

INDIAN INSTITUTE OF TECHNOLOGY ROPAR
GE 103: Introduction to Computing and Data Structures
First Semester of Academic Year 2024-2025
End Semester Examination



Duration: 3 Hours [2:30 PM – 5:30 PM] Max Points: 50

Date: 6-MAY-2024

Instructions:

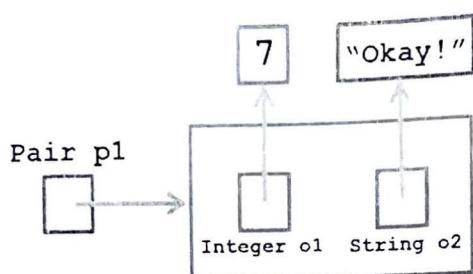
- There are 14 questions in the exam. All of the questions are mandatory.
- The points for each question are mentioned next to the question.
- Answer each question in the space provided for that question only.
- Structure your answer such that it does not go beyond the space allocated.
- No clarifications will be entertained during the examination. If you feel that a question is not clear, state your assumptions while answering.

Maximum Points	1•	2	3•	4•	5•	6•	7	8•
	3	3	4	4	5	2	4	4
Obtained Points								
Maximum Points	9•	10•	11•	12•	13•	14•	Total	
	3	3	4	4	3	4		
Obtained Points								

Please Turn Over.

Q1. The below diagram shows an object `p1` of the `Pair` class containing two values, one for each member variable associated with the object. Write the Object Oriented Python code corresponding to the diagram shown below. The following may be taken into account: [3 points]

- A. You need to declare a class called `Pair`, with two member variables, as shown below.
- B. Include a constructor function.



~~class Pair:~~

~~def __init__(self, integerO1, stringO2):~~
~~self.integerO1 = integerO1,~~

~~class Pair:~~

~~def __init__(self):~~
~~self.integerO1 = 7.~~
~~self.stringO2 = "Okay!"~~

`p=pair()`

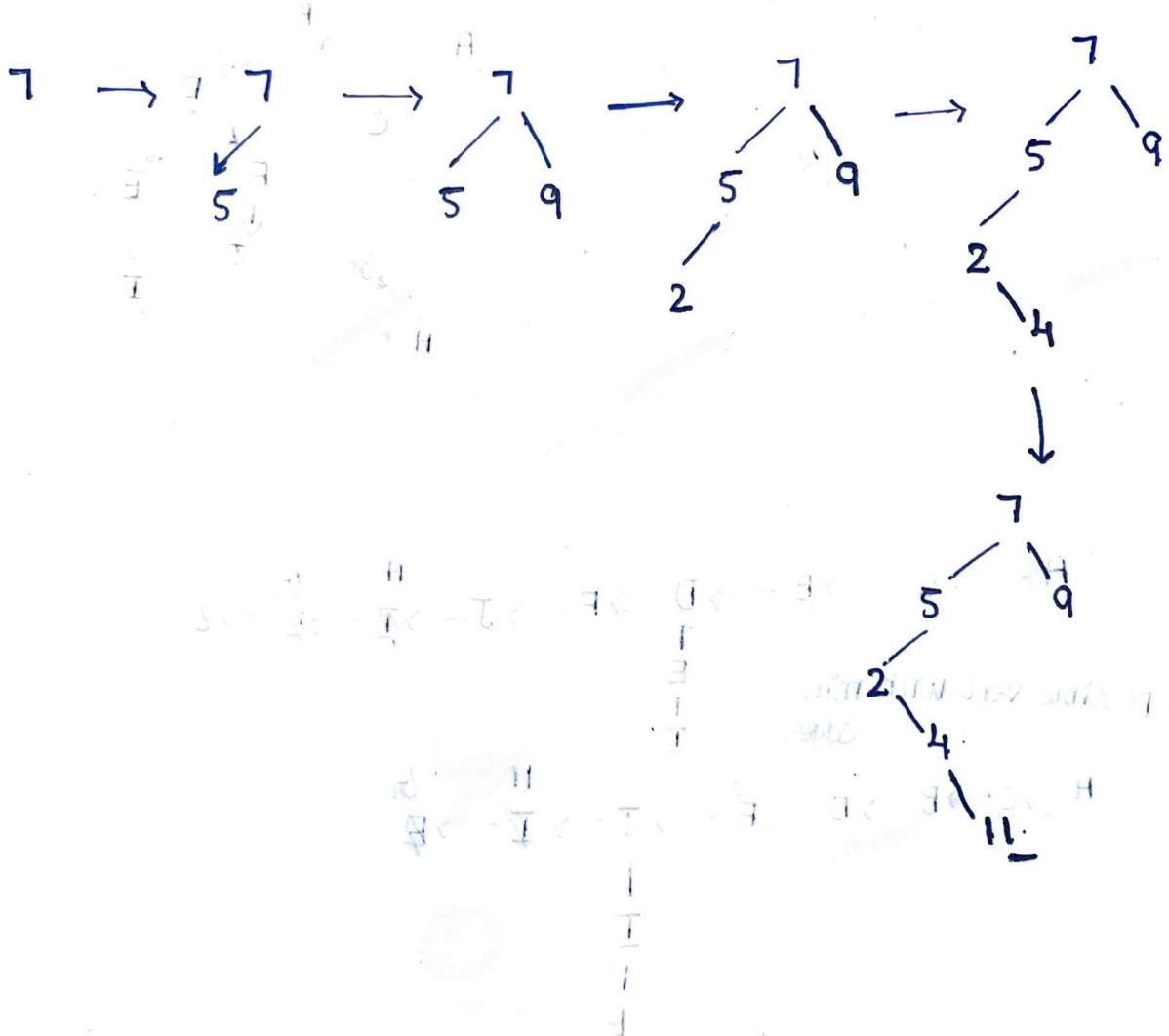
`print(p)`

Q2. Binary Search Trees

- A. Draw the resulting Binary Search Tree at each step after inserting and deleting the nodes in the following order – (i) insert 7, (ii) insert 5, (iii) insert 9, (iv) insert 2, (v) insert 4, (vi) insert 11, (vii) delete 4. Assume an empty tree at the beginning. Show the tree after performing each step. [2 points]

Binary Search tree → Left node is always less than parent & right node
always greater than parent

following
insert left
insert right

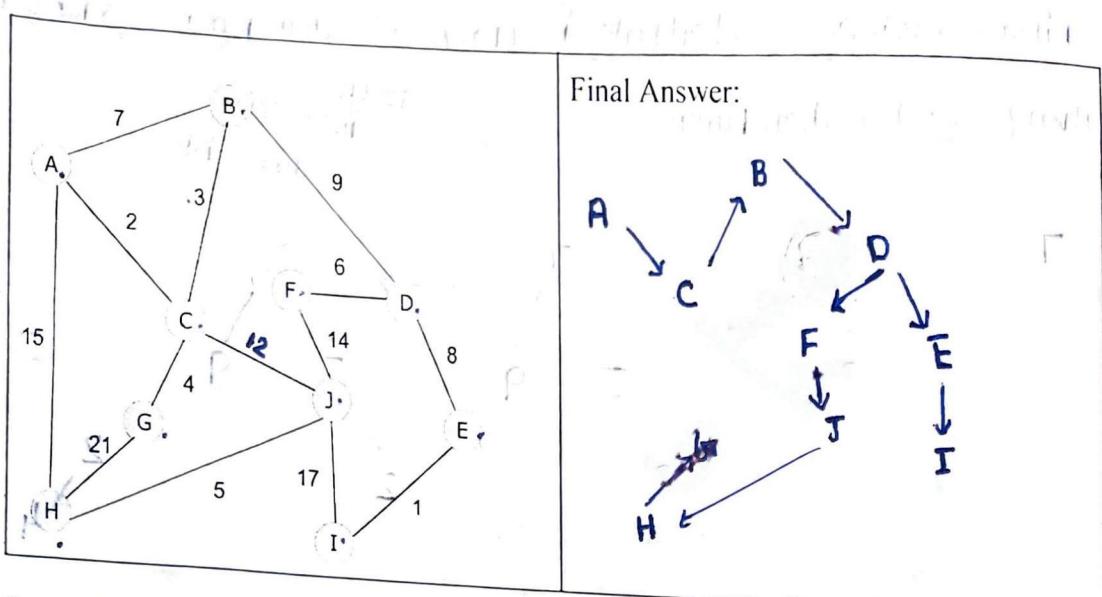


- B. Is the final tree balanced? Answer with just 'yes' or 'no'.

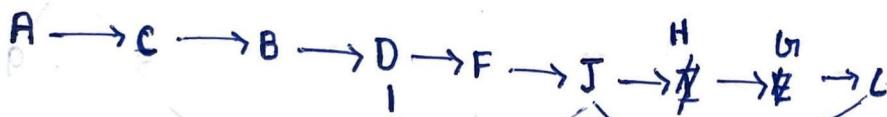
[1 point]

No!

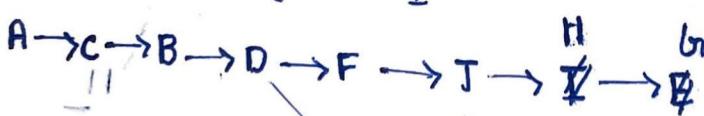
Q3. Draw the minimum spanning tree that is generated after applying Prim's algorithm to the graph below. Start with node A. Only draw the final minimum spanning tree, no explanation is required. [4 points]



Space for Rough Work:



All possible vert with min edges



I
I
I
E

Q4. Write down the expression of the worst case Big O complexity for the following operations. No explanation is necessary. [1 x 4 = 4 points]

A. Contains function: returns true or false depending on whether a particular object is a member of a list.

a. Sorted list implemented as an array. Answer: O(n)

b. Sorted list implemented as a single linked list (head reference only).
Answer: O(n)

B. Insert function: add an object to the list in the appropriate place.

a. Unsorted list implemented as an array. Answer: O(n)

b. Sorted list implemented as an array. Answer: O(n)

Q5. Circle the correct answer. Only one answer correct per question. [1 x 5 = 5 points]

a) Which of the following functions grows the fastest? (b)

a. $n \log n$	<u>b. 2^n</u>
c. $\log n$	d. n^2

b) For a linked list implementation of a queue, if both front and rear have identical valid node values, the size of the queue is: (d)

a. 0	b. 1
c. 2	<u>d. the answer cannot be determined</u>

c) For the linked implementation of a stack, where are the push and pop operations performed? (a)

<u>a. Push in front of first element, pop the first element</u>	b. Push after last element, pop the last element
c. Push after last element, pop the first element	d. Push after first element, pop the first element

d) Which of the following does the binary heap implement? (b)

a. Binary search tree	<u>b. Priority queue</u>
c. Stack	d. None of the above

e) Which of the following would require the most extra space, on average? (a)

<u>a. Bubble sort</u>	b. Merge sort
c. Quick sort	d. Selection sort

Q6. List two applications each for Stacks and Queues. [2 points]

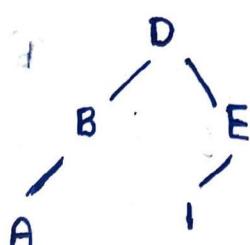
Stack (Application 1) Stacks are used in depth first search algorithm	Stack (Application 2) Stacks are used for application involving backtracking & recursion in algorithms
Queue (Application 1) Queues are used in breadth first search algorithm in graphs	Queue (Application 2) Queues are used in computer networks, Router, Switch queues and email queues

Q7. Consider the results of the following binary tree traversals: (i) Inorder = DBHEIAFCG, and (ii) Preorder = ABDEHICFG. **NLR**

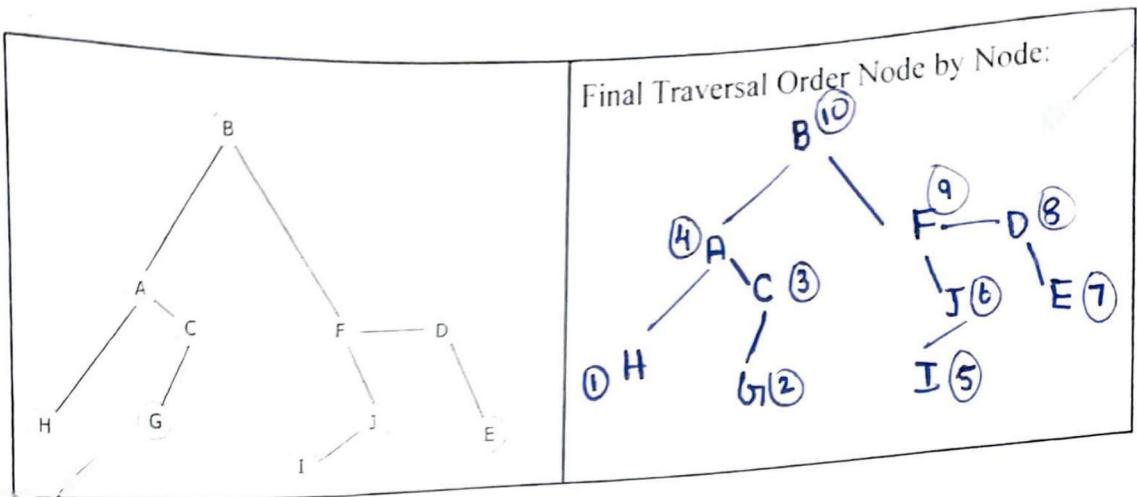
Construct a single corresponding binary tree that satisfies both these traversal sequences. [4 points]

Inorder → root, left, right

Preorder → left, root, right



Q8. Consider the following graph. Draw the postorder traversal of the following binary tree, starting at node B. Show all the steps, and the contents of the stack at each step. [4 points]



Stepwise Answer:

Stepwise Answer:
post order traversal \rightarrow visits left then right then visits the root node

root node B $\xrightarrow{\text{left}}$ A $\xrightarrow{\text{left}}$ H (visited first)

A diagram of a binary search tree node. The node has two children, both labeled 'right'. The left child is labeled 'C' and the right child is labeled 'G1 (visited second)'. An arrow points from the label 'left' to the node's left child.

\downarrow back tracking yet parent
 F
left $\swarrow \searrow \dots$ C (visited third)

left right C (visual area)
T D ↓, back tracking

left ↓ right
E (7th) A (visited fourth)

I(visited 5th) - (,)
| back tracking) backtracking

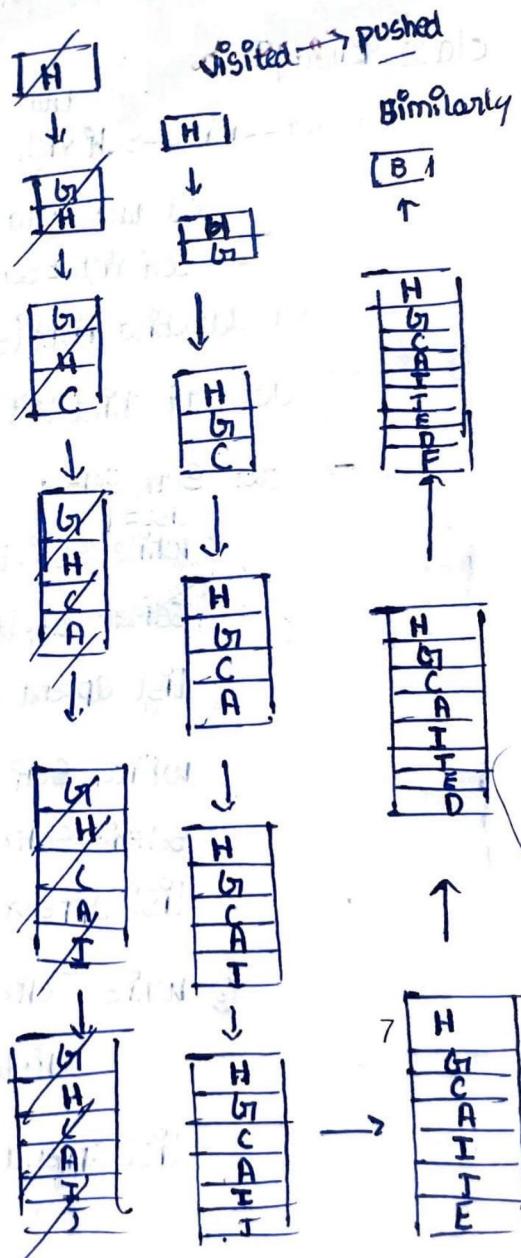
$\tau(6^{\text{th}})$ $D(8^{\text{th}})$

↓ backtracking

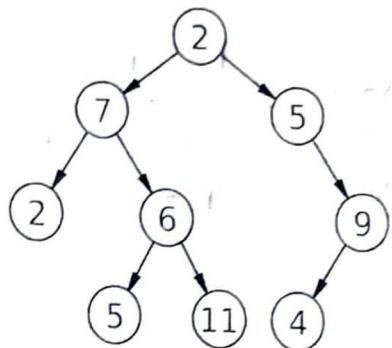
$F(9^{th})$

To rootnode

B(10th) 100% 



Q9. Given the definition of binary tree discussed in class, write the pseudocode for a recursive function named `SumLeaf()`, that takes the root of a binary tree, and returns the sum of all the leaf nodes. For example, if you pass the root of the following binary tree, the function should return $2 + 5 + 11 + 4 = 22$. [3 points]



Binary tree: A tree data structure with all the root/parent nodes having atmost two children and are termed as right and left child.

class Binary Tree:

→ def __init__(self, right, left)

 self.data = data

 self.right = self.left = None

→ def addchild_right(self, child)

→ def addchild_left(self, child)

→ def sum(self):

 list = []

 ① while self.left:

 self.left = self.left.left

 list.append(self.left) → (2)

 ② while self.right:

 self.right = self.right.right

 list.append(self.right) → (4)

 ③ while self.left:

 self.left = self.left.right

 list.append(self.left) → (11)

 list = [2, 4, 11, 5]

 for i in list

 sum += i

 ④ while self.left:

 self.left = self.left.left

 list.append(self.left) → (5)

Q10. Postfix notation

- A. Write the following arithmetic expression in postfix notation: $5 + 3 * (2 + 4)$. Use the BODMAS (Brackets, Order, Division, Multiplication, Addition, Subtraction) rule for determining the order of operations. [1 point]

Prefix

$$5+3*(2+4) \rightarrow +5*3(+24)$$

Ans: $+5*3+(24)$

*maybe
(or) +5*3()+24
in order brackets application in stack
after addition of
reversing tokens*

- B. Next, use a stack to evaluate the above postfix expression. Show the push and pop operations used for each input character step by step, and the resulting stack after each step. [2 points]

define class Stack:

wish push, pop, isEmpty operations

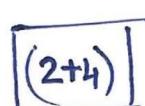
and proceed with expression evaluation

reverse tokens and push in if it is operand pop two out apply operator & push in
if it is operator

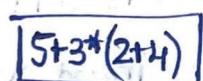
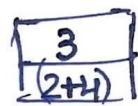
$$(24)+3*5+ \rightarrow \text{reversed}$$



pop '2' and '4'
apply operator '+'
and push in



PUSH 3

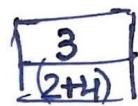


pop '5' & '3*(2+4)'

apply operator '+'
and push



PUSH 5



pop '3' and
'(2+4)'
apply '*' operator
push

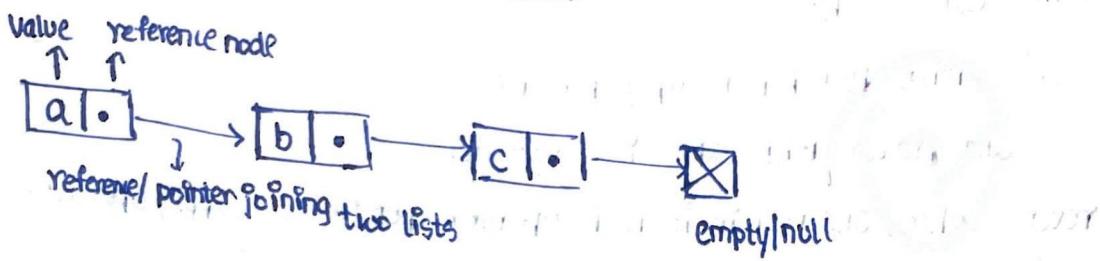
This is our required

Value

Q11. Linked Lists:

(a) What is a linked list data structure? Explain with the help of a pictorial representation. [2 points]

1. Linked lists are the data structure which contain nodes in which one of the node contains value and other contains reference to the next node
2. In linked lists unlike arrays each element points to the next
3. Due to this property of linked lists addition and removal of elements is easier
4. Search time in linked lists is linear



(b) Give one advantage and one disadvantage of the linked list data structure over an array. [2 points]

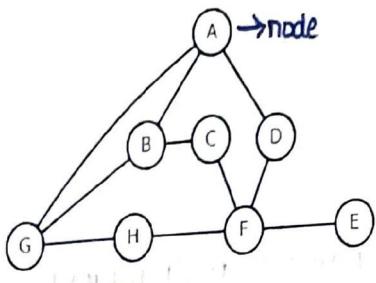
Advantage :

1. Addition and removal of elements in linked lists is easier than of array since linked lists are not given by memory locations instead every element points to the next.
2. Linked lists are used to implement queues and stacks more efficiently than lists.

Disadvantage :

1. The search time over linked lists is linear. $O(n)$
2. Arrays are more abstract classes than linked lists which make the life of user easy.

Q12. Given the following graph:



Explain and show the steps for breadth-first search traversal using the ~~stack~~ data structure, starting from node A.

[4 points]

Breadth first search:

Class Tree node:

```
def __init__(self, data)
    self.data = data
    self.left = None
    self.right = None
```

```
class Queue():
    def bfs_traversal(Node):
        if Node == None:
            return []
        Queue = Queue()
        Queue.enqueue(node)
        visited = []
```

Implement class Queue():

```
# while not queue.isEmpty()
    current_node = queue.dequeue()
    visited.append(current_node)
    if queue.isLeft():
        queue.enqueue(queue.isLeft())
    if queue.isRight():
        queue.enqueue(queue.isRight())
    return visited
```

The queue initially contains [A] \Rightarrow visited = [A]

left of root A = B \rightarrow enqueued
at same depth \rightarrow visited

visited = [A, B]

right of A = C \rightarrow visited
at same depth

visited = [A, B, C]

right of A = D \rightarrow visited

visited = [A, B, C, D]

left of B = G \rightarrow visits

visited = [A, B, C, D, B]

at same depth of B = H \rightarrow visits

[A, B, C, D, B, H]

at left of D = F

(A, B, C, D, B, H, F)

at same level of F = E

[A, B, C, D, B, H, F, E],

Q13. Mutability:

- (a) Give an example to illustrate the difference between mutable and immutable data types in Python. [2 points]

Mutable data types: The data types which can be changed (or) modified without changing memory address Ex: Lists, dictionaries, Sets

Immutable data types: Memory address changes on changing data types Ex: tuple, integers, float, bool

Ex:

(Mutable)

List = [1, 2, 3]

hex(id(List)) = A

List.append(4)

>> List = [1, 2, 3, 4]

hex(id(List)) = A

list → before
list → after

(Immutable)

tuple = (1, 2, 3)

b = (4,)

hex(id(tuple)) = B

tuple → initially
tuple → C

tuple = tuple + b

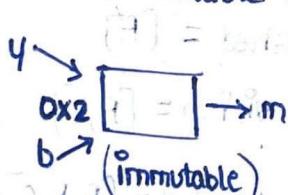
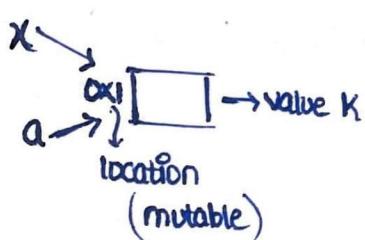
>> tuple = (1, 2, 3, 4)

hex(id(tuple)) = C

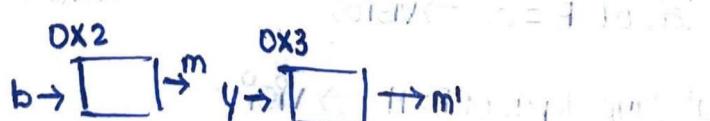
- (b) How does the mutability of function arguments affect the behavior of functions in Python? [1 point]

In functions we know when the function is called the formal parameters are bound to the actual parameters.

Consider '2' parameters one mutable and another immutable



a, b → actual parameters
x, y → formal parameters



For mutable arguments any change in arguments inside function also affects the actual parameter outside function. For immutable arguments both formal & actual parameters are distinct.

Q14. Access Modifiers:

- A. List and explain the different access modifiers available in Python with an example. [2 points]

Different access modifiers available in python are private & protected

- private members can be only accessed ^{& modified} within the class cannot be accessed in the child class (or) outside (given by adding double underscore before variable).
- protected members are only accessed and changed within parent and child classes

class Parent:

```
def __init__(self):  
    self._d = 42
```

class Child(Parent):

```
def __init__(self):  
    Parent.__init__(self)
```

```
    self._d = 42
```

```
def display(self):
```

```
    print(self._d)
```

#error

O₂=parent

O₁=child()

O₁.display()

O₂.self._d

1 & 2 produce error

class Parent:
(single underscore)

```
def __init__(self):  
    self._d = 42
```

class Child(Parent):

```
def __init__(self):
```

```
    Parent.__init__(self)
```

```
def display(self):
```

```
    print(self._d) # will work
```

- B. What is inheritance property in the Python OOP paradigm? List two advantages of the inheritance property. [2 points]

Inheritance is the property (or) capability of a class to derive (or) inherit the properties of other class.

The class which inherits the properties is called child/subclass and from which properties are inherited is the parent/super class.

The child class derives all the properties from parent class and can also have methods defined unique to itself.

Advantages:

1. Inheritance provides reusability of code.
2. Inheritance represents the realworld relationships well.
3. Inheritance helps to reduce the amount of code to be written in child class providing a level of abstraction.