Subject Name Solutions

4321602 – Summer 2024

Semester 1 Study Material

Detailed Solutions and Explanations

Question 1(a) [3 marks]

Give the difference between Tuple and List in python.

Solution

Feature	Tuple	List
Mutability	Immutable (cannot be changed)	Mutable (can be changed)
Syntax	Created using ()	Created using []
Performance	Faster	Slower
Methods	Limited methods (count, index)	Many methods (append, remove, etc.)

- **Memory efficient**: Tuples use less memory than lists
- **Use case**: Tuples for fixed data, lists for dynamic data

Mnemonic

"Tuples are Tight, Lists are Loose"

Question 1(b) [4 marks]

Define Set and how is it created in python?

Solution

Set is an unordered collection of unique elements in Python.
Creating Sets:

```
\# Empty set
my\_set = set()

\# Set with elements
fruits = \{"apple", "banana", "orange"\}

\# From list
numbers = set([1, 2, 3, 4])
```

- **Unique elements**: No duplicates allowed
- **Unordered**: Elements have no specific order
- **Operations**: Union, intersection, difference supported

Mnemonic

"Sets are Special - Unique and Unordered"

Question 1(c) [7 marks]

What is Dictionary in Python? Write a program to concatenate two dictionary into new one.

Solution

Dictionary is an ordered collection of key-value pairs in Python.
Program:

```
\# Two dictionaries
dict1 = \{1: 10, 2: 20\}
dict2 = \{3: 30, 4: 40\}

\# Method 1: Using update()
result1 = dict1.copy()
result1.update(dict2)

\# Method 2: Using ** operator
result2 = \{**dict1, **dict2\}

print("Result:", result2)
\# Output: \{1: 10, 2: 20, 3: 30, 4: 40\}
```

- **Key-value pairs**: Each element has a key and value
- **Mutable**: Can be modified after creation
- **Fast access**: O(1) average time complexity

Mnemonic

"Dictionaries are Dynamic Key-Value stores"

Question 1(c) OR [7 marks]

What is a list in python? Write a program that finds maximum and minimum numbers from a list.

Solution

List is an ordered, mutable collection of elements in Python.
Program:

```
\# Input list
numbers = [45, 12, 78, 23, 56, 89, 34]

\# Find maximum and minimum
maximum = max(numbers)
minimum = min(numbers)

print(f"Maximum: \{maximum\}")
print(f"Minimum: \{minimum\}")

\# Manual method
max\_val = numbers[0]
min\_val = numbers[0]
for num in numbers:
    if num {} max\_val:
        max\_val = num
    if num {} min\_val:
        min\_val = num
```

- **Ordered**: Elements maintain insertion order
- **Indexing**: Accessed using index [0, 1, 2...]
- **Built-in functions**: min(), max(), len() available

Mnemonic

"Lists are Linear and Indexed"

Question 2(a) [3 marks]

Explain Nested Tuple with example.

Solution

Nested Tuple is a tuple containing other tuples as elements.
Example:

```
\# Nested tuple
student\_data = (
    ("John", 85, "A"),
    ("Alice", 92, "A+"),
    ("Bob", 78, "B")
)

\# Accessing elements
print(student\_data[0][1])  \# Output: 85
print(student\_data[1][0])  \# Output: Alice
```

- **Multi-dimensional**: Tuples within tuples
- **Indexing**: Use multiple indices [i][j]
- **Immutable**: Cannot change nested elements

Mnemonic

"Nested means Tuples inside Tuples"

Question 2(b) [4 marks]

What is random module? Explain with example.

Solution

Random module generates random numbers and performs random operations.
Example:

```
import random

\# Random integer
num = random.randint(1, 10)
print(f"Random number: \{num\}")

\# Random choice from list
colors = ["red", "blue", "green"]
choice = random.choice(colors)
print(f"Random color: \{choice\}")

\# Random float
decimal = random.random()
print(f"Random decimal: \{decimal\}")
```

- **Import required**: import random
- **Various functions**: randint(), choice(), random()
- **Useful for**: Games, simulations, testing

Mnemonic

"Random makes things Unpredictable"

Question 2(c) [7 marks]

Explain different ways of importing package. Give one example of it.

Solution

Import Methods:

Method	Syntax	Usage
Normal import	`import package`	package.function()
From import	`from package import function`	function()
Import all	`from package import *`	function()
Alias import	`import package as alias`	alias.function()

Example:

```
\# Normal import
import math
result1 = math.sqrt(16)

\# From import
from math import sqrt
result2 = sqrt(16)

\# Import with alias
import math as m
result3 = m.sqrt(16)

\# Import all (not recommended)
from math import *
result4 = sqrt(16)
```

- **Namespace**: Normal import keeps separate namespace
- **Direct access**: From import allows direct function call
- **Alias**: Shorter names for convenience

Mnemonic

"Import methods: Normal, From, All, Alias"

Question 2(a) OR [3 marks]

Write down the properties of dictionary in python.

Solution

Dictionary Properties:

Property	Description
Ordered	Maintains insertion order (Python 3.7+)
Mutable	Can be modified after creation
Key-unique	No duplicate keys allowed

Heterogeneous Keys and values can be different types

- **Fast access**: O(1) average lookup time
- **Dynamic size**: Can grow or shrink
- **Key restrictions**: Keys must be immutable

Mnemonic

"Dictionaries are Ordered, Mutable, Unique, Heterogeneous"

Question 2(b) OR [4 marks]

What is the dir() function in python. Explain with example.

Solution

dir() function returns all attributes and methods of an object.
Example:

```
\# List all attributes of string
text = "hello"
attributes = dir(text)
print(attributes[:5])  \# First 5 attributes

\# Check available methods
print("upper" in dir(text))  \# True

\# For modules
import math
math\_methods = dir(math)
print("sqrt" in math\_methods)  \# True

\# For custom objects
class MyClass:
    def my\_method(self):
        pass

obj = MyClass()
print(dir(obj))
```

- **Introspection**: Examines object properties
- **Debugging**: Helps find available methods
- **All objects**: Works with any Python object

Mnemonic

"dir() shows Directory of object attributes"

Question 2(c) OR [7 marks]

Write a program to define module to find sum of two numbers. Import module to another program.

Solution

Module file (calculator.py):

```
\# calculator.py
```

```
def add\_numbers(a, b):
    """Function to add two numbers"""
    return a + b

def multiply\_numbers(a, b):
    """Function to multiply two numbers"""
    return a * b

def get\_sum(num1, num2):
    """Alternative sum function"""
    result = num1 + num2
    return result
```

Main program:

```
\# main.py
import calculator

\# Using the module
result1 = calculator.add\_numbers(10, 20)
print(f"Sum: \{result1\}")

\# From import
from calculator import get\_sum
result2 = get\_sum(15, 25)
print(f"Sum using from import: \{result2\}")
```

- **Module creation**: Save functions in .py file
- **Import**: Use import statement to access
- **Code reusability**: Use same module in multiple programs

Mnemonic

"Modules make code Reusable and Organized"

Question 3(a) [3 marks]

What is Runtime error and Logical error. Explain with example.

Solution

Error Type	Definition	Example
Runtime Error	Occurs during program execution	Division by zero, file not found
Logical Error	Program runs but gives wrong output	Wrong formula, incorrect condition

Examples:

```
\# Runtime Error
x = 10
y = 0
result = x / y  \# ZeroDivisionError

\# Logical Error
def calculate\_area(radius):
    return 3.14 * radius  \# Should be radius * radius
```

- **Runtime**: Crashes program execution
- **Logical**: Program continues but wrong result

Mnemonic

"Runtime Crashes, Logical Confuses"

Question 3(b) [4 marks]

Write points on Except and explaining it.

Solution

Except clause handles specific exceptions in try-except block.
Key Points:

Feature	Description
Syntax	`except ExceptionType:`
Multiple	Can have multiple except blocks
Generic	`except:` catches all exceptions
Variable	`except Exception as e:` stores error

```
try:
    number = int(input("Enter number: "))
    result = 10 / number
except ValueError:
    print("Invalid input")
except ZeroDivisionError:
    print("Cannot divide by zero")
except Exception as e:
    print(f"Error: \{e\}")
```

- **Specific handling**: Different exceptions handled differently
- **Error recovery**: Program continues after handling

Mnemonic

"Except Catches and Handles errors"

Question 3(c) [7 marks]

Write a program to catch Divide by zero Exception. Also use finally block.

Solution

```
def safe\_division():
    try:
        \# Get input from user
        numerator = float(input("Enter numerator: "))
        denominator = float(input("Enter denominator: "))

        \# Perform division
        result = numerator / denominator
        print(f"Result: \{numerator\} / \{denominator\} = \{result\}")

    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
        print("Please enter a non{-zero denominator"})

    except ValueError:
        print("Error: Please enter valid numbers only")

    except Exception as e:
        print(f"Unexpected error occurred: \{e\}")

    finally:
        print("Division operation completed")
        print("Thank you for using the calculator")

\# Call the function
safe\_division()
```

- **Try block**: Contains risky code
- **Except**: Handles ZeroDivisionError specifically
- **Finally**: Always executes regardless of exception

Mnemonic

"Try risky code, Except handles errors, Finally always runs"

Question 3(a) OR [3 marks]

What are the built-in exceptions and gives its types.

Solution

Built-in Exception Types:

Type	Description	Example
ValueError	Invalid value for operation	int("abc")
TypeError	Wrong data type	"5" + 5
IndexError	Index out of range	list[10] for 5-element list
KeyError	Key not found in dictionary	dict["missing_key"]
FileNotFoundError	File doesn't exist	open("missing.txt")

```
\# Examples
try:
    int("hello")  \# ValueError
    "5" + 5       \# TypeError
    [1,2,3][5]    \# IndexError
except (ValueError, TypeError, IndexError) as e:
    print(f"Error: \{type(e).\_\_name\_\_\}")
```

Mnemonic

"Value, Type, Index, Key, File - common error types"

Question 3(b) OR [4 marks]

Explain Syntax error and how do we identify it? Give an example.

Solution

Syntax Error occurs when Python cannot parse the code due to incorrect syntax.
Identification:

Method	Description
Python interpreter	Shows error message with line number
IDE highlighting	Code editors highlight syntax errors
Error message	Points to exact location of error

Examples:

```
\# Missing colon
if x {} 5
    print("Greater")  \# SyntaxError

\# Unmatched parentheses
print("Hello"  \# SyntaxError

\# Incorrect indentation
def my\_function():
print("Hello")  \# IndentationError

\# Invalid variable name
2variable = 10  \# SyntaxError
```

- **Detection**: Before program execution
- **Error message**: Shows line and character position
- **Common causes**: Missing colons, brackets, wrong indentation

Mnemonic

"Syntax errors Stop code from Starting"

Question 3(c) OR [7 marks]

What is Exception handling in python? Explain with proper example.

Solution

Exception Handling is a mechanism to handle runtime errors gracefully without crashing the program.
Structure:

```
try:
    \# Risky code
    pass
except SpecificException:
    \# Handle specific error
    pass
except Exception as e:
```

```
    \# Handle any other error
    pass
else:
    \# Runs if no exception
    pass
finally:
    \# Always runs
    pass
```

Complete Example:

```
def file\_processor():
    filename = None
    try:
        filename = input("Enter filename: ")
        with open(filename, {r}) as file:
            content = file.read()
            numbers = [int(x) for x in content.split()]
            average = sum(numbers) / len(numbers)
            print(f"Average: \{average\}")

    except FileNotFoundError:
        print(f"Error: File {}\{filename\}{ not found"})

    except ValueError:
        print("Error: File contains non{-numeric data"})

    except ZeroDivisionError:
        print("Error: No numbers found in file")

    except Exception as e:
        print(f"Unexpected error: \{e\}")

    else:
        print("File processed successfully")

    finally:
        print("File processing operation completed")

\# Run the function
file\_processor()
```

- **Graceful handling**: Program continues after error
- **Multiple exceptions**: Different error types handled separately
- **Else clause**: Runs only if no exception occurs
- **Finally clause**: Always executes for cleanup

Mnemonic

"Try-Except-Else-Finally: Complete error handling"

Question 4(a) [3 marks]

What kind of different operations we can perform in a file?

Solution

File Operations:

Operation	Description	Method
Read	Read file content	read(), readline(), readlines()
Write	Write data to file	write(), writelines()
Append	Add data to end	open with 'a' mode
Create	Create new file	open with 'w' or 'x' mode
Delete	Remove file	os.remove()
Seek	Move file pointer	seek()

```
\# Example operations
with open({file.txt}, {w}) as f:
    f.write("Hello")  \# Write

with open({file.txt}, {r}) as f:
    content = f.read()  \# Read
```

Mnemonic

"Read, Write, Append, Create, Delete, Seek"

Question 4(b) [4 marks]

Give list of file modes. Write Description of any four mode.

Solution

File Modes:

Mode	Description	Purpose
'r'	Read mode (default)	Read existing file
'w'	Write mode	Create new or overwrite existing
'a'	Append mode	Add to end of existing file
'x'	Exclusive creation	Create new file, fail if exists
'b'	Binary mode	Handle binary files
't'	Text mode (default)	Handle text files
'+'	Read and write	Both operations allowed

Four Mode Descriptions:

1. **'r' (Read)**: Opens file for reading only, file pointer at beginning
2. **'w' (Write)**: Opens for writing, truncates file or creates new one
3. **'a' (Append)**: Opens for writing, file pointer at end of file
4. **'r+' (Read/Write)**: Opens for both reading and writing

Mnemonic

"Read, Write, Append, eXclusive - main file modes"

Question 4(c) [7 marks]

Write a program to sort all the words in a file and put it in list.

Solution

```
def sort\_words\_from\_file():
    try:
        \# Input filename
```

```
        filename = input("Enter filename: ")

        \# Read file content
        with open(filename, {r}) as file:
            content = file.read()

        \# Split into words and clean them
        words = content.lower().split()

        \# Remove punctuation and empty strings
        import string
        clean\_words = []
        for word in words:
            clean\_word = word.translate(str.maketrans({}, {}, string.punctuation))
            if clean\_word:  \# Add only non{-empty words}
                clean\_words.append(clean\_word)

        \# Sort the words
        sorted\_words = sorted(clean\_words)

        \# Display results
        print("Sorted words:")
        print(sorted\_words)

        \# Save to new file
        with open({sorted\_words.txt}, {w}) as output\_file:
            for word in sorted\_words:
                output\_file.write(word + {}{n}{})

        print(f"Total words: \{len(sorted\_words)\}")
        print("Sorted words saved to {sorted\_words.txt"})

    except FileNotFoundError:
        print("Error: File not found")
    except Exception as e:
        print(f"Error: \{e\}")

\# Run the program
sort\_words\_from\_file()
```

- **File reading**: Read entire file content
- **Word processing**: Split, clean, and sort words
- **List creation**: Store sorted words in list

Mnemonic

"Read, Split, Clean, Sort, Save"

Question 4(a) OR [3 marks]

What is file handling? List files handling operation and explain it.

Solution

File Handling is the process of working with files to store and retrieve data permanently.
File Handling Operations:

Operation	Function	Description
Open	open()	Opens file in specified mode

Read	read(), readline()	Reads data from file
Write	write(), writelines()	Writes data to file
Close	close()	Closes file and frees resources
Seek	seek()	Moves file pointer position
Tell	tell()	Returns current file pointer position

```
\# Basic file operations
file = open({data.txt}, {w})  \# Open
file.write({Hello World})     \# Write
file.close()                  \# Close

file = open({data.txt}, {r})  \# Open for reading
content = file.read()         \# Read
file.close()                  \# Close
```

Mnemonic

"Open, Read, Write, Close - basic file cycle"

Question 4(b) OR [4 marks]

Explain load() method with example.

Solution

load() method is used to deserialize data from a file (usually with pickle module).
Pickle load() Example:

```
import pickle

\# First, save some data
data\_to\_save = \{
    {name}: {John},
    {age}: 25,
    {scores}: [85, 92, 78]
\}

\# Save data to file
with open({data.pkl}, {wb}) as file:
    pickle.dump(data\_to\_save, file)

\# Load data from file
with open({data.pkl}, {rb}) as file:
    loaded\_data = pickle.load(file)

print("Loaded data:", loaded\_data)
print("Name:", loaded\_data[{name}])
print("Scores:", loaded\_data[{scores}])
```

JSON load() Example:

```
import json

\# Load JSON data
with open({config.json}, {r}) as file:
    config = json.load(file)

print("Configuration:", config)
```

- **Deserialization**: Converts file data back to Python objects

- **Binary mode**: Use 'rb' mode for pickle files
- **Error handling**: Handle FileNotFoundError

Mnemonic

"load() brings file data back to Python objects"

Question 4(c) OR [7 marks]

Write a program that inputs a text file. The program should print all of the unique words in the file in alphabetical order.

Solution

```
def find\_unique\_words():
    try:
        \# Get filename from user
        filename = input("Enter text filename: ")

        \# Read file content
        with open(filename, {r}, encoding={utf{-}8}) as file:
            content = file.read().lower()

        \# Clean and extract words
        import re
        import string

        \# Remove punctuation and split into words
        words = re.findall(r{}{b}[a{-zA{-}Z]}+{b}{}, content.lower())

        \# Create set to get unique words
        unique\_words = set(words)

        \# Convert to sorted list
        sorted\_unique\_words = sorted(list(unique\_words))

        \# Display results
        print("{n}Unique words in alphabetical order:")
        print("{-"} * 40)

        for i, word in enumerate(sorted\_unique\_words, 1):
            print(f"\{i:3d\}. \{word\}")

        print(f"{n}Total unique words: \{len(sorted\_unique\_words)\}")

        \# Save results to file
        with open({unique\_words\_output.txt}, {w}) as output\_file:
            output\_file.write("Unique Words in Alphabetical Order{n}")
            output\_file.write("=" * 40 + "{nn}")
            for word in sorted\_unique\_words:
                output\_file.write(word + {}{n}{})

        print("Results saved to {unique\_words\_output.txt"})

    except FileNotFoundError:
        print(f"Error: File {}\{filename\}{ not found"})
    except PermissionError:
        print("Error: Permission denied to read file")
    except Exception as e:
```

```
        print(f"Unexpected error: \{e\}")

\# Example usage
def create\_sample\_file():
    sample\_text = """
    Python is a powerful programming language.
    Python is easy to learn and Python is versatile.
    Programming with Python is fun and programming is rewarding.
    """

    with open({sample.txt}, {w}) as f:
        f.write(sample\_text)
    print("Sample file {sample.txt created"})

\# Create sample and run program
create\_sample\_file()
find\_unique\_words()
```

- **Regular expressions**: Extract only alphabetic words
- **Set data structure**: Automatically removes duplicates
- **Sorted function**: Arranges words alphabetically
- **File output**: Saves results for future reference

Mnemonic

"Read, Extract, Unique, Sort, Display"

Question 5(a) [3 marks]

Explain the use of the following turtle function with an appropriate example. (a) turn() (b) move().

Solution

Note: Standard turtle module uses `left()`, `right()` instead of `turn()`, and `forward()`, `backward()` instead of `move()`.

Turtle Movement Functions:

Function	Purpose	Example
left(angle)	Turn left by degrees	turtle.left(90)
right(angle)	Turn right by degrees	turtle.right(45)
forward(distance)	Move forward	turtle.forward(100)
backward(distance)	Move backward	turtle.backward(50)

```
import turtle

\# Create turtle
t = turtle.Turtle()

\# Turn functions
t.left(90)    \# Turn left 90 degrees
t.right(45)   \# Turn right 45 degrees

\# Move functions
t.forward(100)  \# Move forward 100 units
t.backward(50)  \# Move backward 50 units

\# Keep window open
turtle.done()
```

Mnemonic

"Turn changes direction, Move changes position"

Question 5(b) [4 marks]

Explain the various inbuilt methods to change the direction of the turtle.

Solution

Direction Control Methods:

Method	Description	Example
left(angle)	Turn counterclockwise	turtle.left(90)
right(angle)	Turn clockwise	turtle.right(45)
setheading(angle)	Set absolute direction	turtle.setheading(0)
towards(x, y)	Point towards coordinates	turtle.setheading(turtle.towards(100, 100))

```
import turtle

t = turtle.Turtle()

\# Relative turning
t.left(90)        \# Turn left 90^
t.right(45)       \# Turn right 45^

\# Absolute direction
t.setheading(0)   \# Point East (0^)
t.setheading(90)  \# Point North (90^)

\# Point towards specific point
angle = t.towards(100, 100)
t.setheading(angle)
```

- **Relative**: left() and right() change current direction
- **Absolute**: setheading() sets exact direction
- **Coordinate-based**: towards() calculates direction to point

Mnemonic

"Left-Right relative, Heading absolute, Towards calculates"

Question 5(c) [7 marks]

Write a program to draw square, rectangle and circle using turtle.

Solution

```
import turtle

def draw\_shapes():
    \# Create turtle and screen
    screen = turtle.Screen()
    screen.title("Drawing Shapes with Turtle")
    screen.bgcolor("white")
    screen.setup(800, 600)
```

```
    \# Create turtle
    pen = turtle.Turtle()
    pen.speed(3)
    pen.color("blue")

    \# Draw Square
    pen.penup()
    pen.goto({-}200, 100)
    pen.pendown()
    pen.write("Square", font=("Arial", 12, "bold"))
    pen.goto({-}200, 50)

    for i in range(4):
        pen.forward(80)
        pen.right(90)

    \# Draw Rectangle
    pen.penup()
    pen.goto(0, 100)
    pen.pendown()
    pen.color("red")
    pen.write("Rectangle", font=("Arial", 12, "bold"))
    pen.goto(0, 50)

    for i in range(2):
        pen.forward(120)  \# Length
        pen.right(90)
        pen.forward(60)   \# Width
        pen.right(90)

    \# Draw Circle
    pen.penup()
    pen.goto(200, 100)
    pen.pendown()
    pen.color("green")
    pen.write("Circle", font=("Arial", 12, "bold"))
    pen.goto(200, 50)

    pen.circle(40)  \# Radius = 40

    \# Hide turtle and keep window open
    pen.hideturtle()
    screen.exitonclick()

\# Alternative function for each shape
def draw\_square(turtle\_obj, size):
    """Draw a square with given size"""
    for \_ in range(4):
        turtle\_obj.forward(size)
        turtle\_obj.right(90)

def draw\_rectangle(turtle\_obj, width, height):
    """Draw a rectangle with given dimensions"""
    for \_ in range(2):
        turtle\_obj.forward(width)
        turtle\_obj.right(90)
        turtle\_obj.forward(height)
        turtle\_obj.right(90)

def draw\_circle(turtle\_obj, radius):
    """Draw a circle with given radius"""
```

```
    turtle\_obj.circle(radius)

\# Run the main program
draw\_shapes()
```

- **Square**: 4 equal sides with 90°*turns*
- **Rectangle**: 2 pairs of equal sides
- **Circle**: Built-in circle() method with radius

Mnemonic

"Square: 4 equal sides, Rectangle: 2 pairs, Circle: radius method"

Question 5(a) OR [3 marks]

What are the various types of pen command in turtle? Explain them all.

Solution

Pen Control Commands:

Command	Purpose	Example
penup()	Lift pen (no drawing)	turtle.penup()
pendown()	Put pen down (start drawing)	turtle.pendown()
pensize(width)	Set pen thickness	turtle.pensize(5)
pencolor(color)	Set pen color	turtle.pencolor("red")
fillcolor(color)	Set fill color	turtle.fillcolor("blue")
begin_fill()	Start filling shape	turtle.begin_fill()
end_fill()	End filling shape	turtle.end_fill()

```
import turtle

t = turtle.Turtle()

\# Pen control
t.penup()           \# Lift pen
t.goto(50, 50)      \# Move without drawing
t.pendown()         \# Put pen down
t.pensize(3)        \# Set thickness
t.pencolor("red")   \# Set color
```

Mnemonic

"Up-Down controls drawing, Size-Color controls appearance"

Question 5(b) OR [4 marks]

Draw circle and star shapes using turtle and fill them with red color.

Solution

```
import turtle

def draw\_filled\_shapes():
    \# Setup screen
    screen = turtle.Screen()
```

```
    screen.bgcolor("white")
    screen.title("Filled Circle and Star")

    \# Create turtle
    artist = turtle.Turtle()
    artist.speed(5)

    \# Draw filled circle
    artist.penup()
    artist.goto({-}150, 0)
    artist.pendown()

    \# Set colors for circle
    artist.color("red", "red")  \# pen color, fill color
    artist.begin\_fill()
    artist.circle(50)
    artist.end\_fill()

    \# Draw filled star
    artist.penup()
    artist.goto(100, 0)
    artist.pendown()

    \# Set colors for star
    artist.color("red", "red")
    artist.begin\_fill()

    \# Draw 5{-pointed star}
    for i in range(5):
        artist.forward(100)
        artist.right(144)

    artist.end\_fill()

    \# Add labels
    artist.penup()
    artist.goto({-}180, {-}80)
    artist.color("black")
    artist.write("Filled Circle", font=("Arial", 12, "bold"))

    artist.goto(70, {-}80)
    artist.write("Filled Star", font=("Arial", 12, "bold"))

    \# Hide turtle
    artist.hideturtle()
    screen.exitonclick()

\# Run the program
draw\_filled\_shapes()
```

Key Points:

- **begin_fill()**: Start filling the shape
- **end_fill()**: Complete the fill
- **color()**: Set both pen and fill colors
- **Star angle**: $144° for 5-pointed star$

Mnemonic

"Begin fill, Draw shape, End fill = Filled shape"

Question 5(c) OR [7 marks]

Write a program to draw Indian Flag using turtle.

Solution

```
import turtle

def draw\_indian\_flag():
    \# Create screen
    screen = turtle.Screen()
    screen.bgcolor("white")
    screen.title("Indian Flag")
    screen.setup(800, 600)

    \# Create turtle
    flag = turtle.Turtle()
    flag.speed(5)
    flag.pensize(2)

    \# Flag dimensions
    flag\_width = 300
    flag\_height = 200

    \# Starting position
    start\_x = {-}150
    start\_y = 100

    \# Draw flag pole
    flag.penup()
    flag.goto(start\_x {-} 20, start\_y + 50)
    flag.pendown()
    flag.color("brown")
    flag.pensize(8)
    flag.setheading(270)  \# Point downward
    flag.forward(400)

    \# Reset pen
    flag.pensize(2)
    flag.color("black")

    \# Draw saffron rectangle (top)
    flag.penup()
    flag.goto(start\_x, start\_y)
    flag.pendown()
    flag.color("orange", "orange")
    flag.begin\_fill()
    flag.setheading(0)

    for \_ in range(2):
        flag.forward(flag\_width)
        flag.right(90)
        flag.forward(flag\_height // 3)
        flag.right(90)
    flag.end\_fill()

    \# Draw white rectangle (middle)
    flag.penup()
    flag.goto(start\_x, start\_y {-} flag\_height // 3)
    flag.pendown()
    flag.color("black", "white")
    flag.begin\_fill()
```

```
    for \_ in range(2):
        flag.forward(flag\_width)
        flag.right(90)
        flag.forward(flag\_height // 3)
        flag.right(90)
    flag.end\_fill()

    \# Draw green rectangle (bottom)
    flag.penup()
    flag.goto(start\_x, start\_y {-} 2 * flag\_height // 3)
    flag.pendown()
    flag.color("green", "green")
    flag.begin\_fill()

    for \_ in range(2):
        flag.forward(flag\_width)
        flag.right(90)
        flag.forward(flag\_height // 3)
        flag.right(90)
    flag.end\_fill()

    \# Draw Ashoka Chakra (wheel)
    chakra\_center\_x = start\_x + flag\_width // 2
    chakra\_center\_y = start\_y {-} flag\_height // 2

    flag.penup()
    flag.goto(chakra\_center\_x, chakra\_center\_y {-} 30)
    flag.pendown()
    flag.color("navy blue")
    flag.pensize(3)

    \# Draw outer circle
    flag.circle(30)

    \# Draw spokes
    flag.penup()
    flag.goto(chakra\_center\_x, chakra\_center\_y)
    flag.pendown()

    for i in range(24):  \# 24 spokes in Ashoka Chakra
        flag.setheading(i * 15)  \# 360/24 = 15 degrees
        flag.forward(30)
        flag.backward(30)

    \# Draw inner circle
    flag.penup()
    flag.goto(chakra\_center\_x, chakra\_center\_y {-} 5)
    flag.pendown()
    flag.circle(5)

    \# Add title
    flag.penup()
    flag.goto({-}100, 200)
    flag.color("black")
    flag.write("INDIAN FLAG", font=("Arial", 16, "bold"))

    \# Hide turtle
    flag.hideturtle()
    screen.exitonclick()

\# Run the program
```

```
draw\_indian\_flag()
```

Flag Components:

- **Saffron**: Courage and sacrifice (top)
- **White**: Truth and peace (middle)
- **Green**: Faith and chivalry (bottom)
- **Ashoka Chakra**: 24-spoke wheel in navy blue

Mnemonic

"Saffron-White-Green stripes with 24-spoke Chakra"