

AKTU BCC302 – PYTHON PROGRAMMING

B.Tech Semester III • Theory Examination 2024–25 • Solved Question Paper

SECTION A

1(a). State how to handle exceptions in Python? Provide a simple example. [2 marks]

Answer: Python handles runtime errors using exception-handling blocks. The **try** block contains code that may generate an exception and the **except** block handles it. The optional **else** block runs when no exception occurs, while **finally** runs in all cases and is useful for cleanup.

```
try:
    a = int(input("Enter a number: "))
    b = int(input("Enter divisor: "))
    print("Result =", a / b)
except ZeroDivisionError:
    print("Division by zero is not allowed.")
except ValueError:
    print("Enter valid integers.")
finally:
    print("Program execution completed.")
```

1(b). What will be the output of the following Python code? [2 marks]

Answer: The list comprehension selects the even elements 2 and 4 and squares them. Therefore, the output is **[4, 16]**.

```
def compute(x):
    return [i**2 for i in x if i%2==0]

print(compute([1, 2, 3, 4, 5]))

Output:
[4, 16]
```

1(c). Explain floor division with an example. [2 marks]

Answer: Floor division uses the **//** operator and returns the greatest integer less than or equal to the actual quotient.

```
print(17 // 5)
print(-17 // 5)

Output:
3
-4
```

1(d). Describe the purpose of the 'with' statement in file handling. [2 marks]

Answer: The **with** statement provides a safe way to work with files. It automatically closes the file after the block finishes, including when an exception occurs. It therefore makes file handling cleaner and safer.

```
with open("data.txt", "r") as file:
    data = file.read()
    print(data)
```

1(e). Briefly describe the use of lambda functions in Python. [2 marks]

Answer: A lambda function is a small anonymous function written with the **lambda** keyword. It contains a single expression and is useful for short temporary operations, particularly with **map()**, **filter()** and **sorted()**.

```
square = lambda x: x * x
print(square(5))

Output:
25
```

1(f). Demonstrate how to assign a single value to a tuple. [2 marks]

Answer: A single-element tuple must contain a trailing comma. Without the comma, Python treats the expression as the value itself.

```
t = (10,)
print(t)
print(type(t))
```

Output:
(10,)
<class 'tuple'>

1(g). Explain why NumPy is used instead of Python arrays for mathematical calculations. [2 marks]

Answer: NumPy provides efficient multidimensional arrays and vectorized mathematical operations. It is designed for numerical computing and provides many mathematical, statistical and linear-algebra functions. NumPy operations are generally more efficient for numerical work than ordinary Python sequences.

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)
```

Output:
[5 7 9]

SECTION B

2(a). Design a basic calculator in Python that supports addition, subtraction, multiplication and division. [7 marks]

Answer: A calculator can accept two numbers and an operator. An **if-elif** structure performs the selected operation. Division by zero is checked separately.

```
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))
op = input("Enter operator (+, -, *, /): ")

if op == "+":
    result = a + b
elif op == "-":
    result = a - b
elif op == "*":
    result = a * b
elif op == "/":
    if b != 0:
        result = a / b
    else:
        print("Division by zero is not allowed.")
        result = None
else:
    print("Invalid operator.")
    result = None

if result is not None:
    print("Result =", result)
```

2(b). Define Membership and Identity Operators. Given a = 3 and b = 3, distinguish between (a is b) and (a == b). [7 marks]

Answer: **Membership operators** (**in**, **not in**) test whether a value exists in a sequence or collection. **Identity operators** (**is**, **is not**) test whether two references refer to the same object. **==** compares values, whereas **is** compares identity. For small integers, CPython commonly interns the objects, so **a is b** may also be True; however, **is** should not be used for ordinary value comparison.

```

a = 3
b = 3

print(a == b)    # True: same value
print(a is b)    # commonly True for small integers

print(3 in [1, 2, 3])    # True
print(4 not in [1, 2, 3])    # True

```

2(c). Write a Python function to count the frequency of each character in a given string and return the output in a dictionary. Example: `char_frequency("HELLO")` returns `{"H":1, "E":1, "L":2, "O":1}`. [7 marks]

Answer: The function maintains a dictionary of character counts. When a character appears for the first time its count is initialized to 1; otherwise its existing count is increased.

```

def char_frequency(text):
    frequency = {}

    for ch in text:
        if ch in frequency:
            frequency[ch] += 1
        else:
            frequency[ch] = 1

    return frequency

print(char_frequency("HELLO"))

```

Output:
`{'H': 1, 'E': 1, 'L': 2, 'O': 1}`

2(d). Write a program to reverse the contents of a file character by character, separating each character with a comma. [7 marks]

Answer: The file contents are read as a string, traversed from the last character to the first, and joined using commas.

```

with open("input.txt", "r") as file:
    data = file.read()

reversed_data = ",".join(reversed(data))

with open("reversed.txt", "w") as file:
    file.write(reversed_data)

print("Reversed content written successfully.")

```

2(e). Create a pie chart using matplotlib to represent the following data: Python 30, Java 25, C++ 20, JavaScript 15, Ruby 10. [7 marks]

Answer: The given popularity values can be represented using `matplotlib.pyplot.pie()`. The labels identify the languages and `autopct` displays the percentage values.

```

import matplotlib.pyplot as plt

languages = ["Python", "Java", "C++", "JavaScript", "Ruby"]
popularity = [30, 25, 20, 15, 10]

plt.pie(popularity, labels=languages,
        autopct="%1.1f%%", startangle=90)

plt.title("Programming Language Popularity")
plt.axis("equal")
plt.show()

```

SECTION C

3(a). Write short notes on the following with examples: (i) Operator Precedence, (ii) Python Indentation, (iii) Type Conversion. [7 marks]

Answer: (i) Operator Precedence: It determines the order in which operators are evaluated. For example, multiplication is evaluated before addition unless parentheses change the order.

(ii) Python Indentation: Indentation defines blocks of code in Python. Statements belonging to the same block must have consistent indentation.

(iii) Type Conversion: It changes a value from one data type to another using functions such as `int()`, `float()` and `str()`.

```
# Operator precedence
x = 2 + 3 * 4
print(x)          # 14

# Indentation
if x > 10:
    print("x is greater than 10")

# Type conversion
a = "25"
b = int(a)
print(b + 5)      # 30
```

3(b). Write a program to validate email addresses using regular expressions. The email must contain @, must contain a domain name, and must not have spaces. [7 marks]

Answer: The `re` module can validate the required structure. The pattern below requires an @ symbol, a domain part and a dot in the domain, while disallowing spaces.

```
import re

email = input("Enter email address: ")

pattern = r"^[^\\s@]+@[^\\s@]+\\.([^\\s@]+)$"

if re.fullmatch(pattern, email):
    print("Valid email address")
else:
    print("Invalid email address")
```

4(a). Write a program to create a hollow pyramid pattern given below. [7 marks]

Answer: The program prints the boundary stars for the middle rows and the complete base. It follows the five-row pattern shown in the question paper.

```
rows = 5

for i in range(1, rows + 1):
    if i == 1:
        print("")
    elif i == 2:
        print("***")
    elif i == rows:
        print("*** * (2 * rows - 1))")
    else:
        print("*** + " " * (i - 1) + "***")
```

4(b). Explain why loops are needed and the types of loops in Python. [7 marks]

Answer: Loops are required when a statement or block must be executed repeatedly. They reduce repetition and make programs shorter and easier to manage. Python mainly provides **for** and **while** loops. A **for** loop iterates over a

sequence or range, while a **while** loop repeats while a condition is true. **break**, **continue** and **pass** are useful loop-control statements.

```
# for loop
for i in range(1, 4):
    print(i)

# while loop
i = 1
while i <= 3:
    print(i)
    i += 1
```

5(a). Write a function to find the longest word in a given list of words. Example: longest_word(['apple', 'banana', 'cherry']) returns 'banana'. [7 marks]

Answer: The function stores the first word as the current longest word and compares the length of every remaining word with it. The word having the greatest length is returned.

```
def longest_word(words):
    longest = words[0]

    for word in words:
        if len(word) > len(longest):
            longest = word

    return longest

print(longest_word(["apple", "banana", "cherry"]))
```

Output:
banana

5(b). Distinguish between a Tuple and a List with examples. Explain with examples at least 4 built-in methods of Dictionary. [7 marks]

Answer: Tuple: immutable and normally written using parentheses. **List:** mutable and written using square brackets. List elements can be changed, added or removed, whereas tuple elements cannot be modified after creation.

Dictionary methods: **keys()** returns keys; **values()** returns values; **items()** returns key-value pairs; **get()** returns the value for a specified key.

```
my_list = [10, 20, 30]
my_tuple = (10, 20, 30)

my_list[0] = 100

student = {"name": "Aman", "marks": 85}

print(student.keys())
print(student.values())
print(student.items())
print(student.get("marks"))

Output:
dict_keys(['name', 'marks'])
dict_values(['Aman', 85])
dict_items([('name', 'Aman'), ('marks', 85)])
85
```

6(a). Discuss different types of file modes in Python. Explain with examples. [7 marks]

Answer: **r** – read an existing file. **w** – write and overwrite/create a file. **a** – append at the end. **x** – create a new file and fail if it already exists. **b** – binary mode, such as **rb** and **wb**. **t** – text mode, the default. **+** – read and write mode, such as **r+**.

```
# Read
```

```

with open("data.txt", "r") as f:
    data = f.read()

# Write
with open("data.txt", "w") as f:
    f.write("Hello Python")

# Append
with open("data.txt", "a") as f:
    f.write("\nNew line")

# Binary read
with open("image.jpg", "rb") as f:
    data = f.read()

```

6(b). Write a program to read a CSV file and display the rows where a specific column value exceeds a given threshold. [7 marks]

Answer: The **csv** module reads the file. The selected column is converted to a number and compared with the threshold. Only rows satisfying the condition are printed.

```

import csv

threshold = 80

with open("students.csv", "r", newline="") as file:
    reader = csv.DictReader(file)

    for row in reader:
        marks = float(row["Marks"])
        if marks > threshold:
            print(row)

# Example:
# Name,Marks
# Aman,85
# Ravi,72
# Neha,91

# Output:
# {'Name': 'Aman', 'Marks': '85'}
# {'Name': 'Neha', 'Marks': '91'}

```

7(a). Discuss the role of event handling in Tkinter. How can you bind events? Provide examples. [7 marks]

Answer: Event handling enables a Tkinter application to respond to user actions such as mouse clicks and keyboard presses. The **bind()** method connects an event pattern with a callback function. For example, **<Button-1>** represents a left mouse click and **<Key>** represents a keyboard event.

```

import tkinter as tk

def clicked(event):
    label.config(text="Mouse clicked")

def key_pressed(event):
    label.config(text="Key: " + event.keysym)

root = tk.Tk()
root.title("Event Handling")

label = tk.Label(root, text="Perform an event")
label.pack(padx=30, pady=30)

root.bind("<Button-1>", clicked)
root.bind("<Key>", key_pressed)

root.mainloop()

```

7(b). Write a program to read data from a CSV file 'students.csv', calculate the average marks for each student, and display the results. [7 marks]

Answer: Assume the CSV contains a **Name** column followed by subject-mark columns. The program reads each row, converts the subject marks to numbers, calculates their sum and divides by the number of subjects.

```
import csv

with open("students.csv", "r", newline="") as file:
    reader = csv.DictReader(file)

    for row in reader:
        name = row["Name"]

        marks = [
            float(value)
            for key, value in row.items()
            if key != "Name" and value != ""
        ]

        average = sum(marks) / len(marks)

        print(name, "Average Marks =", round(average, 2))

# Example:
# Name,Python,DBMS,Maths
# Aman,80,75,85
# Neha,90,88,92

# Output:
# Aman Average Marks = 80.0
# Neha Average Marks = 90.0
```