

Data Structure Using Python (313306)

Model Question Paper — Set C · Model Answers

Q.1

a) Types of operators.

2 M

1. Arithmetic operators (+, −, *, /, //, %, **)
2. Relational / comparison operators (==, !=, >, <, >=, <=)
3. Logical operators (and, or, not)
4. Assignment operators (=, +=, -=, :=)
5. Bitwise operators (&, |, ^, ~, <<, >>)
6. Membership (in, not in) and identity (is, is not) operators

(Any four, ½ M each)

b) Global vs local variables.

2 M

Global variable	Local variable
Declared outside all functions.	Declared inside a function.
Accessible in the whole program (all functions).	Accessible only inside that function.
Exists till the program ends.	Created when the function is called; destroyed when it returns.
Changed inside a function only with the global keyword.	—

```
count = 10                # global variable
def show():
    count_local = 5       # local variable
    print("Inside :", count, count_local)
show()
print("Outside:", count)
```

Output:

```
Inside : 10 5
Outside: 10
```

(Any two points, 1 M each)

c) Stack and queue.

2 M

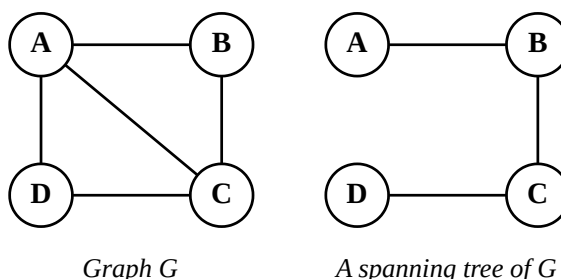
i) Stack: A linear data structure in which insertion (PUSH) and deletion (POP) are done at only one end called **TOP**. It follows **LIFO** (Last In First Out).

ii) Queue: A linear data structure in which insertion is done at the **REAR** end and deletion at the **FRONT** end. It follows **FIFO** (First In First Out).

d) Spanning tree and properties.

2 M

Spanning tree: A spanning tree of a connected graph G is a sub-graph that is a tree and includes **all the vertices** of G with the minimum number of edges (no cycle).



- A spanning tree of a graph with n vertices has exactly **$n - 1$ edges**.
- It has **no cycle** and is connected; removing any edge disconnects it, adding any edge forms a cycle.

(Definition 1 M, two properties 1 M)

e) Algorithm for inorder traversal.

2 M

INORDER(ROOT) — Left, Root, Right

1. If ROOT is None, return.
2. INORDER(ROOT.left).
3. Visit (print) ROOT.data.
4. INORDER(ROOT.right).

f) OOP and its features.

2 M

Object oriented programming is a programming approach in which a program is organised as a collection of **objects** created from **classes**; each object combines data and the functions (methods) that act on that data.

Features: class, object, encapsulation, abstraction, inheritance, polymorphism.

(Definition 1 M, any two features 1 M)

g) List vs tuple.

2 M

List	Tuple
Mutable — items can be changed.	Immutable — items cannot be changed.
Written in square brackets [].	Written in parentheses ().
More methods (append, remove, sort ...).	Only count() and index().
Slower, uses more memory.	Faster, uses less memory.

(Any two, 1 M each)

Q.2

a) if-elif-else and for loop.

4 M

if-elif-else: checks conditions one by one and executes the block of the first true condition; else block runs if none is true.

```
if condition1:
    block1
elif condition2:
    block2
else:
    block3
```

for loop: repeats a block for each item of a sequence (list, string, range).

```
for variable in sequence:
    block
```

```
# if-elif-else and for loop
marks = [82, 67, 45, 30]
for m in marks:
    if m >= 75:
        grade = "Distinction"
    elif m >= 60:
        grade = "First Class"
    elif m >= 40:
        grade = "Pass"
    else:
        grade = "Fail"
    print(m, "->", grade)
```

Output:

```
82 -> Distinction
67 -> First Class
45 -> Pass
30 -> Fail
```

(if-elif-else 2 M, for loop 2 M)

b) Data abstraction vs data encapsulation.

4 M

Data abstraction	Data encapsulation
Shows only essential features and hides implementation details.	Binds data and methods into one unit (class) and protects the data.
Focuses on <i>what</i> an object does.	Focuses on <i>how</i> data is kept safe inside the object.
Solves the problem at the design level.	Solves the problem at the implementation level.
Achieved using classes, abstract classes (abc module) and methods.	Achieved using classes and access control (private __members, getters / setters).
Example: calling len(L) without knowing its code.	Example: __balance accessed only through deposit() / get_balance().

(Any four points, 1 M each)

c) Steps to create a package.

4 M

A **package** is a folder that contains related modules and a file `__init__.py`.

1. Create a folder with the package name, e.g. **mypkg**.

2. Create an `__init__.py` file inside it (may be empty).
3. Write the modules (`geometry.py`, `stats.py`) inside the folder.
4. Import the package modules in a program with `import mypkg.geometry` or from `mypkg import geometry`.

```
project/
├── use_pkg.py
└── mypkg/
    ├── __init__.py
    ├── geometry.py
    └── stats.py
```

mypkg/geometry.py

```
def area_circle(r):
    return 3.14 * r * r
```

mypkg/stats.py

```
def average(nums):
    return sum(nums) / len(nums)
```

use_pkg.py

```
from mypkg import geometry
from mypkg.stats import average
print("Area of circle r=5 :", geometry.area_circle(5))
print("Average          :", average([12, 18, 30]))
```

Output:

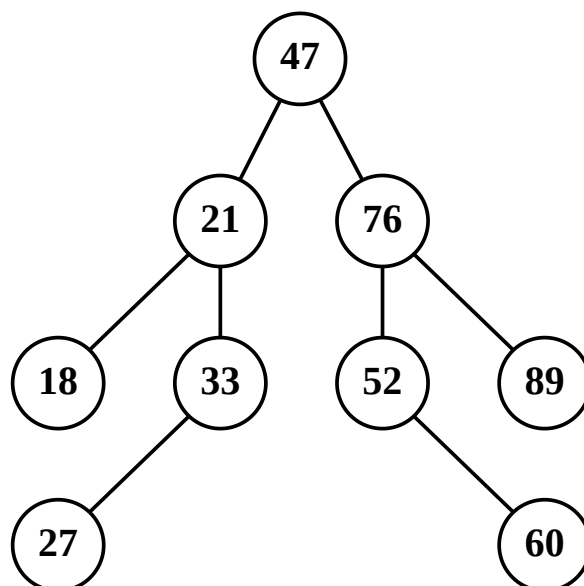
```
Area of circle r=5 : 78.5
Average          : 20.0
```

[\(Steps 2 M, example 2 M\)](#)

d) BST for 47, 21, 76, 18, 33, 52, 89, 27, 60.

4 M

1. **47:** root
2. **21:** $21 < 47 \rightarrow$ left of 47
3. **76:** $76 > 47 \rightarrow$ right of 47
4. **18:** $18 < 47, 18 < 21 \rightarrow$ left of 21
5. **33:** $33 < 47, 33 > 21 \rightarrow$ right of 21
6. **52:** $52 > 47, 52 < 76 \rightarrow$ left of 76
7. **89:** $89 > 47, 89 > 76 \rightarrow$ right of 76
8. **27:** $27 < 47, 27 > 21, 27 < 33 \rightarrow$ left of 33
9. **60:** $60 > 47, 60 < 76, 60 > 52 \rightarrow$ right of 52



Final binary search tree

Inorder: 18, 21, 27, 33, 47, 52, 60, 76, 89 (inorder of a BST is always in ascending order)

(Tree 3 M, inorder 1 M)

Q.3

a) Features of Python.

4 M

1. **Easy to learn and read:** simple English-like syntax; blocks are shown by indentation.
2. **Interpreted:** executed line by line by the interpreter, so debugging is easy; no separate compilation.
3. **Dynamically typed:** the type of a variable is decided at run time; no declaration needed (x = 10, then x = "hi").
4. **Object oriented:** supports classes, objects, inheritance and polymorphism.
5. **Platform independent / portable:** the same program runs on Windows, Linux and macOS.
6. **Large standard library and packages:** math, random, os, and external packages like NumPy, Pandas.

(Any four features with explanation, 1 M each)

b) Selection sort program.

4 M

In each pass the smallest element of the unsorted part is selected and swapped with the first element of that part.

```

# Selection sort with output after each pass
def selection_sort(a):
    n = len(a)
    for i in range(n - 1):
        min_idx = i
        for j in range(i + 1, n):
            if a[j] < a[min_idx]:
                min_idx = j
        a[i], a[min_idx] = a[min_idx], a[i]      # swap
        print("Pass", i + 1, ":", a)
  
```

```

return a

data = [38, 9, 71, 24, 55, 16]
print("Original :", data)
selection_sort(data)
print("Sorted   :", data)

```

Output:

```

Original : [38, 9, 71, 24, 55, 16]
Pass 1 : [9, 38, 71, 24, 55, 16]
Pass 2 : [9, 16, 71, 24, 55, 38]
Pass 3 : [9, 16, 24, 71, 55, 38]
Pass 4 : [9, 16, 24, 38, 55, 71]
Pass 5 : [9, 16, 24, 38, 55, 71]
Sorted   : [9, 16, 24, 38, 55, 71]

```

Check (computed separately): after pass 5 → 9, 16, 24, 38, 55, 71

(Program 3 M, output 1 M)

c) Four tuple operations.

4 M

A **tuple** is an ordered, immutable sequence written in ().

- **Concatenation (+)** joins two tuples.
- **Repetition (*)** repeats a tuple n times.
- **Indexing and slicing** — t[i], t[start:stop].
- **Membership (in / not in)** checks presence of an item.
- **Built-in functions / methods** — len(), max(), min(), count(), index().

```

t1 = (5, 10, 15)
t2 = ("a", "b")
print("Concatenation :", t1 + t2)
print("Repetition    :", t2 * 3)
print("Indexing       :", t1[0], t1[-1])
print("Slicing        :", t1[0:2])
print("Membership     :", 10 in t1)
print("len/max/min    :", len(t1), max(t1), min(t1))
print("count/index    :", t1.count(5), t1.index(15))

```

Output:

```

Concatenation : (5, 10, 15, 'a', 'b')
Repetition    : ('a', 'b', 'a', 'b', 'a', 'b')
Indexing       : 5 15
Slicing        : (5, 10)
Membership     : True
len/max/min    : 3 15 5
count/index    : 1 2

```

(Any four operations with example, 1 M each)

d) Linear queue using list.

4 M

A queue is FIFO. With a list, **append()** inserts at REAR and **pop(0)** deletes from FRONT.

```
# Linear queue using Python list (FIFO)
queue = []
SIZE = 4

def enqueue(x):
    if len(queue) == SIZE:
        print("Queue Overflow")
    else:
        queue.append(x)          # insert at REAR
        print(x, "inserted")

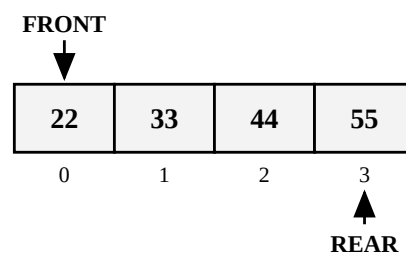
def dequeue():
    if not queue:
        print("Queue Underflow")
    else:
        print(queue.pop(0), "deleted") # delete from FRONT

def display():
    print("Queue (front -> rear):", queue)

for v in (11, 22, 33):
    enqueue(v)
display()
dequeue()
enqueue(44); enqueue(55); enqueue(66)
display()
dequeue(); dequeue(); dequeue(); dequeue(); dequeue()
```

Output:

```
11 inserted
22 inserted
33 inserted
Queue (front -> rear): [11, 22, 33]
11 deleted
44 inserted
55 inserted
Queue Overflow
Queue (front -> rear): [22, 33, 44, 55]
22 deleted
33 deleted
44 deleted
55 deleted
Queue Underflow
```



Queue after inserting 44 and 55 (full, size 4)

(Insert 1½ M, delete 1½ M, output 1 M)

Q.4

a) Sum of digits.

4 M

```
# Sum of digits of a number
num = int(input("Enter a number: "))
n = num
total = 0
while n > 0:
    total += n % 10      # last digit
    n //= 10            # remove last digit
print("Sum of digits of", num, "=", total)
```

Output:

```
Enter a number: 70394
Sum of digits of 70394 = 23
```

Digits $7 + 0 + 3 + 9 + 4 = 23$. Each iteration takes the last digit with $\% 10$ and removes it with $// 10$.

(Logic 3 M, output 1 M)

b) Creating arrays with NumPy.

4 M

NumPy (import numpy as np) provides an n-dimensional array object **ndarray**. Standard methods to create and initialise arrays:

Method	Creates
np.array(list)	array from a Python list / nested list
np.zeros(shape)	array filled with 0 (float by default)
np.ones(shape, dtype)	array filled with 1
np.arange(start, stop, step)	evenly spaced values (stop excluded)
np.linspace(start, stop, n)	n equally spaced values (stop included)
np.eye(n)	$n \times n$ identity matrix
np.full(shape, value)	array filled with the given value

```
import numpy as np
a = np.array([3, 6, 9])          # from list
b = np.array([[1, 2], [3, 4]])   # 2-D from nested list
z = np.zeros(4)                  # all zeros (float)
o = np.ones((2, 3), dtype=int)   # all ones
r = np.arange(2, 12, 3)          # start, stop, step
l = np.linspace(0, 1, 5)         # 5 equal points
e = np.eye(3, dtype=int)         # identity matrix
f = np.full((2, 2), 7)           # filled with 7
for name, arr in (("array", a), ("2-D array", b), ("zeros", z), ("ones", o),
                  ("arange", r), ("linspace", l), ("eye", e), ("full", f)):
    print(name, ":\n", arr)
```

Output:

```
array :
[3 6 9]
```

```
2-D array :
[[1 2]
 [3 4]]
zeros :
[0. 0. 0. 0.]
ones :
[[1 1 1]
 [1 1 1]]
arange :
[ 2  5  8 11]
linspace :
[0.   0.25 0.5  0.75 1.   ]
eye :
[[1 0 0]
 [0 1 0]
 [0 0 1]]
full :
[[7 7]
 [7 7]]
```

(Any four methods with example, 1 M each)

c) Rectangle class.

4 M

Rectangle
length, breadth read() area() perimeter() show()

```
class Rectangle:
    def __init__(self):
        self.length = 0
        self.breadth = 0

    def read(self):
        self.length = float(input("Enter length : "))
        self.breadth = float(input("Enter breadth: "))

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def show(self):
        print("Area      =", self.area())
        print("Perimeter =", self.perimeter())

r = Rectangle()
r.read()
r.show()
```

Output:

```
Enter length : 12.5
Enter breadth: 8
Area          = 100.0
Perimeter     = 41.0
```

(Class with data members 1 M, methods 2 M, output 1 M)

d) Applications of trees.

4 M

1. **Hierarchical data:** file system folders, organisation chart, HTML DOM.
2. **Searching:** binary search tree gives fast search, insert and delete.
3. **Expression trees:** used by compilers to evaluate / convert expressions.
4. **Heaps:** priority queue and heap sort.
5. **Database indexing:** B-trees and B+ trees.
6. **Decision trees** in machine learning; **Huffman tree** for data compression; spanning trees in network routing.

(Any four, 1 M each)

e) Dictionary of programming languages.

4 M

```
# Dictionary of programming languages
langs = {1: "Python", 2: "Java", 3: "C++", 4: "JavaScript"}
langs[5] = "Kotlin"           # add
langs[3] = "C"                 # update
print("All languages:")
for k, v in langs.items():
    print(k, ":", v)
print("Keys   :", list(langs.keys()))
print("Values :", list(langs.values()))
print("Removed:", langs.pop(4))
print("Final  :", langs)
```

Output:

```
All languages:
1 : Python
2 : Java
3 : C
4 : JavaScript
5 : Kotlin
Keys   : [1, 2, 3, 4, 5]
Values : ['Python', 'Java', 'C', 'JavaScript', 'Kotlin']
Removed: JavaScript
Final  : {1: 'Python', 2: 'Java', 3: 'C', 5: 'Kotlin'}
```

(Create 1 M, add / update / delete 2 M, output 1 M)

Q.5

a) Single and multilevel inheritance.

6 M

- **Single inheritance:** one child class derives from one parent class (Car ← Vehicle).

- **Multilevel inheritance:** a class derives from a class which is itself derived from another class (ElectricCar ← Car ← Vehicle). The last class gets the features of all classes above it.

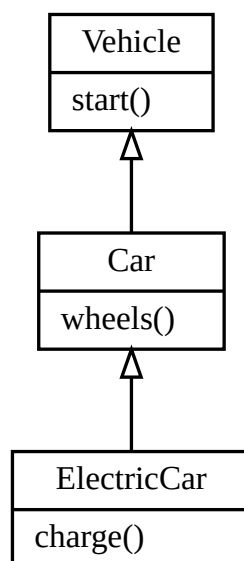


Fig.: Single (Car–Vehicle) and multilevel inheritance

```

# Single and multilevel inheritance
class Vehicle:                                # base class
    def start(self):
        print("Vehicle started")

class Car(Vehicle):                            # single: Car <- Vehicle
    def wheels(self):
        print("Car has 4 wheels")

class ElectricCar(Car):                        # multilevel: ElectricCar <- Car <- Vehicle
    def charge(self):
        print("Electric car is charging")

e = ElectricCar()
e.start()      # from Vehicle
e.wheels()     # from Car
e.charge()     # own method
print(issubclass(ElectricCar, Vehicle))
  
```

Output:

```

Vehicle started
Car has 4 wheels
Electric car is charging
True
  
```

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

b) Module for multiplication and division.

6 M

Save the functions in **muldiv.py**; keep the main program in the same folder.

File 1: muldiv.py (module)

```

# muldiv.py -- module
def multiply(a, b):
  
```

```

    return a * b

def divide(a, b):
    return a / b if b != 0 else "Error: divide by zero"

```

File 2: main program

```

import muldiv
from muldiv import divide as dv
print("15 x 4 =", muldiv.multiply(15, 4))
print("15 / 4 =", dv(15, 4))
print("15 / 0 =", dv(15, 0))

```

Output:

```

15 x 4 = 60
15 / 4 = 3.75
15 / 0 = Error: divide by zero

```

- `import muldiv` — functions are called as `muldiv.multiply()`.
- `from muldiv import divide as dv` — imports only `divide()` and gives it a short name `dv`.

(Module 2 M, import forms 2 M, output 2 M)

c) Infix to postfix: $(P + Q) * R - S / T \wedge U$

6 M

Algorithm (scan left to right):

1. If the symbol is an operand, add it to the postfix expression.
2. If it is '(', push it on the stack.
3. If it is ')', pop and add operators to postfix until '(' is found; discard '('.
4. If it is an operator, pop operators of higher or equal precedence (except $\wedge$ which is right-associative) from the stack and add to postfix, then push the operator.
5. At the end, pop all remaining operators.

Precedence: $\wedge$ (highest) $> * / > + -$

Symbol	Stack	Postfix expression
(	(	
P	(	P
+	(+	P
Q	(+	PQ
)	—	PQ+
*	*	PQ+
R	*	PQ+R
-	-	PQ+R*
S	-	PQ+R*S
/	-/	PQ+R*S

T	-/	PQ+R*ST
^	-/^	PQ+R*ST
U	-/^	PQ+R*STU
(end)	—	PQ+R*STU^/-

Postfix expression: PQ+R*STU^/-

(Algorithm 2 M, steps 3 M, answer 1 M)

Q.6

a) Singly and doubly linked list.

6 M

i) Singly linked list: Each node has two fields — **data** and **next** (address of the next node). START / head points to the first node; next of the last node is NULL. Traversal is possible only in the forward direction.

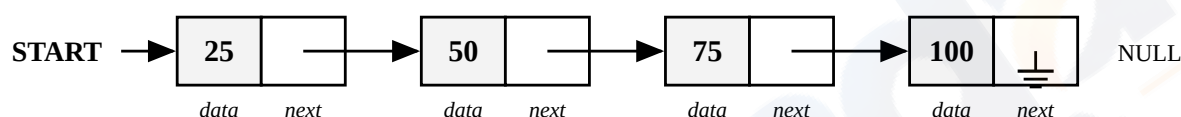


Fig.: Singly linked list

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

ii) Doubly linked list: Each node has three fields — **prev**, **data** and **next**. It can be traversed forward and backward. prev of the first node and next of the last node are NULL.

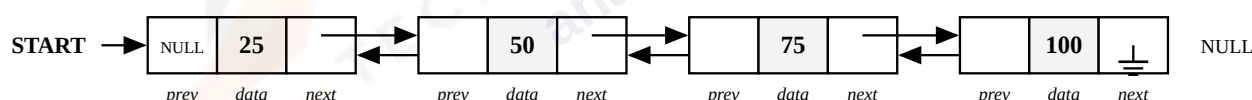


Fig.: Doubly linked list

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

Singly linked list	Doubly linked list
One link (next) per node.	Two links (prev, next) per node.
Forward traversal only.	Forward and backward traversal.

Less memory.	More memory.
--------------	--------------

(Each with diagram and example 3 M)

b) Set operations program.

6 M

Mathematical set operations: **union** (| or union()), **intersection** (& or intersection()), **difference** (– or difference()), **symmetric difference** (^ or symmetric_difference()).

```
A = {1, 3, 5, 7, 9}
B = {3, 6, 9}
print("Union      :", A | B)
print("Intersection :", A & B)
print("Difference   :", A - B)
print("Sym. diff.   :", A ^ B)
print("5 in A ?     :", 5 in A)
A.add(11); A.update({13, 15})
print("After add/update :", A)
A.remove(1); A.discard(100); x = B.pop()
print("After remove/discard :", A)
print("pop() removed", x, "-> B =", B)
```

Output:

```
Union      : {1, 3, 5, 6, 7, 9}
Intersection : {9, 3}
Difference   : {1, 5, 7}
Sym. diff.   : {1, 5, 6, 7}
5 in A ?     : True
After add/update : {1, 3, 5, 7, 9, 11, 13, 15}
After remove/discard : {3, 5, 7, 9, 11, 13, 15}
pop() removed 9 -> B = {3, 6}
```

Note: a set is unordered, so the printed order of elements may differ on another system.

(Four operations 3 M, add / update / delete 2 M, output 1 M)

c) Binary tree traversals.

6 M

- **Inorder (L, Root, R):** traverse left sub-tree in inorder, visit root, traverse right sub-tree in inorder.
- **Preorder (Root, L, R):** visit root, traverse left sub-tree in preorder, traverse right sub-tree in preorder.
- **Postorder (L, R, Root):** traverse left sub-tree in postorder, traverse right sub-tree in postorder, visit root.

```
def inorder(r):
    if r:
        inorder(r.left); print(r.data, end=' '); inorder(r.right)
def preorder(r):
    if r:
        print(r.data, end=' '); preorder(r.left); preorder(r.right)
def postorder(r):
    if r:
        postorder(r.left); postorder(r.right); print(r.data, end=' ')
```

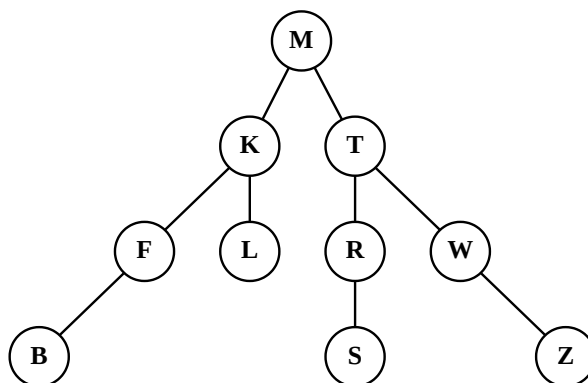


Fig. 1

Inorder: B, F, K, L, M, R, S, T, W, Z

Preorder: M, K, F, B, L, T, R, S, W, Z

Postorder: B, F, L, K, S, R, Z, W, T, M

(Algorithms 3 M, traversals 3 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