Lec-41

Q1

100%

- [x] An **argument** is an expression which is passed to a function by its caller, in order for the function to perform its task.
- [x] Arguments are local to the particular function.
- [x] Parameters are available only with in the specified function and parameters belong to the called function.
- [ ] Parameters occupy memory through out the program execution.

Q2

Func_Ex2.py

```
# take the values from the use

def add(a, b):
    return a+b
 # Write your logic here
a=int(input("a: "))
b=int(input("b: "))
print(add(a,b))
```

Q3

Func_Ex1.py

```
def sayhello(username):
    for i in username:
        print("Hello "+i)

users = ['Ram', 'Mahesh', 'Vasudha', 'Uma', 'Sekhar', 'John']

sayhello(users)
```

Q4

Pascal.py

```
def pt(n):
    tr=[1]
    y=[0]
    for x in range(max(n,0)):
        print(tr)
        tr=[l+r for l,r in zip(tr+y,y+tr)]
    return n>=1

n=int(input("num: "))
pt(n)

```

Q1

- [x] When the values are passed, as arguments to the function in any order and these values get assi arguments.
- [x] Calling the function add(a = 10,b = 20) or by add(10, 20) produce the same result.
- [ ] Calling the function add(b = 20, a = 10) or by add(10, 20) do not produce the same result.

Q2

Func_Ex5.py

```
#Program to illustrate keyword parameters
def simplecalc(a,b):
    print("addition:",a+b)
    print("subtraction:",a-b)
    print("multiplication:",a*b)

#define your function here and perform arithmetic operations addition,

simplecalc(a = 3, b = 5)
simplecalc(b = 4, a = 5)
#This function can also be called with positional arguments
simplecalc(8, 4)
```

Q3

Func_Ex6.py

```
def say(n,m):
    print("Good",n,m)
n=str(input("name: "))
m=str(input("morning/night: "))
say(m,n)
say(n=m,m=n)

# print("Good",m,n)
# print("Good",m,n)

```

order.

Q4

KeywordArgs.py

```
n=str(input("name: "))
a=int(input("age: "))

print(n,a)
print(n,a)
```

42

Q1

Func_Ex7.py

```
def simplecalc(a, b = 100):
    # write your code here
    print("addition:",a+b)
    print("subtraction:",a-b)
    print("multiplication:",a*b)


num1=int(input("num1: "))
num2=int(input("num2: "))


simplecalc(a = num1)
simplecalc(b = num2, a = num1)

```

is

Q2

Func_Ex8.py

```
def ct(s,p=20):
    print((s*p)/100)

s=float(input("salary: "))
t=float(input("tax percentage: "))

ct(s)
ct(s,t)
```

Q3

- [x] Function arguments can have default values in **Python**.
- [ ] The **default value** is assigned even when **a value** is passed for that argument.
- [x] Any **number of arguments** in a function can have a default values.
- [x] **non-default arguments** cannot follow default arguments.

Q4

KeywDefaPos.py

```
def f(name="padma",age=12):
    print(name,"is",age,"years old.")

f("Aruna",16)
f("Karuna",16)
f("Padma",16)
f("Karuna",12)
f("Padma",12)
```

Q1

- [x] If the **correct number of arguments** that will be passed to a function at the time of execution is nc arguments.
- [x] Arbitrary arguments. is specified by using an `asterisk (*)` in the function definition before the p
- [ ] Arbitrary/Variable length arguments are two different functionalities.

Q2

at are

Func_Ex9.py

```
#Program to illustrate variable number of arguments
def mySum(*args):
    sum=0
    for i in args:
        sum+=i
    return sum

#Write your code here

print(mySum(1, 2, 3, 4, 5, 6, 7))   #7 arguments
print(mySum(1, 2))  #2 arguments
print(mySum(1, 2, 3))   #3 arguments
```

Q3

print

Func_Ex10.py

```
#Program to illustrate Variable number of arguments
def largestNumber( * numbers):
    maxi=0
    for i in numbers:
        maxi=max(maxi,i)
    print("largest:",maxi)


# write your code here



largestNumber(1, 2, 3, 4)           #4 arguments
largestNumber(8, 9, 3, 4, 2, 5)     #6 arguments
```

Q1

- [x] Anonymous functions are also called **lambda functions.**
- [x] Lambda functions can be used **anywhere** we need a regular function obje
- [ ] Lambda functions can have one argument and many expressions.
- [x] The **expression is evaluated first** and a value is returned.

Q2

Func_Ex11.py

```
1  tax = lambda salary:(salary*20)/100
2
3  salary = int(input("Please enter your salary: "))
4  print("Tax to be paid is", tax(salary))
```

Q3

rint the

Func_Ex12.py

```
1  dn=lambda x:x*2
2
3  a=int(input("a: "))
4  print(dn(a))
```

Q4

MapFunc.py

```python
def squares(x):
    return x ** 2
list1 = [1, 2, 3, 4, 5]
#use the squares func inside map function print the result
print(list(map(squares,list1)))

#use the lambda func inside map function print the result
print(list(map(lambda x:x**2,list1)))

#use list comprehension to get equivalent behaviour as map and print th
print([x**2 for x in list1])

```

Q5

FilterFunc.py

```python
a = [1, 2, 3, 5, 7, 9]
b = [2, 3, 6, 7, 9, 8]
 # apply lambda function to the filter function and print the result

print(list(filter(lambda x:x in a,b)))
print(list(filter(lambda x:x in a,b)))
 # print the result using list comprehension

 # follow the example given in description
```

44

Q1

- [x] A **fruitful function** is a function, it returns a **value** when it is called.
- [x] A return statement consists of the **return** keyword followed by an **expression**.
- [x] Python returns immediately when it reaches a **return** statement .
- [ ] Any statement after **return is executed after the value is returned**.

ınd

Q2

three

Func_Ex15.py

```
def largestinthree(a, b, c):

    # write your code here to find the largest number in a, b and c
    return max(a,max(b,c))

num1 = int(input("Please enter a value for num1: "))
num2 = int(input("Please enter a value for num2: "))
num3 = int(input("Please enter a value for num3: "))

result = largestinthree(num1, num2, num3)

print("Largest of the values entered is", result)
```

Q3

rs a

Func_Ex16.py

```
def largestintwo(a, b):
    return max(a,b)

# write your code here
num1=int(input("num1: "))
num2=int(input("num2: "))


result = largestintwo(num1, num2)
print("largest:", result)
```

Q4

Gcd.py

```
import math
def computeGCD(x, y):
    print(math.gcd(x,y))

# write your code here
x=int(input("x: "))
y=int(input("y: "))

computeGCD(x,y)
```

Q1

Func_Ex17.py

```
def test1():
    a=50
    b=80
    print(a,b)

def test2():
    a=22
    b=44
    print(a,b)

test1()
test2()
```

Q2

Func_Ex18.py

```
#Program to illustrate Global variable access
globvar = "Hello"
def test1():
    global globvar
    globvar = "Good Morning"
def test2():
    # Here this is a local variable
    globvar = "Good Evening"
print(globvar) # The first value "Hello" is printed
# call the function test1
test1()
# call the function test2
test2()
print(globvar) # The updated value of test1 is printed
```

Q3

Func_Ex19.py

```python
#Program to illustrate Global and Local Variables

# take input a from the user
a=int(input("a: "))
def changeglobal():
    global a
    a = 200
def changelocal():
    a = 500
    print("local a value:", a)
print("global a before function call:", a)
# call the function changeglobal
changeglobal()

# call the function changelocal
changelocal()
print("global a after function call:",a ) # print value of a here
```

Q1

Func_Ex20.py

```python
def square(x):
    # find square of a given number and return the result
    return x**2

def double(x):

    # double the given number and return the result
    return x*2

# take the user input
x=int(input("num: "))

print("double, squaring the value:", square(double(x)))
```

Q2

Func_Ex21.py

```python
print("12.25")
print("36")
```

Q1

100%

- [x] A function call when made within the definition of the same function is known as a recursion function.
- [ ] A successful recursive function can be written without a base criteria (i.e., a condition which stops calling the same fun
- [x] Every time a recursive call is made, the arguments passed to the function should be closer to the base criteria.
- [ ] There cannot be more than one recursive call inside the same function.

Q2

- [ ] In indirect recursion, the function calls itself.
- [x] In circular recursion more than one function is involved.
- [ ] Only indirect recursion can lead to infinite loop.

Q3

Func_Ex22.py

```
1  #Program to illustrate recursion
2  def recurfact(n):
3      if(n==0 or n==1):
4          return 1
5      return n*recurfact(n-1)
6
7
8  # write your code here
9  n=int(input("num: "))
10 if(n>=0):
11     print("factorial:",recurfact(n))
12 else:
13     print("factorial not exist for negative number")
```

Q4

Func_Ex23.py

```python
def r_sum(nested_num_list):

# write your code here
    tot=0
    for i in nested_num_list:
        if type(i)==type([]):
            tot+=r_sum(i)
        else:
            tot+=i
    return tot



L1 = [1, 10, 9, [3, 5, 7], [5, [6, 7], 97]]
print(r_sum(L1))
```

Q5

RecurAdd.py

```python
#Program to add two numbers using recursion
def add(x, y):
    return x+y



# write your code here
a=int(input("a: "))
b=int(input("b: "))



print(add(a, b))
```

Q1

DaysLived.py

```python
# write your code here
from datetime import date

def day(d1,d2):
    return (d2-d1).days

s=str(input("dob in ddmmyyyy format: "))
t=str(input("today's date in ddmmyyyy format: "))

d1=date(int(s[4:]),int(s[2:4]),int(s[:2]))
d2=date(int(t[4:]),int(t[2:4]),int(t[:2]))
print("days since birthday:",day(d1,d2))

```

Q2

valley.py

```python
def valley(l):
    if len(l)<4:
        return False
    else:
        for i in range(0,l.index(min(l))):
            if(l[i]>l[i+1]):
                i+=1
            else:
                return False
        for j in range(l.index(min(l)),len(l)-1):
            if(l[j]<l[j+1]):
                j+=1
            else:
                return False
        if i==l.index(min(l)) and j==len(l)-1:
            return True
        else:
            return False


l=list(map(int,input("integers space separated: ").split()))
print(valley(l))
```

Q3

100

Frequency.py

```
def frequency (seq):


    # write your code here
    com=set(seq)
    dic=dict()
    l=[]

    for i in com:
        dic[i]=seq.count(i)
        l.append(seq.count(i))

    # sort={k: v for k,v in sorted(dic.items(),key =lambda item:item[1])}
    maxi=max(l)
    mini=min(l)

    minl=[]
    maxl=[]

    for i in dic:
        if(dic[i]==mini):
            minl.append(i)
        if(dic[i]==maxi):
            maxl.append(i)

    tup=tuple()
    l.clear()
    minl.sort()
    maxl.sort()
    l.append(minl)
    l.append(maxl)
    l.append(mini)
    l.append(maxi)
    return tuple(l)



l1 = [int(x) for x in input("Please enter integers separated by spaces: ").split()]
print (frequency(l1))
```

48

Q1

- ○ Files are stored in RAM
- ◉ File are stored on external media
- ○ Files are stored on paper
- ○ Files are not stored

Q2

- [ ] The file name cannot contain spaces
- [ ] The maximum length of the extension is 3 characters
- [ ] All extensions are predefined
- [x] File extension tells the type of file

Q3

- [x] File path indicates the directory in which the file is located
- [x] In the string "**c:\users\ravi\desktop\documents\profile.doc**", **doc** is the extension
- [x] We can specify the file path relative to the current directory
- [x] When the file is in the current directory, the file path need not be specified

Q4

- [x] File name and file path
- [ ] File name and location
- [ ] File name and extension
- [ ] File extension and file path

Q1

- [ ] An open statement returns the contents of the file
- [x] The mode arguments tells how the file is to be opened, e.g. for reading only, for writing
- [ ] The mode argument is a mandatory argument
- [ ] If we do not specify any path the file is assumed to be in the current directory

Q2

xt.

- [ ] Image files should be opened in text mode
- [x] Student data containing name(string), age(integer), percentage (float) shoul
- [ ] When you read a binary file, the statement returns a string
- [x] Binary data is read and written in bytes objects and cannot be printed direc

Q3

- [ ] Read only
- [ ] Write only with override of existing file
- [x] Write only with error if file is existing
- [ ] Append

Q4

FileIO.py | TextData.txt

```
fr = open('TextData.txt', 'r')                          # Open the file for read
fw = open('NewFile.txt', 'w')                           # Open the file for write (new fil
# Read the file and copy it to the new file
fw.write(fr.read())

fr.close()                                              # Close the input file
fw.close()                                              # Close the new file
fr1 = open('NewFile.txt', 'r+')                         # Open the new file as read / writ
# read and print the first 12 characters
print(fr1.read(12))

# Print the read cursor position( position is 0 based)
print(fr1.tell())

print(fr1.write("this is the new text"))                # Write some text (length 20). Thi
# Position the cursor at 12
fr1.seek(12)
# Read and print the next character (at cursor position 12)
print(fr1.tell())
print(fr1.read(1))

# Position the cursor at 15

fr1.seek(15)
print(fr1.tell())
# Read and print 10 characters from this position
print(fr1.read(10))
print(fr1.read())
# read() always reads the entire file irrespective of cursor position and changes the
fr1.close()                                             # Close the file
```

Q5

BinaryFile.py

```
import pickle                                           # inports the pic
Students = [['John', '501', 20, 92.5],['Kohli', '502', 21, 95.5],
      ['Ganga','503', 20, 90.5],['Gayathri','504', 20, 82.5],
      ['Krishna','505', 20, 96.5]]                      # Define the stud
fst = open("students.dat",'wb')                         # Open the output
for student in Students:
    pickle.dump(student,fst )   #Fill the missing code      # Wri
fst.close()                                             # Close the outpu
fst = open("students.dat",'rb')                         # Open the file
data = pickle.load(fst) #Fill the missing code      # Read the fi
try:                                                    # The Endof file
  while(True ):
    print(data )
    data = pickle.load(fst )
except:
  print("Bye")
```

Q6

LineReverse.py | InputData1.txt

```
f=open("InputData1.txt","r")

l=f.readlines()
for i in l:
    print(i.strip()[::-1])
```

Q7

FileCounts.py | InputData2.txt

```
fin = open('InputData2.txt' , 'r')
charCount = wordCount = lineCount = 0          #Initialize Counters
for line in fin:                               #Read each Line

    # write your logic here

    #Increment Line count
    lineCount+=1

    #split() gives the words in a list
    wordCount+=len(line.split())

    #Increment Character Count
    charCount+=len(line)

print("Line count = ", lineCount)  #Print the Counts
print("Word count = ", wordCount)
print("Char count = ", charCount)
```

Q8

CopyFile1.py | InputData3.txt

```
fin = open('InputData3.txt', 'r')          #Open the text File
fout = open('OutputData3.txt', 'w')        #if the file is big

fout.write(fin.read())                                #for each line
                                #Write the line
fin.close()                     #Close the input file
fout.close()                    #Close the output file
                                #Close the input file
                                #Close the output file
fin = open('OutputData3.txt' , 'r')        #Open the new file
                                #for each line
print(fin.read())                                #Print line
fin.close()                     #Close the File
```

Q9

PickleDictionary.py

```
1  import pickle
2
3  dic={"Cat":2,"Dog":7,"Lion":4,"Tiger":9,"Leopard":11,"Bear":8,"Elephant":15}
4
5  f=open("file.txt","wb")
6  pickle.dump(dic,f)
7
8  f=open("file.txt","rb")
9  print(pickle.load(f))
10 print("Bye")
11 f.close()
```

50

Q1

- [x] **A regular expression**, is called a pattern.
- [x] An atom is a single point within the **regex pattern** which it tries to ma
- [x] The simplest atom is a **literal**.
- [ ] `.` Matches any character of newline.

Q2

- [x] **[ ]** These are used for specifying a character class.
- [ ] **/** cannot be also used to escape all the metacharacters.
- [x] ***** is used as a repeated qualifier for **0** or more times.
- [ ] **Metacharacters** are active inside classes.

Q3

- [x] Pattern objects have several methods and attributes like match(), search() etc.
- [ ] **findall()** finds all the substrings of the RE where it matches, and returns them as a **iterato**
- [x] The **match() and search()** return a `None` if no match is found.
- [x] **start()** method on match object returns the starting position of the match.

Q4

Re_Example1.py

```
1  import re
2  mystring = "Hello!! Good Morning, Welcome to python
3
4  # write your code here
5  print(re.findall("^Hello",mystring))
6  print(re.findall('\d+',mystring))
7  print(re.findall("[abc]+",mystring))
```

Q5

Re_Example2.py

```
1  import re
2  mystring = "Hello!! Good Morning, Welcome to python tutorial class 24."
3  matches = re.findall("[eo]+",mystring)
4
5  for m in matches:
6      print(m)
7
8  # print m
9
```

Q6

Re_Example3.py

```
import re

s=str(input("string with email address: "))

e=re.findall("\S+@\S+",s)
if(len(e)!=0):
    print("email address:",e)
else:
    print("No email address found")

```

Q7

such as

- [x] Compilation flags lets a user modify a few parts of regular expressions.
- [x] **IGNORECASE, I** is used for case-insensitive matches.
- [ ] Enabling verbose REs helps in matching any character.

Q8

Re_Example4.py

```
import re

mystring = "The alternate email address is victory@ct.com"
match = re.search('(\w+)@(\w+).(\w+)', mystring)
if match:
    # find full match text and print the result
    print("Full match:",match.group())
    # find the match text corresponding to the 1st left parenthesis and print
    print("Group 1:",match.group(1))
    # find the match text corresponding to the 2nd left parenthesis and print
    print("Group 2:",match.group(2))
    # find the match text corresponding to the 3rd left parenthesis and print
    print("Group 3:",match.group(3))
```

Q9

Re_Example5.py

```
# write your code here
import re
s=str(input("data: "))
match=re.search(r'^(\w+),(\w+),(\w+),(\w+)$',s)
if match is None:
    print("you have not entered 4 words")
else:
    print("full:",match.group(0))
    print("group 1:",match.group(1))
    print("group 2:",match.group(2))
    print("group 3:",match.group(3))
    print("group 4:",match.group(4))
```

51

Q1

- [x] In early times, all the programming languages had huge lines of co
- [x] A module contains a set of classes, functions, variables hat got toge
- [ ] Modules are intended for separating code , not reusing.
- [x] In Python, a module is a python file (.py).

Q2

Module1.py

```python
courses = ['Java','Python','C','C plus']

def arithoperations(num1,num2):

    print("addition:",num1+num2)
    print("subtraction:",num1-num2)
    print("multiplication:",num1*num2)
    print("division:",num1/num2)

def compoperations(num1,num2):

# Compare num1 is greater than num2
    print("is greater than:",num1>num2)
# Compare num1 is less than num2 or not
    print("is less than:",num1<num2)
# Compare if num1 is equal to num2 or not
    print("is equal to:",num1==num2)
# Compare if num1 is not equal to num2 or not
    print("is not equal to:",num1!=num2)


arithoperations(10, 20)

compoperations(10, 20)

#print the length of the courses
print("length:",len(courses))

```

Q3

Take

Module2.py

```python
def checkNegativeNumber(num):
    if num<0:
        print("negative")
    else:
        print("positive")

a=int(input("a: "))
checkNegativeNumber(a)

```

Q1

Module3.py | CheckNegative.py

```
import CheckNegative

# Write your code here
# Call the method from module checkNegative
x=int(input("x: "))
CheckNegative.CheckNegativeNumber(x)
```

Q2

Module4.py | Module_Imp2.py

```
import Module_Imp2

a=int(input("a: "))
b=int(input("b: "))

Module_Imp2.arithoperations(a,b)

```

Q3

Module5.py | Module_Imp2.py

```
import Module_Imp2 as mi
a=int(input("a: "))
b=int(input("b: "))

mi.arithoperations(a,b)
```

Q4

Module6.py | Module_Imp1.py | Module_Imp2.py

```python
import Module_Imp1
import Module_Imp2

a=int(input("num1: "))
b=int(input("num2: "))

Module_Imp1.checkNegativeNumber(a)
Module_Imp1.checkNegativeNumber(b)
Module_Imp2.arithoperations(a,b)

```

Q4

Module6.py | Module_Imp1.py | Module_Imp2.py

```python
def checkNegativeNumber(num):
    if(num < 0):
        print("negative")
    else:
        print("positive")
```

Q4

Module6.py | Module_Imp1.py | Module_Imp2.py

```python
def arithoperations(num1, num2):
    print('addition:', num1 + num2)
    print('subtraction:', num1 - num2)
    print('multiplication:', num1 * num2)
    print('division:', num1 / num2)
```

Q5

- [x] If the import and imported files are in the same directory, then the interpreter does
- [x] The interpreter first searches the built-in module.
- [ ] If the file is not found in the current directory, it throws an error.

53

Q1

Module7.py | Module_Imp3.py

```
from Module_Imp3 import calculatearea,calculatediameter

#Write your code here
s=int(input("side: "))
calculatediameter(s)
calculatearea(s,s)

```

Q2

Module8.py | Module_Imp3.py

```
from Module_Imp3 import pivalue, shapes
pivalue()
n=int(input("n: "))
print(shapes[1:n])
```

Q3

Module10.py | Module_Imp3.py

```
import Module_Imp3

s=int(input("side: "))
Module_Imp3.calculatearea(s,s)
Module_Imp3.calculatediameter(s)
```

Q4

Module11.py | Module_Imp3.py

```python
# Write your code here
import Module_Imp3

s=int(input("side: "))
Module_Imp3.calculatearea(s,s)
Module_Imp3.calculatediameter(s)
Module_Imp3.pivalue()
print(Module_Imp3.shapes[1:2])

```

Q5

Module12.py | Module_Imp3.py

```python
from Module_Imp3 import *


# write your code here
s=int(input("side: "))
calculatearea(s,s)
pivalue()
print(shapes[2:3])
```

54

Q1

- [x] Packages can be thought as directories with some specific rules.
- [ ] A package in Python cannot contain other packages.
- [x] Each package should have a file called `__init__.py`
- [ ] The `__init__.py` should not be empty inside a package.

Q2

Person.py | Robots/Car.py | Robots/House.py

```
import Robots.Car, Robots.House  # here modules are imported

Robots.Car.cleancar() # function cleancar is called form the Car modules


# follow the above process and call the functions cleanhouse and payrent
Robots.House.cleanhouse()
Robots.House.payrent()
```

Q3

Pack_ex1.py | Robots/Car2.py

```
import Robots.Car2

Robots.Car2.cleancar()
Robots.Car2.checkcar()
```

Q4

Pack_Ex3.py | Robots/Car.py | Robots/House.py

```
from Robots import House, Car


# call the two functions present in House.py file
House.cleanhouse()
House.payrent()
# call the two functions present in Car.py file

Car.checkcar()
Car.cleancar()
```

Q5

Pack_Ex3.py | Robots/Car2.py

```
import  Robots.Car2

Robots.Car2.checkcar()
Robots.Car2.cleancar()
```

Q1

- [x] Object-oriented programming was first introduced at MIT in 1950's.
- [x] In 1990, **James Gosling** of Sun Micro Systems developed a simple version of
- [ ] **Java** was not widely used to build internet applications.
- [ ] Three OOP concepts are: Objects and Classes, Inheritance and Encapsulation.

Q2

- [x] A programming paradigm informs how problems are analysed and solved in a prog
- [ ] Procedural programming is more closer to the real world than OOP.
- [x] Large systems developed using procedural programming language can be difficult
- [ ] For small algorithm implementation programs, procedural programming languages

Succ

Q3

- [x] A class is a combination of data and its associated m
- [x] Object based programming implements information
- [ ] Object-oriented programming does not implement
- [ ] **Visual Basic** is an OOP language.

Q4

- [x] Objects are **instances of classes**.
- [ ] A class can be an object.
- [x] A class is like the blueprint or design o
- [x] A class contains both attributes and m

Q5

- [x] A class is a combination of data members anc
- [x] An object is an instance of a class.
- [x] Data in an object captures the characteristics
- [x] Methods of an object captures the behaviour

Q6

- [x] A class is a combination of data and functions, which is nothing but im
- [ ] By using public access specifier in a class, data will be hidden.
- [x] An object is a run-time instance of a class.

Q7

- [x] Abstraction.
- [ ] Encapsulation.
- [ ] Inheritance.
- [ ] Polymorphism.

Type here

ClassesExample1.py

```
class Student:
    pass
Stud_1 = Student()
Stud_2 = Student()
Stud_1.name = 'SriRam'
Stud_1.age = 25
Stud_1.graduate = 'MBA'
Stud_2.name = 'Lakshman'
Stud_2.age = 23
Stud_2.graduate = 'Engineer'
print("Stud_1.name:", Stud_1.name)
print("Stud_1.age:", Stud_1.age)
print("Stud_1.graduate:", Stud_1.graduate)
print("Stud_2.name:", Stud_2.name)
print("Stud_2.age:", Stud_2.age)
print("Stud_2.graduate:", Stud_2.graduate)
```

Q9

ClassesExample1.py

```
class Student:
    pass
Stud_1 = Student()
Stud_1.name = 'SriRam'
Stud_1.age = 25
Stud_1.graduate = "MBA"
print("Stud_1.name:", Stud_1.name)
print("Stud_1.age:", Stud_1.age)
print("Stud_1.graduate:", Stud_1.graduate)

```

Q10

ClassesExample1.py

```
class Student:
    pass

# # write your code here

s1n=str(input("s1 name: "))
s1a=int(input("s1 age: "))
s1d=str(input("s1 degree: "))

s2n=str(input("s2 name: "))
s2a=int(input("s2 age: "))
s2d=str(input("s2 degree: "))

print("Stud_1.name:",s1n)
print("Stud_1.age:",s1a)
print("Stud_1.graduate:",s1d)

print("Stud_2.name:",s2n)
print("Stud_2.age:",s2a)
print("Stud_2.graduate:",s2d)
```

Q11

ClassesExample2.py

```
class Employee:
    pass

n=str(input("name: "))
print("Emp_1.name:",n)
print("Emp_1.salary:",25000)
```

Q12

sult as

ClassesExample1.py

```
class Student:
    pass

# # write your code here
n=str(input("name: "))
a=int(input("age: "))
d=str(input("degree: "))

print("Stud_1.name:",n)
print("Stud_1.age:",a)
print("Stud_1.graduate:",d)
```

Q13

rint

ClassesExample2.py

```
class Employee:
    pass
n=str(input("name: "))
s=int(input("salary: "))
print("Emp_1.name:",n)
print("Emp_1.salary:",s)
```


Q1

Selfvar_Example1.py

```
class Student:

    # define init method with self , name, age, email attributes
    def __init__(self,name,age,email):
        self.name = name
        self.age = age
        self.email = email
Stud_1 = Student('SriRam', 25, 'ram@sch.com') # type: Student
Stud_2 = Student('Lakshman', 28, 'laks@sch.com')
print('Stud_1 details =', Stud_1.name, Stud_1.age, Stud_1.email)
print('Stud_2 details =', Stud_2.name, Stud_2.age, Stud_2.email)
```

Q2

Self_Example1.py

Successfully save

```python
class Car:
    def setDetails(self, model, regno):
        # write your code here
        self.model=model
        self.regno=regno
    def getModel(self):
        # write your code here
        return self.model
        # print(self.model)
    def getRegno(self):
        # write your code here
        return self.regno

Hyundai = Car()
Maruthi = Car()
#Take details of the car as input from user. Write your code here

c1=str(input("car1 model: "))
cr1=str(input("car1 regno: "))
c2=str(input("car2 model: "))
cr2=str(input("car2 regno: "))

Hyundai.setDetails(c1,cr1)
Maruthi.setDetails(c2,cr2)
print("Hyundai car details:",Hyundai.getModel(),Hyundai.getRegno() )
print("Maruthi car details:",Maruthi.getModel(),Maruthi.getRegno() )
```

Q3

Self_Example3.py

```python
# write your code here
c=str(input("car name: "))
print("Honda car name:",c)
```

Q4

- [ ] self is a keyword.
- [x] self should be the first argument in the parameter list.
- [x] self is always a reference to the current instance.

Q1

- [x] A constructor can be viewed as a specific method used by the class to perf
- [x] In Java language, the constructor has the same name as the class with no r
- [ ] Constructors cannot be overloaded in **Python**.
- [x] A constructor in **Python** in any class is defined as **__init__(self)** method.

Q2

ClassesExample4.py

```
sn1=str(input("s1 name: "))
sna1=int(input("s1 age: "))
sn2=str(input("s2 name: "))
sna2=int(input("s2 age: "))
print("Stud_1.name:",sn1)
print("Stud_2.name:",sn2)
```

Q3

ClassesExample4.py

```
class Student:
    def __init__(self, name, age, email):
        # Write your code here
        self.name=name
        self.age=age
        self.email=email

    def studentDetails(self):
        # Write your code here
        print("name:",self.name,", age:",self.age,", email:",self.email)

n=str(input("name: "))
a=int(input("age: "))
e=str(input("email: "))

s=Student(n,a,e)
s.studentDetails()
```

Q4

ClassEmployee.py

```python
class Employee:
    def __init__(self, name, salary):
        # Initialize name and salary of the employee
        self.name=name
        self.salary=salary


    def displayEmployee(self):
        # Write a function to display employee details
        print("name:",self.name,", salary:",self.salary)


# Print the details of the employee
n=str(input("name: "))
s=int(input("salary: "))
s=Employee(n,s)
s.displayEmployee()
```

58

Q1

- [x] Methods are a set of statements that are called to perform a specific task.
- [x] Methods are used in OOPS.
- [ ] Functions need to be a part of a class.

ass
ısed

Q2

Method_Example1.py

```python
n1=str(input("p1 name: "))
n2=str(input("p2 name: "))
print("p1 name:",n1)
print("p2 name:",n2)
```

Q3

Method_Example2.py

```
n1=str(input("name1: "))
n2=str(input("name2: "))
print(n1)
print(n2)
```

Type here

Overloading_Example1.py

```
class Greeting:
    def sayHello(self, name = None, wish = None):
        if name is not None and wish is not None:
            print ('Hello' + name + wish)
        elif name is not None and wish is None:
            print ('Hello' + name)
        else:
            print ('Hello')
greet = Greeting()
# Call the method with zero, one and two parameters
greet.sayHello()
greet.sayHello('Ram')
greet.sayHello('Ram,', 'Good Morning!!!')
```

59

Q1

- [x] Polymorphism can be defined as ability where an entity can behave different
- [x] The purpose of inheritance is to avoid redundancy.
- [x] Modularity reduces the complexity of the program.

Q2

- [x] Inheritance is implemented by creating a derived class that c
- [x] In Python , a base class is created like a regular class.
- [x] A derived class is created by passing the base class name as
- [ ] A base class should be in the same module .

Type here

Inheritance_Example1.py

```
class Car:
    def setenginemodel(self, engine):
        self.engine = engine
    def getenginemodel(self):
        print(self.engine)
class Honda(Car):
    def setcarmodel(self, model):
        self.model = model
    def getcarmodel(self):
        print(self.model)
mycar = Honda()
mycar.setenginemodel('EK-1')
mycar.setcarmodel('V6')
print('Car Details:')
mycar.getenginemodel()
mycar.getcarmodel()
```

Q4

Inheritance_Example2.py

```
class Person:
    pass

n=str(input("Please enter a name: "))
a=int(input("Please enter age: "))
print(n)
print(a)
```

Q5

- [ ] Ability of an entity to correct its syntax errors automatically.
- [ ] Ability of an entity to correct its logical errors automatically.
- [x] Ability of an entity to behave differently in different contexts.

Q6

100%

MultipleInher.py

```
class vehicle:
    '''General Vehicle class'''
    def __init__(self,name,price,regno):
        self.name=name
        self.price=price
        self.regno=regno

class car(vehicle):
    ''' Class car inherits from Vehicle'''
    def __init__(self,name,price,regno,gear):
        self.name=name
        self.price=price
        self.regno=regno
        self.gear=gear

class boat(vehicle):
    ''' Class boat inherits from vehicle'''


class hover(car, boat):
    '''Class hovercraft inherits from both car and boat'''


c1 = car('toyota', 1500000, 'car2121', 'auto')
b1 = boat('maruti', 1000000, 'boat0121')
h1 = hover('toyota', 1500000, 'hover1212', 'manual')
print(type(c1).__name__, "\t", c1.name, "\t", c1.price, "\t", c1.regno, "\t", c1.gear)
print(type(b1).__name__, "\t", b1.name, "\t", b1.price, "\t", b1.regno, "\t")
print(type(h1).__name__, "\t", h1.name, "\t", h1.price, "\t", h1.regno, "\t", h1.gear)
print(c1.__doc__)
```

60

Type here

Inheritance_Example4.py

```
class Person:
    def setname(self, name):
        self.name = name
    def getname(self):
        print(self.name)
class Student(Person):
    def setage(self, age):
        self.age = age
    def getage(self):
        print(self.age)
class Address(Student):
    def setaddress(self, address):
        self.address = address
    def getaddress(self):
        print(self.address)
s1 = Address()
s1.setname('SriRam')
s1.setage('19')
s1.setaddress('Hyderabad')
s1.getname()
s1.getage()
s1.getaddress()
```

Type here

Inheritance_Example3.py

```
class Car:
    def setenginemodel(self, engine):
        self.engine = engine
    def getenginemodel(self):
        print(self.engine)
class Tyre:
    def settyrenumber(self, num):
        self.num = num
    def gettyrenumber(self):
        print(self.num)
class Honda(Car, Tyre):
    def setcarmodel(self, model):
        self.model = model
    def getcarmodel(self):
        print(self.model)
accord = Honda()
accord.setenginemodel('EK-1')
accord.setcarmodel('V6')
accord.settyrenumber(236)
print('Car Details: ')
accord.getenginemodel()
accord.getcarmodel()
accord.gettyrenumber()
```

Q1

- [x] Method overloading is a feature that allows methods with multiple param
- [x] Method overriding is a feature where we modify a behaviour of a method
- [x] This overriding feature happens only when we have a base class and deriv

ed from

Q2

Overriding_Example1.py

```
class Car:
    def setbrandname(self, brandname):
        # Write your code here
        pass

    def getbrandname(self):
        # Write your code here
        pass

    def setmodel(self, model):
        # Write your code here
        pass
    def getmodel(self):
        # Write your code here
        pass
class Accord(Car):
    def setbrandname(self, brandname):
        # Write your code here
        pass
    def getbrandname(self):
        # Write your code here
        pass
blueaccord = Accord()
# set the brand and model name by taking the input from the user
b=str(input("brand: "))
m=str(input("model: "))
print(b)
print(m)

```

Type here

ClassOverride.py

```
class Abstract(object):
    def foo(self):
        raise NotImplementedError('subclasses must override foo()!')
class Derived(Abstract):
    def foo(self):
        print('Hooray!')
d = Derived()
d.foo()
```

Q4

Overiding_Example2.py

```
print("Rabbit1 is an instance of Animal")
print("Rabbit eats Grass")
print("Rabbit eats Peanuts")
print("Cow1 is an instance of Mammal")
print("Cow does not eat. It only drinks")
print("Vulture1 is an instance of WingedAnimal")
print("Vulture eats anything and everything")
print("Bat1 is an instance of Bat")
print("Bat eats anything and everything")
print("fbat is an instance of FruitBat")
print("Fruitbat does not eat. It only drinks")
```

62

Q1

- [x] The Object Oriented Programming evolved to overcome the limitations of procedural programming approach.
- [x] Data abstraction refers to the act of representing essential features without including the background details a
- [x] Data encapsulation is the way of combining both data and functions that operate on the data under a single u
- [x] The insulation of data from direct access by the program is called data hiding.

Q2

- [x] When we write an object oriented python program, we can restrict the access to methods and variable
- [x] A private variable can only be accessed by a method that is of the same clas and not outside of the c
- [ ] A private method can be called by a methods of any class.
- [x] A public variable/method can be accessed from anywhere.
- [ ] Python has protected variables or methods which specifies the variables and methods can be used by

Successfully saved.

Q3

Datahiding_Example1.py

```
1  class Student:
2
3      def __init__(self,name,age):
4          print("SriRam 19 CSC pass")
5
6
7
8
9
10 print("Student Report Card")
11 s1 = Student('SriRam', '19')
12 s1.setDetails("CSC", "pass")
13 s1.getdetails()
```

wing

Q4

Datahiding_Example2.py

```
1  n=str(input("name: "))
2  a=int(input("age: "))
3  g=str(input("group: "))
4  r=str(input("report: "))
5  print(n,a,g,r)
```

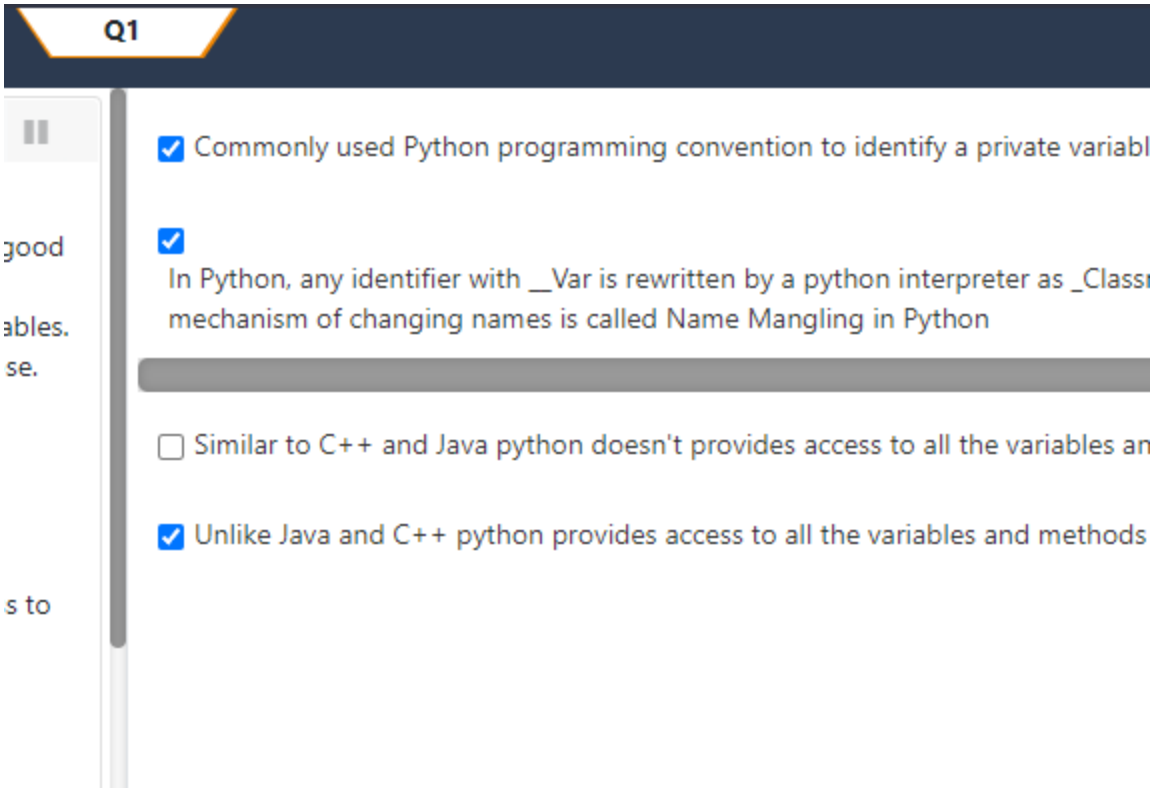
Datahiding_Example3.py

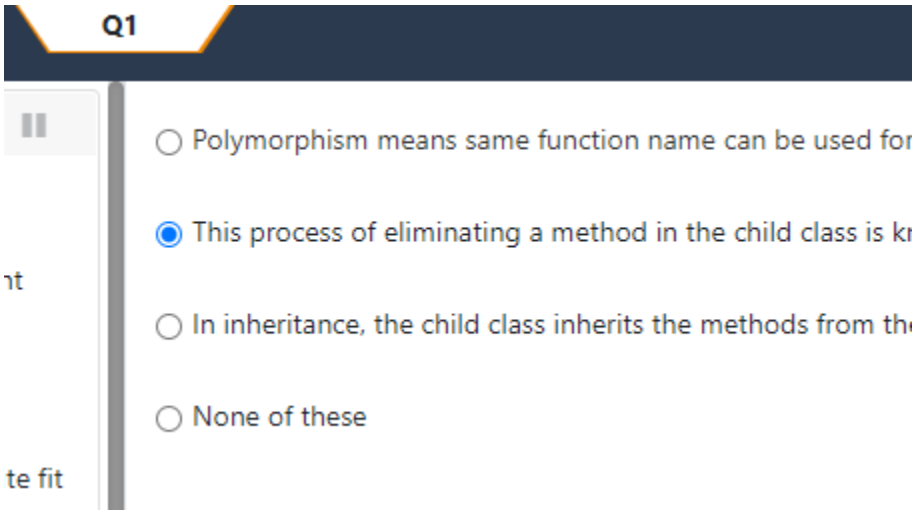
```python
# class Student:
#    __name = "SriRam"
#    __age = "19"
#    __group = "ECE"
#    __report = "fail"


# # define private method setdetails here

# # define a public method setgroup here

# # define a public method getdetails here

#    #print student name, age, group, report here

# s1 = Student()
# s1.setgroup("CSC", "pass")
# s1.getdetails()
print("SriRam 19 CSC pass")
```

Q6

Datahiding_Example4.py

```python
class Car:
    pass
# write your code here
e=str(input("engine number: "))
print(e,"SX6 Blue 2018")
```


Q1
Commonly used Python programming convention to identify a private variabl
In Python, any identifier with __Var is rewritten by a python interpreter as _Class
mechanism of changing names is called Name Mangling in Python
Similar to C++ and Java python doesn't provides access to all the variables an
Unlike Java and C++ python provides access to all the variables and methods


Q1
Polymorphism means same function name can be used for
This process of eliminating a method in the child class is kr
In inheritance, the child class inherits the methods from th
None of these

Q1

- [x] PyTorch is developed by Facebook's artificial intelligence research group
- [ ] The plotting of numerical data is the responsibility of Scipy
- [x] The NumPy Array Python object defines an N-dimensional array with rows and columns.
- [x] PyBrain's main aim is to provide ML algorithms

t holds

and

65

Q1

- (●) 96 8 12 3
- ( ) 96 8 12 2
- ( ) 48 4 12 3
- ( ) 48 4 12 2

Q2

```
Addition of a and b : [15 30 45]
Multiplication of a and b : [ 50 200 450]
Subtraction of a and b : [ 5 10 15]
a raised to b is: [ 100000 797966336 -2080931840]
```

```
Addition of a and b : [25 20 35]
Multiplication of a and b : [ 40 100 350]
Subtraction of a and b : [ 15 10 25]
a raised to b is: [ 1000 787866334 -2089310240]
```

```
Addition of a and b : [10 30 25]
Multiplication of a and b : [ 30 200 250]
Subtraction of a and b : [ 15 10 45]
a raised to b is: [ 10000 787844336 -2081930840]
```

```
Addition of a and b : [5 20 45]
Multiplication of a and b : [ 40 300 450]
Subtraction of a and b : [ 10 15 5]
a raised to b is: [ 10000 797955446 -2088931840]
```

Q3

- [x] To do numerical calculations
- [x] To do scientific computing
- [ ] Contiguous allocation of memory.
- [ ] None of the provided option.

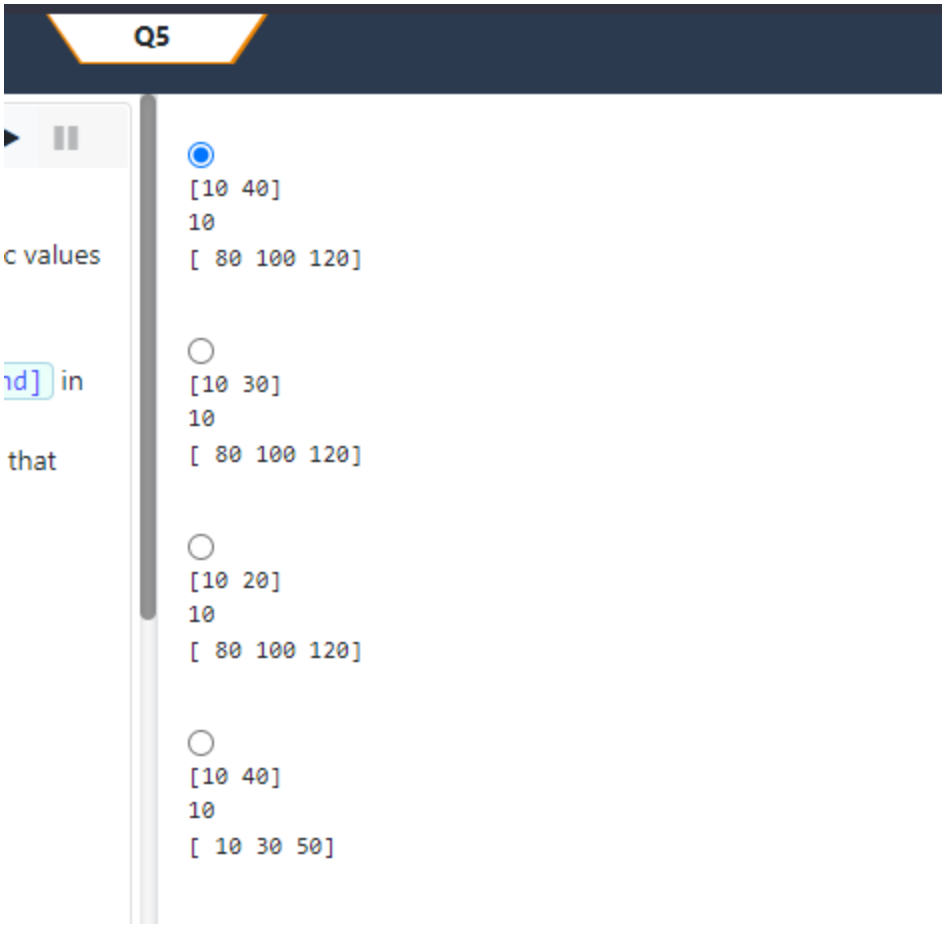
Q4

◉
```
[ 0  4  8 12 16]
[ 0. 20. 40. 60. 80.]
20.0
```

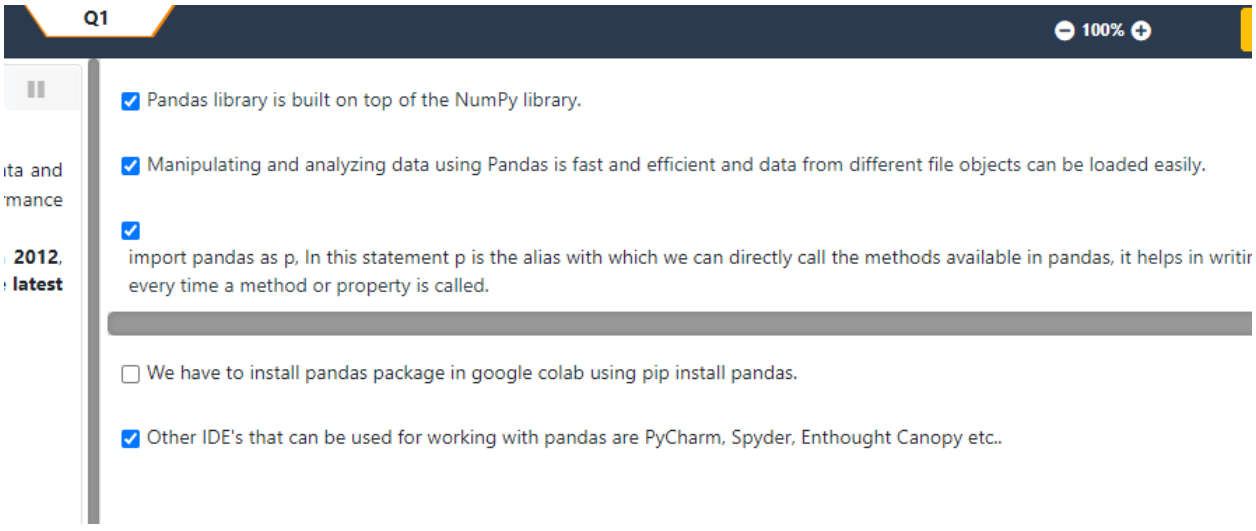
○
```
[ 0  4  8 12 16]
[ 0 20 40 60 80]
20
```

○
```
[ 0  4  8 12 16]
[ 0. 20. 40. 60. 80.]
20
```

○
```
[ 0  4  8 12 16 20]
[ 0. 20. 40. 60. 80.]
20.0
```


Q5
c values
]] in
that
[10 40]
10
[80 100 120]
[10 30]
10
[80 100 120]
[10 20]
10
[80 100 120]
[10 40]
10
[10 30 50]

66


Q1
100%
Pandas library is built on top of the NumPy library.
Manipulating and analyzing data using Pandas is fast and efficient and data from different file objects can be loaded easily.
import pandas as p, In this statement p is the alias with which we can directly call the methods available in pandas, it helps in writir
every time a method or property is called.
We have to install pandas package in google colab using pip install pandas.
Other IDE's that can be used for working with pandas are PyCharm, Spyder, Enthought Canopy etc..

Q2

- [x] In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull()
- [x] Hierarchical indexing is a method of creating structured group relationships in data.
- [ ] In order to drop null values from a dataframe, we use fillna() function.
- [ ] All of these

Successfully saved

Q3

- [x] Pandas provide data analysts a way to delete and filter data frame usir
- [x] DataFrame.align() method aligns two objects on their axes with the sp
- [ ] Rows can be removed using index label or column name using Data al
- [ ] All of these

Q4

- [ ] csv file is also called as Subset Selection
- [x] Pandas Index's task is to organize the data and to provide fast access of data
- [ ] The 'reset_index' is used to set the DataFrame index using existing columns.
- [x] Multiple indexing is defined as a very essential indexing because it deals with data an

Q5

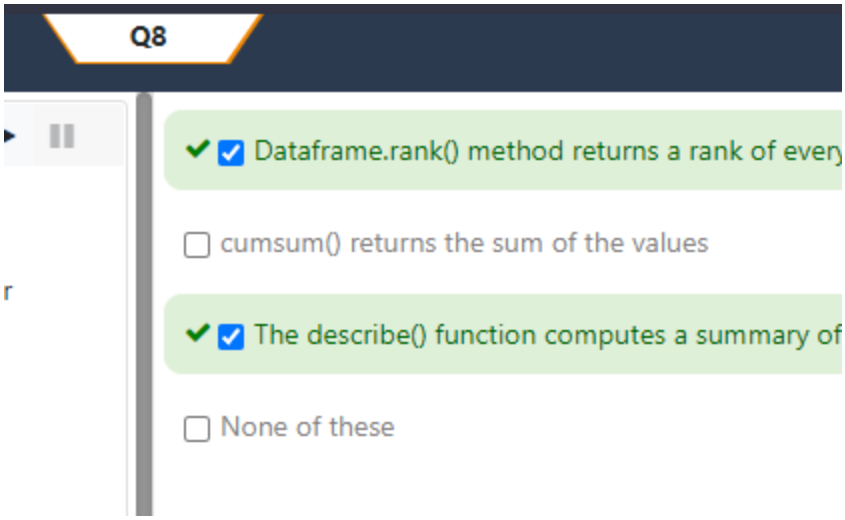
- [x] The main task of the Pandas reindex is to conform DataFrame to a new index with optional
- [x] Reindexing is used to change the index of the rows and columns of the DataFrame.
- [ ] We can reindex only a single row by using the reindex() method.
- [ ] None of these

Q6

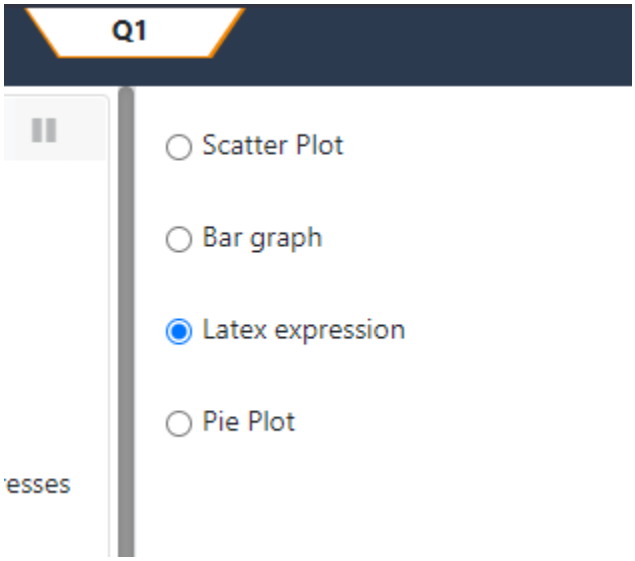
- (•) 0 1011 2012 1303 4014 2505 600dtype: int640 1011 2012 1303 4014 250dtype: int642
- ( ) Series([], dtype: int64)0 1011 2012 1303 4014 250dtype: int642 1303 4014 2505 600dty
- ( ) 0 1011 2012 1303 4014 2505 600dtype: int640 1011 2012 1303 4014 2505 600dtype: in
- ( ) 0 1011 2012 1303 4014 2505 600dtype: int640 1011 2012 1303 4014 250dtype: int64Se

Q7

- [x] The default value for the separator in CSV file is ','.
- [x] Default data displayed by head() and tail() methods is 5 t
- [ ] Default indexing of data frame created from NumPy arra
- [x] Default indexing of data frame created from NumPy arra


Q8
Dataframe.rank() method returns a rank of every
cumsum() returns the sum of the values
The describe() function computes a summary of
None of these

67


Q1
Scatter Plot
Bar graph
Latex expression
Pie Plot

68

Q1

- [x] scikit-learn has a wide range of Machine Learning (ML) algorithms.
- [x] To check the accuracy of our model, we can split the dataset into two pieces-a training
- [x] Scikit learn is Built on the top of NumPy, SciPy, and matplotlib.
- [ ] None of these

Q1

- [x] Machine learning uses data to detect various patterns in a given dataset.
- [x] Reinforcement learning is a feedback-based learning method
- [x] The accuracy of predicted output depends upon the amount of data.
- [ ] The goal of unsupervised learning is to restructure the Output data