**Logo

Description automatically generated**

**San Francisco Bay University**

**CE450 Fundamentals of Embedded Engineering**

**2022 Fall Final Exam**

**Student Name: Mahmud H. Omer ID: 19660**

**Part I Python Programming**

1. Write a function as a decorator of other function calls for the following operations

ANS:

|  |
| --- |
| # 1.  Write a function as a decorator of other function  # calls for the following operations  def  trc1(g):         def f(n) :            return g(n)         return f    @trc1  def  sqr(x):      return x\*x  @trc1  def  sum\_sqr(n):      sum = 0      for i in range(n+1):         sum += sqr(i)      return sum  print(sqr(3))  print(sum\_sqr(3)) |

Output:

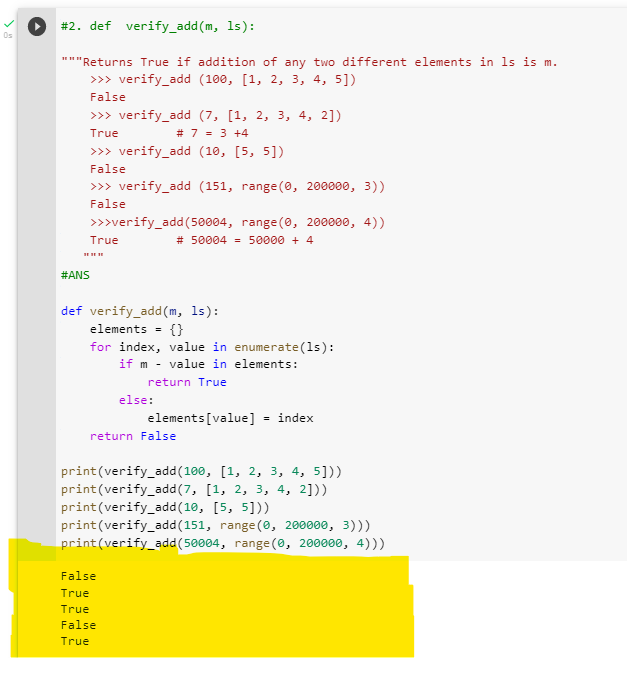
|  |
| --- |
|  |

1. Generate a function to implement the following operations

ANS:

|  |
| --- |
| #2. def  verify\_add(m, ls):  """Returns True if addition of any two different elements in ls is m.      >>> verify\_add (100, [1, 2, 3, 4, 5])      False      >>> verify\_add (7, [1, 2, 3, 4, 2])      True        # 7 = 3 +4      >>> verify\_add (10, [5, 5])      False      >>> verify\_add (151, range(0, 200000, 3))      False      >>>verify\_add(50004, range(0, 200000, 4))      True        # 50004 = 50000 + 4     """  #ANS  def verify\_add(m, ls):      elements = {}      for index, value in enumerate(ls):          if m - value in elements:              return True          else:              elements[value] = index      return False  print(verify\_add(100, [1, 2, 3, 4, 5]))  print(verify\_add(7, [1, 2, 3, 4, 2]))  print(verify\_add(10, [5, 5]))  print(verify\_add(151, range(0, 200000, 3)))  print(verify\_add(50004, range(0, 200000, 4))) |

Output



1. Write a function to implement deep-reverse for taking a (possibly deep) tuple argument and reverses it including deep tuple element.

***def deep\_rvrs(tup):***

*"""Reverses tuple with possible tuple elements*

*>>> a = (11, 12, 13, 14)*

*>>> deep\_rvrs (a)*

*(14, 13, 12, 11)*

*>>>tpl = (11, (12, (13,113), 14), 15)*

*>>> deep\_rvrs (tpl)*

*(15, (14, (113, 13), 12), 11))*

*"""*

*""" YOUR SOURCE CODE HRER """*

*ANS:*

|  |
| --- |
| #3. Write a function to implement deep-reverse for taking a (possibly deep)  # tuple argument and reverses it including deep tuple element.  # def  deep\_rvrs(tup):  """Reverses tuple with possible tuple elements      >>> a = (11, 12, 13, 14)     >>> deep\_rvrs (a)      (14, 13, 12, 11)      >>>tpl = (11, (12, (13,113), 14), 15)      >>> deep\_rvrs (tpl)      (15, (14, (113, 13), 12), 11))      """  #ANS  def deep\_rvrs(t):      return tuple(deep\_rvrs(x) if isinstance(x, tuple) else x for x in reversed(t))    a = (11, 12, 13, 14)  print(deep\_rvrs(a))  tpl = (11, (12, (13,113), 14), 15)  print(deep\_rvrs (tpl)) |

*Output:*

|  |
| --- |
|  |

1. Write a Fibonacci class to calculate next number in the ***'Fibonacci'***class by the

***'nxt'*** method. In this class, the ***'val'*** member is a ***'Fibonacci'*** number. The ***'nxt'*** method will return a ***'Fibonacci'*** object whose value is the next number in Fibonacci series.

***class*** *Fibonacci ():*

*"""A Fibonacci number.*

*>>> a = Fibonacci():*

*>>> a*

*0*

*>>> a.nxt()*

*1*

*>>> a.nxt().nxt()*

*1*

*>>> a.nxt().nxt().nxt()*

*2*

*>>> a.nxt().nxt().nxt().nxt()*

*3*

*>>> a.nxt().nxt().nxt().nxt().nxt()*

*5*

*>>> a.nxt().nxt().nxt().nxt().nxt().nxt()*

*8*

*"""*

***def*** ***\_\_init\_\_(self):***

*self.val = 0*

***def******nxt(self):***

*""" YOUR SOURCE CODE HRER """*

***def*** ***\_\_repr\_\_(self):***

*return str(self.val)*

*Hint: A new* ***'Fibonacci'*** *object is needed to create and assign****'val'*** *and****'pre'*** *members within****'nxt'*** *method.*

*ANS:*

|  |
| --- |
| class Fibonacci():      """A Fibonacci number.      >>>a = Fibonacci():      >>>a      0      >>>a.nxt()      1      >>>a.nxt().nxt()      1      >>>a.nxt().nxt().nxt()      2      >>>a.nxt().nxt().nxt().nxt()      3      >>>a.nxt().nxt().nxt().nxt().nxt()      5      >>>a.nxt.nxt().nxt().nxt().nxt().nxt()      8      """      def \_\_init\_\_(self):          self.val = 0          self.next\_val = 1      def nxt(self):          result = Fibonacci()          result.val = self.next\_val          result.next\_val = self.val + self.next\_val          return result      def \_\_repr\_\_(self):          return str(self.val)  # Testing the class here. ignore/remove the code below if not required  a = Fibonacci()  print(a)  print(a.nxt())  print(a.nxt().nxt())  print(a.nxt().nxt().nxt())  print(a.nxt().nxt().nxt().nxt())  print(a.nxt().nxt().nxt().nxt().nxt())  print(a.nxt().nxt().nxt().nxt().nxt().nxt()) |

Output:

|  |
| --- |
|  |

1. Create a class *'****Student'*** first and construct objects with student ***'name'*** and ***'number'*** of course(s) she/he is taking in the current semester. The following operations can be allowed by using magic methods (*'dunder'* method), such as ***\_\_add\_\_(), \_\_str\_\_(), \_\_repr\_\_(), \_\_lt\_\_(), \_\_eq\_\_(), \_\_ne\_\_(), and \_\_gt\_\_()*.**

***class*** *Student():*

*"""*

*>>> a= Student ('Peter', 3)*

*>>> b= Student ('Mike', 4)*

*>>> c= Student ('John', 5)*

*>>> d= Student ('Kelvin', 3)*

*>>> a+b+d*

*10*

*>>> a!=d*

*False*

*>>> b<c*

*True*

*"""*

*ANS:*

|  |
| --- |
| class Student:      def \_\_init\_\_(self, name, courses):          self.name = name          self.courses = courses      def \_\_str\_\_(self):           return str(self.courses)      def \_\_repr\_\_(self):            return str(self)      def \_\_add\_\_(self, st):        return Student("", self.courses + st.courses)      def \_\_lt\_\_(self, st):            return self.courses < st.courses      def \_\_eq\_\_(self, st):            return self.courses == st.courses      def \_\_ne\_\_(self, st):             return self.courses != st.courses      def \_\_gt\_\_(self, st):          return self.name > st.courses  a = Student('Peter', 3)  b= Student('Mike', 4)  c = Student('John',5)  d = Student('Kelvin', 3)  print(a+b+d)  print(a!=d)  print( b<c) |

Output:

|  |
| --- |
|  |

**Part II Embedded System Design Theory**

1. Describe the application areas of the real time operating system (RTOS)

ANS: Real time operating system (RTOS), **Real-time operating system (RTOS)** is an operating system intended to serve real time applications that process data as it comes in, mostly without buffer delay. RTOS is used in many real time systems such as airlines reservation system, air traffic control system, systems that provide immediate updating, system that provides up to date and minute information on stock prices, defense application systems like RADAR, networked Multimedia Systems, Command Control Systems, Internet Telephony, Anti-lock Brake Systems, Heart Pacemaker etc because:

* It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing.
* The Real time OS provides API functions that allow cleaner and smaller application code.
* Abstracting timing dependencies and the task-based design results in fewer interdependencies between modules.
* RTOS offers modular task-based development, which allows modular task-based testing.
* The task-based API encourages modular development as a task, will typically have a clearly defined role. It allows designers/teams to work independently on their parts of the project.
* An RTOS is event-driven with no time wastage on processing time for the event which is not occur

1. Explain why the middleware is needed and where

ANS: Middleware is software that is used to bridge the gap between applications and other tools or databases. Middleware sits between an operating system and the applications that run on it or on each side of a distributed computer network. Middleware is needed to speed the development of distributed applications by simplifying connectivity between applications, application components and back-end data sources. Below is an example of where middleware is used:

* 1. **Transaction management:** Transactional middleware can be used to manage and control individual transactions to ensure that no problems corrupt the system or database in business or banking industry. It does this by controlling transaction apps, pushing database updates related to the transaction, and enforcing the business rules and logic of the transaction.
  2. **Message queues:** Message Oriented Middleware (MOM) is used in communication to allow messages to be sent and received between applications or systems.
  3. **Web server:** Middleware can also accept client requests from web browsers and channel them to the main server/database and then deliver the responses to the browsers.

1. Describe each component’s function of any operating system

ANS:

The components of an operating system play a key role in making a variety of computer system parts work together. There are the following components of an operating system with their functions, such as:

**Process Management:** the process management component is a procedure for managing many processes running simultaneously on the operating system. Every running software application program has one or more processes associated with them. For example, Process creation and deletion, suspension and resumption, synchronization process, communication process

**File Management:** The fille management part of the operating system has the following important functions:

* File and directory creation and deletion.
* For manipulating files and directories.
* Mapping files onto secondary storage.
* Backup files on stable storage media

**Network Management**: Network management provides the following functions, such as:

* Distributed systems help you to various computing resources in size and function. They may involve minicomputers, microprocessors, and many general-purpose computer systems.
* A distributed system also offers the user access to the various resources the network shares.
* It helps to access shared resources that help computation to speed up or offers data availability and reliability.

**Main Memory Management**: An Operating System performs the following functions for Memory Management in the operating system:

* It helps you to keep track of primary memory.
* Determine what part of it are in use by whom, what part is not in use.
* In a multiprogramming system, the OS decides which process will get memory and how much.
* Allocates the memory when a process requests.
* It also de-allocates the memory when a process no longer requires or has been terminated.

**Secondary Storage Management**: major functions of secondary storage management in the operating system:

* Storage allocation
* Free space management
* Disk scheduling

**I/O Device Management**: The I/O management system offers the following functions, such as:

* It offers a buffer caching system
* It provides general device driver code
* It provides drivers for particular hardware devices.
* I/O helps you to know the individualities of a specific device

**Security Management:** ensure the security and proper authorization of the processes that want to operate files, memory CPU, and other hardware resources from the operating system

Command Interpreter System: reads and interprets control statements automatically to be executed. The command statements deal with process management, I/O handling, secondary storage management, main memory management, file system access, protection, and networking.

1. What general functions are there in any device drivers, including the description for each?

ANS:

**Device Drivers** are important for a computer system to work properly. Device drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface. The main purpose of device drivers is to provide abstraction by acting as a translator between a hardware device and the applications or operating systems that use it. Almost every device associated with the computer system, a Device Driver exists for the particular hardware. But it can be broadly classified into the following two types

* + 1. **Kernel-mode Device Driver**

This Kernel-mode device driver includes some generic hardware that loads with an operating system as part of the OS. These are ***BIOS, motherboard, processor***, and some other hardware that are part of kernel software. These include the minimum system requirement device drivers for each operating system.

* **BIOS:** BIOS (basic input/output system) is the most basic computer driver in existence. It is designed to be the first program that boots when a PC turns on. The BIOS is stored on memory built into the motherboard and is designed to boot the hardware connected to the PC, including the hard drives, video display output, keyboard and mouse.
* **Motherboard Drivers:** Motherboard drivers are small programs that are read by either Windows or Linux and allow for basic computer functions while inside the operating system. These drivers normally include programs that allow broadband ports, USB ports and I/O ports for the mouse and keyboard. Depending on the making of the motherboard, the drivers may also have basic drivers for video and audio support.
  + 1. **User-mode Device Driver**

Other than the devices brought by the kernel for working of the system, the user also brings some devices for use during the using of a system that devices need device drivers to function those drivers fall under User mode device driver. For example, the user needs any plug and play action that comes under this.