**Python and OOP (Lab- Final) Answer Script**

| Question No. 01` |
| --- |
| Is there a difference between the type of the expressions "python" and ’python’ |
| Answer:  In general for python, double quotes and single quotes create no difference in string.  Here, "python" and python’, both are same. This can be tested by a simple code, shown below,  if "python" == 'python':  print("true")  # ans is true |

| Question No. 02 |
| --- |
| What is the difference between the expressions  c = 299792458 and c = 2.99792458 \* 10 \*\* 8? |
| Answer:  in python \* is the multiplication symbol, and \*\* is the mathematical power symbol.  here,  10 \*\* 8 = 100000000 (meand 10^8)  then 2.99792458 \*100000000 = 299792458.0 which has the same valus as 299792458  so there is no difference in value between these two numbers, apart from resresentation difference.  shown below in a code,  c = 299792458  c1 = c = 2.99792458 \* 10 \*\* 8  if c==c1:  print("true")  else:  print("f")  # answer true |

| Question No. 03 |
| --- |
| a. Using books, the Web, or any other resource, find out what a logic gate is, and, in particular, what a half-adder is. An output of the work should be a table showing the inputs and outputs for a half-adder. 2  b. Implement a half adder as a few lines of Python code. 10  c. Write a test program that shows that your implementation works as expected. You should be able to test that your code works for all possible input values. 10 |
| Answer:   1. **Logic gate**: a logic gate is a device or circuits that implements logical functions like AND, OR, NOT.   **Half Adder**: a half adder is a special type of device or circuit that adds two single binary digits and produce two outputs sum and carry. two half adders can be used to make a full adder circuit.  Below are equations and Truth table for sum and carry calculation for a half adder.  Sum = A XOR B  Carry = A AND B   | A | B | Sum | Carry | | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | | 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 1 |  1. **Half adder code in python**:   # save as my\_adder.py  def sum(a, b):  sum= a^b # ex-OR  return sum  def carry(a, b):  carry= a and b # AND  return carry  if \_\_name\_\_=="\_\_main\_\_":  a=1  b=1  s=sum(a,b)  c=carry(a,b)  print(f"sum={s}, carry={c}")   1. **Test code:**  | # saved as my\_adder.py  def sum(a, b):  sum= a^b  return sum  def carry(a, b):  carry= a and b  return carry  if \_\_name\_\_=="\_\_main\_\_":  a=1  b=1  s=sum(a,b)  c=carry(a,b) | | --- | | # saved as test\_adder.py  import my\_adder  # import pytest  a=1  b=1  def test\_my\_sum(): # must have test word  # a=1  # b=1  s=my\_adder.sum(a,b)  assert 0 == s  def test\_my\_carry(): # must have test word  # a=1  # b=1  c=my\_adder.carry(a,b)  assert 1 == c | |

| Question No. 04 |
| --- |
| Create a program for storing a week’s worth of rainfall data. Use a list to store each day’s value, entered sequentially by the user. When an entire week has been input, display the days with the minimum and maximum rainfall. 10 |
| Answer:  rainfall=[]  for i in range(7):  print(f"input ranfall data for day {i+1}: ")  a=input()  rainfall.append(a)  # print(rainfall)  print(f"max rainfall day number: {rainfall.index(max(rainfall))+1} and max ranfll is {max(rainfall)}")  print(f"min rainfall day number: {rainfall.index(min(rainfall))+1} and min ranfll is {min(rainfall)}") |

| Question No. 05 |
| --- |
| Extend your answer to the previous exercise (4) so that it also displays the mean and standard deviation of the values. The mean is the sum of all the values divided by the number of values. The standard deviation is the square root of the sum of the squares of the difference between each value and the mean, divided by the number of items. 10 |
| Answer:  import statistics  import math  rainfall=[]  # rainfall=[2, 2.5, 1.25, 3.1, 1.75, 2.8, 1.8]  for i in range(7):  print(f"input ranfall data for day {i+1}: ")  a=int(input())  rainfall.append(a)  print(rainfall)  print(f"max rainfall day number: {rainfall.index(max(rainfall))+1} and max ranfll is {max(rainfall)}")  print(f"min rainfall day number: {rainfall.index(min(rainfall))+1} and min ranfll is {min(rainfall)}")  # mean and SD  mean=sum(rainfall)/7.00  print(f"mean: {mean}")  std=statistics.stdev(rainfall)  print(f"std: {std}")  # manual calculation of SD  Sum= 0  for i in rainfall :  Sum +=(i-mean)\*\*2  stdeV = math.sqrt(Sum/7)  print(stdeV) |

| Question No. 06 |
| --- |
| What does the term referential transparency mean? 2 |
| Answer:  Referential transparency means an expression can be replaced with the result of that expression and everything remains the same.  for example, equation z=2+1 can be replaced bt z=3 and everything will remain same.  in other words, it means, it doesnot matter how many times someone runs this equation, it will always give the same result. |

| Question No. 07 |
| --- |
| In Python modules, what is the following code idiom used for? 2  if \_\_name\_\_ == '\_\_main\_\_: |
| Answer: in short,  This conditional block allows the python file to run as a script but stops automatic execution (of the whole code) when imported as a module in another python file.  **Example:**  def sum(a, b):  sum= a+b  return sum  if \_\_name\_\_=="\_\_main\_\_":  s=sum(7,3)  print(s)  When executed, this code produces output 10. But when it is imported into another file, it does not execute automatically, the sum method needs to be called to produce output.  On the other hand, if there is no “if **\_\_name\_\_=="\_\_main\_\_":**” in the code,  when imported into another file, it will execute and produce 10, doesn’t need to call the sum method. |

| Question No. 08 |
| --- |
| Convert the following recursive functions into iterative ones: 3+3+3 |
| Answer:   | 1 | def sum\_odd(n, total):  while True:  if n==1:  # print("one:", n, total)  return total  break  elif n%2==0:  # print("even:", n, total)  n=n-1  else:  # print("odd:", n, total)  total=total+n  n=n-2  res=sum\_odd(23, 199)  print(res) | | --- | --- | | 2 | def max(l,n):  while True:  if l==[]:  # print(l,n)  return n  # break  elif l[0]>n:  # print(l,n)  n=l[0]  l=l[1:]  else:  # print(l,n)  l=l[1:]  n=n  # l=[1,2,3,4,5,6]  l=[5,4,3,1,2,3,8]  n=max(l,3)  print(n) | | 3 | def mylen(l,n):  while True:  if l==[]:  # print(l,n)  return n  else:  # print(l,n)  n=n+1  l=l[1:]  l=[5,4,3,1,2,3,8]  n=mylen(l,8)  print(n) | |

| Question No. 09 |
| --- |
|  |
| Answer:   1. **File name:** words.dat 2. word count is not fixed and depends on the number of inputs.   **Example inputs,**  Enter a word (enter END to quit): chicken  Enter a word (enter END to quit): apple  Enter a word (enter END to quit): fox  Enter a word (enter END to quit): tiger  Enter a word (enter END to quit): END  in this case, The total number of words stored in the file is 5.   1. **Read and display file:**   file = open ( 'words.dat' , 'r' )  i=1  for item in file:  print(i,":",item,end="")  i+=1   1. **Modified code to exclude the word “END”:**   file = open ( 'words2.dat' , 'w' )  word = ''  while word != 'END' :  word = input( 'Enter a word (enter END to quit): ')  if word=="END":  break  file.write ( word + '\n' )  file.close ( )   1. **Modified the code to terminate when an empty string is given:**   **#** code terminates with empty string  file = open ( 'words2.dat' , 'w' )  word = '' # empty string  while True :  word = input( 'Enter a word (enter END to quit): ')  if not word: # if word == empty string, break  break  if word=="END": # not saving the word "END"  continue  file.write ( word + '\n' )  file.close () |

| Question No. 10 |
| --- |
| Write a Python Program to demonstrate Observer Design Pattern, explain your code in writing. 15 |
| Answer:  """  An example of an observer design pattern.  --------------------------------------  calling the top 3 football players  observable/publisher: FIFA  observers/ subscribers: players  Note: when players are called they will be notified and they will respond too.  """  from abc import ABC, abstractmethod  """  observable/publisher interface  """  class IObservable(ABC):    @abstractmethod  def add(self, observer):  pass    @abstractmethod  def remove(self, observer):  pass    @abstractmethod  def notify():  pass  """  implement observable interface: FIFA  """  class FIFA(IObservable):  def \_\_init\_\_(self) -> None:  self.subscribers=[] # players list  def add(self, observer):  if observer not in self.subscribers:  self.subscribers.append(observer)  print(f"{observer.name}: added")  else:  print(f"{observer.name}: alreadt in the list")  return  def remove(self, observer):  print(f"{observer.name}: removed")  print(f"i am {observer.name}, i will rise!")  self.subscribers.remove(observer)    def notify(self, message):  for observer in self.subscribers:  observer.update(message)    """  observer / subscriber / player interface  """  class IObserver(ABC):  @abstractmethod  def update(self, message):  pass  """  implement observer/player interface  """  class Messi(IObserver):  name="MESSI"  def update(self, message):  print("Messi called-", message)  return  class Ronaldo(IObserver):  name="RONALDO"  def update(self, message):  print("Ronaldo called-",message)  return  class Zidane(IObserver):  name="Zidan"  def update(self, message):  print("Zidan called-",message)  return  class Neymar(IObserver):  name="Neymar"  def update(self, message):  print("Neymer called-", message)  return  if \_\_name\_\_=="\_\_main\_\_":  print("Top 3 selected players by FIFA:")  # instance of observable  publisher=FIFA()  # instance of observers  messi=Messi()  ronaldo=Ronaldo()  zidane=Zidane()  neymar=Neymar()  # add subscribers/observers to publisher  publisher.add(messi)  publisher.add(ronaldo)  publisher.add(neymar)  print("\ncall top 3 players:\n")  # notify selected players  publisher.notify("yes, i am ready!")  print()  # remove neymar  publisher.remove(neymar)  print()  # add zidane and notify again  publisher.add(zidane)  publisher.notify("yes, i am ready!") |
| **Code Explanation:**  In the observer design pattern, there is in general a publisher (also called the observable) and several subscribers (also called observers). If there is any change in the publisher’s state, all the observers will be notified.  Observable or publisher:  In this code example, there is an observable interface class named “IObservable” which has 3 methods add, remove and notify.  This observable interface class in inherited and implemented in the “FIFA” class. The “subscribers” list holds all the observers information. “add” method adds any new observer in the list, “remove” method removes any selected observer and “notify” method notifies all the observers or subscribers using “update method” from observer.  Observers or subscribers:  Here there is one observer interface “IObserver”, which has one method “update”.  In this code, observer interface is implemented by 4 classes named, “Messi”, “Ronaldo”, “Zidane”, and “Neymar” , they all implemented the update method.  Main code:  Later, one “publisher” instance is created of the observable class “FIFA” and 4 instances of 4 observer’s class. Then 3 players are added in the publisher list and they are notified from the publisher.  Afterwards, one player is removed and an alternate player is added. Finally all selected 3 players are notified from the publisher again. |

Submitted by: Subrata Saha

Email:[Subratabaec@gmail.com](mailto:Subratabaec@gmail.com)

Date:26.11.2022

THE END