OOPS & Python Programming (4351108) - Summer 2024 Solution

Milav Dabgar

May 18, 2024

Question 1(a) [3 marks]

Explain for loop working in Python.

Solution

For loop repeats code block for each item in sequence like list, tuple, or string.
Syntax Table:

Table 1. For Loop Syntax

Component	Syntax	Example
Basic	`for variable in sequence:`	`for i in [1,2,3]:`
Range	`for i in range(n):`	`for i in range(5):`
String	`for char in string:`	`for c in "hello":`

Diagram:

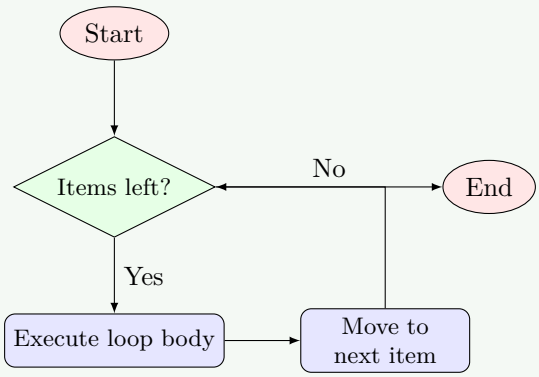

Figure 1. For Loop Execution Flow

- **Iteration**: Loop variable gets each value from sequence one by one
- **Automatic**: Python handles moving to next item automatically
- **Flexible**: Works with lists, strings, tuples, ranges

Mnemonic

"For Each Item, Execute Block"

Question 1(b) [4 marks]

Explain working of if-elif-else in Python.

Solution

Multi-way decision structure that checks multiple conditions in sequence.
Structure Table:

Table 2. If-Elif-Else Structure

Statement	Purpose	Syntax
if	First condition	`if condition1:`
elif	Alternative conditions	`elif condition2:`
else	Default case	`else:`

Flow Diagram:

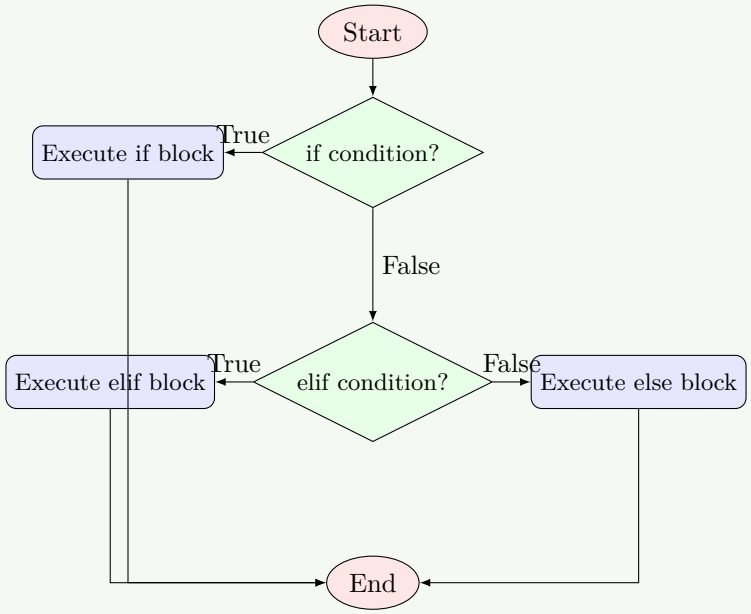

Figure 2. If-Elif-Else Logic Flow

- **Sequential**: Checks conditions top to bottom
- **Exclusive**: Only one block executes
- **Optional**: elif and else are optional

Mnemonic

"If This, Else If That, Else Default"

Question 1(c) [7 marks]

Explain structure of a Python Program.

Solution

Python program has organized structure with specific components in logical order.
Program Structure Table:

Table 3. Python Program Structure

Component	Purpose	Example
Comments	Documentation	`# This is comment`
Import	External modules	`import math`
Constants	Fixed values	`PI = 3.14`
Functions	Reusable code	`def function_name():`
Classes	Objects blueprint	`class ClassName:`
Main code	Program execution	`if __name__ == "__main__":`

Program Architecture:

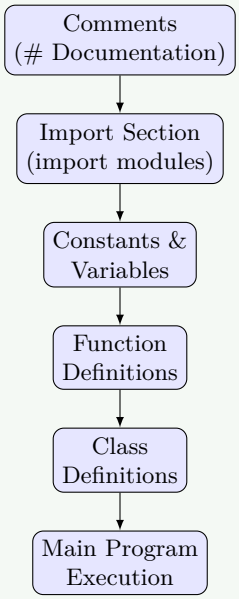

Figure 3. Python Program Structure

- **Modular**: Each section has specific purpose
- **Readable**: Clear organization helps understanding
- **Maintainable**: Easy to modify and debug
- **Standard**: Follows Python conventions

Simple Example:

```python
# Program to calculate area
import math

PI = 3.14159

def calculate_area(radius):
    return PI * radius * radius

# Main execution
radius = float(input("Enter radius: "))
area = calculate_area(radius)
print(f"Area = {area}")
```

Mnemonic

"Comment, Import, Constant, Function, Class, Main"

Question 1(c OR) [7 marks]

Explain features of Python Programming Language.

Solution

Python has unique characteristics that make it popular for beginners and professionals.
Python Features Table:

Table 4. Python Features

Feature	Description	Benefit
Simple	Easy syntax	Quick learning
Interpreted	No compilation	Fast development
Object-Oriented	Classes and objects	Code reusability
Open Source	Free to use	No licensing cost
Cross-Platform	Runs everywhere	High portability

Feature Categories:

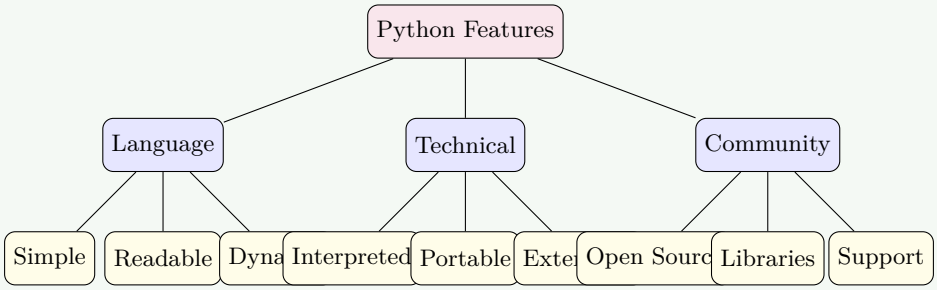

Figure 4. Python Features Hierarchy

- **Beginner-Friendly**: Simple syntax like English language
- **Versatile**: Used for web, AI, data science, automation
- **Rich Libraries**: Huge collection of pre-built modules
- **Dynamic Typing**: No need to declare variable types

Code Example:

```
# Simple Python syntax
name = "Python"
print(f"Hello, {name}!")
```

Mnemonic

"Simple, Interpreted, Object-Oriented, Open, Cross-platform"

Question 2(a) [3 marks]

Explain any 3 operations done on Strings.

Solution

String operations manipulate and process text data in various ways.
String Operations Table:

Table 5. String Operations

Operation	Method	Example	Result
Concatenation	+	"Hello" + "World"	"HelloWorld"
Length	len()	len("Python")	6
Uppercase	.upper()	"hello".upper()	"HELLO"

Operation Examples:

```
text = "Python"
# 1. Concatenation
result1 = text + " Programming"
# 2. Find length
result2 = len(text)
# 3. Convert to uppercase
result3 = text.upper()
```

Mnemonic

"Combine, Count, Convert"

Question 2(b) [4 marks]

Develop a Python program to convert temperature from Fahrenheit to Celsius unit using eq: C=(F-32)/1.8

Solution

Program converts temperature using mathematical formula with user input.
Algorithm Table:

Table 6. Conversion Algorithm

Step	Action	Code
1	Get input	fahrenheit = float(input())
2	Apply formula	celsius = (fahrenheit - 32) / 1.8
3	Display result	print(f"Celsius: {celsius}")

Complete Program:

```
# Temperature conversion program
fahrenheit = float(input("Enter temperature in Fahrenheit: "))
celsius = (fahrenheit - 32) / 1.8
print(f"Temperature in Celsius: {celsius:.2f}")
```

Test Cases:
- Input: 32°F → Output: 0.00°C
- Input: 100°F → Output: 37.78°C

Mnemonic

"Input, Calculate, Output"

Question 2(c) [7 marks]

Explain in detail working of list data types in Python.

Solution

List is ordered, mutable collection that stores multiple items in single variable.
List Characteristics Table:

Table 7. List Characteristics

Property	Description	Example
Ordered	Items have position	`[1, 2, 3]`
Mutable	Can be changed	`list[0] = 10`
Indexed	Access by position	`list[0]`
Mixed Types	Different data types	`[1, "hello", 3.14]`

List Operations Diagram:

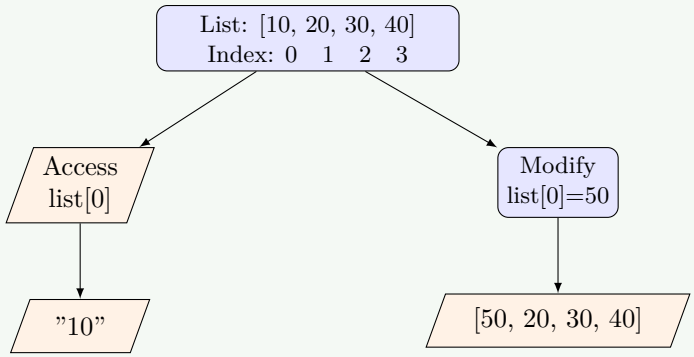

Figure 5. List Operations

Common List Methods:
- `append()`: Add item at end
- `insert()`: Add at position
- `remove()`: Delete item
- `pop()`: Remove last item

Example Code:

```
# Creating and using lists
numbers = [1, 2, 3, 4, 5]
numbers.append(6)        # Add 6 at end
numbers.insert(0, 0)     # Add 0 at beginning
print(numbers[2])        # Access 3rd element
numbers.remove(3)        # Remove value 3
```

Mnemonic

"Ordered, Mutable, Indexed, Mixed"

Question 2(a OR) [3 marks]

Explain String formatting in Python.

Solution

String formatting creates formatted strings by inserting values into templates.
Formatting Methods Table:

Table 8. Formatting Methods

Method	Syntax	Example
f-strings	`f"text {variable}"`	`f"Hello {name}"`
format()	`"text {}".format(value)`	`"Age: {}".format(25)`
% operator	`"text %s" % value`	`"Name: %s" % "John"`

Example Usage:

```
name = "Alice"
age = 25
# f-string formatting
message = f"Hello {name}, you are {age} years old"
```

Mnemonic

"Format, Insert, Display"

Question 2(b OR) [4 marks]

Develop a Python program to identify whether the scanned number is even or odd and print an appropriate message.

Solution

Program checks if number is divisible by 2 to determine even or odd.
Logic Table:

Table 9. Even/Odd Logic

Condition	Result	Message
number % 2 == 0	Even	"Number is even"
number % 2 != 0	Odd	"Number is odd"

Complete Program:

```
# Even/Odd checker program
number = int(input("Enter a number: "))
if number % 2 == 0:
    print(f"{number} is even")
else:
    print(f"{number} is odd")
```

Test Cases:
- Input: 4 → Output: "4 is even"
- Input: 7 → Output: "7 is odd"

Mnemonic

"Input, Check Remainder, Display Result"

Question 2(c OR) [7 marks]

Explain in detail working of Set data types in Python.

Solution

Set is unordered collection of unique items with no duplicate values allowed.
Set Characteristics Table:

Table 10. Set Characteristics

Property	Description	Example
Unordered	No fixed position	`{1, 3, 2}`
Unique	No duplicates	`{1, 2, 3}`
Mutable	Can be modified	`set.add(4)`
Iterable	Can loop through	`for item in set:`

Set Operations Diagram:

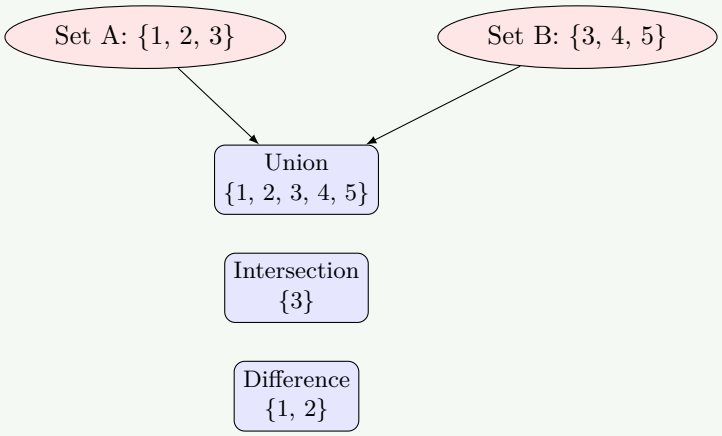

Figure 6. Set Operations

Set Methods Table:

Table 11. Set Methods

Method	Purpose	Example
add()	Add single item	`set.add(6)`
update()	Add multiple items	`set.update([7, 8])`
remove()	Delete item	`set.remove(3)`
union()	Combine sets	`set1.union(set2)`
intersection()	Common items	`set1.intersection(set2)`

Example Code:

```
# Creating and using sets
fruits = {"apple", "banana", "orange"}
fruits.add("mango")              # Add single item
fruits.update(["grape", "kiwi"]) # Add multiple
fruits.remove("banana")          # Remove item
print(len(fruits))               # Count items
```

Mnemonic

"Unique, Unordered, Mutable, Mathematical"

Question 3(a) [3 marks]

Explain working of any 3 methods of math module.

Solution

Math module provides mathematical functions for complex calculations.
Math Methods Table:

Table 12. Math Methods

Method	Purpose	Example	Result
math.sqrt()	Square root	`math.sqrt(16)`	`4.0`
math.pow()	Power calculation	`math.pow(2, 3)`	`8.0`
math.ceil()	Round up	`math.ceil(4.3)`	`5`

Usage Example:

```
import math
number = 16
result1 = math.sqrt(number)  # Square root
result2 = math.pow(2, 4)     # 2 to power 4
result3 = math.ceil(7.2)     # Round up to 8
```

Mnemonic

"Square root, Power, Ceiling"

Question 3(b) [4 marks]

Develop a Python program to find sum of all elements in a list using for loop.

Solution

Program iterates through list and accumulates sum of all elements.
Algorithm Table:

Table 13. Summation Algorithm

Step	Action	Code
1	Initialize sum	`total = 0`
2	Loop through list	`for element in list:`
3	Add to sum	`total += element`
4	Display result	`print(total)`

Complete Program:

```
# Sum of list elements
numbers = [10, 20, 30, 40, 50]
total = 0
for element in numbers:
    total += element
print(f"Sum of all elements: {total}")
```

Test Case:
- Input: [1, 2, 3, 4, 5] → Output: 15

Mnemonic

"Initialize, Loop, Add, Display"

Question 3(c) [7 marks]

Develop a Python program to check if two lists are having similar length. If yes then merge them and create a dictionary from them.

Solution

Program compares list lengths and creates dictionary if they match.
Logic Flow Table:

Table 14. Merge Logic

Step	Condition	Action
1	Check lengths	`len(list1) == len(list2)`
2	If equal	Merge and create dictionary
3	If not equal	Display error message

Process Diagram:

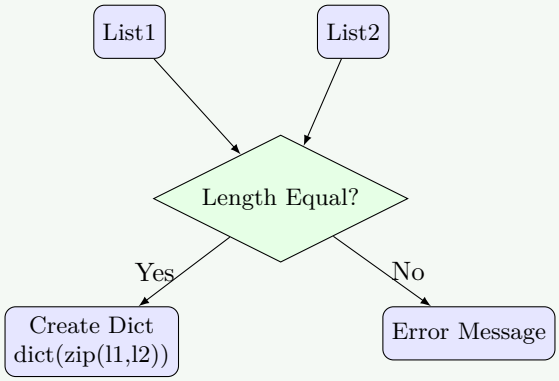

Figure 7. List Merge Logic

Complete Program:

```
# Merge lists into dictionary
list1 = ['name', 'age', 'city']
list2 = ['John', 25, 'Mumbai']

if len(list1) == len(list2):
    # Create dictionary using zip
    result_dict = dict(zip(list1, list2))
    print("Dictionary created:", result_dict)
else:
    print("Lists have different lengths, cannot merge")
```

Expected Output:

```
Dictionary created: {'name': 'John', 'age': 25, 'city': 'Mumbai'}
```

Mnemonic

"Check Length, Zip, Create Dictionary"

Question 3(a OR) [3 marks]

Explain working of any 3 methods of statistics module.

Solution

Statistics module provides functions for statistical calculations on numeric data.
Statistics Methods Table:

Table 15. Statistics Methods

Method	Purpose	Example	Result
statistics.mean()	Average value	`mean([1,2,3,4,5])`	`3.0`
statistics.median()	Middle value	`median([1,2,3,4,5])`	`3`
statistics.mode()	Most frequent	`mode([1,1,2,3])`	`1`

Usage Example:

```
import statistics
data = [10, 20, 30, 40, 50]
avg = statistics.mean(data)      # Calculate average
mid = statistics.median(data)    # Find middle value
```

Mnemonic

"Mean, Median, Mode"

Question 3(c OR) [7 marks]

Develop a Python program to count the number of times a character appears in a given string using a dictionary.

Mnemonic

"Loop, Check, Count, Store"

Question 4(a) [3 marks]

Explain working of Python class and objects with example.

Mnemonic

"Class Blueprint, Object Instance"

Question 4(b) [4 marks]

Develop a Python program to print all odd numbers in a list.

Solution

Program filters list elements and displays only odd numbers.
Odd Number Check Table:

Table 18. Odd Number Logic

Number	Mod 2 (mod)	Result
1	1	Odd
2	0	Even

Complete Program:

```
# Print odd numbers from list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print("Odd numbers in the list:")
for number in numbers:
    if number % 2 != 0:
        print(number, end=" ")
```

Expected Output:

```
Odd numbers in the list:
1 3 5 7 9
```

Mnemonic

"Loop, Check Remainder, Print Odd"

Question 4(c) [7 marks]

Explain working of user defined functions in Python.

Solution

User-defined functions are custom functions created by programmers to perform specific tasks.
Function Components Table:

Table 19. Function Components

Component	Purpose	Syntax
def keyword	Function declaration	`def function_name():`
Parameters	Input values	`def func(param1, param2):`
Body	Function code	Indented statements
return	Output value	`return value`

Function Structure:

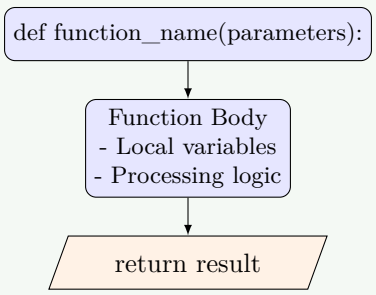

Figure 10. Function Anatomy

Types of Functions:
- No parameters: `def greet():`

- With parameters: `def add(a, b):`
- Return value: `return a + b`
- No return: `print("Hello")`

Example Functions:

```
# Function with parameters and return value
def calculate_area(length, width):
    area = length * width
    return area

# Using functions
result = calculate_area(5, 3)
print(f"Area: {result}")
```

Mnemonic

"Define, Parameters, Body, Return"

Question 4(a OR) [3 marks]

Explain working constructors in Python.

Solution

Constructor is special method that initializes objects when they are created.

Constructor Details Table:

Table 20. Constructor Details

Aspect	Description	Syntax
Method name	Always `__init__`	`def __init__(self):`
Purpose	Initialize object	Set initial values
Automatic call	Called during object creation	`obj = Class()`

Constructor Example:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print("Student object created")

# Object creation automatically calls constructor
student1 = Student("Alice", 20)
```

- **Automatic Execution**: Runs immediately when object is created
- **Initialization**: Sets up object's initial state
- **self Parameter**: Refers to current object being created

Mnemonic

"Initialize, Automatic, Self"

Question 4(b OR) [4 marks]

Develop a Python program to find smallest number in a list without using min function.

Solution

Program manually compares all elements to find the smallest value.
Finding Minimum Algorithm:

Table 21. Min Finding Algorithm

Step	Action	Code
1	Assume first is smallest	`smallest = list[0]`
2	Compare with others	`for num in list[1:]:`
3	Update if smaller found	`if num < smallest:`
4	Display result	`print(smallest)`

Complete Program:

```
# Find smallest number without min()
numbers = [45, 23, 67, 12, 89, 5, 34]

smallest = numbers[0]  # Assume first is smallest

for i in range(1, len(numbers)):
    if numbers[i] < smallest:
        smallest = numbers[i]

print(f"Smallest number: {smallest}")
```

Expected Output:

```
Smallest number: 5
```

Mnemonic

"Assume, Compare, Update, Display"

Question 4(c OR) [7 marks]

Explain working of user defined Modules in Python.

Solution

User-defined modules are custom Python files containing functions, classes, and variables that can be imported and used in other programs.
Module Components: Functions, Classes, Variables, Constants.
Module Creation Process:

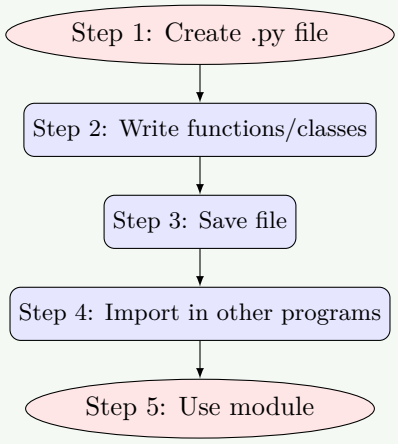

Figure 11. Module Lifecycle

Example Module (math_operations.py):

```
PI = 3.14159

def calculate_circle_area(radius):
    return PI * radius * radius
```

Main Program:

```
import math_operations

# Using module functions
radius = 5
area = math_operations.calculate_circle_area(radius)
print(f"Circle area: {area}")
```

Module Benefits:

- **Code Reusability**: Write once, use in multiple programs
- **Organization**: Keep related functions together
- **Namespace**: Avoid naming conflicts

Mnemonic

"Create File, Define Functions, Import, Use"

Question 5(a) [3 marks]

Explain single inheritance in Python with example.

Solution

Single inheritance is when one class inherits properties and methods from exactly one parent class.
Inheritance Structure: Parent Class (Base) → Child Class (Derived).
Inheritance Diagram:

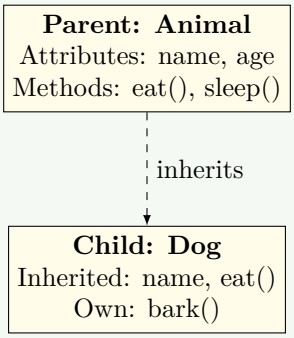

Figure 12. Single Inheritance

Example Code:

```
class Animal:
    def eat(self):
        print("Eating")

class Dog(Animal):
    def bark(self):
        print("Barking")

my_dog = Dog()
my_dog.eat()    # Inherited
my_dog.bark()   # Own
```

Mnemonic

"One Parent, One Child"

Question 5(b) [4 marks]

Explain concept of abstraction in Python with its advantages.

Solution

Abstraction hides complex implementation details and shows only essential features to the user.

Abstraction Concepts:

- **Abstract Class**: Cannot be instantiated (`class Shape(ABC):`)
- **Abstract Method**: Must be implemented by child (`@abstractmethod`)

Implementation:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(Shape):
    def area(self):
        return self.length * self.width
```

Advantages:

- **Simplicity**: Hides complex details
- **Security**: Hides internal implementation

- **Maintainability**: Change implementation safely

Mnemonic

"Hide Details, Show Interface"

Question 5(c) [7 marks]

Develop a Python program to demonstrate working of multiple and multi-level inheritances.

Solution

Program shows both inheritance types: multiple (multiple parents) and multi-level (chain of inheritance).
Inheritance Hierarchy:

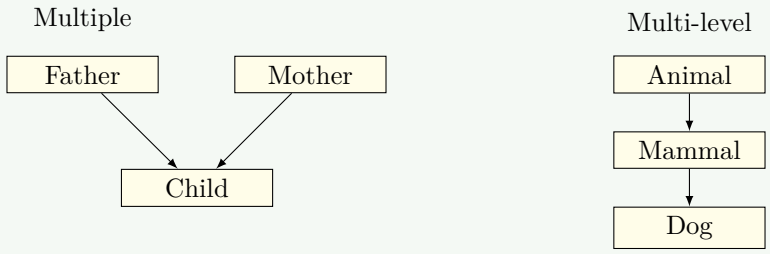

Figure 13. Inheritance Types

Complete Program:

```
print("=== Multi-level Inheritance ===")
class Animal:
    def eat(self): print("Eating")
class Mammal(Animal):
    def breathe(self): print("Breathing")
class Dog(Mammal):
    def bark(self): print("Barking")

d = Dog()
d.eat(); d.breathe(); d.bark()

print("\n=== Multiple Inheritance ===")
class Father:
    def f_method(self): print("Father")
class Mother:
    def m_method(self): print("Mother")
class Child(Father, Mother):
    pass

c = Child()
c.f_method(); c.m_method()
```

Mnemonic

"Multiple Parents, Multi-level Chain"

Question 5(a OR) [3 marks]

Explain working of 3 types of methods in Python.

Solution

Python classes have three types of methods based on how they access class data.
Method Types Table:

Table 22. Method Types

Method Type	First Parameter	Purpose
Instance Method	`self`	Access instance data
Class Method	`cls`	Access class data
Static Method	None	Utility functions

Example Code:

```
class Student:
    school = "ABC"
    def display(self): pass           # Instance
    @classmethod
    def get_school(cls): pass         # Class
    @staticmethod
    def is_adult(age): pass           # Static
```

Mnemonic

"Instance Self, Class Cls, Static None"

Question 5(b OR) [4 marks]

Explain polymorphism through inheritance in Python.

Solution

Polymorphism allows objects of different classes to be treated as objects of common base class.
Key Concept: Same method name, different implementation.
Example:

```
class Shape:
    def area(self): pass

class Rectangle(Shape):
    def area(self): return self.l * self.w

class Circle(Shape):
    def area(self): return 3.14 * self.r * self.r

shapes = [Rectangle(5,3), Circle(4)]
for s in shapes:
    print(s.area())  # Polymorphic call
```

- **Flexibility**: Same code works with different object types
- **Extensibility**: Easy to add new classes

Mnemonic

"Same Name, Different Behavior"

Question 5(c OR) [7 marks]

Develop a Python program to demonstrate working of hybrid inheritance.

Solution

Hybrid inheritance combines multiple and multi-level inheritance.
Structure:

- Animal → Mammal (Single)
- Mammal → Dog, Cat (Hierarchical)
- Dog, Cat → Pet (Multiple)

Diagram:

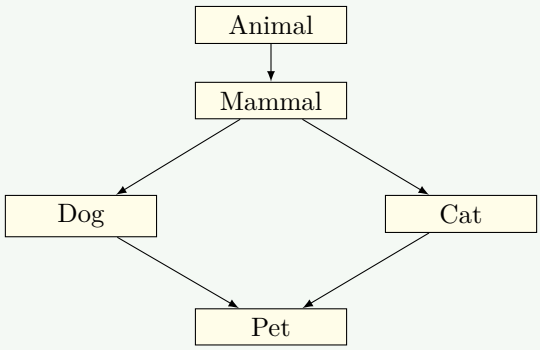

Figure 14. Hybrid Inheritance

Complete Program:

```
# Hybrid Inheritance Demo
class Animal:
    def __init__(self, name): self.name = name

class Mammal(Animal):
    def breathe(self): print("Breathing")

class Dog(Mammal):
    def bark(self): print("Barking")

class Cat(Mammal):
    def meow(self): print("Meowing")

class Pet(Dog, Cat):
    def play(self): print("Playing")

# Usage
p = Pet("Buddy")
p.breathe()  # From Mammal
p.bark()     # From Dog
p.meow()     # From Cat
p.play()     # Own
```