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 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.

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.

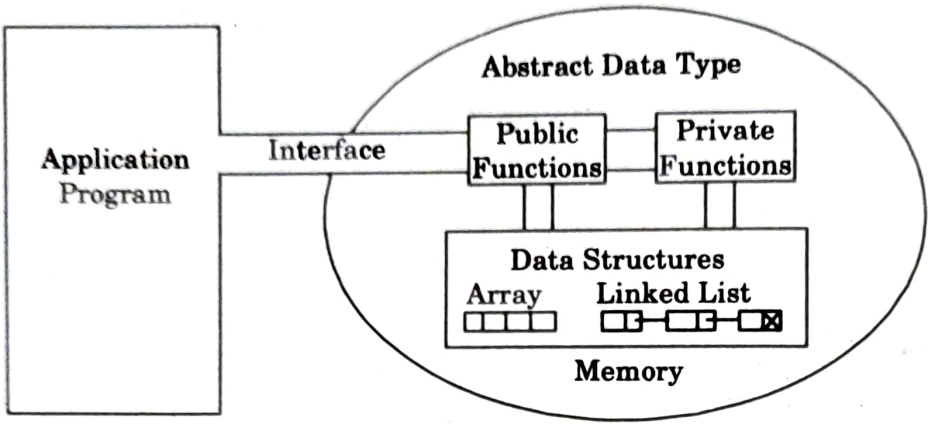

Fig. 4.19.1.

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*.

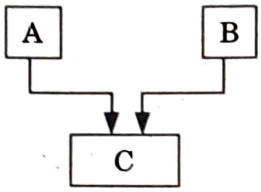

Fig. 4.32.1. Multiple inheritance.

4 **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.

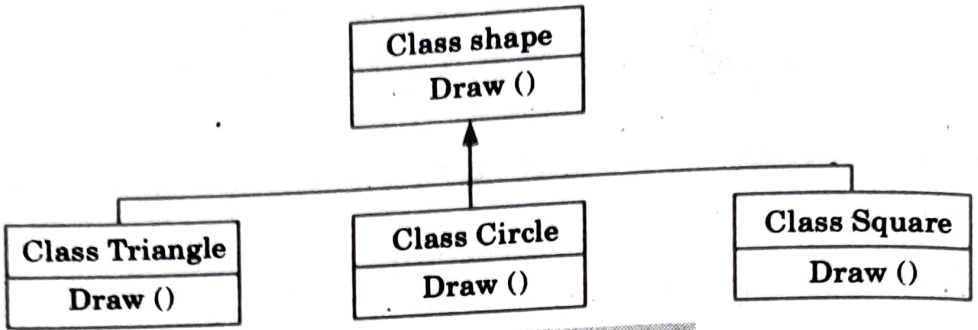

Fig. 4.34.1. Pythomorphism.

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