Core CSE

Q1. List out different OOPS principles and explain? (10 marks)

There are four types of Principles in OPPs they are:

1. Encapsulation (Putting Something in Capsule)
2. Inheritance
3. Abstraction
4. Polymorphism

Encapsulation:

Encapsulation is the process of keeping the data and the code (methods) that manipulates that data together as a unit. Any variable can only be changed through a method in the same class that way data is protected from any intentional or accidental modification by any outside entity.

A class is an example of encapsulation as it wraps all the variables and methods defined with in that class.

Since class is an example of Encapsulation so defining a class in Python which wraps all the variables and methods is first step towards encapsulation. But the question is how to stop outside access to the variables as there are no explicit access modifiers like public, private, protected in Python and all the variables are public by default. Here is an example to clarify it-

class User:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display\_user(self):

print('User Name:', self.name)

print('User Age:', self.age)

user = User('Mike Dallas', 34)

# calling class method

user.display\_user()

# Accessing variables directly

print(user.name)

print(user.age)

**Output**

User Name: Mike Dallas

User Age: 34

Mike Dallas

34

Abstraction:

Abstraction is an extension of encapsulation. It means providing only the necessary information to the outside world while hiding the internal details of implementation. It reveals only the appropriate operations for other objects. The advantage of this is that we can change the implementation without affecting the class, as the method interface remains the same.

Let us take the example of a calculator, which takes the input from us, and at the press of a button, gives us the desired output while sparing us the internal details of how it has arrived at that answer

#### 3. Inheritance

Often, objects are similar in functionality, sharing part of the logic but differing in the rest. So how do we reuse the common logic and separate the different logic? This can be achieved by inheritance. In inheritance, we create a new class called the child class, which is derived from the class called the parent class, thus forming a hier0archy of classes. The child class reuses the data fields and methods required from the parent class and implements its unique functionality on its own.

For example, a vehicle can be a parent class, from which we can derive child classes like Bike and Car. They share the common properties of running on fuel and carrying passengers but differ in the number of passengers they can carry and more such properties.

#### 4. Polymorphism

Polymorphism is the ability to take more than one form. For example, suppose we have a parent class and a few of its child classes. Now we want to use attributes from both the parent and the child classes, so how will it be achieved? This can be done using Polymorphism. In Polymorphism, abstract entities are executed in multiple ways. It gives a way to consume a class exactly like the parent class, such that there is no confusion with mixing the type of classes, and each child class continues to keep its methods the way it was. This can be done by reusing a parent interface so that the child class can implement these methods in their own version.

Q2. List out Layers of TCP/IP Model and explain? (10 marks)

* The TCP/IP model was developed prior to the OSI model.
* The TCP/IP model is not exactly similar to the OSI model.
* The TCP/IP model consists of five layers: the application layer, transport layer, network layer, data link layer and physical layer.
* The first four layers provide physical standards, network interface, internetworking, and transport functions that correspond to the first four layers of the OSI model and these four layers are represented in TCP/IP model by a single layer called the application layer.
* TCP/IP is a hierarchical protocol made up of interactive modules, and each of them provides specific functionality.

Here, hierarchical means that each upper-layer protocol is supported by two or more lower-level protocols.

## Functions of TCP/IP layers:



## Network Access Layer

* A network layer is the lowest layer of the TCP/IP model.
* A network layer is the combination of the Physical layer and Data Link layer defined in the OSI reference model.
* It defines how the data should be sent physically through the network.
* This layer is mainly responsible for the transmission of the data between two devices on the same network.
* The functions carried out by this layer are encapsulating the IP datagram into frames transmitted by the network and mapping of IP addresses into physical addresses.
* The protocols used by this layer are ethernet, token ring, FDDI, X.25, frame relay.

## Internet Layer

* An internet layer is the second layer of the TCP/IP model.
* An internet layer is also known as the network layer.
* The main responsibility of the internet layer is to send the packets from any network, and they arrive at the destination irrespective of the route they take.

### **Following are the protocols used in this layer are:**

**IP Protocol:** IP protocol is used in this layer, and it is the most significant part of the entire TCP/IP suite.

Following are the responsibilities of this protocol:

* **IP Addressing:** This protocol implements logical host addresses known as IP addresses. The IP addresses are used by the internet and higher layers to identify the device and to provide internetwork routing.
* **Host-to-host communication:** It determines the path through which the data is to be transmitted.
* **Data Encapsulation and Formatting:** An IP protocol accepts the data from the transport layer protocol. An IP protocol ensures that the data is sent and received securely, it encapsulates the data into message known as IP datagram.
* **Fragmentation and Reassembly:** The limit imposed on the size of the IP datagram by data link layer protocol is known as Maximum Transmission unit (MTU). If the size of IP datagram is greater than the MTU unit, then the IP protocol splits the datagram into smaller units so that they can travel over the local network. Fragmentation can be done by the sender or intermediate router. At the receiver side, all the fragments are reassembled to form an original message.
* **Routing:** When IP datagram is sent over the same local network such as LAN, MAN, WAN, it is known as direct delivery. When source and destination are on the distant network, then the IP datagram is sent indirectly. This can be accomplished by routing the IP datagram through various devices such as routers.

Q3. Construct a binary tree by using postorder and inorder sequences given below. Inorder: N, M, P, O, Q Postorder: N, P, Q, O, M (10 marks)

from typing import List, Optional # for annotations

# post\_order\_index = 0

class Node :

def \_\_init\_\_ (self, val=0, left=None, right=None) :

self.val = val

self.left = left

self.right = right

class Tree :

def \_\_init\_\_ (self, in\_order\_seq : List[int], post\_order\_seq : List[int]) :

self.post\_order = post\_order\_seq

self.in\_order = in\_order\_seq

self.nodes = len(post\_order\_seq)

# Use a dictionary to find the index of a node in the inorder sequence

self.map\_in\_order = {}

for index in range(self.nodes) :

self.map\_in\_order[in\_order\_seq[index]] = index

# global post\_ordex\_index

self.post\_order\_index = self.nodes - 1

# Recursively construct a binary tree from an in\_order and post\_order sequence

def Construct (self, start : int, end : int) -> Optional[Node] :

# Check if all the nodes in the post-order sequence are processed. i.e if end < start

if (start > end) :

return None

root = Node (self.post\_order[self.post\_order\_index])

# Fetch the index of the root node in the in\_order sequence

# to get the range of nodes in the left and right subtrees.

index = self.map\_in\_order[root.val]

# Process the nodes from right to left

self.post\_order\_index -= 1

# Recursively construct right sub-tree

root.right = self.Construct (index + 1, end)

# Recursively construct left sub-tree

root.left = self.Construct (start, index - 1)

return root

# Recursive function for inorder traversal

def In\_Order\_Traversal (root : Node) :

if (root) :

In\_Order\_Traversal (root.left)

print(root.val, end=' ')

In\_Order\_Traversal (root.right)

# Recursive function for postorder traversal

def Post\_Order\_Traversal (root : Node) :

if (root) :

Post\_Order\_Traversal (root.left)

Post\_Order\_Traversal (root.right)

print(root.val, end=' ')

# Construct a binary tree from inorder and preorder traversals.

def BuildTree (in\_order\_seq : List[int], post\_order\_seq : List[int]) -> Optional[Node] :

t = Tree (in\_order\_seq, post\_order\_seq)

nodes = len(post\_order\_seq)

return t.Construct (0, nodes-1)

def main() :

# Construct the following tree

# 10

# / \

# 12 30

# / / \

# 44 50 16

post\_order = [ 44, 12, 50, 16, 30, 10 ]

in\_order = [ 44, 12, 10, 50, 30, 16 ]

print("\nGiven in-order traversal : ", end=' ')

for node in in\_order :

print(node, end=' ')

print("\nGiven post-order traversal : ", end=' ')

for node in post\_order :

print(node, end=' ')

root = BuildTree (in\_order, post\_order)

print("\nIn-order traversal of the constructed tree : ", end=' ')

In\_Order\_Traversal(root)

print("\nPost-order traversal of the constructed tree : ", end=' ')

Post\_Order\_Traversal(root)

if \_\_name\_\_ == "\_\_main\_\_" :

main()

Q5. Explain LRU cache and its implementation by taking some examples and explaining all steps. (5 marks)

**Caching**, in general, is a technique that improves software performance. A cache is a location in memory or storage that is computationally cheaper and faster to access. The LRU strategy evicts the **least**recently used items from the cache, only keeping the **most** recently used items. The LRU cache is used when one wants to reuse previously computed values.

In Python, the *lru\_cache* function decorator implements LRU caching. The decorator wraps the function and memoizes up to the specified amount of function calls. Recall that **memoization** stores results of function calls and returns the cached result if and when the same inputs re-occur. Recall also that a **decorator** in Python is a function that takes another function as its argument, and returns yet another function.

Internally, Python uses a **dictionary** data structure to implement caching.

Let’s write some code. Here, we will use employ LRU caching in a famous use case:

from functools import lru\_cache

@lru\_cache(maxsize = 256)

def fibonacci\_sequence(n):

if n <= 1:

return n

else:

return(fibonacci\_sequence(n-1) + fibonacci\_sequence(n-2))

n = 25

print("Fibonacci sequence:")

for i in range(n):

print(fibonacci\_sequence(i))

#Output

#Fibonacci sequence:

#0

#1

#1

#2

#3

#...

#10946

#17711

#28657

#46368

Let’s explain what is happening here:

1. From the *functools* library we import *lru\_cache*.
2. We include our *@lru\_cache*decorator which takes two potential parameters: *maxsize*(defaults to 128)and *typed*(which defaults to False). The *maxsize*parameter is the size of our LRU cache and the maximum amount of function calls that will memoized or cached. We used a size of 256 in this case.
3. We perform the usual steps of the Fibonacci Sequence algorithm which we include in the *fibonacci\_sequence()*function. Note that we are using a **recursive**algorithm here so this will require high potential memory utilization. This is one use case that could always benefit from LRU caching.
4. We call the function in a loop to print the first *n*members of the sequence.

Easy right? Once the above code executes we will see a list of the first *n*members of the Fibonacci. But how do we know if the cache actually improved the performance of our script? Seeing is believing so let’s actually see *how* the cache was used:

print(fibonacci\_sequence.cache\_info())

#Output

#CacheInfo(hits=46, misses=25, maxsize=256, currsize=25)

The *cache\_info()*function is how we will know if the cache is doing its job. It returns a named tuple showing *hits*, *misses*, *maxsize* and *currsize*. The *hits* value is of particular importance because it shows how many times the cache is actually being used. The *currsize*is the current amount of cached function calls and the *misse*s value is the amount of times the cache was referenced but the value entry sought didn’t exist in the cache. In our case, the cache is actually being used!

The *fibonacci\_sequence()* function takes only one parameter argument but in LRU caching, different argument patterns will have different calls and different cache entries. For example, if we had argument x=10 and y=20 for function z, this would be different in the cache to y=10 and x=20 for function z. Both argument patterns for function z will require their own separate calls to the cache.

Now, let’s see how the speed of our code improved due to LRU caching. We will do a speed comparison of an LRU Cached vs Non-LRU Cached version of the code. Let’s go:

from functools import lru\_cache

import time

#lru cached version

@lru\_cache(maxsize = 256)

def fibonacci\_sequence(n):

if n <= 1:

return n

else:

return(fibonacci\_sequence(n-1) + fibonacci\_sequence(n-2))

#non-lru cached version

def non\_cached\_fibonacci\_sequence(n):

if n <= 1:

return n

else:

return(non\_cached\_fibonacci\_sequence(n-1) + non\_cached\_fibonacci\_sequence(n-2))

n = 25

print("Fibonacci sequence:")

start = time.time()

for i in range(n):

print(fibonacci\_sequence(i))

finish = time.time()

print(f"The cached version took: {finish-start}")

print(fibonacci\_sequence.cache\_info())

start = time.time()

for i in range(n):

print(non\_cached\_fibonacci\_sequence(i))

finish = time.time()

print(f"The non-cached version took: {finish-start}")

Let’s explain what’s going on here:

1. Aside from the usual *lru\_cache*import, we also import the *time* library which we will use to see the difference in the time taken for the functions to execute.
2. We define two functions: the LRU cached version *fibonacci\_sequence* with the usual decorator and the non-LRU cached version *non\_cached\_fibonacci\_sequence* without the decorator.
3. The value of *n*is 25 which means we want the first 25 members of the Fibonacci sequence.
4. We execute both in sequence. First the cached version and then the non-cached version of the function.

Great! Once this program executes we will get the following output or something very similar:

# Fibonacci sequence:

# 0

# 1

#...

# 28657

# 46368

# The cached version took: 0.0

# CacheInfo(hits=46, misses=25, maxsize=256, currsize=25)

# 0

# 1

# ...

# 28657

# 46368

# The non-cached version took: 0.0625150203704834

Note that the cached version took 0 seconds (really a very very small number) and the cached version took approximately 0.00625 seconds. So not only is the LRU cached being used but it is actually improving the speed at which our code executes.

When you try this code, time taken will not be the exact same as mine, but they will be close.

We used small quantities for this example, but at scale you can certainly imagine how using a cache will be CRITICAL to maintaining high performance.

Q6. Explain virtual memory. (5 marks)

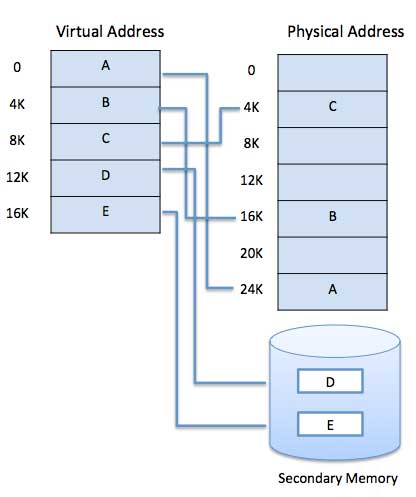
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

* User written error handling routines are used only when an error occurred in the data or computation.
* Certain options and features of a program may be used rarely.
* Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
* The ability to execute a program that is only partially in memory would counter many benefits.
* Less number of I/O would be needed to load or swap each user program into memory.
* A program would no longer be constrained by the amount of physical memory that is available.
* Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

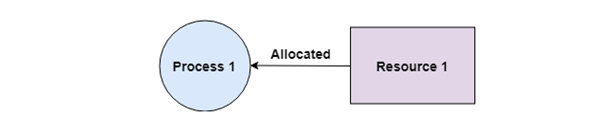
Q7. Explain Deadlock and its characteristics. (5 marks)

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows −

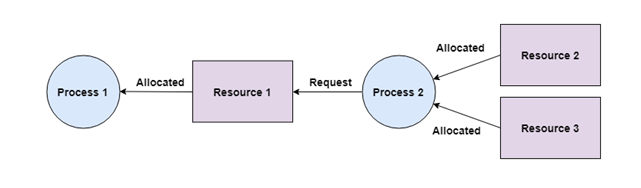
**Mutual Exclusion**

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



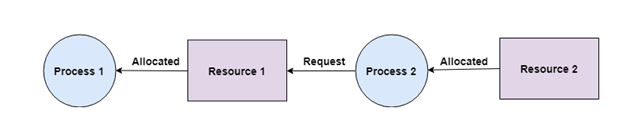
**Hold and Wait**

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



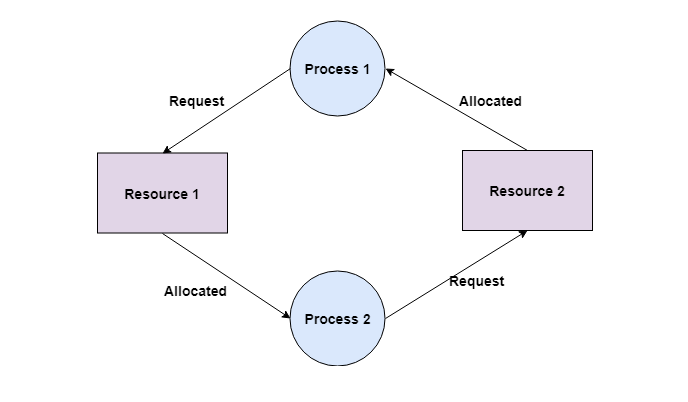
**No Preemption**

A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



**Circular Wait**

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



Q8. Explain NAT and ARP protocol? ( 5 marks)

To access the Internet, one public IP address is needed, but we can use a private IP address in our private network. The idea of NAT is to allow multiple devices to access the Internet through a single public address. To achieve this, the translation of a private IP address to a public IP address is required. **Network Address Translation (NAT)** is a process in which one or more local IP address is translated into one or more Global IP address and vice versa in order to provide Internet access to the local hosts. Also, it does the translation of port numbers i.e. masks the port number of the host with another port number, in the packet that will be routed to the destination. It then makes the corresponding entries of IP address and port number in the NAT table. NAT generally operates on a router or firewall.

**Network Address Translation (NAT) working –**   
Generally, the border router is configured for NAT i.e the router which has one interface in the local (inside) network and one interface in the global (outside) network. When a packet traverse outside the local (inside) network, then NAT converts that local (private) IP address to a global (public) IP address. When a packet enters the local network, the global (public) IP address is converted to a local (private) IP address.

If NAT runs out of addresses, i.e., no address is left in the pool configured then the packets will be dropped and an Internet Control Message Protocol (ICMP) host unreachable packet to the destination is sent.

**Why mask port numbers ?**  
Suppose, in a network, two hosts A and B are connected. Now, both of them request for the same destination, on the same port number, say 1000, on the host side, at the same time. If NAT does only translation of IP addresses, then when their packets will arrive at the NAT, both of their IP addresses would be masked by the public IP address of the network and sent to the destination. Destination will send replies to the public IP address of the router. Thus, on receiving a reply, it will be unclear to NAT as to which reply belongs to which host (because source port numbers for both A and B are the same). Hence, to avoid such a problem, NAT masks the source port number as well and makes an entry in the NAT table.

**NAT inside and outside addresses –**   
Inside refers to the addresses which must be translated. Outside refers to the addresses which are not in control of an organization. These are the network Addresses in which the translation of the addresses will be done.

* **Inside local address –** An IP address that is assigned to a host on the Inside (local) network. The address is probably not an IP address assigned by the service provider i.e., these are private IP addresses. This is the inside host seen from the inside network.
* **Inside global address –** IP address that represents one or more inside local IP addresses to the outside world. This is the inside host as seen from the outside network.
* **Outside local address –** This is the actual IP address of the destination host in the local network after translation.
* **Outside global address –** This is the outside host as seen from the outside network. It is the IP address of the outside destination host before translation.

**Network Address Translation (NAT) Types –**   
There are 3 ways to configure NAT: 

1. **Static NAT –** In this, a single unregistered (Private) IP address is mapped with a legally registered (Public) IP address i.e one-to-one mapping between local and global addresses. This is generally used for Web hosting. These are not used in organizations as there are many devices that will need Internet access and to provide Internet access, a public IP address is needed.

Suppose, if there are 3000 devices that need access to the Internet, the organization has to buy 3000 public addresses that will be very costly. 

1. **Dynamic NAT –** In this type of NAT, an unregistered IP address is translated into a registered (Public) IP address from a pool of public IP addresses. If the IP address of the pool is not free, then the packet will be dropped as only a fixed number of private IP addresses can be translated to public addresses.

Suppose, if there is a pool of 2 public IP addresses then only 2 private IP addresses can be translated at a given time. If 3rd private IP address wants to access the Internet then the packet will be dropped therefore many private IP addresses are mapped to a pool of public IP addresses. NAT is used when the number of users who want to access the Internet is fixed. This is also very costly as the organization has to buy many global IP addresses to make a pool. 

1. **Port Address Translation (PAT) –** This is also known as NAT overload. In this, many local (private) IP addresses can be translated to a single registered IP address. Port numbers are used to distinguish the traffic i.e., which traffic belongs to which IP address. This is most frequently used as it is cost-effective as thousands of users can be connected to the Internet by using only one real global (public) IP address.

**Advantages of NAT –** 

* NAT conserves legally registered IP addresses.
* It provides privacy as the device’s IP address, sending and receiving the traffic, will be hidden.
* Eliminates address renumbering when a network evolves.

**Disadvantage of NAT –** 

* Translation results in switching path delays.
* Certain applications will not function while NAT is enabled.
* Complicates tunneling protocols such as IPsec.
* Also, the router being a network layer device, should not tamper with port numbers(transport layer) but it has to do so because of NAT.

**Address Resolution Protocol:**

* ARP stands for **Address Resolution Protocol**.
* ARP is a network layer protocol which is used to find the physical address from the IP address.
* **The two terms are mainly associated with the ARP Protocol:**
  + **ARP request:** When a sender wants to know the physical address of the device, it broadcasts the ARP request to the network.
  + **ARP reply:** Every device attached to the network will accept the ARP request and process the request, but only recipient recognize the IP address and sends back its physical address in the form of ARP reply. The recipient adds the physical address both to its cache memory and to the datagram header