Data Structure Using Python (313306)

Model Question Paper — Set A · Model Answers

Q.1

a) Features of Python. 2 M

1. **Simple and easy to learn** — English-like syntax.
2. **Interpreted** — code is executed line by line.
3. **Dynamically typed** — no need to declare variable type.
4. **Object oriented** — supports classes and objects.
5. **Free and open source, platform independent (portable).**
6. **Large standard library** and extensible with C/C++.

(Any four, ½ M each)

b) Linear and non-linear data structure. 2 M

Linear data structure: Elements are arranged in a sequence, one after another; each element has a unique predecessor and successor. Example: array / list, stack, queue, linked list.

Non-linear data structure: Elements are not arranged sequentially; one element can be connected to many elements (hierarchical / network). Example: tree, graph.

(Each definition with example 1 M)

c) Basic tuple operations. 2 M

- **Concatenation (+):** (1, 2) + (3,) → (1, 2, 3)
- **Repetition (*):** ("a",) * 3 → ("a", "a", "a")
- **Indexing / slicing:** t = (5, 10, 15); t[1] → 10, t[0:2] → (5, 10)
- **Membership:** 10 in t → True

(Any two with example, 1 M each)

d) Data abstraction and data encapsulation. 2 M

i) Data abstraction: Showing only the essential features of an object and hiding the internal details of how they are implemented. Example: we call `list.sort()` without knowing the sorting logic.

ii) Data encapsulation: Wrapping (binding) data members and the methods that work on them together into a single unit called a class, and restricting direct access to data.

e) Types of inheritance. 2 M

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance

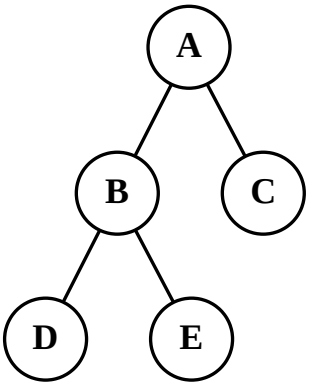
5. Hybrid inheritance

(Any four, ½ M each)

f) Leaf node and degree of a node. 2 M

i) Leaf node: A node which has no child (degree 0) is called a leaf or terminal node.

ii) Degree of a node: The number of children (sub-trees) of a node.



C, D, E are leaf nodes; degree of A = 2, degree of C = 0

g) Renaming a module. 2 M

Syntax:

```
import module_name as new_name
```

Example:

```
import math as m
print(m.sqrt(49))      # 7.0
```

The module is then used only through the new (alias) name.

Q.2

a) Built-in functions. 4 M

Built-in functions are always available in Python without importing any module.

Function	Use
len(x)	Returns number of items in a sequence.
max(x) / min(x)	Returns largest / smallest item.
sorted(x)	Returns a new sorted list.
type(x)	Returns the data type of x.
abs(x), round(x, n)	Absolute value; round to n decimal places.

```
nums = [12, 5, 27, 8]
print(len(nums))        # number of items
print(max(nums), min(nums))
print(sorted(nums))     # new sorted list
```

```
print(type(3.5))        # data type
print(abs(-15), round(7.468, 2))
```

Output:

```
4
27 5
[5, 8, 12, 27]
<class 'float'>
15 7.47
```

(Any four functions with example, 1 M each)

b) List vs array. 4 M

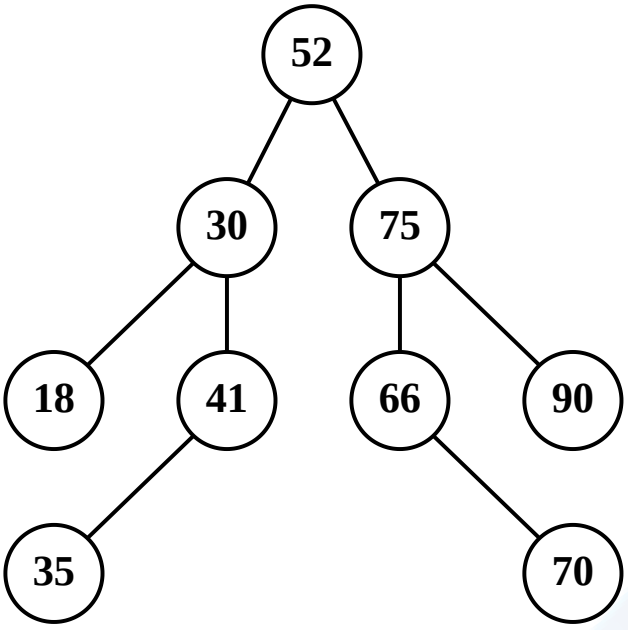
List	Array
Built-in data type; no import needed.	Needs array module or NumPy (import array / numpy).
Can store elements of different data types.	Stores elements of the same data type only.
Arithmetic on all elements needs a loop.	Element-wise operations are direct (e.g. a * 2 in NumPy).
Uses more memory.	Compact; uses less memory.
Example: L = [1, "a", 2.5]	Example: np.array([1, 2, 3])

(Any four points, 1 M each)

c) BST for 52, 30, 75, 18, 41, 66, 90, 35, 70. 4 M

First value is the root. Smaller values go to the left sub-tree and greater values go to the right sub-tree.

1. **52**: root
2. **30**: 30 < 52 → left of 52
3. **75**: 75 > 52 → right of 52
4. **18**: 18 < 52 , 18 < 30 → left of 30
5. **41**: 41 < 52 , 41 > 30 → right of 30
6. **66**: 66 > 52 , 66 < 75 → left of 75
7. **90**: 90 > 52 , 90 > 75 → right of 75
8. **35**: 35 < 52 , 35 > 30 , 35 < 41 → left of 41
9. **70**: 70 > 52 , 70 < 75 , 70 > 66 → right of 66



Final binary search tree

Check — inorder: 18, 30, 35, 41, 52, 66, 70, 75, 90 (sorted)

(Correct steps 2 M, final tree 2 M)

d) Membership and identity operators. 4 M

Membership operators test whether a value is present in a sequence (string, list, tuple, set, dict):

- **in** — True if the value is found.
- **not in** — True if the value is not found.

Identity operators test whether two variables refer to the **same object** in memory:

- **is** — True if both refer to the same object.
- **is not** — True if they refer to different objects.

```
L = [10, 20, 30]
print(20 in L, 50 not in L)      # membership
x = [1, 2]
y = x
z = [1, 2]
print(x is y, x is z)            # identity
print(x == z, x is not z)
```

Output:

```
True True
True False
True True
```

x and z have equal values (== is True) but are different objects, so **x is z** is False.

(Membership 2 M, identity 2 M)

Q.3

a) Reverse a number and check palindrome. 4 M

```
# Reverse a number and check palindrome
num = int(input("Enter a number: "))
n = num
rev = 0
while n > 0:
    digit = n % 10
    rev = rev * 10 + digit
    n = n // 10
print("Reverse of", num, "is", rev)
if rev == num:
    print(num, "is a palindrome number")
else:
    print(num, "is not a palindrome number")
```

Output:

```
Enter a number: 12321
Reverse of 12321 is 12321
12321 is a palindrome number
```

Second run:

```
Enter a number: 4503
Reverse of 4503 is 3054
4503 is not a palindrome number
```

(Logic 3 M, output 1 M)

b) Set — create, access, update, delete. 4 M

A **set** is an unordered collection of unique (no duplicate) items, written in { }. Sets are mutable but have no index.

- **Create:** s = {10, 20, 30} or s = set([10, 20, 30])
- **Access:** by a for loop or the **in** operator (indexing is not allowed).
- **Update:** add(x) adds one item; update([..]) adds many items.
- **Delete:** remove(x) (error if absent), discard(x) (no error), pop(), clear(); del s deletes the set.

```
# Set: create, access, update, delete
s = {10, 20, 30}
print("Created      :", s)
for x in s:                          # access by iteration (no index)
    print("Element:", x)
print("20 in s ?    :", 20 in s)
s.add(40)                            # add one element
s.update([50, 60])                   # add many elements
print("After update :", s)
s.remove(10)                         # error if not present
s.discard(99)                        # no error if not present
print("After delete :", s)
```

Output:

```
Created      : {10, 20, 30}
Element: 10
Element: 20
Element: 30
20 in s ?    : True
After update : {40, 10, 50, 20, 60, 30}
After delete : {40, 50, 20, 60, 30}
```

(Each operation 1 M)

c) Doubly linked list.

4 M

A **doubly linked list** is a linear collection of nodes in which each node has three parts: **prev** (address of previous node), **data** and **next** (address of next node). It can be traversed in both forward and backward directions. prev of the first node and next of the last node are NULL (None).

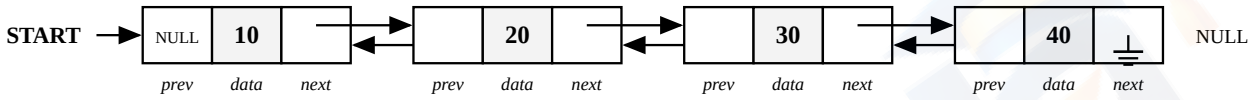


Fig.: Doubly linked list

```
class Node:
    def __init__(self, data):
        self.prev = None
        self.data = data
        self.next = None
```

- Insertion and deletion are easier because the previous node is directly known.
- Needs extra memory for the prev pointer.
- Example use: browser back/forward navigation, undo-redo.

(Description 2 M, diagram 2 M)

d) Linear search program.

4 M

Linear search compares the key with each element from the first to the last until it is found.

```
# Linear search
def linear_search(lst, key):
    for i in range(len(lst)):
        if lst[i] == key:
            return i          # position found
    return -1                 # not found

data = [34, 7, 19, 56, 23, 81, 45]
print("List:", data)
for key in (23, 50):
    pos = linear_search(data, key)
    if pos != -1:
        print(key, "found at index", pos, "(position", pos + 1, ")")
    else:
        print(key, "not found in the list")
```

Output:

```
List: [34, 7, 19, 56, 23, 81, 45]
23 found at index 4 (position 5 )
50 not found in the list
```

(Program 3 M, output 1 M)

Q.4

a) Four NumPy functions. 4 M

NumPy (Numerical Python) is a package for fast operations on multi-dimensional arrays (ndarray). It is imported as `import numpy as np`.

- **np.array()** — creates an array from a list.
- **np.zeros(shape)** — array filled with 0.
- **np.arange(start, stop, step)** — array of evenly spaced values.
- **reshape(r, c)** — changes the shape without changing data.
- **np.sqrt(), np.sum(), np.max(), np.mean()** — mathematical functions.

```
import numpy as np

a = np.array([4, 9, 16, 25, 36])
print("array()     :", a)
z = np.zeros((2, 3), dtype=int)
print("zeros()     :\n", z)
r = np.arange(10, 30, 5)
print("arange()    :", r)
m = np.array([[1, 2, 3], [4, 5, 6]])
print("reshape()   :\n", m.reshape(3, 2))
print("sqrt()      :", np.sqrt(a))
print("sum(), max(), mean():", np.sum(a), np.max(a), np.mean(a))
```

Output:

```
array()     : [ 4  9 16 25 36]
zeros()     :
 [[0 0 0]
 [0 0 0]]
arange()    : [10 15 20 25]
reshape()   :
 [[1 2]
 [3 4]
 [5 6]]
sqrt()      : [2. 3. 4. 5. 6.]
sum(), max(), mean(): 90 36 18.0
```

(Any four functions with example, 1 M each)

b) Traversals of the tree in Fig. 1. 4 M

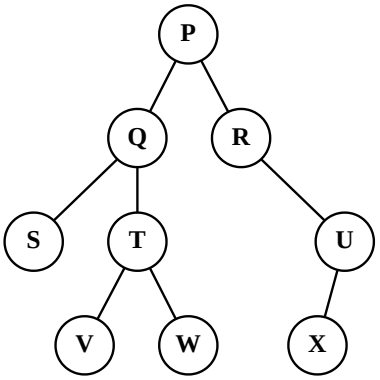


Fig. 1

Inorder (Left, Root, Right): S, Q, V, T, W, P, R, X, U

Preorder (Root, Left, Right): P, Q, S, T, V, W, R, U, X

Postorder (Left, Right, Root): S, V, W, T, Q, X, U, R, P

(Inorder 1 M, preorder 1½ M, postorder 1½ M)

c) Single inheritance program. 4 M

In **single inheritance** a child (derived) class inherits from only one parent (base) class.

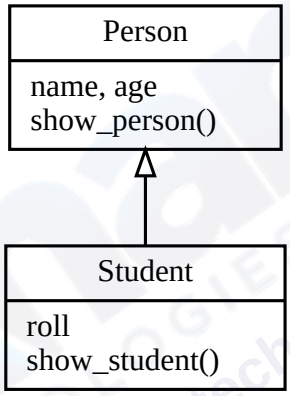


Fig.: Single inheritance

```
# Single inheritance: Student (child) inherits Person (parent)
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def show_person(self):
        print("Name :", self.name)
        print("Age  :", self.age)

class Student(Person):              # Student IS-A Person
    def __init__(self, name, age, roll):
        super().__init__(name, age)   # call parent constructor
        self.roll = roll

    def show_student(self):
        self.show_person()            # inherited method
        print("Roll :", self.roll)
```

```
s = Student("Riya", 18, 21)
s.show_student()
```

Output:

```
Name : Riya
Age  : 18
Roll : 21
```

(Program 3 M, output 1 M)

d) Infix to postfix: A + B * (C − D) / E

4 M

Scan from left to right: operands go to postfix; '(' is pushed; on ')' pop till '('; an operator pops operators of higher or equal precedence before it is pushed.

Symbol	Stack	Postfix expression
A	—	A
+	+	A
B	+	AB
*	+*	AB
(	+*(	AB
C	+*(	ABC
-	+*(-	ABC
D	+*(-	ABCD
)	+*	ABCD-
/	+/	ABCD-*
E	+/	ABCD-*E
(end)	—	ABCD-*E/+

Postfix expression: ABCD-*E/+

(Steps 3 M, answer 1 M)

e) break, continue and pass.

4 M

- **break** — terminates the loop immediately; control goes to the statement after the loop.
- **continue** — skips the remaining statements of the current iteration and goes to the next iteration.
- **pass** — a null statement; does nothing. Used as a placeholder where a statement is required syntactically.

```
# break, continue and pass
for i in range(1, 8):
    if i == 6:
        break          # stop the loop
    if i % 2 == 0:
        continue       # skip even numbers
    print("odd:", i)
```

```
for ch in "AB":
    pass                # placeholder, does nothing
print("Loop finished")
```

Output:

```
odd: 1
odd: 3
odd: 5
Loop finished
```

Even numbers are skipped by continue; the loop stops at 6 by break.

(Explanation 2 M, example 2 M)

Q.5

a) Bubble sort program. 6 M

Working: Adjacent elements are compared and swapped if they are in the wrong order. After each pass the largest remaining element moves (bubbles) to its correct place at the end.

```
# Bubble sort with output after each pass
def bubble_sort(a):
    n = len(a)
    for i in range(n - 1):
        swapped = False
        for j in range(n - 1 - i):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]   # swap
                swapped = True
        print("Pass", i + 1, ":", a)
        if not swapped:          # already sorted
            break
    return a

data = [47, 12, 85, 30, 6, 59]
print("Original :", data)
bubble_sort(data)
print("Sorted   :", data)
```

Output:

```
Original : [47, 12, 85, 30, 6, 59]
Pass 1 : [12, 47, 30, 6, 59, 85]
Pass 2 : [12, 30, 6, 47, 59, 85]
Pass 3 : [12, 6, 30, 47, 59, 85]
Pass 4 : [6, 12, 30, 47, 59, 85]
Pass 5 : [6, 12, 30, 47, 59, 85]
Sorted   : [6, 12, 30, 47, 59, 85]
```

Pass-wise list (computed):

Pass	List after pass
Initial	47, 12, 85, 30, 6, 59
1	12, 47, 30, 6, 59, 85

2	12, 30, 6, 47, 59, 85
3	12, 6, 30, 47, 59, 85
4	6, 12, 30, 47, 59, 85
5	6, 12, 30, 47, 59, 85

In pass 5 no swap occurs, so the list is sorted.

(Program 4 M, output 2 M)

b) User-defined module for arithmetic operations. 6 M

A **module** is a .py file containing functions, classes and variables. It is used in another file with the **import** statement. Both files are kept in the same folder.

File 1: calc.py (module)

```
# calc.py  -- user-defined module
def add(a, b):
    return a + b

def sub(a, b):
    return a - b

def mul(a, b):
    return a * b

def div(a, b):
    if b == 0:
        return "Division by zero not allowed"
    return a / b
```

File 2: main program

```
# main.py -- importing the calc module
import calc

x, y = 24, 6
print("Addition       :", calc.add(x, y))
print("Subtraction    :", calc.sub(x, y))
print("Multiplication :", calc.mul(x, y))
print("Division       :", calc.div(x, y))
print("Division by 0  :", calc.div(x, 0))
```

Output:

```
Addition       : 30
Subtraction    : 18
Multiplication : 144
Division       : 4.0
Division by 0  : Division by zero not allowed
```

(Module 3 M, import and use 2 M, output 1 M)

c) BFS vs DFS. 6 M

Breadth First Search (BFS)	Depth First Search (DFS)

Visits all neighbours of a vertex first, level by level.	Goes as deep as possible along one path, then backtracks.
Uses a queue (FIFO).	Uses a stack (LIFO) or recursion.
Gives shortest path (in edges) in an unweighted graph.	Does not guarantee shortest path.
Needs more memory for wide graphs.	Needs less memory (stores one path).
Applications: shortest path, peer-to-peer networks, web crawlers.	Applications: cycle detection, topological sort, maze / puzzle solving.

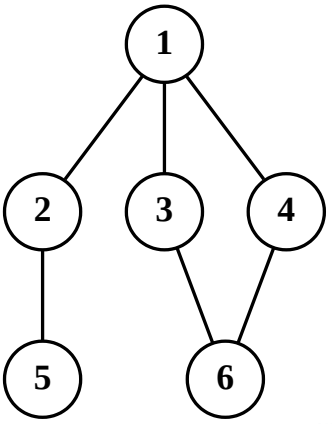


Fig. 2

```
from collections import deque
graph = {1: [2, 3, 4], 2: [1, 5], 3: [1, 6], 4: [1, 6], 5: [2], 6: [3, 4]}

def bfs(s):
    seen, q, out = {s}, deque([s]), []
    while q:
        v = q.popleft(); out.append(v)
        for w in graph[v]:
            if w not in seen:
                seen.add(w); q.append(w)
    return out

def dfs(s):
    seen, st, out = set(), [s], []
    while st:
        v = st.pop()
        if v not in seen:
            seen.add(v); out.append(v)
            st.extend(reversed(graph[v]))
    return out

print("BFS:", bfs(1))
print("DFS:", dfs(1))
```

Output:

```
BFS: [1, 2, 3, 4, 5, 6]
DFS: [1, 2, 5, 3, 6, 4]
```

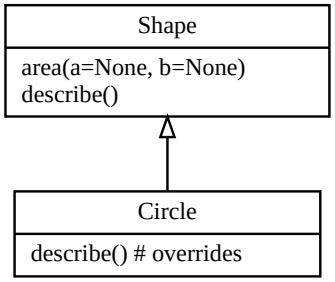
BFS from 1: 1, 2, 3, 4, 5, 6 **DFS from 1:** 1, 2, 5, 3, 6, 4 (neighbours taken in increasing order).

(Any four differences 4 M, BFS and DFS 2 M)

Q.6

a) Method overloading and overriding. 6 M

- **Method overloading:** same method name with different number / type of arguments in the same class. Python does not support it directly (the last definition replaces earlier ones), so it is done using **default arguments** or *args.
- **Method overriding:** a child class gives its own definition of a method already defined in the parent class with the same name and parameters; the child version is called for child objects.




```
# Method overloading (default arguments) and method overriding
class Shape:
    def area(self, a=None, b=None):        # overloaded: 0, 1 or 2 arguments
        if a is not None and b is not None:
            return a * b                    # rectangle
        elif a is not None:
            return a * a                    # square
        return 0

    def describe(self):
        print("I am a shape")

class Circle(Shape):
    def describe(self):                     # overrides parent method
        print("I am a circle")

s = Shape()
print("Area of square (side 5)     :", s.area(5))
print("Area of rectangle (4 x 7)   :", s.area(4, 7))
print("No arguments                :", s.area())
Shape().describe()
Circle().describe()
```

Output:

```
Area of square (side 5)       : 25
Area of rectangle (4 x 7)     : 28
No arguments                  : 0
I am a shape
I am a circle
```

(Overloading 3 M, overriding 3 M)

b) Dictionary operations. 6 M

A **dictionary** is an unordered (insertion-ordered from Python 3.7), mutable collection of **key : value** pairs written in { }. Keys must be unique and immutable.

- **Create:** d = {"Python": 78, "DSA": 82}
- **Access:** d[key] or d.get(key)
- **Update:** d[key] = new_value (adds the pair if key is new); d.update({...})
- **Delete:** del d[key], d.pop(key), d.popitem(), d.clear()

```
# Dictionary: create, access, update, delete
marks = {"Python": 78, "DSA": 82, "Maths": 69}
print("Dictionary      :", marks)
print("Access by key   :", marks["DSA"])
print("Using get()     :", marks.get("Maths"))
marks["Maths"] = 74          # update existing value
marks["DBMS"] = 88           # add new key-value pair
print("After update    :", marks)
del marks["Python"]          # delete one item
print("After del       :", marks)
v = marks.pop("DSA")         # remove and return value
print("pop('DSA') gave :", v, "->", marks)
marks.clear()                # remove all items
print("After clear()   :", marks)
```

Output:

```
Dictionary      : {'Python': 78, 'DSA': 82, 'Maths': 69}
Access by key   : 82
Using get()     : 69
After update    : {'Python': 78, 'DSA': 82, 'Maths': 74, 'DBMS': 88}
After del       : {'DSA': 82, 'Maths': 74, 'DBMS': 88}
pop('DSA') gave : 82 -> {'Maths': 74, 'DBMS': 88}
After clear()   : {}
```

(Explanation 2 M, program 3 M, output 1 M)

c) Largest, smallest and sum of list. 6 M

```
# Accept n numbers in a list; find largest, smallest and sum
n = int(input("How many numbers? "))
nums = []
for i in range(n):
    nums.append(int(input("Enter number: ")))
print("List     :", nums)
print("Largest  :", max(nums))
print("Smallest :", min(nums))
print("Sum      :", sum(nums))
```

Output:

```
How many numbers? 5
Enter number: 34
Enter number: -8
Enter number: 71
Enter number: 15
```

```
Enter number: 0
List     : [34, -8, 71, 15, 0]
Largest  : 71
Smallest : -8
Sum      : 112
```

max(), min() and sum() are built-in functions; the same result can be obtained with a loop by comparing each element.

(Input 2 M, logic 3 M, output 1 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in