4. What do you mean by keyword and default arguments?

Answer

Keyword arguments :

1. Keyword arguments are related to the function calls.
2. When we use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
3. This allows us to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

For example :

```
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;
# Now you can call printme function
printme( str = 'My string')
```

Output :

My string

Default arguments :

1. A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

For example :

```
# Function definition is here
def printinfo( name, age = 35 ):
    "This prints a passed info into this function"
    print "Name:", name
    print "Age", age
    return ;
# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
printinfo( name = "miki")
```

Output :

Name : miki

Age 50

Name : miki

Age 35

Que 3.5. Discuss the scope rules in Python.

Answer

1. Scope of a name is the part of the program in which the name can be used.
2. Two variables can have the same name only if they are declared in separate scopes.
3. A variable cannot be used outside its scopes.
4. Fig. 3.5.1 illustrates Python's four scopes.

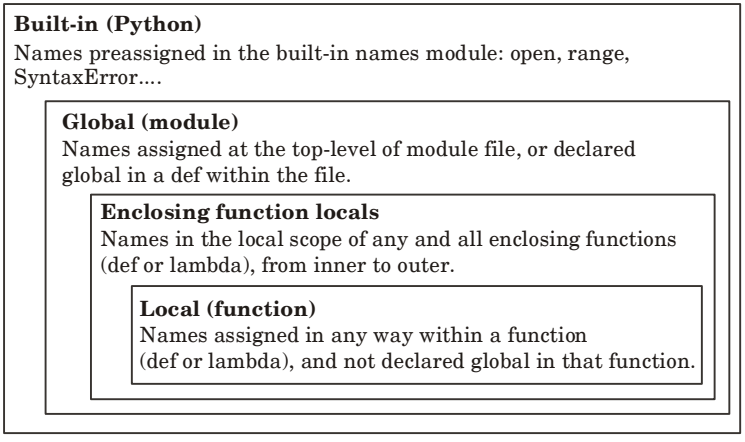


Fig. 3.5.1. The LEGB scope.

5. The LEGB rule refers to local scope, enclosing scope, global scope, and built-in scope.
6. Local scope extends for the body of a function and refers to anything indented in the function definition.
7. Variables, including the parameter, that are defined in the body of a function are local to that function and cannot be accessed outside the function. They are local variables.
8. The enclosing scope refers to variables that are defined outside a function definition.
9. If a function is defined within the scope of other variables, then those variables are available inside the function definition. The variables in the enclosing scope are available to statements within a function.

Que 3.6. **Discuss function in Python with its parts and scope. Explain with example. (Take simple calculator with add, subtract, division and multiplication).** **AKTU 2019-20, Marks 10**

Answer

Function : Refer Q. 3.1, Page 3–2T, Unit-3.

Parts of function : Refer Q. 3.2, Page 3–2T, Unit-3.

Scope : Refer Q. 3.5, Page 3–6T, Unit-3.

For example : Simple calculator using python :

```
# This function adds two numbers
def add(x, y) :
    return x + y
# This function subtracts two numbers
def subtract(x, y) :
    return x – y
# This function multiplies two numbers
def multiply(x, y) :
    return x * y
# This function divides two numbers
def divide(x, y) :
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
# Take input from the user
choice = input("Enter choice(1/2/3/4) : ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1' :
print(num1,"+",num2,"=", add(num1,num2))
elif choice == '2' :
print(num1,"–",num2,"=", subtract(num1,num2))
elif choice == '3' :
print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
print(num1,"/",num2,"=", divide(num1,num2))
else :
print("Invalid input")
```

PART-2

Strings : Length of the String and Perform Concatenation and Repeat Operations in it, Indexing and Slicing of Strings.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.7. What do you mean by the term strings ?

Answer

1. Strings are created by enclosing various characters within quotes. Python does not distinguish between single quotes and double quotes.
2. Strings are of literal or scalar type. The Python interpreter treats them as a single value.
3. Strings, in Python, can be used as a single data type, or, alternatively, can be accessed in parts. This makes strings really useful and easier to handle in Python.

For example :

```
>>> var1 = 'Hello Python!'
>>> var2 = "Welcome to Python Programming!"
>>> print var1
Hello Python!   #   Output
>>> print var2
Welcome to Python Programming!   #   Output
```

Que 3.8. What is len( ) function ?

OR

Explain length of the string.

Answer

1. len ( ) is a built-in function in Python. When used with a string, len returns the length or the number of character in the string.
2. Blank symbol and special characters are considered in the length of the string.

For example :

```
>>> var = "Hello Python!"
```

```
>>> len(var)
13   # Output
```

Here, we took a string 'Hello Python!' and used the len function with it. The len function returned the value 13 because not only characters, but also the blank space and exclamation mark in our string will also be counted as elements.

Que 3.9. Discuss the concatenation and repeat operation in Python with example.

Answer

Concatenation :

1. Concatenation means joining two operands by linking them end-to-end.
2. In list concatenation, + operator concatenate two lists with each other and produce a third list.
3. For example, to concatenate two lists :

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> z = x + y                    # concatenate two lists
>>> print z
[1, 2, 3, 4, 5, 6]
```

Repeat / replicate :

1. Lists can be replicated or repeated or repeatedly concatenated with the asterisk (*) operator.
2, For example,

```
>>> aList = [1, 2, 3]
>>> print aList*3                # list aList repeats three times
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Here, the list aList multiplied by 3 and printed three times.

Que 3.10. What do you mean by string slices ?

Answer

1. A piece or subset of a string is known as slice.
2. Slice operator is applied to a string with the use of square braces ([]).
3. Operator $[n : m]$ will give a substring which consists of letters between n and m indices, including letter at index n but excluding that at m, *i.e.*, letter from n^{th} index to $(m - 1)^{th}$ index.

For example :

```
>>> var = 'Hello Python'
```

>>> print var [0 : 4]

Hell # Output

In the above example, we can see that in the first case the slice is [0 : 4], which means that it will take the 0^{th} element and will extend to the 3^{rd} element, while excluding the 4^{th} element.

4. Similarly, operator [n : m : s] will give a substring which consists of letters from n^{th} index to $(m-1)^{th}$ index, where s is called the step value, *i.e.*, after letter at n, that at $n + s$ will be included, then $n + 2s$, $n + 3s$, etc.

For example :

>>> alphabet = "abcdefghij"

>>> print alphabet [1 : 8 : 3]

beh # Output

In the given example, the slice is [1 : 8 : 3], which means it will take the element at 1^{st} index which is *b* and will extend till 7^{th} element. Since step is 3, it will will print 1^{st} element, then 4^{th} element and then 7^{th} element *i.e.*, beh.

Que 3.11. Explain the term indexing in Python.

Answer

1. Indexing means referring to an element of an iterable by its position within the iterable.
2. **For example :** Create a list using a list comprehension :

   my_list = [_for_in 'abcdefghi']

   my_list

   ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
3. Now to retrieve an element of the list, we use the index operator ([ ]).
   For example :

   my_list[0]

   'a'
4. In Python, lists are "zero indexed", so [0] returns the zeroth (*i.e.*, the left-most) item in the list.
5. In the given example there are 9 elements in list ([0] through [8]) and if we want to access my_list[9] then it throws an indexError : list index out of range.
6. Python also allows to index from the end of the list using a negative number. In negative indices, we start counting from the right instead from the left.
7. Negative indices start from – 1.

PART-3

Pythons Data Structure : Tuples, Unpacking Sequences, Lists.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.12. Define tuples. How are tuples created in Python ?

Answer

1. Tuples are the sequence or series values of different types separated by commas (,).
2. Values in tuples can also be accessed by their index values, which are integers starting from 0.

For example :

The names of the months in a year can be defined in a tuple :

>>> months = ('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December')

Creating tuples in Python :

1. To create a tuple, all the items or elements are placed inside parentheses separated by commas and assigned to a variable.
2. Tuples can have any number of different data items (that is, integer, float, string, list, etc.).

For examples :

1. A tuple with integer data items :

```
>>> tuple = (4, 2, 9, 1)
>>> print tuple
(4, 2, 9, 1) # Output
```

2. A tuple with items of different data types :

```
>>>tuple_mix = (2, 30, "Python", 5.8, "Program")
>>>print tuple_mix
(2, 30, 'Python', 5.8, 'Program')   #   Output
```

3. Nested tuple :

```
>>>nested_tuple = ("Python", [1, 4, 2], ["john", 3.9])
>>> print nested_tuple
('Python', [1, 4, 2], ['john', 3.9])   #   Output
```

4. Tuple can also be created without parenthesis :

```
>>>tuple = 4.9, 6, 'house'
>>>print tuple
(4.9, 6, 'house')   #   Output
```

Que 3.13. **How are the values in a tuple accessed ?**

Answer

Accessing values in tuples :

1. In order to access the values in a tuple, it is necessary to use the index number enclosed in square brackets along with the name of the tuple.

   For example : Using square brackets

   ```
   >>>tup1 = ('Physics', 'Chemistry', 'Mathematics')
   >>>tup2 = (10, 20, 30, 40, 50)
   >>>print tup1 [1]
   Chemistry   #   Output
   >>>print tup2 [4]
   50   #   Output
   ```

2. We can also use slicing in order to print the continuous values in a tuple.

   For example :

   ```
   >>>tup1 = ('Physics', 'Chemistry', 'Mathematics')
   >>>tup2 = (10, 20, 30, 40, 50)
   >>>tup2 [1 : 4]
   (20, 30, 40)   # Output
   >>>tup1[ : 1]
   ('Physics',)   #   Output
   ```

Que 3.14. **Explain tuples as return values and variable-length arguments tuples with the help of example.**

Answer

Tuple as return values :

1. Tuples can also be returned by the function as return values.
2. Generally, the function returns only one value but by returning tuple, a function can return more than one value.

   For example : If we want to compute a division with two integers and want to know the quotient and the remainder, both the quotient and the remainder can be computed at the same time. Two values will be returned, *i.e.*, quotient and remainder, by using the tuple as the return value of the function.

   ```
   >>>def div_mod(a, b) :   #   defining function
   ```

```
    quotient = a/b
    remainder = a%b
    return quotient, remainder    #    function returning two values
# function calling
>>>x = 10
>>>y = 3
>>>t = div_mod(x, y)
>>>print t
(3, 1)    #    Output
>>>type(t)
<type 'tuple'>    #    Output
```

Variable length argument tuples :

1. Variable number of arguments can also be passed to a function.
2. A variable name that is preceded by an asterisk (*) collects the arguments into a tuple.

   For example : In the given example, we have defined a function traverse with argument *t*, which means it can take any number of arguments and will print each of them one by one.

```
>>>def traverse (* t) :
...         i = 0
...    while i<len(t) :
...         print t[i]
...         i = i + 1
>>>traverse(1, 2, 3, 4, 5)
```

Output :

```
1
2
3
4
5
```

Que 3.15. Explain the basic operation of tuples with example.

Answer

Basic operations of tuples are :

1. **Concatenation :** The concatenation in tuples in used to concatenates two tuples. This is done by using + operator in Python.

For example :

```
>>>t1 = (1, 2, 3, 4)
>>>t2 = (5, 6, 7, 8)
>>>t3 = tl + t2
>>>print t3
(1, 2, 3, 4, 5, 6, 7, 8)  #  Output
```

2. **Repetition :** The repetition operator repeats the tuples a given number of times. Repetition is performed by the * operator in Python.

For example :

```
>>>tuple = ('ok',)
>>>tuple * 5
('ok', 'ok', 'ok', 'ok', 'ok')  # Output
```

3. **In operator :**
   i. The in operator also works on tuples. It tells user that the given element exists in the tuple or not.
   ii. It gives a Boolean output, that is, true or false.
   iii. If the given input exists in the tuple, it gives the true as output, otherwise false.

For example :

```
>>>tuple = (10, 20, 30, 40)
>>>20 in tuple
True  #  Output
>>>50 in tuple
False  #  Output
```

4. **Interation :** It can be done in tuples using for loop. It helps in traversing the tuple.

For example :

```
>>>tuple = (1, 2, 3, 4, 5, 6)
>>> for x in tuple :
... print x
```

Output :

```
1
2
3
4
5
6
```

Que 3.16. What do you mean by unpacking sequences ? Give example to explain.

Answer

1. Unpacking allows to extract the components of the sequence into individual variable.
2. Several different assignments can be performed simultaneously.
3. We have multiple assignments in Python where we can have multiple LHS assigned from corresponding values at the RHS. This is an example of unpacking sequence.
4. There is one restriction, the LHS and RHS must have equal length. That is, every value that is created at RHS should be assigned to LHS.
5. Strings and tuples are example of sequences. Operations applicable on sequences are: Indexing, repetition, concatenation.

   For example :

   >>> student

   ('Aditya', 27, ('Python', 'Abha', 303))

   >>> name, roll, regdcourse = student

   >>> name

   Output : Aditya

   >>> roll

   27

   >>> regdcourse

   ('Python', 'Abha', 303)
6. Since strings are also sequences, we can also get individual characters in the string using unpacking operation.

   For example :

   >>> x1, x2, x3, x4 = 'abha'

   >>> print (x1, x2, x3, x4)

   a b h a

Que 3.17. Write short note on list.

Answer

1. A list is also a series of values in Python.
2. In a list, all the values are of any type.
3. The values in a list are called elements or items.
4. A list is a collection of items or elements.
5. The sequence of data in a list is ordered and can be accessed by their positions, *i.e.*, indices.

   For example :

   >>> list = [1, 2, 3, 4]

   >>> list [1]

```
2   #   Output
>>> list [3]
4   #   Output
```

Que 3.18. **Explain the copying method in list.**

Answer

We can make a duplicate or copy of an existing list.

There are two ways to make copy of a list.

1. **Using [:] Operator :**

```
>>> list_original = [1, 2, 3, 4]
>>> list_copy = list_original [:]   #   Using [:] operator
>>> print list_copy
[1, 2, 3, 4]
```

2. **Using built-in function :** Python has a built-in copy function which can be used to make copy of an existing list. In order to use the copy function, first we have to import it.

   For example :

```
>>> from copy import copy   #   Import library
>>> list_original = [1, 2, 3, 4]
>>> list_copy = copy(list_original)   #   Copying list
>>> print list_copy
[1, 2, 3, 4]
```

Que 3.19. **How elements are deleted from a list ?**

Answer

1. Python provides many ways in which the elements in a list can be deleted.

Methods of deleting element from a list :

1. **pop operator :**
   a. If we know the index of the element that we want to delete, then we can use the pop operator.

      For example :

```
>>> list = [10, 20, 30, 40,]
>>> a = list.pop (2)
>>> print list
[10, 20, 40]   #   Output
```

   b. The pop operator deletes the element on the provided index and stores that element in a variable for further use.

2. **del operator :** The del operator deletes the value on the provided index, but it does not store the value for further use.

For example :

```
>>> list = ['w', 'x', 'y', 'z',]
>>> del list (1)
>>> print list
['w', 'y', 'z']    #    Output
```

3. Remove operator :

a. We use the remove operator if we know the item that we want to remove or delete from the list.

For example :

```
>>> list = [10, 20, 30, 40,]
>>> list.remove (10)
>>> print list
[20, 30, 40]    #    Output
```

b. In order to delete more than one value from a list, del operator with slicing is used.

For example :

```
>>>list = [1, 2, 3, 4, 5, 6, 7, 8]
>>>del list [1 : 3]
>>>print list
[1, 4, 5, 6, 7, 8] # Output
```

Que 3.20. Discuss built-in list operators in detail.

Answer

Built-in list operators are as follows :

1. Concatenation : The concatenation operator is used to concatenate two lists. This is done by the + operator in Python.

For example :

```
>>>list1 = [10, 20, 30, 40]
>>>list2 = [50, 60, 70]
>>>list3 = list1 + list2
>>>print list3
[10, 20, 30, 40, 50, 60, 70]    #    Output
```

2. Repetition : The repetition operator repeats the list for a given number of times. Repetition is performed by the * operator.

For example :

```
>>>list1 = [1, 2, 3]
>>>list 1 * 4
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]    # Output
```

3. In operator :

a. The in operator tells the user whether the given string exists in the list or not.

b. It gives a Boolean output *i.e.*, true or false.

c. If the given input exists in the string, it given true as output, otherwise, false.

For example :

```
>>>list = ['Hello', 'Python', 'Program']
>>> 'Hello' in list
True  #  Output
>>>'World' in list
False  #  Output
```

Que 3.21. Write various built-in list methods. Explain any three with example.

Answer

Python includes many built-in methods for use with list as shown in Table 3.21.1.

Table 3.21.1. List of built-in list methods.

S. No.	Method	Description
1.	cmp (list1, list2)	It compares the elements of both lists, list1 and list2.
2.	max (list)	It returns the item that has the maximum value in a list.
3.	min (list)	It returns the item that has the minimum value in a list.
4.	list (seq)	It converts a tuple into a list.
5.	list.append (item)	It adds the item to the end of the list.
6.	list.count (item)	It returns number of times the item occurs in the list.
7.	list.extend (seq)	It adds the elements of the sequence at the end of the list.
8.	list.remove (item)	It deletes the given item from the list.
9.	list.reverse ( )	It reverses the position (index number) of the items in the list.
10.	list.sort ([func])	It sorts the elements inside the list and uses compare function if provided.

1. **Append method :** This method can add a new element or item to an existing list.

For example :

```
>>>list = [1, 2, 3, 4]
>>>list.append (0)
[1, 2, 3, 4, 0]                    # Output
```

2. **Extend method :** This method works like concatenation. It takes a list as an argument and adds it to the end of another list.

For example :

```
>>>list1 = ['x', 'y', 'z']
>>>list2 = [1, 2, 3]
>>>list1.extend (list2)
>>>print list1
['x', 'y', 'z', 1, 2, 3]           # Output
```

3. **Sort method :** This method arranges the list in ascending order.

For example :

```
>>>list = [4, 2, 5, 8, 1, 9]
>>>list.sort ( )
>>>print list
[1, 2, 4, 5, 8, 9]                 # Output
```

PART-4

Python Data Structure : Mutable Sequences, List Comprehension, Sets and Dictionaries.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.22. What are mutable sequences ? Discuss with example.

Answer

1. Python represents all its data as objects. Mutability of object is determined by its type.
2. Some of these objects like lists and dictionaries are mutable, meaning we can change their content without changing their identity.
3. Other objects like integers, floats, strings and tuples are immutable, meaning we cannot change their contents.

4. Dictionaries are mutable in Python :

a. Dictionaries in Python are mutable.

b. The values in a dictionary can be changed, added or deleted.

c. If the key is present in the dictionary, then the associated value with that key is updated or changed; otherwise a new key : value pair is added.

For example :

```
>>> dict1 = {'name' : 'Akash', 'age' : 27}
>>> dict1['age'] = 30    #    updating a value
>>> print dict
{'age' : 30, 'name': 'Akash'}    #    Output
>>> dict1['address'] = 'Alaska'    #    adding a key : value
>>>print dict1
{'age': 30, 'name': 'Akash', 'address': 'Alaska'}    #    Output
```

In the given example, we tried to reassign the value '30' to the key 'age', Python interpreter first searches the key in the dictionary and then update it. Hence, the value of 'age' is updated to 30. However, in the next statement, it does not find the key 'address'; hence, the key: value 'address' : 'Alaska' is added to the dictionary.

5. Strings are immutable :

a. String are immutable which means that we cannot change any element of a string.

b. If we want to change an element of a string, we have to create a new string.

For example :

```
>>> var = 'hello Python'
>>> var [0] = 'p'
```

Output :

Type error : 'str' object does not support item assignment

Here, we try to change the 0^{th} index of the string to a character *p*, but the python interpreter generates an error.

Now, the solution to this problem is to generator a new string rather than change the old string.

For example :

```
>>> var = 'hello Python'
>>> new_var = 'p' + var[1 :]
>>> print new_var
pello Python    #    Output
```

In the given example, we cut the slice from the original string and concatenate it with the character we want to insert in the string. It does not have any effect on the original string.

6. Lists are mutable :

a. Lists are mutable means that the value of any element inside the list can be changed at any point of time.

b. The elements of the list are accessible with their index value.

c. The index always starts with 0 and ends with $n - 1$, if the list contains n elements.

d. The syntax for accessing the elements of a list is the same as in the case of a string. We use square brackets around the variable and index number.

For example :

```
>>> list = [10, 20, 30, 40]
>>> list [1]
20  #  Output
```

In the given example, we access the 2nd element of the list that has 1 as index number and the interpreter prints 20.

Now, if we want to change a value in the list given in the example :

For example :

```
>>> list [3] = 50
>>> print list
[10, 20, 30, 50]  #  Output
```

Note that the value of the 4th element is changed to 50.

Que 3.23. Define the term list comprehension.

Answer

1. List comprehension is used to create a new list from existing sequences.
2. It is a tool for transforming a given list into another list.
3. Using list comprehension, we can replace the loop with a single expression that produces the same result.
4. The syntax of list comprehension is based on set builder notation in mathematics.
5. Set builder notation is a notation is a mathematical notation for describing a set by stating the property that its members should satisfy. The syntax is

   [<expression> for <element> in <sequence> if <conditional>]

   The syntax is read as "Compute the expression for each element in the sequence, if the conditional is true".

For example :

```
>>> List1 = [10, 20, 30, 40, 50]
>>> List1
        [10, 20, 30, 40, 50]
>>> for i in range (0, len(List1)) :
        List1 [i] = List1[i] + 10
>>> List1
 [20, 30, 40, 50, 60]
```

Without list comprehension

```
>>> List1 = [10, 20, 30, 40, 50]
>>> List1= [x + 10 for x in List1]
>>> List1
 [20, 30, 40, 50, 60]
```

Using list comprehension

5. In the given example, the output for both without list comprehension and using list comprehension is the same.
6. The use of list comprehension requires lesser code and also runs faster.
7. From above example we can say that list comprehension contains :
   a. An input sequence
   b. A variable referencing the input sequence
   c. An optional expression
   d. An output expression or output variable

Que 3.24. What do you mean by sets ? Explain the operations performed on sets.

Answer

1. In Python we also have one data type which is an unordered collection of data known as set.
2. A set does not contain any duplicate values or elements.
3. **Operations performed on sets are :**
   i. **Union :** Union operation performed on two sets returns all the elements from both the sets. It is performed by using and operator.
   ii. **Intersection :** Intersection operation performed on two sets returns all the element which are common or in both the sets. It is performed by using '|' operator.
   iii. **Difference :** Difference operation performed on two sets set1 and set2 returns the elements which are present on set1 but not in set2. It is performed by using '–' operator.
   iv. **Symmetric difference :** Symmetric difference operation performed on two sets returns the element which are present in either set1 or set2 but not in both. It is performed by using ^ operator.

For example :

```
# Defining sets
>>> set1 = set([1, 2, 4, 1, 2, 8, 5, 4])
>>> set2 = set ([1, 9, 3, 2, 5])
>>> print set1   #   Printing set
set([8, 1, 2, 4, 5])
>>> print set2
set([1, 2, 3, 5, 9])   #   Output
>>> intersection = set1 & set2   #   intersection of set1 and set2
>>> print intersection
set([1, 2, 5])   #   Output
>>> union = set1 | set2   #   Union of set1 and set2
>>> print union
set ([1, 2, 3, 4, 5, 8, 9])   #   Output
>>> difference = set1 – set2   # Difference of set 1 and set2
>>> print difference
set ([8, 4])   #   Output
>>> symm_diff = set1 ^ set2 # symmetric difference of set1 and set2
>>> print symm_diff
set ([3, 4, 8, 9])   #   Output
```

Que 3.25. **What is dictionary in Python ?**

Answer

1. The Python dictionary is an unordered collection of items or elements.
2. All other compound data types in Python have only values as their elements or items whereas the dictionary has a key : value pair. Each value is associated with a key.
3. In dictionary, we have keys and they can be of any type.
4. Dictionary is said to be a mapping between some set of keys and values.
5. The mapping of a key and value is called as a key-value pair and together they are called one item or element.
6. A key and its value are separated by a colon (:) between them.
7. The items or elements in a dictionary are separated by commas and all the elements must be enclosed in curly braces.
8. A pair of curly braces with no values in between is known as an empty dictionary.
9. The values in a dictionary can be duplicated, but the keys in the dictionary are unique.

Que 3.26. **How do we create a dictionary in Python ?**

Answer

1. Creating a dictionary is simple in Python.
2. The values in a dictionary can be of any data type, but the keys must be of immutable data types (such as string, number or tuple).

For example :

1. Empty dictionary :

```
>>> dict1 = {}
>>>print dict1
{}  #  Output
```

2. Dictionary with integer keys :

```
>>> dict1 = {1 : 'red', 2 : 'yellow', 3 : 'green'}
>>>print dict1
{1 : 'red', 2 : 'yellow', 3 : 'green'}  #  Output
```

3. Dictionary with mixed keys :

```
>>> dict1 = {'name' : 'Rahul', 3 : ['Hello', 2, 3]}
>>>print dict1
{3 : ['Hello', 2, 3], 'name' : 'Rahul'}  #  Output
```

Que 3.27. Explain the operations performed on dictionary with example.

Answer

Operations in dictionary :

1. Traversing :

a. Traversing in dictionary is done on the basis of keys.

b. For traversing, for loop is used, which iterates over the keys in the dictionary and prints the corresponding values using keys.

For example :

```
>>>defprint_dict(d):
...     for c in d:
...             print c,d[c]
>>> dictl = {1 : 'a', 2 : 'b', 3 : 'c' 4 : 'd'}
>>> print_dict(dict1)
```

Output :

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

This example prints the key: value pairs in the dictionary dict.

2. Membership :

i. Using the membership operator (in and not in), we can test whether a key is in the dictionary or not.

ii. In operator takes an input key and finds the key in the dictionary.

iii. If the key is found, then it returns true, otherwise, false.

For example :

```
>>>cubes = {1 : 1, 2 : 8, 3 : 27, 4 : 64, 5 : 125, 6 : 216}
>>> 3 in cubes
True                                    # Output
>>> 17 not in cubes
True                                    # Output
```

Que 3.28. Write some built in dictionary methods used in Python with example.

Answer

There are some built-in dictionary methods which are included in Python given in Table 3.28.1.

Table 3.28.1. Built-in dictionary methods.

S. No.	Function	Description
1.	all (dict)	It is a Boolean type function, which returns true if all keys of dictionary are true (or the dictionary is empty).
2.	any (dict)	It is also a Boolean type function, which returns true if any key of the dictionary is true. It returns false if the dictionary is empty.
3.	len (dict)	It returns the number of items (length) in the dictionary.
4.	cmp (dict1, dict2)	It compares the items of two dictionaries.
5.	sorted (dict)	It returns the sorted list of keys.
6.	dict.clear ( )	It deletes all the items in a dictionary at once.
7.	dict. copy ( )	It returns a copy of the dictionary.
8.	dict.get (key, default = None)	For key key, returns value or default if key not in dictionary.
9.	dict.items ( )	It returns a list of entire key : value pair of dictionary.
10.	dict.keys ( )	It returns the list of all the keys in dictionary.
11.	dict.update (dict2)	It adds the items from dict2 to dict.
12.	dict.values ( )	It returns all the values in the dictionary.

For example :

```
>>>cubes = {1 : 1, 2 : 8, 3 : 27, 4 : 64, 5 : 125, 6 : 216}
>>>all (cubes)
True                                    # Output
>>>any (cubes)
True                                    # Output
>>>len (cubes)
6                                       # Output
>>>sorted (cubes)
[1, 2, 3, 4, 5, 6]                      # Output
>>>str (cubes)
'{1 : 1, 2 : 8, 3 : 27, 4 : 64, 5 : 125, 6 : 216}'          # Output
```

PART-5

Higher Order Functions : Treat Functions as First-Class Objects, Lambda Expressions.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.29. Explain treat functions as first-class objects in detail.

Answer

1. In Python both functions and classes are treated as objects which are called as first class objects, allowing them to be manipulated in the same ways as built-in data types.
2. By definition, first class objects are the following which is :
   a. Created at runtime
   b. Assigned as a variable or in a data structure
   c. Passed as an argument to a function
   d. Returned as the result of a function
3. For example, let create and test a function, then read its__doc__and check its type.

```
>>> def factorial(n):
. . . '''returns n!'''
        ... return 1 if n < 2 else n * factorial(n – 1)
...
 >>> factorial (42)
```

140500611775287989854314260$244511569936384000000000

>>> factorial. __doc __

'returns n!'

>>> type(factorial)

<class 'function'>

4. The console session in example shows that Python functions are objects. Here we create a function, call it, read its__doc__attribute, and check that the function object itself is an instance of the function class.

Que 3.30. Explain Lambda expression.

Answer

1. Lambda expressions is used to create the anonymous function.
2. The anonymous functions are the functions created using a lambda keyword.
3. They are not defined by using def keyword. For this reason, they are called anonymous functions.
4. We can pass any number of arguments to a lambda form functions, but still they return only one value in the form of expression.
5. An anonymous function cannot directly call print command as the lambda needs an expression.
6. It cannot access the parameters that are not defined in its own namespace.
7. An anonymous function is a single line statement function.
8. **Syntax :**

   lambda [arg1 [,arg2, ..., argn]] : expression

   The syntax of the lambda function is a single statement

   For example :

   # function definition here

   >>>mult = lambda val1, val2 : val1*val2 ;

   # function call here

   >>> print "value :", mult(20,40)

   Value : 800 # Output

   In the given example, the lambda function is defined with two arguments val1 and val2. The expression val1*va12 does the multiplication of the two values. Now, in function call, we can directly call the mult function with two valid values as arguments and produce the output as shown in given example.

Que 3.31. Explain higher order function with respect to lambda expression. Write a Python code to count occurrences of an element in a list. **AKTU 2019-20, Marks 10**

Answer

1. Reduce(), filter(), map() are higher order functions used in Python.
2. Lambda definition does not include a "return" statement, it always contains an expression which is returned.
3. We can also put a lambda definition anywhere a function is expected, and we do not have to assign it to a variable at all.
4. Lambda functions can be used along with built-in higher order functions like filter(), map() and reduce().

Use of lambda with filter() :

1. The filter() function in Python takes in a function and a list as arguments.
2. This function helps to filter out all the elements of a sequence "sequence", for which the function returns true.

For example : Python program that returns the odd numbers from an input list :

```
# Python code to illustrate filter() with lambda
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambdax: (x%2!=0), li))
print(final_list)
```

Output :

```
[5, 7, 97, 77, 23, 73, 61]
```

Use of lambda() with reduce() :

1. The reduce() function in Python takes in a function and a list as argument.
2. The function is called with a lambda function and a list and a new reduced result is returned. This performs a repetitive operation over the pairs of the list.
3. This is a part of functools module.

For example :

```
# Python code to illustrate reduce() with lambda()  to get sum of a list
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y : x + y), li)
print (sum)
```

Output :

```
193
```

Here the results of previous two elements are added to the next element and this goes on till the end of the list like (((((5+8)+10)+20)+50)+100).

Program to count occurrences of an element in a list :

```
# vowels list
vowels = [‘a’, ‘e’, ‘i’, ‘o’, ‘i’, ‘u’]
# count element ‘i’
count = vowels.count(‘i’)
# print count
print(‘The count of i is:’, count)
# count element ‘p’
count = vowels.count(‘p’)
# print count
print(‘The count of p is:’, count)
```

Output :

The count of i is: 2

The count of p is: 0

Que 3.32. Explain unpacking sequences, mutable sequences, and list comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.

AKTU 2019-20, Marks 10

Answer

Unpacking sequences : Refer Q. 3.16, Page 3–14T, Unit-3.

Mutable sequences : Refer Q. 3.22, Page 3–19T, Unit-3.

List comprehension : Refer Q. 3.23, Page 3–21T, Unit-3.

Program :

```
# Initializing list of dictionaries
lis = [{“name” : “Nandini”, “age” : 20},
{“name” : “Manjeet”, “age” : 20 },
{“name” : “Nikhil” , “age” : 19 }]
# using sorted and lambda to print list sorted by age
print “The list printed sorting by age :”
print sorted(lis, key = lambda i: i[‘age’])
print (“\r”)
# using sorted and lambda to print list sorted by both age and name
print “The list printed sorting by age and name:”
print sorted(lis, key = lambda i: (i[‘age’], i[‘name’]))
print (“\r”)
# using sorted and lambda to print list sorted
# by age in descending order
print “The list printed sorting by age in descending order:”
print sorted(lis, key = lambda i: i[‘age’],reverse=True)
```

Output :
The list printed sorting by age:
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}]
The list printed sorting by age and name :
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Manjeet'}, {'age': 20, 'name': 'Nandini'}]
The list printed sorting by age in descending order:
[{'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}, {'age': 19, 'name': 'Nikhil'}]

Que 3.33. **Write a function in find the HCF of some given numbers.**

Answer

```
>>>def hcf (a, b) :
    if a > b :
        small = b
    else :
        small = a
    for i in range (1, small + 1) :
        if (a % i == 0) and (b % i == 0)) :
            hcf = i
    return hcf
>>> hcf (20, 40)
20                                  #Output
>>>hcf (529, 456)
1                                   #Output
```

Que 3.34. **Write a function to find the ASCII value of the character.**

Answer

```
>>>def ascii_val_of (a) :
    print ("The ASCII value of '" + a + "' is", ord (a))
>>>ascii_val_of ('A')
("The ASCII value of 'A' is", 65)       #Output
>>>ascii_val_of (' ')                   #Finding ASCII value of space
("The ASCII value of ' ' is", 32)       #Output
```

Que 3.35. **Write a function to convert a decimal number to its binary, octal and hexadecimal equivalents.**

Answer

```
>>>def bin_oct_hex (a) :
        print (bin (a), "binary equivalent")
        print (oct (a), "octal equivalent")
        print (hex (a), "hexadecimal equivalent")
>>>bin_oct_hex (10)                    #Finding binary, octal, hex value of 10
('0b1010', 'binary equivalent')        #Output
('012', 'octal equivalent')
('0xa', 'hexadecimal equivalent')
```

4
UNIT

Sieve of Eratosthenes and File I/O

CONTENTS

Part-1 : Sieve of Eratosthenes ..... **4–2T to 4–4T**

Part-2 : File I/O : File Input and Output Operations in Python Programming ..... **4–4T to 4–7T**

Part-3 : Exceptions and Assertions ..... **4–7T to 4–13T**

Part-4 : Modules : Introduction, Importing Modules, Abstract Data Types : Abstract Data Types and ADT Interface in Python Programming ..... **4–13T to 4–17T**

Part-5 : Classes : Class Definition and other Operations in the Classes, Special Methods (Such as__init__, __str__, Comparison Methods, Arithmetic Methods, etc.), Class Example ..... **4–17T to 4–22T**

Part-6 : Inheritance : Inheritance and OOPS ..... **4–22T to 4–32T**

PART- 1

Sieve of Eratosthenes.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. What is Sieve of Eratosthenes ?

Answer

1. Sieve of Eratosthenes is a simple and ingenious ancient algorithm for finding all prime numbers up to any given limit.
2. It does so by iteratively marking as composite (*i.e.*, not prime) the multiples of each prime, starting with the first prime number, 2.
3. The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime.
4. Following is the algorithm to find all the prime numbers less than or equal to a given integer n by Eratosthenes' method :
   a. Create a list of consecutive integers from 2 to n : (2, 3, 4, …, n).
   b. Initially, let p equal 2, the first prime number.
   c. Starting from p^2, count up in increments of p and mark each of these numbers greater than or equal to p^2 itself in the list. These numbers will be $p(p+1)$, $p(p+2)$, $p(p+3)$, etc.
   d. Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this number (which is the next prime), and repeat from step c.

Que 4.2. Explain Sieve of Eratosthenes with example.

Answer

1. Let us take an example when n = 50. So, we need to print all print numbers smaller than or equal to 50.
2. We create a list of all numbers from 2 to 50.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

3. According to the algorithm we will mark all the numbers which are divisible by 2 and are greater than or equal to the square of it.

	2	3	4*	5	6*	7	8*	9	10*
11	12*	13	14*	15	16*	17	18*	19	20*
21	22*	23	24*	25	26*	27	28*	29	30*
31	32*	33	34*	35	36*	37	38*	39	40*
41	42*	43	44*	45	46*	47	48*	49	50*

4. Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3 and are greater than or equal to the square of it.

	2	3	4*	5	6*	7	8*	9*	10*
11	12*	13	14*	15*	16*	17	18*	19	20*
21*	22*	23	24*	25	26*	27*	28*	29	30*
31	32*	33*	34*	35	36*	37	38*	39*	40*
41	42*	43	44*	45*	46*	47	48*	49	50*

5. We move to our next unmarked number 5 and mark all multiples of 5 and are greater than or equal to the square of it.

	2	3	4*	5	6*	7	8*	9*	10*
11	12*	13	14*	15*	16*	17*	18*	19	20*
21*	22*	23	24*	25*	26*	27*	28*	29	30*
31	32*	33*	34*	35*	36*	37*	38*	39*	40*
41	42*	43	44*	45*	46*	47*	48*	49	50*

We continue this process and our final table will :

	2	3	4*	5	6*	7	8*	9*	10*
11	12*	13	14*	15*	16*	17	18*	19	20*
21*	22*	23	24*	25*	26*	27*	28*	29	30*
31	32*	33*	34*	35*	36*	37	38*	39*	40*
41	42*	43	44*	45*	46*	47	48*	49*	50*

So the prime numbers are the unmarked ones : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Que 4.3. **Write a Python program to print all primes smaller than or equal to *n* using Sieve of Eratosthenes.**

Answer

```
def SieveOf Eratosthenes (n) :
        # Create a boolean array "prime[0. . n]" and initialize
        # all entries it as true. A value in prime[i] will
```

```
        # finally be false if i is Not a prime, else true.
        prime = [True for i in range(n+1)]
        p = 2
        while (p * p <= n):
            # If prime[p] is not changed, then it is a prime
            if (prime[p] == True):
                # Update all multiples of p
                for i in range(p * p, n+1, p):
                    prime[i] = False
            p + = 1
        # Print all prime numbers
        for p in range(2, n):
            if prime[p]:
                print p,
# driver program
if__name__'=='__main__':
        n = 30
        print "Following are the prime numbers smaller",
        print "than or equal to", n
        SieveOfEratosthenes(n)
```

PART-2

File I/O : File Input and Output Operations in Python Programming.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.4. What are files ? How are they useful ?

Answer

1. A file in a computer is a location for storing some related data.
2. It has a specific name.
3. The files are used to store data permanently on to a non-volatile memory (such as hard disks).
4. As we know, the Random Access Memory (RAM) is a volatile memory type because the data in it is lost when we turn off the computer. Hence, we use files for storing of useful information or data for future reference.

Que 4.5. Describe the opening a file function in Python.

Answer

1. Python has a built-in open () function to open files from the directory.
2. Two arguments that are mainly needed by the open () function are :
   - **a. File name :** It contains a string type value containing the name of the file which we want to access.
   - **b. Access_mode :** The value of access_mode specifies the mode in which we want to open the file, *i.e.*, read, write, append etc.
3. **Syntax :**

   file_object = open(file_name [, access_mode])

   For example :

```
>>>f = open ("test.txt")                    #Opening file current directory
>>>f = open ("C:/Python27/README.txt")
                                    #Specifying full path        #Output
>>>f
<open file 'C:/Python27/README.txt', mode 'r' at 0x02BC5128>
                                                                 #Output
```

Que 4.6. Explain the closing a file method in Python.

Answer

1. When the operations that are to be performed on an opened file are finished, we have to close the file in order to release the resources.
2. Python comes with a garbage collector responsible for cleaning up the unreferenced objects from the memory, we must not rely on it to close a file.
3. Proper closing of a file frees up the resources held with the file.
4. The closing of file is done with a built-in function close ().
5. **Syntax :**

   fileObject. close ()

   For example :

```
# open a file
>>> f = open ("test. txt", "wb")
# perform file operations
>>> f. close()    # close the file
```

Que 4.7. Discuss writing to a file operation.

Answer

1. After opening a file, we have to perform some operations on the file. Here we will perform the write operation.
2. In order to write into a file, we have to open it with *w* mode or *a* mode, on any writing-enabling mode.
3. We should be careful when using the *w* mode because in this mode overwriting persists in case the file already exists.

   For example :

   # open the file with w mode

   >>> f = open ("C :/Python27/test.txt", "w")

   # perform write operation

   >>>f. write ('writing to the file line 1/n')

   >>>f. write ('writing to the file line 2/n')

   >>>f. write ('writing to the file line 3/n')

   >>>f. write ('writing to the file line 4')

   # clos the file after writing

   >>> f.close ()

   The given example creates a file named test.txt if it does not exist, and overwrites into it if it exists. If we open the file, we will find the following content in it.

   Output :

   Writing to the file line 1

   Writing to the file line 2
   Writing to the file line 3
   Writing to the file line 4

Que 4.8. Explain reading from a file operation with example.

Answer

1. In order to read from a file, we must open the file in the reading mode (*r* mode).
2. We can use read (size) method to read the data specified by size.
3. If no size is provided, it will end up reading to the end of the file.
4. The read() method enables us to read the strings from an opened file.
5. **Syntax :**

   file object. read ([size])

   For example :

   # open the file

```
>>> f = open ("C :/Python27/test.txt", "r")
>>>f . read (7)   #   read from starting 7 bytes of data
'writing'   #   Output
>>>f. read (6)   #   read next 6 bytes of data
'to the'   #   Output
```

Que 4.9. **Discuss file I/O in Python. How to perform open, read, write, and close into a file ? Write a Python program to read a file line-by-line store it into a variable.** **AKTU 2019-20, Marks 10**

Answer

File I/O : Refer Q. 4.4, Page 4–4T, Unit-4.

Open, read, write and close into a file : Refer Q. 4.5, Page 4–4T, Refer Q. 4.8, Page 4–6T, Refer Q. 4.7, Page 4–5T, and Refer Q. 4.6, Page 4–5T; Unit-4.

Program :

```
L = ["Quantum\n", "for\n", "Students\n"]
# writing to file
file1 = open('myfile.txt', 'w')
file1.writelines(L)
file1.close()
# Using readlines()
file1 = open('myfile.txt', 'r')
Lines = file1.readlines()
count = 0
# Strips the newline character
for line in Lines:
print(line.strip())
print("Line{}: {}".format(count, line.strip()))
```

Output :

```
Line1: Quantum
Line2: for
Line3: Students
```

PART-3

Exception and Assertions.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.10. **Describe assertions.**

Answer

1. An assertion is a sanity-check that we can turn on or turn off when we are done with our testing of the program. An expression is tested, and if the result is false, an exception is raised.
2. Assertions are carried out by the assert statement.
3. Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.
4. An AssertionError exception is raised if the condition evaluates to false.
5. The syntax for assert is : assert Expression [, Arguments]
6. If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError.

   For example :

   Consider a function that converts a temperature from degrees Kelvin to degrees Fahrenheit. Since zero degrees Kelvin is as cold as it gets, the function fails if it sees a negative temperature :

```
#!/user/bip/python
def KelvinToFahrenheit(Temperature) :
    assert (Temperature >= 0),"Colder than absolute zero!"
    return ((Temperature – 273)*1.8) + 32
print KelvinToFahrenheit (273)
print int(KelvinToFahrenheit (505.78))
print KelvinToFahrenheit (– 5)
```

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

```
32.0
451
Traceback (most recent call last) :
    File "test.py", line 9, in<module>
        print KelvinToFahrenheit (– 5)
    File "test.py", line 4, in KelvinToFahrenheit
        assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError : Colder than absolute zero!
```

Que 4.11. **What do you mean by exceptions ?**

Answer

1. While writing a program, we often end up making some errors. There are many types of error that can occur in a program.

2. The error caused by writing an improper syntax is termed syntax error or parsing error; these are also called compile time errors.
3. Errors can also occur at runtime and these runtime errors are known as exceptions.
4. There are various types of runtime error in Python.
5. For example, when a file we try to open does not exist, we get a FileNotFoundError. When a division by zero happens, we get a ZeroDivisionError. When the module we are trying to import does not exist, we get an ImportError.
6. Python creates an exception object for every occurrence of these runtime errors.
7. The user must write a piece of code that can handle the error.
8. If it is not capable of handling the error, the program prints a trace back to that error along with the details of why the error has occurred.

For example :

Compile time error (syntax error)

```
>>> a = 3
>>> if (a < 4)   #  semicolon is not included
SyntaxError : invalid syntax   #   Output
```

ZeroDivisionError :

```
>>>5/0
```

Output :

```
Traceback (most recent call last) :
File "<pyshell#71>", line 1, in <module>
5/0
ZeroDivisionError : Integer division or modulo by zero
```

Que 4.12. Explain exceptions handling with syntax.

Answer

1. Whenever an exception occurs in Python, it stops the current process and passes it to the calling process until it is handled.
2. If there is no piece of code in our program that can handle the exception, then the program will crash.
3. For example, assume that a function *X* calls the function *Y*, which in turn calls the function *Z*, and an exception occurs in *Z*. If this exception is not handled in *Z* itself, then the exception is passed to *Y* and then to *X*. If this exception is not handled, then an error message will be displayed and our program will suddenly halt.

i. Try...except :

a. Python provides a try statement for handling exceptions.

b. An operation in the program that can cause the exception is placed in the try clause while the block of code that handles the exception is placed in the except clause.

c. The block of code for handling the exception is written by the user and it is for him to decide which operation he wants to perform after the exception has been identified.

Syntax :

try :

the operation which can cause exception here,

.....

except Exceptionl1 :

if there is exception1, execute this.

except Exception2 :

if there is exception2, execute this.

.....

else :

if no exception occurs, execute this.

ii. try finally :

a. The try statement in Python has optional finally clause that can be associated with it.

b. The statements written in finally clause will always be executed by the interpreter, whether the try statement raises an exception or not.

c. With the try clause, we can use either except or finally, but not both.

d. We cannot use the else clause along with a finally clause.

Que 4.13. Give example of try....except.

Answer

```
>>>try:
...         file = open("C:/Python27/test.txt","w")
...         file write("hello python")
...     exceptIOError :
...         print "Error: cannot find file or read data
...     else :
...         print "content written successfully"
>>> file. close ( )
```

1. In the given example, we are trying to open a file test.txt with write access mode, and want to write to that file. We have added try and except blocks.
2. If the required file is not found or we do not have the permission to write to the file, an exception is raised.
3. The exception is handled by the except block and the following statement printed :
   Error : cannot find file or read data
4. On the other hand, if the data is written to the file then the else block will be executed and it will print the following.
   Output :
   Content written successfully

Que 4.14. Give example of try....finally.

Answer

```
>>> try :
...         file = open("testfile","w")
...         try :
...             file.write("Write this to the file")
...         finally :
...             print "Closing file"
...             file.close()
...     exceptIOError:
            print "Error Occurred"
```

1. In the given example, when an exception is raised by the statements of try block, the execution is immediately passed to the finally block.
2. After all the statements inside the finally block are executed, the exception is, raised again and is handled by the except block that is associated with the next higher layer try block.

Que 4.15. Discuss exceptions and assertions in Python. How to handle exceptions with try-finally ? Explain five built-in exceptions with example. **AKTU 2019-20, Marks 10**

Answer

Exception : Refer Q. 4.11, Page 4–8T, Unit-4.
Assertions : Refer Q. 4.10, Page 4–8T, Unit-4.
Handle exceptions : Refer Q. 4.12, Page 4–9T, Unit-4.
Five built-in exceptions :

1. **exception LookupError :** This is the base class for those exceptions that are raised when a key or index used on a mapping or sequence is invalid or not found. The exceptions raised are :
   a. KeyError
   b. IndexError

For example :

```
try:
a = [1, 2, 3]
print a[3]
except LookupError :
    print "Index out of bound error."
else:
    print "Success"
```

Output :

Index out of bound error.

2. **TypeError :** TypeError is thrown when an operation or function is applied to an object of an inappropriate type.

For example :

```
>>> '2'+2
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

3. **exception ArithmeticError :** This class is the base class for those built-in exceptions that are raised for various arithmetic errors such as :
   a. OverflowError
   b. ZeroDivisionError
   c. FloatingPointError

For example :

```
>>> x=100/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
x=100/0
ZeroDivisionError: division by zero
```

4. **exception AssertionError :** An AssertionError is raised when an assert statement fails.

For example :

```
assert False, 'The assertion failed'
```

Output :

```
Traceback (most recent call last):
File "exceptions_AssertionError.py", line 12, in
assert False, 'The assertion failed'
AssertionError: The assertion failed
```

5. **exception AttributeError :**

An AttributeError is raised when an attribute reference or assignment fails such as when a non-existent attribute is referenced.

For example :

```
class Attributes(object):
    pass
object = Attributes()
print object.attribute
```

Output :

Traceback (most recent call last):
File "d912bae549a2b42953bc62da114ae7a7.py", line 5, in
print object.attribute
AttributeError: 'Attributes' object has no attribute 'attribute'

PART-4

Modules : Introduction, Importing Modules, Abstract Data Types : Abstract Data Types and ADT Interface in Python Programming.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.16. **Define the term modules.**

Answer

1. A module is a file containing Python definitions and statements. A module can define functions, classes and variables.
2. It allows us to logically organize our Python code.
3. The file name is the module name with the suffix .py appended.
4. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.
5. Definitions from a module can be imported into other modules or into the main module.

For example :

Here is an example of a simple module named as support.py

```
def print_func( par ):
    print "Hello : ", par
    return
```

Que 4.17. **Explain the import statement with the help of example.**

Answer

1. The import statement is the most common way of invoking the import machinery.
2. An import statement is made up of the import keyword along with the name of the module.
3. The import statement combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope.

For example :

```
# to import standard module math
import math
print("The value of pi is", math.pi)
```

When we run the program, the output will be :

```
The value of pi is 3.141592653589793
```

Que 4.18. Explain abstract data types with its types.

Answer

1. Abstract Data type (ADT) is a type for objects whose behaviour is defined by a set of value and a set of operations.
2. There are three types of ADTs :
   - **a. List ADT :** The data is stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list.
   - **b. Stack ADT :**
     - i. In stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
     - ii. The program allocates memory for the data and address is passed to the stack ADT.
     - iii. The head node and the data nodes are encapsulated in the ADT.
     - iv. The calling function can only see the pointer to the stack.
     - v. The stack head structure also contains a pointer to top and count of number of entries currently in stack.
   - **c. Queue ADT :**
     - i. The queue abstract data type (ADT) follows the basic design of the stack abstract data type.
     - ii. Each node contains a void pointer to the data and the link pointer to the next element in the queue.
     - iii. The program allocates the memory for storing the data.

Que 4.19. Explain ADT interface in Python programming.

Answer

1. ADT only defines as what operations are to be performed but not how these operations will be implemented.
2. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
3. It is called "abstract" because it gives an implementation-independent view.

4. The process of providing only the essentials and hiding the details is known as abstraction.

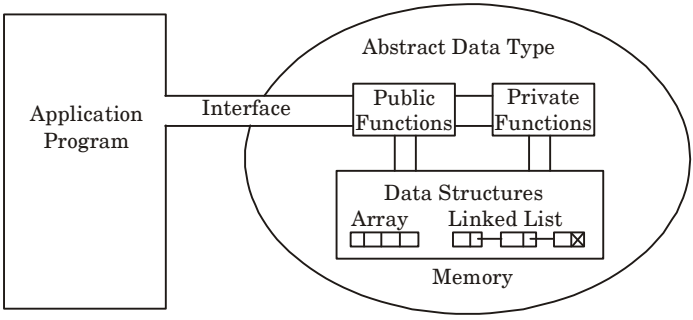


Fig. 4.19.1.

5. The user of data type does not need to know how that data type is implemented, for example, we have been using primitive values like int, float, char data types only with the knowledge that these data type can operate and be performed on without any idea of how they are implemented.
6. So, a user only needs to know what a data type can do, but not how it will be implemented.

Que 4.20. Discuss ADT in Python. How to define ADT ? Write code for a student information. **AKTU 2019-20, Marks 10**

Answer

ADT in python : Refer Q. 4.18, Page 4–14T, Unit-4.

The Queue and Stack are used to define Abstract Data Types (ADT) in Python.

Code for student information :

```
class Student :
# Constructor
def __init__(self, name, rollno, m1, m2):
    self.name = name
        self.rollno = rollno
        self.m1 = m1
        self.m2 = m2
    # Function to create and append new student
    def accept(self, Name, Rollno, marks1, marks2 ):
    # use 'int(input())' method to take input from user
        ob = Student(Name, Rollno, marks1, marks2 )
        ls.append(ob)
```

```
# Function to display student details
def display(self, ob):
    print("Name  :", ob.name)
    print("RollNo :", ob.rollno)
    print("Marks1 :", ob.m1)
    print("Marks2 :", ob.m2)
    print("\n")
# Search Function
def search(self, rn):
    for i in range(ls.__len__()):
    if(ls[i].rollno == rn):
                    return i
# Delete Function
def delete(self, rn):
    i = obj.search(rn)
    del ls[i]
# Update Function
def update(self, rn, No):
    i = obj.search(rn)
    roll = No
    ls[i].rollno = roll;
# Create a list to add Students
ls =[ ]
# an object of Student class
obj = Student(' ', 0, 0, 0)
print("\nOperations used, ")
print("\n1.Accept Student details\n2.Display Student Details\n"/
/"3.Search Details of a Student\n4.Delete Details of Student" /
/"\n5.Update Student Details\n6.Exit")
ch = int(input("Enter choice:"))
if(ch == 1):
    obj.accept("A", 1, 100, 100)
    obj.accept("B", 2, 90, 90)
    obj.accept("C", 3, 80, 80)
elif(ch == 2):
    print("\n")
    print("\nList of Students\n")
    for i in range(ls.__len__()):
    obj.display(ls[i])
elif(ch == 3):
    print("\n Student Found,")
    s = obj.search(2)
```

```
        obj.display(ls[s])
    elif(ch == 4):
        obj.delete(2)
        print(ls.__len__())
        print("List after deletion")
        for i in range(ls.__len__()):
        obj.display(ls[i])
    elif(ch == 5):
        obj.update(3, 2)
        print(ls.__len__())
        print("List after updation")
        for i in range(ls.__len__()):
        obj.display(ls[i])
    else:
        print("Thank You !")
```

PART-5

Classes : Class Definition and other Operations in the Classes, Special Methods (Such as__init__, __str__, Comparison Methods, Arithmetic Methods, etc.), Class Example.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.21. Define class.

Answer

1. A class can be defined as a blue print or a previously defined structure from which objects are made.
2. Classes are defined by the user; the class provides the basic structure for an object.
3. It consists of data members and method members that are used by the instances of the class.
4. In Python, a class is defined by a keyword Class.
5. Syntax : class class_name;

   For example : Fruit is a class, and apple, mango and banana are its objects. Attribute of these objects are color, taste, etc.

Que 4.22. **What do you mean by objects ?**

Answer

1. An object is an instance of a class that has some attributes and behaviour.
2. The object behaves according to the class of which it is an object.
3. Objects can be used to access the attributes of the class.
4. The syntax of creating an object in Python is similar to that for calling a function.
5. **Syntax :**

   obj_name = class_name ( )

   For example :
   s1 = Student ()
   In the given example, Python will create an object s1 of the class student.

Que 4.23. **Give an example of class.**

Answer

```
>>>class Student :
...     'student details'
...     def fill_details(self, name, branch, year):
...             self.name = name
...             self.branch = branch
...             self.year = year
...             print("A Student detail object is created")
...     def print details (self) :
...             print('Name: ', self.name)
...             print('Branch: ',self.branch)
...              print('Year: ',self.year)
```

In the given example, we have created a class Student that contains two methods: fill_details and print_details. The first method fill_details takes four arguments: self, name, branch and year. The second method print_details takes exactly one argument: self.

Que 4.24. **Explain object creation with the help of example.**

Answer

```
>>>class Student :
...     'student details'
...     def fill_details(self, name, branch, year):
...             self.name = name
```

```
...         self.branch = branch
...         self.year = year
...         print("A Student detail object is created")
...     def print details (self) :
...         print('Name: ', self.name)
...         print('Branch: ',self.branch)
...          print('Year: ',self.year)
# creating an object of Student class
>>>s1 = Student ()
# creating another object of Student class
>>>s2 = Student ()
# using the method fill_details with proper attributes
>>> s1.fill_details('John','CSE','2002')
A Student detail object is created
>>>s2.fill_details('Jack','ECE','2004')
A Student detail object is created
# using the print_details method with proper attributes
>>>s1.print_details ()
Name : John   # Output
Branch : CSE # Output
Year : 2002     # Output
>>>s2.print_details()
Name : Jack   # Output
Branch : ECE # Output
Year : 2004     # Output
```

Que 4.25. Discuss the __init__ function in detail with example.

Answer

1. The __init__ function is a reserved function in classes in Python which is automatically called whenever a new object of the class is instantiated.
2. As a regular function, the init function is also defined using the def keyword. As per the object-oriented programming paradigm, these types of functions are called constructors.
3. We use constructors to initialize variables.
4. We can pass any number of arguments while creating the class object as per the definition of the init function.

For example :

```
class IntellipaatClass :
    def__init__(self, course) :
        self.course = course
    def display(self) :
        print(self.course)object1 = IntellipaatClass("Python")
        object1.display( )
```

Output :

Python

4. In the above example, the init function behaves as a constructor and is called automatically as soon as the 'object1 = IntellipaatClass("Python")' statement is executed.
5. Since it is a function inside a class, the first argument passed in it is the object itself which is caught in the 'self' parameter.
6. The second parameter, *i.e.*, course, is used to catch the second argument passed through the object that is 'Python'. And then, we have initialized the variable course inside the init function.
7. Further, we have defined another function named 'display' to print the value of the variable. This function is called using object1.

Que 4.26. What is __str__ method in class ?

Answer

1. __str__method is the "informal" or nicely printable string representation of an object. This is for the end user.
2. It is called by str(object) and the built-in functions such as format() and print().
3. The return value of this method is a string object.

For example :

```
Class Account :
    def __str__(self) :
        return 'Account of { } with starting amount : { }' format
        (self.owner, self.amount)
```

Now we can query the object in various ways and always get a nice string representation :

```
>>> str(acc)
'Account of bob with starting amount : 10'
```

Que 4.27. Define the comparison method.

Answer

1. The comparison methods are used whenever <, >, <=, >=, !=, == are used with the object.
2. The comparison methods are also called 'rich comparison methods' or 'comparison magic methods'.
3. Following are the comparison methods : _it_, _le_, _eq_, _ne_, _gt_, _ge_.

   object.__it__(self, other) # For x < y

   object.__le__(self, other) # For x < = y

   object.__eq__(self, other) # For x == y

   object.__ne__(self, other) # For x ! = y OR x <> y

   object.__gt__(self, other) # For x > y

   object.__ge__(self, other) # For x > = Y
4. The most common usage is to return False or True when using one of the comparison methods, but we can actually return any value.
5. If $x == y$ it does not mean that x != y. The best practice is always to define __ne__() if __eq__() is defined.

Que 4.28. Explain various arithmetic methods in detail.

Answer

Method :	Description
__add__(self, other)	To get called on add operation using + operator
__sub__(self, other)	To get called on subtraction operation using – operator.
__mut__(self, other)	To get called on multiplication operation using * operator.
__floordiv__(self, other)	To get called on floor division operation using // operator.
__div__(self, other)	To get called on division operation using / operator.

For example :

```
class IntervalMath(object) :
def __init__(self, lower, upper) :
      self. to = float (lower)
      self. up = float (upper)
def __add__(self, other) :
```

```
        a, b, c, d = self.lo, self.up, other.lo, other.up
        return IntervalMath (a + c, b + d)
    def __sub__(self, other) :
        a, b, c, d = self. lo, self. up, other. lo, other. up
        return IntervalMath (a – d, b – c )
    def __mul__(self, other) :
        a, b, c, d = self.lo, self.up, other.lo, other.up
        return IntervalMath(min(a*c, a*d, b*c, b*d),
                                                max(a*c, a*d, b*c, b*d))
    def __div__(self, other):
        a, b, c, d = self.lo, self.up, other.lo, other.up
        # [c,d] cannot contain zero:
        if c*d <= 0:
            raise ValueError\
            ('Interval %s cannot be denominator because '\'
            it contains zero' % other)
        return IntervalMath(min(a/c, a/d, b/c, b/d),   max(a/c, a/d, b/c, b/d))
```

The code of this class is found in the file IntervalMath.py.

```
I = IntervalMath
a = I(– 3, – 2)
b = I(4, 5)
expr = 'a + b', 'a – b', 'a*b', 'a/b'
for e in expr :
    print '%s =' % e, eval(e)
```

Output :

```
a + b = [1, 3]
a – b = [– 8, – 6]
a * b = [– 15, – 8]
a / b = [– 0.75, – 0.4]
```

PART-6

Inheritance : Inheritance and OOPS.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.29. **What is object-oriented programming ?**

Answer

1. The object-oriented programming approach mainly focuses on the object and classes while procedural programming focuses on the function and methods.
2. The object is an instance of class.
3. It is a collection of data (variables) and methods (functions).
4. A class can also be called the basic structure of object.
5. Class is set of attributes, which can be data members and methods members.

Que 4.30. **Define the term inheritance.**

Answer

1. A class '*A*' that can use the characteristics of another class '*B*' is said to be a derived class, *i.e*., a class inherited from '*B*'. The process is called inheritance.
2. In OOP, it means that reusability of code.
3. It is the capability of a class to derive the properties of another class that has already been created.

   For example : Vehicle is a class that is further divided into two subclasses, automobiles (driven by motors) and pulled vehicles (driven by men). Therefore, vehicle is the base class and automobiles and pulled vehicles are its subclasses. These subclasses inherit some of the properties of the base class vehicle.

Que 4.31. **Give syntax of inheritance and explain with the help of example.**

Answer

Syntax :

```
Class sub_classname(Parent_classname):
    'Optional Docstring'
    Class_suite
```

For example :

```
#Define a parent class Person
>>>class Person(object) :
    'returns a Person object with given name'
    def get_name (self ,name) :
```

```
            self.name = name
        def get_details (self) :
            'returns a string containing name of person'
            return self.name
#Define a subclass Student
>>>class Student (Person) :
        'return a Student object, takes 2 arguments'
        def fill_details (self, name, branch) :
            Person.get_name(self,name)
            self.branch = branch
def get_details(self):
            'returns student details'
            print("Name:", self.name)
            print("Branch: ", self.branch)
#Define a subclass Teacher
>>>class Teacher(Person) :
        'returns a Teacher object, takes 1 arguments'
        def fill_details(self, name, branch) :
            Person.get_name(self,name)
        def get_details(self) :
        print("Name:", self.name)
#Define one object for each class
>>>person 1 = Person ()
>>>student 1 = Student ()
>>>teacher 1 = Teacher ()
#Fill details in the objects
>>>  person1.get_name('John')
>>> student1.fill_details('Jinnie', 'CSE')
>>> teacher1.fill_details('Jack')
#Print the details using parent class function
>>>print(personl.get_details())
John       # Output
>>>print(studentl.get_details())
Name: Jinnie # Output
Branch: CSE  # Output
>>>print(teacherl.get_details())
```

Name: Jack # Output

Que 4.32. What do you mean by multiple inheritance ? Explain in detail.

Answer

1. In multiple inheritance, a subclass is derived from more than one base class.
2. The subclass inherits the properties of all the base classes.
3. In Fig. 4.32.1, subclass *C* inherits the properties of two base classes *A* and *B*.

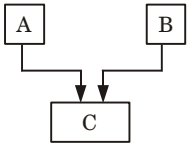


Fig. 4.32.1. Multiple inheritance.

Syntax :

```
# Define your first parent class
class A
..... class_suite .....
# Define your second parent class
class B
.....class-suite.....
# Define the subclass inheriting both A and B
class C(A, B)
.....class-suite.....
```

For example :

```
>>> class A :                          # Defining class A
      def x(self):
          print("method of A")
>>> class B :                          # Defining Class B
      def x(self):
          print("method of B")
>>> class C(A, B) :                    # Defining class C
      pass
>>> y = c ()
>>> B.x(y)
```

```
method of B                    # Output
>>> A.x(Y)
method of A                    # Output.
```

Que 4.33. Define method overriding with the help of example.

Answer

1. Method overriding is allowed in Python.
2. Method overriding means that the method of parent class can be used in the subclass with different or special functionality.

For example :

```
>>>class Parent :
    def ovr_method (self) :
        print 'This is in Parent Class'
>>>class Child (Parent) :
    def ovr_method (self) :
        print 'This is in Child Class'
>>>c = Child ()
>>>c.ovr_method()
This is in Child Class      #  Output
```

Que 4.34. Describe the term polymorphism.

Answer

1. The word 'Poly' means 'many'.
2. The term 'polymorphism' means that the object of a class can have many different forms to respond in different ways to any message or action.
3. Polymorphism is the capability for a message or data to be processed in one or more ways.

For example :

1. If a base class is mammals, then horse, human, and cat are its subclasses. All the mammals can see in the daytime.
2. Therefore, if the message 'see in the daytime' is passed to mammals, all the mammals including the human, the horse and the cat will respond to it.
3. Whereas, if the message 'see during the night time' is passed to the mammals, then only the cat will respond to the message as it can see during the night as well as in daytime.
4. Hence, the cat, which is a mammal, can behave differently from the other mammals.

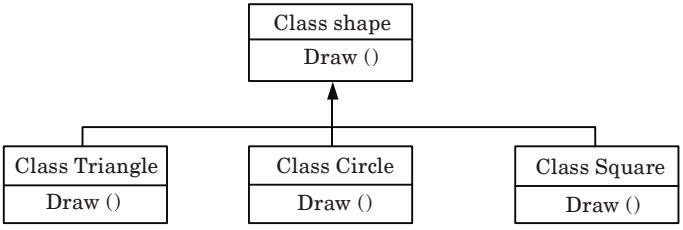


Fig. 4.34.1. Pythomorphism.

Que 4.35. Explain data encapsulation with example.

Answer

1. In Python programming language, encapsulation is a process to restrict the access of data members. This means that the internal details of an object may not be visible from outside of the object definition.
2. The members in a class can be assigned in three ways *i.e.*, public, protected and private.
3. If the name of a member is preceded by single underscore, it is assigned as a protected member.
4. If the name of a member is preceded by double underscore, it is assigned as a private member.
5. If the name is not preceded by anything then it is a public member.

Name	Notation	Behaviour
varname	Public	Can be accessed from anywhere
_varname	Protected	They are like the public members but they cannot be directly accessed from outside
__varname	Private	They cannot be seen and accessed from outside the class

For example :

```
>>>class MyClass (object) :                    #Defining class
      def __init__ (self, x, y, z) :
            self.var1 = x                      #public data member
            self_var2 = y                      #projected data member
            self__var3 = z                     #private data member
>>>obj = MyClass (3, 4, 5)
```

```
>>>obj.var1
3                                   #Output
>>>obj.var1 = 10
>>>obj.var1
10                                  #Output
>>>obj._var2
4                                   #Output
>>>obj._var2 = 12
>>>obj.var2
12                                  #Output
>>>obj.__var3                       #Private member is not
                                     accessible
Traceback (most recent call last) :
    File "<pyshell#71>", line 1, in<module>
        obj.__var3
AttributeError : 'MyClass' object has no attribute '__var3'
```

Que 4.36. Discuss data hiding in detail.

Answer

1. In Python programming, there might be some cases when we intend to hide the attributes of objects outside the class definition.
2. To accomplish this, use double score (__) before the name of the attributes and these attributes will not be visible directly outside the class definition.

For example :

```
>>> class MyClass :                 # defining class
    __ a = 0 ;
def sum (self, increment) :
        self.__a += increment
        self print.__ a
>>>b = MyClass()                    # creating instance of class
>>>b.sum(2)
2    #Output
>>> b.sum(5)
7    #Output
>>> print b. __a
Traceback (most recent call last) :
    File "<pyshell#24>", line 1, in <module>
        print b.__a
```

AttributeError : MyClass instance has no attribute '__a'

3. In the given example, the variable *a* is not accessible as we tried to access it; the Python interpreter generates an error immediately.
4. In such a case, the Python secures the members by internally changing the names, to incorporate the name of the class.
5. In the given code, if we use the aforementioned syntax to access the attributes, then the following changes are seen in the output:

```
>>> class MyClass :                    # Defining class
      __a = 0;
      def sum (self, increment):
          self. __a += increment
          print self. __a
>>> b = MyClass()                      # Creating instance of class
>>> b.sum(2)
2     #Output
>>> b.sum(5)
7     #Output
>>> print b._ MyClass_a                # Accessing the hidden variable
7     #Output
```

Que 4.37. **What will be the output after the following statements ?**

```
class Furniture:
    def legs():
        print('is made of wood')
Furniture.legs()
```

Answer

is made of wood

Que 4.38. **What will be the output after the following statements ?**

```
class Furniture:
    def chair(x):
        print('It has %s legs' % x)
    def table(x):
        print('It has %s legs' % x)
Furniture.table(6)
```

Answer

It has 6 legs

Que 4.39. **What will be the output after the following statements ?**

```
class Furniture:
    def chair():
        print('It has 4 legs')
```

```
    def table():
        print('It has 6 legs')
Furniture.chair()
```

Answer

It has 4 legs

Que 4.40. **What will be the output after the following statements ?**

```
import random
x = [3, 8, 6, 5, 0]
print(random.choice(x))
```

Answer

A random element from the list *x*.

Que 4.41. **What will be the output after the following statements ?**

```
import random
x = [3, 8, 6, 5, 0]
random.shuffle(x)
print(x)
```

Answer

The shuffled list *x* with the elements mixed up.

Que 4.42. **What will be the output after the following statements ?**

```
import re
x = re.compile(r'(Sun)+day')
y = x.search('Today is a nice day and a Sunday')
print(y.group())
```

Answer

Sunday

Que 4.43. **What will be the output after the following statements ?**

```
import re
x = re.compile(r'(Python){2}')
y = x.search('PythonPythonPython')
print(y.group())
```

Answer

PythonPython

Que 4.44. **What will be the output after the following statements ?**

```
import re
x = re.compile(r'(Python){2,3}')
```

```
y = x.search('PythonPythonPython')
print(y.group())
```

Answer

PythonPythonPython

Que 4.45. **What will be the output after the following statements ?**

```
import re
x = re.compile(r'(Python){1,3}?')
y = x.search('PythonPythonPython')
print(y.group())
```

Answer

Python

Que 4.46. **What will be the output after the following statements ?**

```
import re
x = re.compile(r'day')
y = x.findall('Today is a nice day and a Sunday')
print(y)
```

Answer

['day', 'day', 'day']

Que 4.47. **What will be the output after the following statements ?**

```
import re
x = re.compile(r'(Sun)?day')
y = x.findall('Today is a nice day and a Sunday')
print(y)
```

Answer

[' ', ' ', 'Sun']

Que 4.48. **What will be the output after the following statements ?**

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

Answer

The current working directory

Que 4.49. **What do the following statements do ?**

```
import webbrowser
webbrowser.open('http://google.com')
```

Answer

Launch a browser window to http://google.com

Que 4.50. **What will be the output after the following statements ?**

```
import sys
print(sys.argv)
```

Answer

A list of the program's filename and command line arguments

5
UNIT

Iterators and Recursion

CONTENTS

Part-1 : Iterators and Recursion : ..... **5–2T to 5–7T**
Recursive Fibonacci,
Tower of Hanoi

Part-2 : Search : Simple Search and ..... **5–7T to 5–10T**
Estimating Search Time,
Binary Search and Estimating
Binary Search Time

Part-3 : Sorting and Merging : ..... **5–10T to 5–18T**
Selection Sort, Merge List,
Merge Sort, Higher
Order Sort

PART- 1

Iterators and Recursion : Recursive Fibonacci, Tower of Hanoi.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.1. What do you mean by iterator ?

Answer

1. An iterator is an object that contains a countable number of values.
2. An iterator is an object that can be iterated upon, meaning that we can traverse through all the values.
3. Python iterator, implicitly implemented in constructs like for-loops, comprehensions, and python generators.
4. Python lists, tuples, dictionary and sets are all examples of in-built iterators.
5. These types are iterators because they implement following methods :
   - **a. __iter__ :** This method is called on initialization of an iterator. This should return an object that has a next() method.
   - **b. next() (or __next__) :** The iterator next method should return the next value for the iterable. When an iterator is used with a 'for in' loop, the for loop implicitly calls next() on the iterator object. This method should raise a StopIteration to signal the end of the iteration.

For example :

```
# An iterable user defined type
class Test:
    # Constructor
    def __init__(self, limit):
        self.limit = limit
    # Called when iteration is initialized
    def __iter__(self):
        self.x = 10
        return self
    # To move to next element.
    def next(self):
        # Store current value of x
        x = self.x
```

```
            # Stop iteration if limit is reached
            if x > self.limit:
                raise StopIteration
            # Else increment and return old value
            self.x = x + 1;
            return x
    # Prints numbers from 10 to 15
    for i in Test(15):
        print(i)
    # Prints nothing
    for i in Test(5):
        print(i)
```

Output :

```
10
11
12
13
14
15
```

Que 5.2. **Define recursion. Also, give example.**

Answer

1. In Python, recursion occurs when a function is defined by itself.
2. When a function calls itself, directly or indirectly, then it is called a recursive function and this phenomenon is known as recursion.
3. Recursion is the property how we write a function. A function which performs the same task can be written either in a recursive form or in an iterative form.
4. Recursion is the process of repeating something self-similar way.

For example :

```
def fact (n) :
    if n == 0:
        return 1
    else :
        return n * fact(n – 1)
    print(fact(0))
    print(fact(5))
```

Output :

```
1
120
```

Que 5.3. **Explain Fibonacci series using Python.**

Answer

1. Fibonacci series is a series of numbers formed by the addition of the preceding two numbers in the series.
2. It is simply the series of numbers which starts from 0 and 1 and then continued by the addition of the preceding two numbers.
3. Example of Fibonacci series: 0, 1, 1, 2, 3, 5.
4. **Python code for recursive Fibonacci :**

```
def FibRecursion(n) :
    if n <= 1 :
        return n
    else :
        return(FibRecursion(n – 1) + FibRecursion(n – 2))
nterms = int(input("Enter the term : ")) # take input from the user
if nterms < = 0: # check if the number is valid
    print ("Please enter a positive integer")
else :
    print ("Fibonacci sequence :")
    for i in range (nterms) :
        print(FibRecursion(i))
```

Output : Enter the term : 5
Fibonacci sequence :

0 1 1 2 3

1. In the given Python program, we use recursion to generate the Fibonacci sequence.
2. The function FibRecursion is called recursively until we get the output.
3. In the function, we first check if the number n is zero or one. If yes, it returns the value of n. If not, we recursively call FibRecursion with the values $n – 1$ and $n – 2$.

Que 5.4. Explain Tower of Hanoi problem in detail.

OR

Explain iterator. Write a program to demonstrate the Tower of Hanoi using function. **AKTU 2019-20, Marks 10**

Answer

Iterator : Refer Q. 5.1, Page 5–2T, Unit-5.

Tower of Hanoi problem :

1. Tower of Hanoi is a mathematical puzzle which consists of three rods and a number of disks of different sizes, which can slide onto any rod.
2. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

3. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules :
   a. Only one disk can be moved at a time.
   b. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
   c. A disk can only be moved if it is the uppermost disk on a stack.
   d. No disk may be placed on top of a smaller disk.

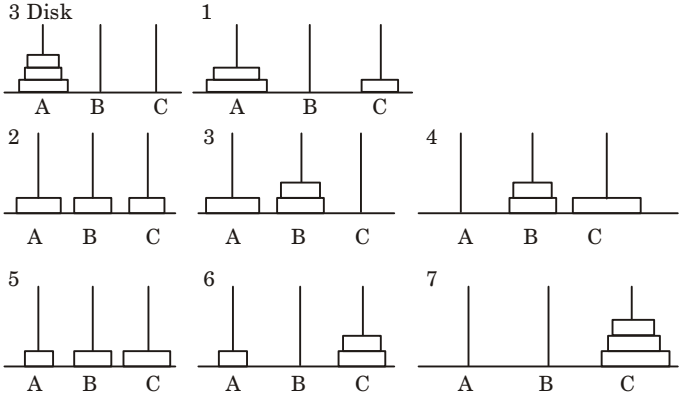


Fig. 5.4.1.

4. Recursive Python function to solve Tower of Hanoi

```
def TowerOfHanoi(n, from_rod, to_rod, aux_rod):
    if n==1:
        print "Move disk 1 from rod",from_rod,"to rod",to_rod
        return
    TowerOfHanoi(n – 1, from_rod, aux_rod, to_rod)
    print "Move disk",n,"from_rod",from_rod,"to rod",to_rod
    TowerOfHanoi(n – 1, aux_rod, to_rod, from_rod)
    n = 4
TowerOfHanoi(n, \'A\', \'C\', \'B\')
# A, C, B are the name of rods
```

Output :

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
```

Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

Que 5.5. **Differentiate between recursion and iteration.**

OR

Discuss and differentiate iterators and recursion. Write a program for recursive Fibonacci series. **AKTU 2019-20, Marks 10**

Answer

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instruction repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size need to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code size	Smaller code size.	Larger code size.
Time Complexity	Very high (generally exponential) time complexity.	Relatively lower time complexity (generally polynomial logarithmic).
Stack	The stack is used to store the set of new local variables and parameters each time the function is called.	Does not use stack.
Overhead	Recursion possesses the overhead of repeated function calls.	No overhead of repeated function call.
Speed	Slow in execution.	Fast in execution.

Program for recursive Fibonacci series : Refer Q. 5.3, Page 5–3T, Unit-5.

PART-2

Search : Simple Search and Estimating Search Time, Binary Search and Estimating Binary Search Time.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.6. What is simple (linear) search ? Explain with the help of example.

Answer

1. Linear search is a method for finding a particular value in a list.
2. Linear search is good to use when we need to find the first occurrence of an item in an unsorted collection.
3. Linear (Simple) search is one of the simplest searching algorithms, and the easiest to understand.
4. It starts searching the value from the beginning of the list and continues till the end of the list until the value is found.
5. **Code :**

```
def seach(arr, n, x) :
    i = 0
    for i in range(i, n) :
        if (arr[i] == x) :
            return – i
    return – 1
```

For example :

```
arr = [3, 10, 30, 45]
x = 10
n = len(arr)
print(x, “is present at index”, search(arr, n, x))
```

Output : 10 is present at index 1

Que 5.7. Explain the time complexity for linear search.

Answer

1. Let $T(n)$ denote the time taken by a search on a sequence of size n.
2. Therefore, recurrence relation is : $T(n) = T(n-1) + C$
3. Solution to the recurrence relation is: $T(n) = Cn$
4. We have three cases to analyse an algorithm:

a. Worst case :

i. In the worst case analysis, we calculate upper bound on running time of an algorithm.

ii. For linear search, the worst case happens when the element to be searched is not present in the array.

iii. When x is not present, the search() functions compares it with all the elements of arr[] one by one.

iv. Therefore, the worst case time complexity of linear search would be $\Theta(n)$.

b. Average case :

i. In average case analysis, we take all possible inputs and calculate computing time for all of the inputs.

ii. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So, we sum all the cases and divide the sum by $(n + 1)$.

iii. Therefore, the average case time complexity of linear search would be $\Theta(n)$.

c. Best case :

i. In the best case analysis, we calculate lower bound on running time of an algorithm.

ii. In the linear search problem, the best case occurs when x is present at the first location.

iii. So, time complexity in the best case would be $\Theta(1)$.

Que 5.8. Discuss binary search in Python.

Answer

1. Binary search follows a divide and conquer approach. It is faster than linear search but requires that the array be sorted before the algorithm is executed.
2. Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned.
3. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item.

4. Otherwise, the item is searched for in the sub-array to the right of the middle item.
5. This process continues on the sub-array as well until the size of the sub-array reduces to zero.
6. **Code :**

```
def binarysearch(arr, 1, r, x) :
    while 1 <= r :
        mid = 1 + (r – 1)/2;
    # Check if x is present at mid
        if arr[mid] == x:
            return mid
    # If x is greater, ignore left half
        elif arr[mid] < x :
            l = mid + 1
    # If x is smaller, ignore right half
        else :
            r = mid – 1
# If we reach here, then the element was not present
    return  – 1
# Test array
arr = [2, 3, 4, 10, 40]
x = 10
# Function call
result = binarySearch(arr, 0, len(arr) – 1, x)
if result ! = – 1:
    print “Element is present at index % d” % result
else :
    print “Element is not present in array”
```

Output :

Element is present at index 3

Que 5.9. Explain the time complexity for binary search.

Answer

1. Let $T(n)$ denote the time taken by a search on a sequence of size n.
2. Therefore, recurrence relation is : $T(n) <= T(n/2) + C$
3. Solution to the recurrence relation is : $T(n) = \log(n)$

Recursive function of binary search :

$T(n) = T(n/2) + 1$

$T(n/2) = T(n/4) + 1 + 1$

Put the value of $T(n/2)$ in above so

$T(n) = T(n/4) + 1 + 1...T(n/2^k) + 1 + 1$

$= T(2^k/2^k) + 1 + 1...+ 1$ upto k

$= T(1) + k$

As we taken $2^k = n$

$k = \log n$

So time complexity is $O(\log n)$

4. We have three cases to analyse an algorithm :

a. Best case :

i. In the best case, the item x is the middle in the array A. A constant number of comparisons (actually just 1) are required.

ii. Time complexity in best case is $O(1)$.

b. Worst case :

i. In the worst case, the item x does not exist in the array A at all. Through each recursion or iteration of binary search, the size of the admissible range is halved.

ii. Time complexity in worst case is $O(\log n)$.

c. Average case :

i. To find the average case, take the sum over all elements of the product of number of comparisons required to find each element and the probability of searching for that element.

ii. Time complexity in average case is $O(\log n)$.

PART-3

Sorting and Merging : Selection Sort, Merge List, Merge Sort, Higher Order Sort.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.10. What do you mean by selection sort ? Discuss in detail.

Answer

1. The selection sort algorithm sorts an array by repeatedly finding the smallest element (considering ascending order) from unsorted list and swapping it with the first element of the list.
2. The algorithm maintains two sub-arrays in a given array:
   i. The sub-array which is already sorted.
   ii. Remaining sub-array which is unsorted.
3. In every iteration of selection sort, the smallest element from the unsorted sub-array is picked and moved to the sorted sub-array.
4. **Code :**

```
def slectionSort(nlist) :
    for fillslot in range(len(nlist) – 1, 0, – 1) :
        maxpos = 0
        for location in range(1, fillslot + 1) :
            if nlist[location]>nlist[maxpos] :
                maxpos = location
        temp = nlist[fillslot]
        nlist[fillslot] = nlist[maxpos]
        nlist[maxpos] = temp
nlist = [14, 46, 43, 27, 57, 41, 45, 21, 70]
selectionSort(nlist)
print(nlist)
```

Output :

[14, 21, 27, 41, 43, 45, 46, 57, 70]

5. **Time complexity :**
   i. Best case : $O(n^2)$
   ii. Worst case : $O(n^2)$
   iii. Average case : $O(n^2)$

Que 5.11. Explain merge list.

Answer

1. Merging is defined as the process of creating a sorted list/array of data items from two other sorted array/list of data items.
2. Merge list means to merge two sorted list into one list.

Code for merging two lists and sort it :

```
a=[ ]
c=[ ]
```

```
n1=int(input("Enter number of elements:"))
for i in range(1, n1+1):
    b=int(input("Enter element:"))
    a.append(b)
n2=int(input("Enter number of elements:"))
for i in range(1, n2+1):
    d=int(input("Enter element:"))
    c.append(d)
new=a+c
new.sort( )
print("Sorted list is:", new)
```

Que 5.12. **Explain merge sort with the help of example.**

Answer

1. Merge sort is a divide and conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.
2. The merge() function is used for merging two halves.
3. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m + 1 ..r] are sorted and merges the two sorted sub-arrays into one.
4. **Code :**

```
def mergeSort(arr)
    if len(arr) >1:
        mid = len(arr)//2 #Finding the mid of the array
        L = arr[:mid] # Dividing the array elements
        R = arr[mid:] # into 2 halves
        mergeSort(L) # Sorting the first half
        mergeSort(R) # Sorting the second half
        i = j = k = 0
# Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i + = 1
            else :
                arr[k] = R[j]
```

```
                j + = 1
            k + = 1
    # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i + = 1
            k + = 1
        while j < len(R):
            arr[k] = R[j]
            j + = 1
            k + = 1
    # Code to print the list
    def printList(arr):
        for i in range(len(arr)):
            print(arr[i],end=" ")
        print()
    # driver code to test the above code
    if __name__ == '__main__ ' :
        arr = [12, 11, 13, 5, 6, 7 ]
        print ("Given array is", end = "\n")
        printList(arr)
        mergeSort(arr)
        print("Sorted array is: ", end = "\n")
        printList(arr)
```

Output :

Given array is

12, 11, 13, 5, 6, 7

Sorted array is

5, 6, 7, 11, 12, 13

5. **Time complexity :** Recurrence relation of merge sort is given by

$T(n) = 2T(n/2) + Cn$

$= 2(2T(n/4) + Cn/2) + Cn = 2^2T(n/4) + 2Cn$

$= 2^2(2T(n/8) + Cn/4) + Cn = 2^3T(n/8) + 3Cn$

$= \ldots$ // keep going for k steps

$= 2^kT(n/2^k) + k*Cn$

Assume $n = 2^k$ for some k.

$k = \log_2 n$

Then, $T(n) = n*T(1) + Cn*\log_2 n$

i. Time complexity of Merge sort is $O(n \log n)$ in all three cases (worst, average and best) as merge sort always divides the array into two halves and take linear time to merge two halves.

Que 5.13. Discuss higher order sort.

Answer

1. Python also supports higher order functions, meaning that functions can accept other functions as arguments and return functions to the caller.
2. **Sorting of higher order functions :**
   a. In order to defined non-default sorting in Python, both the sorted() function and .sort() method accept a key argument.
   b. The value passed to this argument needs to be a function object that returns the sorting key for any item in the list or iterable.
3. **For example :** Consider the given list of tuples, Python will sort by default on the first value in each tuple. In order to sort on a different element from each tuple, a function can be passed that return that element.

```
>>> def second_element (t) :
...         return t[1]
...
>>> zepp = [('Guitar', 'Jimmy'), ('Vocals', 'Robert'), ('Bass', 'John Paul'), ('Drums', 'John')]
>>> sorted(zepp)
[('Bass', 'John Paul'), ('Drums', 'John'), ('Guitar', 'Jimmy'), ('Vocals', 'Robert')]
>>> sorted(zepp, key = second_element)
[('Guitar', 'Jimmy'), ('Drums', 'John'), ('Bass', 'John Paul'), ('Vocals', 'Robert')]
```

Que 5.14. Discuss sorting and merging. Explain different types of sorting with example. Write a Python program for Sieve of Eratosthenes. **AKTU 2019-20, Marks 10**

Answer

Sorting :

1. Sorting refers to arranging data in a particular order.
2. Most common orders are in numerical or lexicographical order.
3. The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner.
4. Sorting is also used to represent data in more readable formats.

Merging : Refer 5.11, Page 5–11T, Unit-5.

Different types of sorting are :

1. Bubble sort : It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

For example :

```
def bubblesort(list):
# Swap the elements to arrange in order
    for iter_num in range(len(list) – 1,0, – 1):
        for idx in range(iter_num):
        if list[idx]>list[idx+1]:
                temp = list[idx]
                 list[idx] = list[idx+1]
                 list[idx+1] = temp
list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

Output :

```
[2, 6, 11, 19, 27, 31, 45, 121]
```

2. Merge sort : Refer 5.12, Page 5–12T, Unit-5.

3. Selection sort : Refer 5.10, Page 5–10T, Unit-5.

4. Higher order sort : Refer 5.13, Page 5–14T, Unit-5.

5. Insertion sort :

a. Insertion sort involves finding the right place for a given element in a sorted list. So in beginning we compare the first two elements and sort them by comparing them.

b. Then we pick the third element and find its proper position among the previous two sorted elements.

c. This way we gradually go on adding more elements to the already sorted list by putting them in their proper position.

For example :

```
def insertion_sort(InputList):
    for i in range(1, len(InputList)):
        j = i – 1
        nxt_element = InputList[i]
# Compare the current element with next one
        while (InputList[j] > nxt_element) and (j >= 0):
        InputList[j+1] = InputList[j]
            j=j – 1
        InputList[j+1] = nxt_element
list = [19,2,30,42,28,11,135,26]
insertion_sort(list)
print(list)
```

Output :

[2, 11, 19, 26, 28, 30, 42, 135]

Program : Refer Q. 4.3, Page 4–3T, Unit-4.

Que 5.15. **What will be the output after the following statements ?**

```
x = [25, 14, 53, 62, 11]
x.sort()
print(x)
```

Answer

[11, 14, 25, 53, 62]

Que 5.16. **What will be the output after the following statements ?**

```
x = ['25', 'Today', '53', 'Sunday', '15']
x.sort()
print(x)
```

Answer

['15', '25', '53', 'Sunday', 'Today']

Que 5.17. **What will be the output after the following statements ?**

```
x = {5:4, 8:8, 3:16, 9:32}
print(sorted(x.items()))
```

Answer

[(3, 16), (5, 4), (8, 8), (9, 32)]

Que 5.18. **What will be the output after the following statements ?**

```
x = ['a', 'b', 'c', 'A', 'B', 'C']
x.sort()
print(x)
```

Answer

['A', 'B', 'C', 'a', 'b', 'c']

Que 5.19. **What will be the output after the following statements ?**

```
x = ['a', 'b', 'c', 'A', 'B', 'C']
x.sort(key=str.lower)
print(x)
```

Answer

['a', 'A', 'b', 'B', 'c', 'C']

Que 5.20. **What will be the output after the following statements ?**

```
x = ['a', 'b', 'c', 'A', 'B', 'C']
x.sort(key=str.swapcase)
print(x)
```

Answer

['a', 'b', 'c', 'A', 'B', 'C']

Que 5.21. **What will be the output after the following statements ?**

```
x = ['a', 'b', 1, 2, 'A', 'B']
x.sort()
print(x)
```

Answer

TypeError

Que 5.22. **What will be the data type of the output after the following statements ?**

```
x = 'Python'
y = list(x)
print(y)
```

Answer

List

Que 5.23. **What will be the data type of the output after the following statements ?**

```
x = 'Python'
y = tuple(x)
print(y)
```

Answer

Tuple

1 UNIT

Introduction and Basics (2 Marks Questions)

1.1. What is Python ?

Ans. Python is a high-level, interpreted, interactive and object-oriented scripting language. It is a highly readable language. Unlike other programming languages, Python provides an interactive mode similar to that of a calculator.

1.2. What are the difference between Java and Python ?

Ans.

Features	Java	Python
Syntax	The syntax of Java is complex than Python.	The syntax of Python is easier than Java.
Speed	Java is statically typed programming, makes it faster.	Python is manually typed, makes it slower.
Code	Longer lines of code than Python.	Shorter lines of code than Java.

1.3. What are the features of Python ?

Ans. **Features of Python :**

1. The code written in Python is automatically compiled to byte code and executed.
2. Python can be used as a scripting language, as a language for implementing web applications, etc.
3. Python supports many features such as nested code blocks, functions, classes, modules and packages.
4. Python makes use of an object oriented programming approach.
5. It has many built-in data types such as strings, lists, tuples, dictionaries, etc.

1.4. What are the different ways of starting Python ?

Ans. There are three different ways of starting Python :

1. Running a script written in Python.

2. Using a graphical user interface (GUI) from an Integrated Development Environment (IDE),
3. Employing an interactive approach.

1.5. Which character is used for commenting in Python ?

Ans. Hash mark (#) is used for commenting in Python.

1.6. How is Python an interpreted language ?

AKTU 2019-20, Marks 02

Ans. Python is called an interpreted language because it goes through an interpreter, which turns the Python code into the language understood by processor of the computer.

1.7. What type of language is python ?

AKTU 2019-20, Marks 02

Ans. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

1.8. Define the type () function.

Ans. Type () function in Python programming language is a built-in function which return the datatype of any arbitrary object. The object is passed as an argument to the type() function. Type() function can take anything as an argument and return its datatype, such as integers, strings, dictionaries, lists, classes, module, tuple, functions, etc.

1.9. Define the unary operator.

Ans. Unary operators are operators with only one operand. These operators are basically used to provide sign to the operand.
+, –, ~ are some unary operators.

1.10. What do you mean by binary operator ?

Ans. Binary operators are operators with two operands that are manipulated to get the result. They are also used to compare numeric values and string values.
**, %, <<, >>, &, |, ^, <, >, <=, >=, ==, !=, <> are some binary operators.

1.11. What is the purpose of PYTHONPATH environment variable ?

Ans. Pythonpath has a role similar to PATH. This variable tells Python Interpreter where to locate the module files imported into a program. It should include Python source library directory and the directories containing Python source code. PYTHONPATH is sometimes preset by Python Installer.

1.12. What is the difference between list and tuples in Python ?

AKTU 2019-20, Marks 02

Ans.

S. No.	List	Tuples
1.	Lists are mutable, *i.e.*, they can be edited.	Tuples are immutable (they are lists that cannot be edited).
2.	Lists are usually slower than tuples.	Tuples are faster than lists.
3.	**Syntax :** list_ 1 = [10, 'Quantum', 20]	**Syntax :** tup_ 1 = (10, 'Quantum', 20)

1.13. What is the difference between Python arrays and lists.

AKTU 2019-20, Marks 02

Ans.

S. No.	Arrays	Lists
1.	Arrays can only store homogeneous data (data of the same type).	Lists can store heterogeneous and arbitrary data.
2.	Arrays use less memory to store data.	Lists require more memory to store data.
3.	The length of an array is pre-fixed while creating it, so more elements cannot be added.	The length of a list is not fixed, so more elements can be added.

1.14. Can we make multiline comments in Python ?

Ans. Python does not have a specific syntax for including multiline comments like other programming languages. However, programmers can use triple-quoted strings (docstrings) for making multiline comments as when a docstring is not used as the first statement inside a method, it is ignored by Python parser.

1.15. Do we need to declare variables with data types in Python ?

Ans. No. Python is a dynamically typed language, *i.e.*, Python Interpreter automatically identifies the data type of a variable based on the type of value assigned to the variable.

1.16. In some languages, every statement ends with a semi-colon (;). What happens if you put a semi-colon at the end of a Python statement ? **AKTU 2019-20, Marks 02**

Ans. Python allows semicolon to use as a line terminator. So no error will occur if we put a semi-colon; at the end of python statement.

1.17. Mention five benefits of using Python.

AKTU 2019-20, Marks 02

Ans. **Benefits of Python :**

1. Python is easy to learn.
2. Most automation, data mining, and big data platforms depend on Python. This is because it is the ideal language to work with for general purpose tasks.
3. Python provides productive coding environment.
4. It supports extensive libraries.
5. Python uses different frameworks to simplify the development process.

1.18. Define floor division with example.

AKTU 2019-20, Marks 02

Ans. Floor division returns the quotient in which the digits after the decimal point are removed. But if one of the operands (dividend and divisor) is negative, then the result is floored, i.e., rounded away from zero (means, towards the negative of infinity). It is denoted by "//".

For example :

```
5.0 // 2
2.0
```

1.19. List some Python IDEs.

Ans. **Some Python IDEs are :**

1. PyCharm
2. Spyder
3. PyDev

1.20. Mention some of the reserved keyword in Python.

Ans.

1. and
2. false
3. is
4. pass
5. return
6. def

1.21. Give an example of assigning one variable value to another.

Ans.
```
>>> name1 = 'Albert'
>>> name2 = name1
>>> name2
'Albert' # Output
>>>
```

1.22. Give an example of different types of values to the same variable.

Ans.
```
>>> amount = 50
>>> amount
50 # Output
>>> amount = 'Fifty'
>>> amount
'Fifty' # Output
>>>
```

1.23. What are the types of type conversion ?

Ans. **Two types of type conversion are :**

1. Implicit type conversion
2. Explicit type conversion

1.24. What are types of assignment statements ?

Ans. **Three type of assignment statements are :**

1. Value-based expression on RHS
2. Current variable on RHS
3. Operation on RHS

1.25. List the categories of operators.

Ans. **Following are the seven categories of operators :**

1. Arithmetic operators.
2. Assignment operators.
3. Bitwise operators.
4. Comparison operators.
5. Identity operators.
6. Logical operators.
7. Membership operators.

1.26. Name the tools that are used for static analysis.

Ans.

1. Pychecker
2. Pylint

1.27. What are the different data types used in Python ?

Ans. **Python has six basic data types which are as follows :**

1. Numeric

2. String
3. List
4. Tuple
5. Dictionary
6. Boolean

1.28. Define operator associativity with its type.

Ans.

1. Associativity decides the order in which the operators with same precedence are executed.
2. There are two types of associativity :
   - **a. Left to right :** In left to right associativity, the operator of same precedence are executed from the left side first.
   - **b. Right to left :** In right to left associativity, the operator of same precedence are executed from the right side first.

2 UNIT

Conditionals and Loops (2 Marks Questions)

2.1. What are the factors for expression evaluation ?

Ans.

1. **Precedence :** It is applied to two different class of operators. That is, + and*, – and *, AND & OR, etc.
2. **Associativity :** It is applied to operators of same class. That is, * and *, + and –, * and /, etc.
3. **Order :** Precedence and associativity identify the operands for each operator. While evaluating an assignment, the RHS is evaluated before LHS.

2.2. What is range () function ?

Ans. The range () function is a built-in function in Python that helps us to iterate over a sequence of numbers. It produces an iterator that follows arithmetic progression.

2.3. Give an example of range () function.

Ans.
```
>>> range (8)
[0, 1, 2, 3, 4, 5, 6, 7]
```
range (8) provides a sequence of number 0-7. That is to say range (n) generates a sequence of number that starts with 0 and end with (n – 1).

2.4. Explain begin and end arguments passed by range () function.

Ans.
```
>>> range (3, 9)
[3, 4, 5, 6, 7, 8]
```
We provided the begin index with 3 and the end index with 9. Hence, the range function generates a sequence iterator of number that starts from 3 and ends at 8.

2.5. Define the term alternative execution.

Ans. The alternative execution provides two possibilities and the condition determines which one is to be executed. This is the second form of the if statement.

2.6. What do you mean by block ?

Ans. The intended statements that follow the conditional statements are called block. The first unintended statement marks the end of the block.

2.7. What will be the output of the following code :
Str [0 : 4] if str="Hello"

Ans. 'Hello'

2.8. What are control statements ?

Ans. A control statement is a statement that determines the control flow of a set of instructions. There are three fundamental forms of control that programming languages provide: sequential control, selection control, and iterative control.

2.9. What is short-circuit evaluation ?

Ans. In short-circuit (lazy) evaluation, the second operand of Boolean operators AND and OR is not evaluated if the value of the Boolean expression can be determined from the first operand alone.

2.10. Define the terms : header, suite and clause.

Ans. A header in Python starts with a keyword and ends with a colon. The group of statements following a header is called a suite. A header and its associated suite are together referred to as a clause.

2.11. What do you mean by iterative control ?

Ans. An iterative control statement is a control statement providing the repeated execution of a set of instructions. An iterative control structure is a set of instructions and the iterative control statement(s) controlling their execution.

2.12. What do you mean by definite loop ?

Ans. A definite loop is a program loop in which the number of times the loop will iterate can be determined before the loop is executed.

2.13. What do you mean by indefinite loop ?

Ans. An indefinite loop is a program loop in which the number of times that the loop will iterate cannot be determined before the loop is executed.

2.14. Is indentation optional in Python ?

Ans. No indentation in Python is compulsory and is part of its syntax. Indentation is a way of defining the scope and extent of the block of codes. Indentation provides better readability to the code.

2.15. What happen if break statement is used in for loop ?

Ans. If the break statement in a for loop is executed then the else part of that for loop is skipped.

2.16. What is raw_input ( ) function ?

Ans. Raw_input ( ) takes the input from the user but it does not interpret the input and also it returns the input of the user without doing any changes.

2.17. Differentiate fruitful functions and void functions.

AKTU 2019-20, Marks 02

Ans. The main difference between void and fruitful function in python is :

1. Void does not return any value
2. Fruitful function returns some value

3 UNIT

Functions and Strings (2 Marks Questions)

3.1. Define traversing of string. Give an example.

Ans. Traversal is a process in which we access all the elements of the string one by one using some conditional statements such as for loop, while loop, etc.

For example :

```
>>> var = 'jack john'
>>> i = 0
>>> while i < len (var) :
...    x = var [i]
...    print x
...    i = i + 1
```

Output :

```
j
a
c
k
j
o
h
n
```

3.2. What are escape characters ?

Ans. The backslash character (/) is used to escape characters. It converts difficult-to-type characters into a string. For example, we need the escaping character concept when we want to print a string with double quotes or single quotes. When single or double quotes are used with the string, Python normally neglects them and prints only the string.

3.3. What do you mean by tuple assignment ?

Ans. Tuple assignment allows the assignment of values to a tuple of variables on the left side of assignment from the tuple of values on the right side of the assignment.

3.4. What do you mean by "Lists are mutable" ?

Ans. Lists are mutable means that we can change the value of any elements inside the list at any point of time. The element inside the list are accessible with their index value. The index will always start with 0 and end with $n - 1$, if the list contains n elements.

3.5. What do you understand by traversing a list ?

Ans. Traversing of the list refers to accessing all the elements or items of the list. Traversing can be done using any conditional statement of Python, but it is preferable to use for loop.

3.6. What are the different methods used in deleting elements from dictionary ?

Ans. **Methods used in deleting elements from dictionary are :**

1. **pop( ) :** pop() method removes that item from the dictionary for which the key is provided. It also returns the value of the item.
2. **popitem( ) :** popitem() method is used to remove or delete and return an arbitrary item from the dictionary.
3. **clear( ) :** clear() method removes all the items or elements from a dictionary at the same time.

3.7. What are the two properties of key in the dictionary ?

Ans. **Properties of key :**

1. One key in a dictionary cannot have two values, *i.e.*, duplicate keys are not allowed in the dictionary; they must be unique.
2. Keys are immutable, *i.e.*, we can use string, integers or tuples for dictionary keys, but cannot use something like ['key'].

3.8. Why we use functions ?

Ans.

1. Break up complex problem into small sub-programs.
2. Solve each of the sub-problems separately as a function, and combine them together in another function.
3. Hide the details and shows the functionality.

3.9. What are mathematical functions ? How are they used in Python ?

Ans. Python provides us a math module containing most of the familiar and important mathematical functions. A module is a file that contain some predefine Python codes. A module can define functions, classes and variables. It is a collection of related functions grouped together.

Before using a module in Python, we have to import it

For example, to import the math module, we use :

```
>>> import math
```

3.10. What are user-defined functions ? Give the syntax.

Ans. Python also allows users to define their own functions. To use their own functions in Python, users have to define the functions first; this is known as function definition. In a function definition, users have to define a name for the new function and also the list of the statements that will execute when the function will be called.

Syntax :

```
def functionname (parameters) :
    "function_docstring"
    statement (s)
    return (expression)
```

3.11. Define the return statement in a function. Give the syntax.

Ans. The return statement is used to exit a function. A function may or may not return a value. If a function returns a value, it is passed back by the return statement as argument to the caller. If it does not return a value, we simply write return with no arguments.

Syntax :

```
return [expression]
```

3.12. Define anonymous function.

Ans. The anonymous functions are the functions created using a lambda keyword.

3.13. You have been given a string 'I live in Cochin. I love pets.' Divide this string in such a very that the two sentences in it are separated and stored in different variables. Print them.

Ans.

```
>>> var = 'I live in Cochin, I love pets.'
>>> var1 = var[: 17]
>>> var2 = var[18 : 30]
>>> print var1
I live in Cochin,      #   Output
>>> print var2
I love pets,           #   Output
```

3.14. An email address is provided : hello@python. org. Using tuple assignment, split the username and domain from the email address.

Ans.

```
>>> addr = 'hello@python. org'
>>> usrname, domain = addr. split ('@')
>>> print usrname
Hello                  #   Output
>>> print domain
python. org            #   Output
```

3.15. Write a function called sumall that takes any number of arguments and returns their sum.

Ans.
```
>>> def suma11 (*t) :
        i = 0
        sum = 0
        whilei < len(t) :
            sum = sum + t[i]
            i = i + 1
            return sum
>>>suma11(1, 2, 3, 4, 5, 6, 7)
28                  #  Output
```

3.16. Write a function called circleinfo which takes the radius of circle as argument and returns the area and circumference of the circle.

Ans.
```
>>>def circleinfo(r) :
        c = 2 * 3.14159 * r
        a = 3.14159  *  r  *  r
        return (c, a)
>>> circleinfo(10)
(62.8318, 314.159)  #  Output
```

3.17. Give examples for len, max and min methods.

Ans.
```
>>> list = [789, 'abcd', 'jinnie', 1234]
>>> len(list)
4                   #  Output
>>>  max(list)
'jinnie'            #  Output
>>>min(list)
789                 #  Output
```

3.18. Give examples for all, any, len and sorted methods in dictionary.

Ans.
```
>>> dict1 = {8 : 'a', 3 : 'b', 5 : 'c', 7 : 'd'}
>>> all (dict1)
True                #  Output
>>> any(dict1)
True                #  Output
>>> len (dict1)
4                   #  Output
>>> sorted(dict1)
[3, 5, 7, 8]        #  Output
```

3.19. Give the syntax required to convert an integer number into string and a float to an integer.

Ans.
```
# integer to string
>>>str (5)
'5'                 #  Output
                    #  float to integer
```

```
>>>float (5.50)
5                          # Output
```

3.20. Write a program to print the calendar for the month of March 1991.

Ans.
```
>>> import calendar
>>> c = calender.month(1991, 3)
>>> print c
```
March 1991

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

3.21. Write a function which accepts two numbers and returns their sum.

Ans.
```
>>> def sum (arg1, arg2) :
        sum = arg1 + arg2
        return sum
                           # Now calling the function here
>>> a = 4
>>> b = 3
>>> total = sum(a, b)      #   calling the sum function
>>> print(total)
7                          # Output
```

3.22. What are the types of arguments used for calling a function ?

Ans. Four types of arguments used for calling a function :

i. Required argument
ii. Keyword argument
iii. Default argument
iv. Variable length argument

3.23. Give one-one example for zip, max and min methods.

Ans.
```
>>> tuple1 = ('a', 'b', 'c')
>>> tuple1 = (1, 2, 3)
>>> max (tuple 2)
3                          #   Output
>>>min(tuple 1)
'a'                        #   Output
>>>zip(tuple 1, tuple 2)
{('a', 1), ('b', 2), ('c', 3)}   #   Output
```

3.23. What is the output of print list[2] when list = ['abcd', 2.23, 'john'] ?

Ans. john

3.24. What is slicing ?

Ans. In Python, we can extract a substring by using a colon inside the square bracket [:]. The resultant substring is a part of the long string.

3.25. What is the use of docstring ?

Ans. Docstring command is use to know about a function using triple-quoted string.

3.26. What are local variables and global variables in Python ?

AKTU 2019-20, Marks 02

Ans.

i. Global variables :

1. Global variables are the variables that are defined outside a function body.
2. Global variables can be accessed throughout the program body by all functions.

ii. Local variables :

1. Local variables are the variables that are defined inside a function body.
2. Local variables can be accessed only inside the function in which they are declared.
3. When we call a function, the variables declared inside it are brought into scope.

4 UNIT

Sieve of Eratosthenes and File I/O (2 Marks Questions)

4.1. What is class variable ?

Ans. A variable that is defined in the class and can be used by all the instances of that class is called class variable.

4.2. Define instance variable.

Ans. A variable that is defined in the method; its scope is only within the object that defines it.

4.3. Define the term instance.

Ans. An object is an instance of the class.

4.4. What do you mean by instantiation ?

Ans. The process of creating an object is called instantiation.

4.5. Define function overloading.

Ans. A function defined more than one time with different behaviours is called function overloading.

4.6. What do you mean by methods ?

Ans. Methods are the functions defined in the definition of class and are used by various instances of the class.

4.7. How are the objects created in Python ? Give an example.

Ans. Objects can be used to access the attributes of the class. The syntax for creating an object in Python is similar to that for calling a function.

Syntax :

obj_name = class_name ()

For example :

```
# define a class
>>>class A :
def print_det (self) :
      print 'This is a class'
```

```
# create object of class A
>>> object = A()
>>> object. print_det()
This is a class    #    Output
```

4.8. What do you understand by "Objects are mutable" ?

Ans. Objects are mutable means that the state of an object can be changed at any point in time by making changes to its attributes.

4.9. What do you understand by arguments "Instances as return values" ?

Ans. The instances of a class can also be returned by a function *i.e.*, function can return the instances or objects.

4.10. Define__dict__, __bases__, __name__ built-in class attributes. Give example.

Ans. **__dict__ :** It displays the dictionary in which the namespace of class is stored.

__name__ : It displays the name of the class.

__bases__ : It displays the tuple that contains the base classes, possibly empty. It displays them in the order in which they occur in the base class list.

For example :

```
>>> print "__name__ :", PrintStatement.__name__
__name__ : PrintStatement
>>> print "__bases__ :", PrintStatement __bases__
__bases__ : ()
>>>print "PrintStatement".__dict__: "PrintStatment" __dict__
PrintStatement.__dict__ : {'__module__' : '__main__', '__doc__' :
None, 'print_method' : < function print_method at 0x.2CE3130>}
```

4.11. Give the advantage of inheritance.

Ans. The main advantage of inheritance is that the code can be written once in the base class and then reused repeatedly in the subclasses.

4.12. Define subclass.

Ans. The class which inherits the feature of another class is called subclass.

4.13. List the order of file operations in Python.

Ans. **The order is as follows :**

1. Opening a file
2. Perform read or write operation
3. Closing a file

4.14. Explain any four modes of opening the file.

Ans. **Modes of opening the file :**

i. ***r* :** It opens a file in reading mode. The file pointer is placed at the starting of the file.

ii. ***r +* :** It opens the file in both reading and writing mode. The file pointer is placed at the starting of the file.

iii. ***w* :** It opens the file in writing mode. If a file exists, it overwrites the existing file; otherwise, it creates a new file.

iv. ***w +* :** It opens the file in both reading and writing mode. If a file exists, it overwrites the existing file; otherwise, it creates a new file.

4.15. Explain the file object attributes in detail.

Ans.

Attribute	Description
file.closed	It will return true if the file is closed ; it will otherwise return false.
file.mode	It will return the access mode with which the file is opened.
file.name	It will return name of the file
file.softspace	It will return false if space explicitly required with print; otherwise it will return true.

4.16. Give the syntax for reading from a file. What is the work of the readline() function ?

Ans. **Syntax :**

fileobject.read([size])

The count parameter size gives the number of bytes to be read from an opened file. It starts reading from the beginning of the file until the size given. If no size is provided, it ends up reading until the end of the file.

4.17. How are renaming and deleting performed on a file ? Give the syntax for each.

Ans. **Renaming a file :** Renaming a file in Python is done with the help of the rename() method. The rename() method is passed with two argument, the current filename and the new filename.

Syntax :

os.rename(current_filename, new_filename)

Deleting a file : Deleting a file in Python is done with the help of the remove() method. The remove() method takes the filename as an argument to be deleted.

Syntax :

os.remove(filename)

4.18. What are the various file positions methods ?

Ans. In Python, the tell() method tells us about the current position of the pointer. The current position tells us where reading will starts from at present.

We can also change the position of the pointer with the help of the seek() method. We pass the number of bytes to be moved by the pointer as arguments to the seek() method.

4.19. What are directories ?

Ans. If there is a large number of file, then related files are placed in different directories. Directory can be said to be a collection of files and sub directories. The module os in Python enables us to use various methods to work with directories.

4.20. What are the basic methods performed on directories ?

Ans. Following are the four basic methods that are performed on directories :

i. mkdir () method (Creating a directory)
ii. chdir() method (Changing the current directory)
iii. getcwd () method (Displaying the current directory)
iv. rmdir () method (Deleting the directory).

4.21. What are user-defined exceptions ? Give one example.

Ans. Python allows users to define their own exceptions by creating a new class. Exception needs to be derived, directly or indirectly from exception class.

For example :

```
>>> class error(Exception)
      pass
```

4.22. Write some built-in exception in Python.

Ans.

i. AssertionError
ii. FloatingPointError
iii. SystemError
iv. RunTimeError
v. ZeroDivisionError

4.23. Define ADT interface. **AKTU 2019-20, Marks 02**

Ans. ADT interface only define as what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation-independent view.

5 UNIT

Iterators and Recursion (2 Marks Questions)

5.1. What are properties of recursive functions ?

Ans. **Properties of recursive function :**

1. The arguments of function change between the recursive calls.
2. The change in arguments should be toward a case for which the solution is known and we call it as a base case. There can be more than one base case.

5.2. What are the advantages of recursion ?

Ans.

1. It requires few variables.
2. The programs are easy to implement if the problem has a recursive definition.

5.3. Give some disadvantages of recursion.

Ans.

1. Debugging is difficult.
2. It is not easy to write the program in a recursive form.
3. It can be inefficient as it requires more time and space.

5.4. What are the applications of Tower of Hanoi problem ?

Ans.

1. The Tower of Hanoi is frequently used in psychological research on problem solving.
2. There also exists a variant of this task called Tower of London for neuropsychological diagnosis and treatment of executive functions.
3. The Tower of Hanoi is also used as a Backup rotation scheme when performing computer data Backups where multiple tapes/media are involved.
4. The Tower of Hanoi is also used as a test by neuropsychologists trying to evaluate frontal lobe deficits.

5.5. What are the advantages and drawbacks of simple search ?

Ans. **Advantages :**

1. It is a very simple search and easy to program.
2. In the best-case scenario, the item we are searching for may be at the start of the list in which case we get it on the very first try.

Drawbacks :

1. Its drawback is that if our list is large, it may take time to go through the list.
2. In the worst-case scenario, the item we are searching for may not be in the list, or it may be at the opposite end of the list.

5.6. Write algorithm of simple search.

Ans.

```
if start > end:
    return False
if a[start]==key:
    return True
return search(a, start + 1, end, key)
```

5.7. Give the algorithm for binary search.

Ans.

```
if start > end :
    return False
mid = (start + end) //2
if a [mid] = = key:
    return True
if (a[mid] > key):
    return binsearch(a, start, mid – 1, key)
else :
    return binsearch(a, mid + 1, end, key)
```

5.8. Define the term sorting.

Ans.

1. Sorting is the arrangement of a given list in ascending order or descending order.
2. In sorting, searching for an element is very fast.
3. Example of sorting in real world are : Contact list in mobile phones, ordering marks before assignment of grades, etc.

5.9. Write the complexity of sorting algorithm.

Ans.

1. Merge sort : $O(n \log n)$
2. Selection sort : $O(n^2)$
3. Bouble sort : $O(n^2)$

5.9. Which operation is used to implement merge sort ?

Ans. Merge operation is used to implement merge sort.

5.11. List some sorting algorithm.

Ans.

1. Insertion sort
2. Selection sort
3. Bubble sort
4. Merge sort

5.12. What is binary search ?

Ans. Binary search is a search algorithm that finds the position of a target value within a sorted list.

5.13. Which method is used to sort a list ?

Ans. Sort( ) method is used to sort a list.

5.14. What are the possibilities while sorting a list of string ?

Ans. **Possibilities while sorting a list of string are :**

1. Sorting in alphabetical/reverse order.
2. Based on length of string character.
3. Sorting the integer values in list of string.

B.Tech.
(SEM. III) ODD SEMESTER THEORY EXAMINATION, 2019-20
PYTHON PROGRAMMING

Time : 3 Hours **Max. Marks : 100**

Note : 1. Attempt **all** Section.

Section-A

1. Answer **all** questions in brief. **(2 × 10 = 20)**

a. What is the difference between list and tuples in Python ?

b. In some languages, every statement ends with a semi-colon (;). What happens if you put a semi-colon at the end of a Python statement ?

c. Mention five benefits of using Python.

d. How is Python an interpreted language ?

e. What type of language is python ?

f. What are local variables and global variables in Python ?

g. What is the difference between Python Arrays and lists ?

h. Define ADT interface.

i. Define floor division with example.

j. Differentiate fruitful functions and void functions.

Section-B

2. Answer any **three** of the following : **(3 × 10 = 30)**

a. Explain iterator. Write a program to demonstrate the Tower of Hanoi using function.

b. Discuss function in Python with its parts and scope. Explain with example. (Take simple calculator with add, subtract, division and multiplication).

c. Discuss ADT in Python. How to define ADT ? Write code for a student information.

d. Explain all the conditional statement in Python using small code example.

e. What is Python? How Python is interpreted? What are the tools that help to find bugs or perform static analysis? What are Python decorators?

Section-C

3. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Write short notes with example : The programming cycle for Python, elements of Python, type conversion in Python, operator precedence, and Boolean expression.

b. How memory is managed in Python? Explain PEP 8. Write a Python program to print even length words in a string.

4. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Explain expression evaluation and float representation with example. Write a Python program for how to check if a given number is Fibonacci number.

b. Explain the purpose and working of loops. Discuss break and continue with example. Write a Python program to convert time from 12 hour to 24-hour format.

5. Answer any **one** part of the following : **(10 × 1 = 10)**

a. Explain higher order function with respect to lambda expression. Write a Python code to count occurrences of an element in a list.

b. Explain unpacking sequences, mutable sequences, and list comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.

6. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Discuss File I/O in Python. How to perform open, read, write, and close into a file ? Write a Python program to read a file line-by-line store it into a variable.

b. Discuss exceptions and assertions in Python. How to handle exceptions with try-finally ? Explain five built-in exceptions with example.

7. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Discuss and differentiate iterators and recursion. Write a program for recursive Fibonacci series.

b. Discuss sorting and merging. Explain different types of sorting with example. Write a Python program for Sieve of Eratosthenes.

☺☺☺

SOLUTION OF PAPER (2019-20)

Note : **1.** Attempt **all** Section.

Section-A

1. Answer **all** questions in brief. **(2 × 10 = 20)**

a. What is the difference between list and tuples in Python ?

Ans.

S. No.	List	Tuples
1.	Lists are mutable, *i.e.*, they can be edited.	Tuples are immutable (they are lists that cannot be edited).
2.	Lists are usually slower than tuples.	Tuples are faster than lists.
3.	**Syntax :** list_ 1 = [10, 'Quantum', 20]	**Syntax :** tup_ 1 = (10, 'Quantum', 20)

b. In some languages, every statement ends with a semi-colon (;). What happens if you put a semi-colon at the end of a Python statement ?

Ans. Python allows semicolon to use as a line terminator. So no error will occur if we put a semi-colon; at the end of python statement.

c. Mention five benefits of using Python.

Ans. **Benefits of Python :**

1. Python is easy to learn.
2. Most automation, data mining, and big data platforms depend on Python. This is because it is the ideal language to work with for general purpose tasks.
3. Python provides productive coding environment.
4. It supports extensive libraries.
5. Python uses different frameworks to simplify the development process.

d. How is Python an interpreted language ?

Ans. Python is called an interpreted language because it goes through an interpreter, which turns the Python code into the language understood by processor of the computer.

e. What type of language is python ?

Ans. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

f. What are local variables and global variables in Python ?

Ans.

i. Global variables :

1. Global variables are the variables that are defined outside a function body.
2. Global variables can be accessed throughout the program body by all functions.

ii. Local variables :

1. Local variables are the variables that are defined inside a function body.
2. Local variables can be accessed only inside the function in which they are declared.
3. When we call a function, the variables declared inside it are brought into scope.

g. What is the difference between Python Arrays and lists ?

Ans.

S. No.	Arrays	Lists
1.	Arrays can only store homogeneous data (data of the same type).	Lists can store heterogeneous and arbitrary data.
2.	Arrays use less memory to store data.	Lists require more memory to store data.
3.	The length of an array is pre-fixed while creating it, so more elements cannot be added.	The length of a list is not fixed, so more elements can be added.

h. Define ADT interface.

Ans. ADT interface only define as what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation-independent view.

i. Define floor division with example.

Ans. Floor division returns the quotient in which the digits after the decimal point are removed. But if one of the operands (dividend and divisor) is negative, then the result is floored, i.e., rounded away from zero (means, towards the negative of infinity). It is denoted by "//".

For example :

```
5.0 // 2
2.0
```

j. Differentiate fruitful functions and void functions.

Ans. The main difference between void and fruitful function in python is :

1. Void does not return any value
2. Fruitful function returns some value

Section-B

2. Answer any **three** of the following : **(3 × 10 = 30)**

a. Explain iterator. Write a program to demonstrate the Tower of Hanoi using function.

Ans.

1. An iterator is an object that contains a countable number of values.
2. An iterator is an object that can be iterated upon, meaning that we can traverse through all the values.
3. Python iterator, implicitly implemented in constructs like for-loops, comprehensions, and python generators.
4. Python lists, tuples, dictionary and sets are all examples of in-built iterators.
5. These types are iterators because they implement following methods :

   a. **__iter__ :** This method is called on initialization of an iterator. This should return an object that has a next() method.

   b. **next() (or __next__) :** The iterator next method should return the next value for the iterable. When an iterator is used with a 'for in' loop, the for loop implicitly calls next() on the iterator object. This method should raise a StopIteration to signal the end of the iteration.

Recursive Python function to solve Tower of Hanoi :

```
def TowerOfHanoi(n, from_rod, to_rod, aux_rod):
    if n==1:
    print "Move disk 1 from rod",from_rod,"to rod",to_rod
    return
TowerOfHanoi(n – 1, from_rod, aux_rod, to_rod)
print "Move disk",n,"from_rod",from_rod,"to rod",to_rod
TowerOfHanoi(n – 1, aux_rod, to_rod, from_rod)
n = 4
TowerOfHanoi(n, \'A\', \'C\', \'B\')
# A, C, B are the name of rods
```

Output :

Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

b. Discuss function in Python with its parts and scope. Explain with example. (Take simple calculator with add, subtract, division and multiplication).

Ans. Function :

1. Functions are self-contained programs that perform some particular tasks.
2. Once a function is created by the programmer for a specific task, this function can be called anytime to perform that task.
3. Each function is given a name, using which we call it. A function may or may not return a value.
4. There are many built-in functions provided by Python such as dir (), len ( ), abs ( ), etc.
5. Users can also build their own functions, which are called user-defined functions.

Parts of function : Function is defined by "def" keyword following by function name and parentheses.

Syntax of function definition : def function_name ( ) :

Syntax of functional call : function_name ( )

For example :

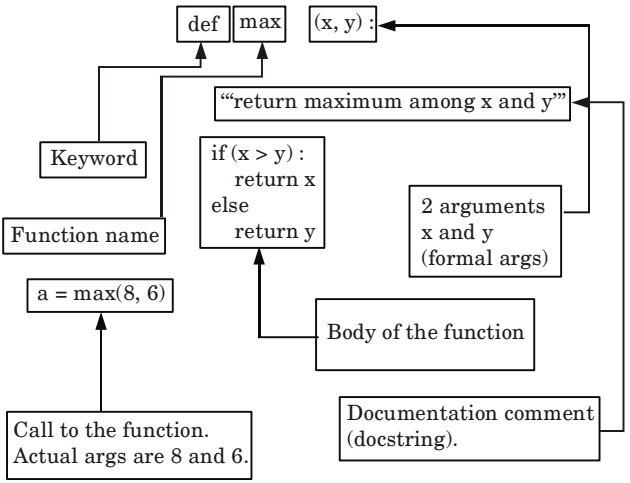


Fig. 2.

1. **Keyword :** The keyword 'def' is used to define a function header.
2. **Function name :** We define the function name for identification or to uniquely identify the function. In the given example, the function name is max. Function naming follows the same rules of writing identifiers in Python.
3. A colon (:) to mark the end of function header.
4. **Arguments :** Arguments are the values passed to the functions between parentheses. In the given example, two arguments are used, x and y. These are called formal arguments.
5. **Body of the function :** The body processes the arguments to do something useful. In the given example, body of the function is intended w.r.t. the def keyword.
6. **Documentation comment (docstring) :** A documentation string (docstring) to describe what the function does. In the given example, "return maximum among x and y" is the docstring.
7. An optional return statement to return a value from the function.
8. **Function call :** To execute a function, we have to call it. In the given example, a = max (8, 6) is calling function with 8 and 6 as arguments.

Scope :

1. Scope of a name is the part of the program in which the name can be used.

2. Two variables can have the same name only if they are declared in separate scopes.
3. A variable cannot be used outside its scopes.
4. Fig. 3 illustrates Python's four scopes.

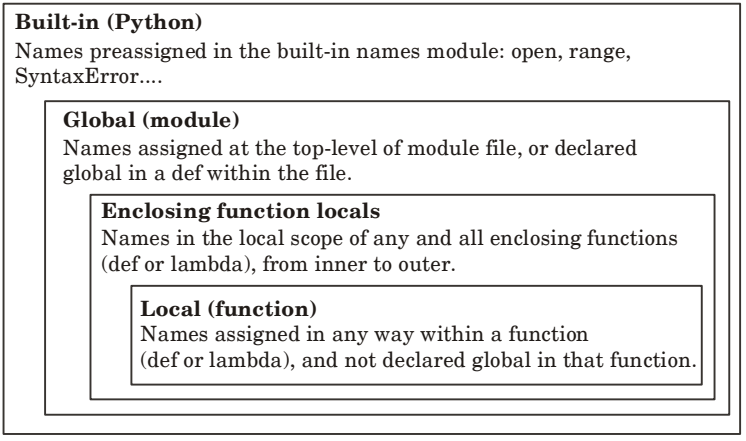


Fig. 3. The LEGB scope.

5. The LEGB rule refers to local scope, enclosing scope, global scope, and built-in scope.
6. Local scope extends for the body of a function and refers to anything indented in the function definition.
7. Variables, including the parameter, that are defined in the body of a function are local to that function and cannot be accessed outside the function. They are local variables.
8. The enclosing scope refers to variables that are defined outside a function definition.
9. If a function is defined within the scope of other variables, then those variables are available inside the function definition. The variables in the enclosing scope are available to statements within a function.

For example : Simple calculator using python :

```
# This function adds two numbers
def add(x, y) :
    return x + y
# This function subtracts two numbers
def subtract(x, y) :
    return x – y
# This function multiplies two numbers
def multiply(x, y) :
```

```
        return x * y
# This function divides two numbers
def divide(x, y) :
        return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
# Take input from the user
choice = input("Enter choice(1/2/3/4) : ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1' :
print(num1,"+",num2,"=", add(num1,num2))
elif choice == '2' :
print(num1,"–",num2,"=", subtract(num1,num2))
elif choice == '3' :
print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
print(num1,"/",num2,"=", divide(num1,num2))
else :
print("Invalid input")
```

c. Discuss ADT in Python. How to define ADT ? Write code for a student information.

Ans. **ADT in python :**

1. Abstract Data type (ADT) is a type for objects whose behaviour is defined by a set of value and a set of operations.
2. There are three types of ADTs :
   - **a. List ADT :** The data is stored in key sequence in a list which has a head structure consisting of count, pointers and address of compare function needed to compare the data in the list.
   - **b. Stack ADT :**
     - i. In stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
     - ii. The program allocates memory for the data and address is passed to the stack ADT.
     - iii. The head node and the data nodes are encapsulated in the ADT.
     - iv. The calling function can only see the pointer to the stack.
     - v. The stack head structure also contains a pointer to top and count of number of entries currently in stack.

c. Queue ADT :

i. The queue abstract data type (ADT) follows the basic design of the stack abstract data type.

ii. Each node contains a void pointer to the data and the link pointer to the next element in the queue.

iii. The program allocates the memory for storing the data.

The Queue and Stack are used to define Abstract Data Types (ADT) in Python.

Code for student information :

```
class Student :
# Constructor
def __init__(self, name, rollno, m1, m2):
    self.name = name
        self.rollno = rollno
        self.m1 = m1
        self.m2 = m2
# Function to create and append new student
def accept(self, Name, Rollno, marks1, marks2 ):
# use 'int(input())' method to take input from user
    ob = Student(Name, Rollno, marks1, marks2 )
    ls.append(ob)
# Function to display student details
def display(self, ob):
    print("Name  :", ob.name)
    print("RollNo :", ob.rollno)
    print("Marks1 :", ob.m1)
    print("Marks2 :", ob.m2)
    print("\n")
# Search Function
def search(self, rn):
    for i in range(ls.__len__()):
        if(ls[i].rollno == rn):
            return i
# Delete Function
def delete(self, rn):
    i = obj.search(rn)
    del ls[i]
# Update Function
def update(self, rn, No):
    i = obj.search(rn)
    roll = No
    ls[i].rollno = roll;
```

```
# Create a list to add Students
ls =[ ]
# an object of Student class
obj = Student(‘ ’, 0, 0, 0)
print(“\nOperations used, ”)
print(“\n1.Accept Student details\n2.Display Student Details\n”/
/“3.Search Details of a Student\n4.Delete Details of Student” /
/“\n5.Update Student Details\n6.Exit”)
ch = int(input(“Enter choice:”))
if(ch == 1):
    obj.accept(“A”, 1, 100, 100)
    obj.accept(“B”, 2, 90, 90)
    obj.accept(“C”, 3, 80, 80)
elif(ch == 2):
    print(“\n”)
    print(“\nList of Students\n”)
    for i in range(ls.__len__()):
        obj.display(ls[i])
elif(ch == 3):
    print(“\n Student Found,”)
    s = obj.search(2)
    obj.display(ls[s])
elif(ch == 4):
    obj.delete(2)
    print(ls.__len__())
    print(“List after deletion”)
    for i in range(ls.__len__()):
        obj.display(ls[i])
elif(ch == 5):
    obj.update(3, 2)
    print(ls.__len__())
    print(“List after updation”)
    for i in range(ls.__len__()):
        obj.display(ls[i])
else:
    print(“Thank You !”)
```

d. Explain all the conditional statement in Python using small code example.

Ans. Different types of conditional statement are :

1. If statement :

i. An if statement consists of a Boolean expression followed by one or more statements.

ii. With an if clause, a condition is provided; if the condition is true then the block of statement written in the if clause will be executed, otherwise not.

Syntax :

If (Boolean expression) : Block of code #Set of statements to execute if
the condition is true

For example :

```
var = 100
if ( var == 100 ) : print "value of expression is 100"
print "Good bye !"
```

Output :

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

2. If else statement :

i. An if statement can be followed by an optional else statement, which executes when the Boolean expression is False.

ii. The else condition is used when we have to judge one statement on the basis of other.

Syntax :

If (Boolean expression): Block of code #Set of statements to execute if
condition is true
else : Block of code #Set of statements to execute if condition is false

For example :

```
num = 5
if (num > 10) :
    print ("Number is greater than 10")
else :
    print ("Number is less than 10")
print ("This statement will always be executed")
```

Output :

```
Number is less than 10.
```

3. Nested-if statement :

i. Nested-if statements are nested inside other if statements. That is, a nested-if statement is the body of another if statement.

ii. We use nested if statements when we need to check secondary conditions only if the fist condition executes as true.

Syntax :

```
if test expression 1 :
#      executes when condition 1 is true
 body of if statement
if test expression 2 :
#      executes when condition 2 is true
Body of nested-if
else :
body of nested-if :
else :
body of if-else statement
```

For example :

```
a = 20
if (a == 20) :
# First if statement
if (a < 25) :
      print ("a is smaller than 25")
else :
      print ("a is greater than 25")
else :
      print ("a is not equal to 20")
```

Output :

a is smaller than 25

4. Elif statement :

i. Elif stands for else if in Python.

ii. We use elif statements when we need to check multiple conditions only if the given if condition executes as false.

For example :

```
a = 50
if (a == 29) :
      print ("value of variable a is 20")
elif (a == 30) :
      print ("value of variable a is 30")
elif (a == 40) :
      print ("value of variable a is 40")
else :
print ("value of variable a is greater than 40")
```

Output :

value of variable a is greater than 40

e. What is Python ? How Python is interpreted ? What are the tools that help to find bugs or perform static analysis ? What are Python decorators ?

Ans. **Python :** Python is a high-level, interpreted, interactive and object-oriented scripting language. It is a highly readable language. Unlike other programming languages, Python provides an interactive mode similar to that of a calculator.

Interpretation of Python :

1. An interpreter is a kind of program that executes other programs.
2. When we write Python programs, it converts source code written by the developer into intermediate language which is again translated into the machine language that is executed.
3. The python code we write is compiled into python bytecode, which creates file with extension .pyc.
4. The bytecode compilation happened internally and almost completely hidden from developer.
5. Compilation is simply a translation step, and byte code is a lower-level, and platform-independent, representation of source code.
6. Each of the source statements is translated into a group of bytecode instructions. This bytecode translation is performed to speed execution. Bytecode can be run much quicker than the original source code statements.
7. The .pyc file, created in compilation step, is then executed by appropriate virtual machines.
8. The Virtual Machine iterates through bytecode instructions, one by one, to carry out their operations.
9. The Virtual Machine is the runtime engine of Python and it is always present as part of the Python system, and is the component that actually runs the Python scripts.
10. It is the last step of Python interpreter.

Following tools are the static analysis tools that help to find bugs in python :

1. **Pychecker :** Pychecker is an open source tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.
2. **Pylint :**
   a. Pylint is highly configurable and it acts like special programs to control warnings and errors, it is an extensive configuration file.
   b. It is an open source tool for static code analysis and it looks for programming errors and is used for coding standard.
   c. It also integrates with Python IDEs such as Pycharm, Spyder, Eclipse, and Jupyter.

Python decorators :

1. Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class.

2. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.
3. In decorators, functions are taken as the argument into another function and then called inside the wrapper function.
4. **Syntax :**

```
@gfg_decorator
def hello_decorator():
    print("Gfg")
```

5. gfg_decorator is a callable function, will add some code on the top of some another callable function, hello_decorator function and return the wrapper function.

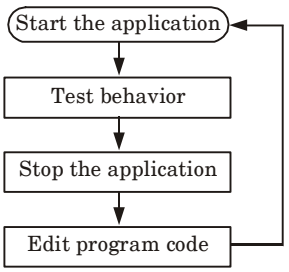
Section-C

3. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Write short notes with example : The programming cycle for Python, elements of Python, type conversion in Python, operator precedence, and Boolean expression.

Ans. Programming cycle for Python :

1. Python's programming cycle is dramatically shorter than that of traditional programming cycle.
2. In Python, there are no compile or link steps.
3. Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.
4. In cases where dynamic module reloading can be used, it is even possible to change and reload parts of a running program without stopping it at all.
5. Fig. 7 shows Python's impact on the programming cycle.



(a) Python's programming cycle

(b) Python's programming cycle with module reloading

Fig. 7.

6. Since Python is interpreted, there is a rapid turnaround after program changes. And because Python's parser is embedded in Python-based systems, it is easy to modify programs at runtime.

Elements of Python :

Data types :

i. The data stored in the memory can be of many types. For example, a person's name is stored as an alphabetic value and his address is stored as an alphanumeric value.

ii. Python has six basic data types which are as follows :

1. Numeric
2. String
3. List
4. Tuple
5. Dictionary
6. Boolean

Numeric :

1. Numeric data can be broadly divided into integers and real numbers (*i.e.*, fractional numbers). Integers can be positive or negative.
2. The real numbers or fractional numbers are called, floating point numbers in programming languages. Such floating point numbers contain a decimal and a fractional part.

For example :

```
>>> num1 = 2        # integer number
>>>num2 = 2.5      # real number (float)
>>>num1
2                   # Output
>>>num2
2.5                 # Output
>>>
```

String :

1. Single quotes or double quotes are used to represent strings.
2. A string in Python can be a series or a sequence of alphabets, numerals and special characters.

For example :

```
>>> sample_string = "Hello"    # store string value
>>> sample_string              # display string value
'Hello'                        # Output
```

List :

1. A list can contain the same type of items.
2. Alternatively, a list can also contain different types of items.
3. A list is an ordered and indexable sequence.
4. To declare a list in Python, we need to separate the items using commas and enclose them within square brackets ([ ]).
5. Operations such as concatenation, repetition and sub-list are done on list using plus (+), asterisk (*) and slicing (:) operator.

For example :

```
>>>first = [1, "two", 3.0, "four" ]    # 1st list
>>>second = ["five", 6]    # 2nd list
>>>first    # display 1st list
[1, 'two', 3.0, 'four']    # Output
```

Tuple :

1. A tuple is also used to store sequence of items.
2. Like a list, a tuple consists of items separated by commas.
3. Tuples are enclosed within parentheses rather than within square brackets.

For example :

```
>>>third = (7, "eight", 9, 10.0)
>>>third
(7, 'eight', 9, 10.0)    # Output
```

Dictionary :

1. A Python dictionary is an unordered collection of key-value pairs.
2. When we have the large amount of data, the dictionary data type is used.
3. Keys and values can be of any type in a dictionary.
4. Items in dictionary are enclosed in the curly-braces {} and separated by the comma (,).
5. A colon (:) is used to separate key from value. A key inside the square bracket [ ] is used for accessing the dictionary items.

For example :

```
>>> dict1 = {1:"first line", "second" : 2}    # declare dictionary
>>> dict1[3] = "third line"    # add  new item
>>> dict1 # display dictionary
{1 : 'first line', 'second' : 2, 3: 'third line'}    # Output
```

Boolean :

1. In a programming language, mostly data is stored in the form of alphanumeric but sometimes we need to store the data in the form of 'Yes' or 'No'.

2. In terms of programming language, Yes is similar to True and No is similar to False.
3. This True and False data is known as Boolean data and the data types which stores this Boolean data are known as Boolean data types.

For example :

```
>>> a = True
>>> type (a)
<type 'bool'>
```

Type conversion in Python :

1. The process of converting one data type into another data type is known as type conversion.
2. There are mainly two types of type conversion methods in Python :

a. Implicit type conversion :

i. When the data type conversion takes place during compilation or during the run time, then it called an implicit data type conversion.

ii. Python handles the implicit data type conversion, so we do not have to explicitly convert the data type into another data type.

For example :

```
a = 5
b = 5.5
sum = a + b
print (sum)
print (type (sum)) # type() is used to display the datatype o f a variable
```

Output :

```
10.5
<class 'float'>
```

iii. In the given example, we have taken two variables of integer and float data types and added them.

iv. Further, we have declared another variable named 'sum' and stored the result of the addition in it.

v. When we checked the data type of the sum variable, we can see that the data type of the sum variable has been automatically converted into the float data type by the Python compiler. This is called implicit type conversion.

b. Explicit type conversion:

i Explicit type conversion is also known as type casting.

ii. Explicit type conversion takes place when the programmer clearly and explicitly defines the variables in the program.

For example :

```
# adding string and integer data types using explicit type conversion
a = 100
b = "200"
result1 = a + b
b = int(b)
result2 = a + b
print (result2)
```

Output :

```
Traceback (most recent call last):
File "", line 1, in
TypeError : unsupported operand type (s) for +: 'int' and 'str' 300
```

iii. In the given example, the variable *a* is of the number data type and variable *b* is of the string data type.

iv. When we try to add these two integers and store the value in a variable named result1, a TypeError occurs. So, in order to perform this operation, we have to use explicit type casting.

v. We have converted the variable *b* into integer type and then added variable *a* and *b*. The sum is stored in the variable named result2, and when printed it displays 300 as output.

Operator precedence :

1. When an expression has two or more operator, we need to identify the correct sequence to evaluate these operators. This is because result of the expression changes depending on the precedence.

   For example : Consider a mathematical expression :

   10 + 5 / 5

   When the given expression is evaluated left to right, the final answer becomes 3.
2. However, if the expression is evaluated right to left, the final answer becomes 11. This shows that changing the sequence in which the operators are evaluated in the given expression also changes the solution.
3. Precedence is the condition that specifies the importance of each operator relative to the other.

Table 1. Operator precedence from lower precedence to higher.

Operator	Description
NOT, OR AND	Logical operators
in , not in	Membership operator
is, not is	Identity operator
=, %=, /=, //=, –=, +=, *=, **==	Assignment operators.
<>, ==, !=	Equality comparison operator
<=, <, >, >=	Comparison operators
^, |	Bitwise XOR and OR operator
&	Bitwise AND operator
<<, >>	Bitwise left shift and right shift
+, –	Addition and subtraction
*, /, %, ??	Multiplication, Division, Modulus and floor division
**	Exponential operator

Boolean expression : A boolean expression may have only one of two values : True or False.

For example : In the given example comparison operator (==) is used which compares two operands and prints true if they are equal otherwise print false :

```
>>> 5 == 5
True    # Output
>>> 5 == 6
False    # Output
```

b. How memory is managed in Python ? Explain PEP 8. Write a Python program to print even length words in a string.

Ans. Memory management :

1. Memory management in Python involves a private heap containing all Python objects and data structures.
2. The management of this private heap is ensured internally by the Python memory manager.
3. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.
4. At the lowest level, a raw memory allocator ensures that there is enough room in the private heap for storing all Python-related

data by interacting with the memory manager of the operating system.

5. On top of the raw memory allocator, several object-specific allocators operate on the same heap and implement distinct memory management policies adapted to the peculiarities of every object type.

PEP 8 :

1. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment.
2. The PEP should provide a concise technical specification of the feature.
3. PEP is actually an acronym that stands for Python Enhancement Proposal.
4. PEP 8 is Python's style guide. It is a set of rules for how to format the Python code to maximize its readability.
5. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment.

Program to print even length words in a string :

```
def printWords(s) :
# split the string
s = s.split(' ')
# iterate in words of string
for word in s:
# if length is even
if len(word)%2==0:
      print(word)
# Driver Code
s = "i am muskan"
printWords(s)
```

Output :

```
am
muskan
```

4. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Explain expression evaluation and float representation with example. Write a Python program for how to check if a given number is Fibonacci number.

Ans. **Expression evaluation :**

1. In Python actions are performed in two forms :
   a. Expression evaluation,
   b. Statement execution.

2. The key difference between these two forms is that expression evaluation returns a value whereas statement execution does not return any value.
3. A Python program contains one or more statements. A statement contains zero or more expressions.
4. Python executes a statement by evaluating its expressions to values one by one.
5. Python evaluates an expression by evaluating the sub-expressions and substituting their values.

For example :

```
>>> program = "Hello Python"
>>> program
'Hello Python'                    #Output
>>> print program
Hello Python                      #Output
```

6. An expression is not always a mathematical expression in Python. A value by itself is a simple expression, and so is a variable.
7. In the given example, we assigned a value "Hello Python" to the variable program. Now, when we type only program, we get the output 'Hello Python'. This is the term we typed when we assigned a value to the variable. When we use a print statement with program it gives the value of the variable *i.e.*, the value after removing quotes.

Float representation :

1. Floating point representations vary from machine to machine.
2. The float type in Python represents the floating-point number.
3. Float is used to represent real numbers and is written with a decimal point dividing the integer and fractional parts.
4. For example: 97.98, 32.3 + e18, – 32.54e100 all are floating point numbers.
5. Python float values are represented as 64-bit double-precision values.
6. The maximum value any floating-point number can be is approx 1.8×10^{308}.
7. Any number greater than this will be indicated by the string inf in Python.
8. Floating-point numbers are represented in computer hardware as base 2 (binary) fractions.
9. For example, the decimal fraction 0.125 has value 1/10 + 2/100 + 5/1000, and in the same way the binary fraction 0.001 has value 0/2 + 0/4 + 1/8.

For example : # Python code to demonstrate float values.

```
Print(1.7e308)
# greater than 1.8 * 10^308
# will print 'inf'
print(1.82e308)
```

Output :

```
1.7e+308
inf
```

Program to check if a given number is Fibonacci number :

```
import math
# A utility function that returns true if x is perfect square
def isPerfectSquare(x):
s = int(math.sqrt(x))
return s*s == x
# Returns true if n is a Fibonacci number, else false
def isFibonacci(n):
return isPerfectSquare(5*n*n + 4) or isPerfectSquare(5*n*n – 4)
# A utility function to test above functions
for i in range(1,6):
if (isFibonacci(i) == True):
    print i,"is a Fibonacci Number"
else:
    print i,"is a not Fibonacci Number"
```

Output :

```
1 is a Fibonacci Number
2 is a Fibonacci Number
3 is a Fibonacci Number
4 is a not Fibonacci Number
5 is a Fibonacci Number
```

b. Explain the purpose and working of loops. Discuss break and continue with example. Write a Python program to convert time from 12 hour to 24-hour format.

Ans. **Purpose and working of loops :**

1. A loop is a programming structure that repeats a sequence of instructions until a specific condition is met.
2. A loop statement allows us to execute a statement or group of statements multiple times.
3. Python programming language provides following types of loops to handle looping requirements:
   a. For
   b. While
   c. Nested
4. **Purpose :** The purpose of loops is to repeat the same, or similar, code a number of times. This number of times could be specified to

a certain number, or the number of times could be dictated by a certain condition being met.

5. **Working :** Consider the flow chart for a loop execution :

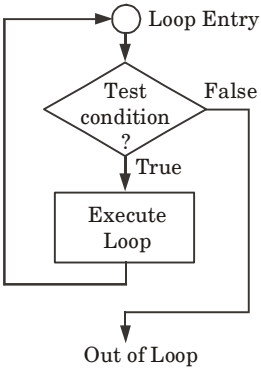


Fig. 8.

a. In the flow chart if the test condition is true, then the loop is executed, and if it is false then the execution breaks out of the loop.

b. After the loop is successfully executed the execution again starts from the loop entry and again checks for the test condition, and this keeps on repeating until the condition is false.

Break statement :

1. The break keyword terminates the loop and transfers the control to the end of the loop.
2. While loops, for loops can also be prematurely terminated using the break statement.
3. The break statement exits from the loop and transfers the execution from the loop to the statement that is immediately following the loop.

For example :

```
>>> count = 2
>>> while True :
        print count
        count = count + 2
        if count > = 12 :
            break    # breaks the loop
```

Output :

```
2
4
```

6

8

10

Continue statement :

1. The continue statement causes execution to immediately continue at the start of the loop, it skips the execution of the remaining body part of the loop.
2. The continue keyword terminates the ongoing iteration and transfers the control to the top of the loop and the loop condition is evaluated again. If the condition is true, then the next iteration takes place.
3. Just as with while loops, the continue statement can also be used in Python for loops.

For example :

```
>>> for i in range (1, 10) :
        if i % 2! = 0 :
            continue # if condition becomes true, it skips the print part
            print i
```

Output :

2

4

6

8

Program to convert time format :

```
# Function to convert the time format
def convert24(str1):
# Checking if last two elements of time # is AM and first two
    elements are 12
    if str1[– 2:] == “AM” and str1[:2] == “12”:
    return “00” + str1[2:– 2]
# remove the AM
    elif str1[-2:] == “AM”:
        return str1[:– 2]
# Checking if last two elements of time is PM and first two elements
    are 12
    elif str1[– 2:] == “PM” and str1[:2] == “12”:
        return str1[:– 2]
    else:
# add 12 to hours and remove PM
    return str(int(str1[:2]) + 12) + str1[2:8]
# Driver Code
print(convert24(“08:05:45 PM”))
```

5. Answer any **one** part of the following : **(10 × 1 = 10)**

a. Explain higher order function with respect to lambda expression. Write a Python code to count occurrences of an element in a list.

Ans.

1. Reduce(), filter(), map() are higher order functions used in Python.
2. Lambda definition does not include a "return" statement, it always contains an expression which is returned.
3. We can also put a lambda definition anywhere a function is expected, and we do not have to assign it to a variable at all.
4. Lambda functions can be used along with built-in higher order functions like filter(), map() and reduce().

Use of lambda with filter() :

1. The filter() function in Python takes in a function and a list as arguments.
2. This function helps to filter out all the elements of a sequence "sequence", for which the function returns true.

For example : Python program that returns the odd numbers from an input list :

```
# Python code to illustrate filter() with lambda
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambdax: (x%2!=0), li))
print(final_list)
```

Output :

[5, 7, 97, 77, 23, 73, 61]

Use of lambda() with reduce() :

1. The reduce() function in Python takes in a function and a list as argument.
2. The function is called with a lambda function and a list and a new reduced result is returned. This performs a repetitive operation over the pairs of the list.
3. This is a part of functools module.

For example :

```
# Python code to illustrate reduce() with lambda()  to get sum of a list from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y : x + y), li)
print (sum)
```

Output :

193

Here the results of previous two elements are added to the next element and this goes on till the end of the list like (((((5+8)+10)+20)+50)+100).

Program to count occurrences of an element in a list :

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
# count element 'i'
count = vowels.count('i')
# print count
print('The count of i is:', count)
# count element 'p'
count = vowels.count('p')
# print count
print('The count of p is:', count)
```

Output :

The count of i is: 2

The count of p is: 0

b. Explain unpacking sequences, mutable sequences, and list comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.

Ans.

A. Unpacking sequences :

1. Unpacking allows to extract the components of the sequence into individual variable.
2. Several different assignments can be performed simultaneously.
3. We have multiple assignments in Python where we can have multiple LHS assigned from corresponding values at the RHS. This is an example of unpacking sequence.
4. There is one restriction, the LHS and RHS must have equal length. That is, every value that is created at RHS should be assigned to LHS.
5. Strings and tuples are example of sequences. Operations applicable on sequences are: Indexing, repetition, concatenation.

For example :

```
>>> student
('Aditya', 27, ('Python', 'Abha', 303))
>>> name, roll, regdcourse = student
>>> name
```

Output : Aditya

```
>>> roll
27
>>> regdcourse
('Python', 'Abha', 303)
```

B. Mutable sequences :

1. Python represents all its data as objects. Mutability of object is determined by its type.
2. Some of these objects like lists and dictionaries are mutable, meaning we can change their content without changing their identity.
3. Other objects like integers, floats, strings and tuples are immutable, meaning we cannot change their contents.
4. **Dictionaries are mutable in Python :**
   a. Dictionaries in Python are mutable.
   b. The values in a dictionary can be changed, added or deleted.
   c. If the key is present in the dictionary, then the associated value with that key is updated or changed; otherwise a new key : value pair is added.

For example :

```
>>> dict1 = {'name' : 'Akash', 'age' : 27}
>>> dict1['age'] = 30                    # updating a value
>>> print dict
{'age' : 30, 'name': 'Akash'}            # Output
>>> dict1['address'] = 'Alaska'          # adding a key : value
>>>print dict1
{'age': 30, 'name': 'Akash', 'address': 'Alaska'}   # Output
```

In the given example, we tried to reassign the value '30' to the key 'age', Python interpreter first searches the key in the dictionary and then update it. Hence, the value of 'age' is updated to 30. However, in the next statement, it does not find the key 'address'; hence, the key: value 'address' : 'Alaska' is added to the dictionary.

C. List comprehension :

1. List comprehension is used to create a new list from existing sequences.
2. It is a tool for transforming a given list into another list.
3. Using list comprehension, we can replace the loop with a single expression that produces the same result.
4. The syntax of list comprehension is based on set builder notation in mathematics.

5. Set builder notation is a notation is a mathematical notation for describing a set by stating the property that its members should satisfy. The syntax is

   [<expression> for <element> in <sequence> if <conditional>]

   The syntax is read as "Compute the expression for each element in the sequence, if the conditional is true".

   For example :

```
>>> List1 = [10, 20, 30, 40, 50]
>>> List1
        [10, 20, 30, 40, 50]
>>> for i in range (0, len(List1)) :
        List1 [i] = List1[i] + 10
>>> List1
 [20, 30, 40, 50, 60]
```

Without list comprehension

```
>>> List1 = [10, 20, 30, 40, 50]
>>> List1= [x + 10 for x in List1]
>>> List1
 [20, 30, 40, 50, 60]
```

Using list comprehension

5. In the given example, the output for both without list comprehension and using list comprehension is the same.
6. The use of list comprehension requires lesser code and also runs faster.
7. From above example we can say that list comprehension contains :
   a. An input sequence
   b. A variable referencing the input sequence
   c. An optional expression
   d. An output expression or output variable

Program :

```
# Initializing list of dictionaries
lis = [{"name" : "Nandini", "age" : 20},
{"name" : "Manjeet", "age" : 20 },
{"name" : "Nikhil" , "age" : 19 }]
# using sorted and lambda to print list sorted by age
print "The list printed sorting by age :"
print sorted(lis, key = lambda i: i['age'])
print ("\r")
# using sorted and lambda to print list sorted by both age and name
print "The list printed sorting by age and name:"
print sorted(lis, key = lambda i: (i['age'], i['name']))
print ("\r")
# using sorted and lambda to print list sorted
# by age in descending order
```

```
print "The list printed sorting by age in descending order:"
print sorted(lis, key = lambda i: i['age'],reverse=True)
```

Output :

```
The list printed sorting by age:
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}]
The list printed sorting by age and name :
[{'age': 19, 'name': 'Nikhil'}, {'age': 20, 'name': 'Manjeet'}, {'age': 20, 'name': 'Nandini'}]
The list printed sorting by age in descending order:
[{'age': 20, 'name': 'Nandini'}, {'age': 20, 'name': 'Manjeet'}, {'age': 19, 'name': 'Nikhil'}]
```

6. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Discuss File I/O in Python. How to perform open, read, write, and close into a file ? Write a Python program to read a file line-by-line store it into a variable.

Ans. **File I/O :**

1. A file in a computer is a location for storing some related data.
2. It has a specific name.
3. The files are used to store data permanently on to a non-volatile memory (such as hard disks).
4. As we know, the Random Access Memory (RAM) is a volatile memory type because the data in it is lost when we turn off the computer. Hence, we use files for storing of useful information or data for future reference.

A. Open a file :

1. Python has a built-in open () function to open files from the directory.
2. Two arguments that are mainly needed by the open () function are :
   - **a. File name :** It contains a string type value containing the name of the file which we want to access.
   - **b. Access_mode :** The value of access_mode specifies the mode in which we want to open the file, *i.e.*, read, write, append etc.
3. **Syntax :**

   file_object = open(file_name [, access_mode])

B. Read a file :

1. In order to read from a file, we must open the file in the reading mode (*r* mode).
2. We can use read (size) method to read the data specified by size.

3. If no size is provided, it will end up reading to the end of the file.
4. The read() method enables us to read the strings from an opened file.
5. **Syntax :**

   file object. read ([size])

C. Write into a file :

1. After opening a file, we have to perform some operations on the file. Here we will perform the write operation.
2. In order to write into a file, we have to open it with *w* mode or *a* mode, on any writing-enabling mode.
3. We should be careful when using the *w* mode because in this mode overwriting persists in case the file already exists.

   For example :

   # open the file with w mode

   >>> f = open (“C :/Python27/test.txt”, “w”)

   # perform write operation

   >>>f. write (‘writing to the file line 1/n’)

   # clos the file after writing

   >>> f.close ()

D. Close a file :

1. When the operations that are to be performed on an opened file are finished, we have to close the file in order to release the resources.
2. Python comes with a garbage collector responsible for cleaning up the unreferenced objects from the memory, we must not rely on it to close a file.
3. Proper closing of a file frees up the resources held with the file.
4. The closing of file is done with a built-in function close ().
5. **Syntax :**

   fileObject. close ()

   Program to read a file line-by-line :

```
L = [“Quantum\n”, “for\n”, “Students\n”]
# writing to file
file1 = open(‘myfile.txt’, ‘w’)
file1.writelines(L)
file1.close()
# Using readlines()
file1 = open(‘myfile.txt’, ‘r’)
```

```
Lines = file1.readlines()
count = 0
# Strips the newline character
for line in Lines:
print(line.strip())
print("Line{}: {}".format(count, line.strip()))
```

Output :

Line1: Quantum
Line2: for
Line3: Students

b. Discuss exceptions and assertions in Python. How to handle exceptions with try-finally ? Explain five built-in exceptions with example.

Ans. **Exception :**

1. While writing a program, we often end up making some errors. There are many types of error that can occur in a program.
2. The error caused by writing an improper syntax is termed syntax error or parsing error; these are also called compile time errors.
3. Errors can also occur at runtime and these runtime errors are known as exceptions.
4. There are various types of runtime error in Python.
5. For example, when a file we try to open does not exist, we get a FileNotFoundError. When a division by zero happens, we get a ZeroDivisionError. When the module we are trying to import does not exist, we get an ImportError.
6. Python creates an exception object for every occurrence of these run-time errors.
7. The user must write a piece of code that can handle the error.
8. If it is not capable of handling the error, the program prints a trace back to that error along with the details of why the error has occurred.

Assertions :

1. An assertion is a sanity-check that we can turn on or turn off when we are done with our testing of the program. An expression is tested, and if the result is false, an exception is raised.
2. Assertions are carried out by the assert statement.
3. Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.
4. An AssertionError exception is raised if the condition evaluates to false.

5. The syntax for assert is : assert Expression [, Arguments]
6. If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError.

Handle exceptions :

1. Whenever an exception occurs in Python, it stops the current process and passes it to the calling process until it is handled.
2. If there is no piece of code in our program that can handle the exception, then the program will crash.
3. For example, assume that a function *X* calls the function *Y*, which in turn calls the function *Z*, and an exception occurs in *Z*. If this exception is not handled in *Z* itself, then the exception is passed to *Y* and then to *X*. If this exception is not handled, then an error message will be displayed and our program will suddenly halt.

try finally :

a. The try statement in Python has optional finally clause that can be associated with it.

b. The statements written in finally clause will always be executed by the interpreter, whether the try statement raises an exception or not.

c. With the try clause, we can use either except or finally, but not both.

d. We cannot use the else clause along with a finally clause.

Five built-in exceptions :

1. exception LookupError : This is the base class for those exceptions that are raised when a key or index used on a mapping or sequence is invalid or not found. The exceptions raised are :

a. KeyError

b. IndexError

For example :

```
try:
a = [1, 2, 3]
print a[3]
except LookupError :
    print “Index out of bound error.”
else:
    print “Success”
```

2. TypeError : TypeError is thrown when an operation or function is applied to an object of an inappropriate type.

For example :

```
>>> '2'+2
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

3. **exception ArithmeticError :** This class is the base class for those built-in exceptions that are raised for various arithmetic errors such as :
   a. OverflowError
   b. ZeroDivisionError
   c. FloatingPointError

   For example :

```
>>> x=100/0
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
x=100/0
ZeroDivisionError: division by zero
```

4. **exception AssertionError :** An AssertionError is raised when an assert statement fails.

   For example :

```
assert False, 'The assertion failed'
```

5. **exception AttributeError :**

   An AttributeError is raised when an attribute reference or assignment fails such as when a non-existent attribute is referenced.

   For example :

```
class Attributes(object):
pass
object = Attributes()
print object.attribute
```

7. Answer any **one** part of the following : **(1 × 10 = 10)**

a. Discuss and differentiate iterators and recursion. Write a program for recursive Fibonacci series.

Ans.

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instruction repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size need to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code size	Smaller code size.	Larger code size.
Time Complexity	Very high (generally exponential) time complexity.	Relatively lower time complexity (generally polynomial logarithmic).
Stack	The stack is used to store the set of new local variables and parameters each time the function is called.	Does not use stack.
Overhead	Recursion possesses the overhead of repeated function calls.	No overhead of repeated function call.
Speed	Slow in execution.	Fast in execution.

Program for recursive Fibonacci series :

```
def FibRecursion(n) :
    if n <= 1 :
        return n
    else :
        return(FibRecursion(n – 1) + FibRecursion(n – 2))
nterms = int(input("Enter the term : ")) # take input from the user
if nterms < = 0: # check if the number is valid
    print ("Please enter a positive integer")
else :
    print ("Fibonacci sequence :")
    for i in range (nterms) :
        print(FibRecursion(i))
```

b. Discuss sorting and merging. Explain different types of sorting with example. Write a Python program for Sieve of Eratosthenes.

Ans. **Sorting :**

1. Sorting refers to arranging data in a particular order.
2. Most common orders are in numerical or lexicographical order.
3. The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner.
4. Sorting is also used to represent data in more readable formats.

Merging :

1. Merging is defined as the process of creating a sorted list/array of data items from two other sorted array/list of data items.
2. Merge list means to merge two sorted list into one list.

Different types of sorting are :

1. Bubble sort : It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

For example :

```
def bubblesort(list):
# Swap the elements to arrange in order
    for iter_num in range(len(list) – 1,0, – 1):
        for idx in range(iter_num):
        if list[idx]>list[idx+1]:
                temp = list[idx]
                list[idx] = list[idx+1]
                list[idx+1] = temp
list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

2. Merge sort :

1. Merge sort is a divide and conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.
2. The merge() function is used for merging two halves.
3. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m + 1 ..r] are sorted and merges the two sorted sub-arrays into one.

4. **Code :**

```
def mergeSort(arr)
if len(arr) >1:
    mid = len(arr)//2 #Finding the mid of the array
    L = arr[:mid] # Dividing the array elements
    R = arr[mid:] # into 2 halves
    mergeSort(L) # Sorting the first half
    mergeSort(R) # Sorting the second half
    i = j = k = 0
# Code to print the list
def printList(arr):
    for i in range(len(arr)):
        print(arr[i],end=" ")
    print()
# driver code to test the above code
if __name__ == '__main__ ' :
    arr = [12, 11, 13, 5, 6, 7 ]
    print ("Given array is", end = "\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end = "\n")
    printList(arr)
```

3. Selection sort :

1. The selection sort algorithm sorts an array by repeatedly finding the smallest element (considering ascending order) from unsorted list and swapping it with the first element of the list.
2. The algorithm maintains two sub-arrays in a given array:
   i. The sub-array which is already sorted.
   ii. Remaining sub-array which is unsorted.
3. In every iteration of selection sort, the smallest element from the unsorted sub-array is picked and moved to the sorted sub-array.
4. **Code :**

```
def slectionSort(nlist) :
    for fillslot in range(len(nlist) – 1, 0, – 1) :
        maxpos = 0
        for location in range(1, fillslot + 1) :
```

```
            if nlist[location]>nlist[maxpos] :
                maxpos = location
        temp = nlist[fillslot]
        nlist[fillslot] = nlist[maxpos]
        nlist[maxpos] = temp
nlist = [14, 46, 43, 27, 57, 41, 45, 21, 70]
selectionSort(nlist)
print(nlist)
```

4. Higher order sort :

1. Python also supports higher order functions, meaning that functions can accept other functions as arguments and return functions to the caller.
2. **Sorting of higher order functions :**
   a. In order to defined non-default sorting in Python, both the sorted() function and .sort() method accept a key argument.
   b. The value passed to this argument needs to be a function object that returns the sorting key for any item in the list or iterable.
3. **For example :** Consider the given list of tuples, Python will sort by default on the first value in each tuple. In order to sort on a different element from each tuple, a function can be passed that return that element.

```
>>> def second_element (t) :
...         return t[1]
...
>>> zepp = [('Guitar', 'Jimmy'), ('Vocals', 'Robert'), ('Bass', 'John Paul'), ('Drums', 'John')]
>>> sorted(zepp)
[('Bass', 'John Paul'), ('Drums', 'John'), ('Guitar', 'Jimmy'), ('Vocals', 'Robert')]
```

5. Insertion sort :

a. Insertion sort involves finding the right place for a given element in a sorted list. So in beginning we compare the first two elements and sort them by comparing them.

b. Then we pick the third element and find its proper position among the previous two sorted elements.

c. This way we gradually go on adding more elements to the already sorted list by putting them in their proper position.

For example :

```
def insertion_sort(InputList):
    for i in range(1, len(InputList)):
        j = i – 1
        nxt_element = InputList[i]
# Compare the current element with next one
        while (InputList[j] > nxt_element) and (j >= 0):
            InputList[j+1] = InputList[j]
            j=j – 1
            InputList[j+1] = nxt_element
list = [19,2,30,42,28,11,135,26]
insertion_sort(list)
print(list)
```

Program for Sieve of Eratosthenes :

```
def SieveOf Eratosthenes (n) :
    # Create a boolean array "prime[0. . n]" and initialize
    # all entries it as true. A value in prime[i] will
    # finally be false if i is Not a prime, else true.
    prime = [True for i in range(n+1)]
    p = 2
    while (p * p <= n):
        # If prime[p] is not changed, then it is a prime
        if (prime[p] == True):
            # Update all multiples of p
            for i in range(p * p, n+1, p):
                prime[i] = False
        p + = 1
    # Print all prime numbers
    for p in range(2, n):
        if prime[p]:
            print p,
# driver program
if__name__'=='__main__':
    n = 30
    print "Following are the prime numbers smaller",
    print "than or equal to", n
    SieveOfEratosthenes(n)
```