1 (a) **Answer the following questions:** (2)

i. **What is Moore’s Law, and how does it relate to computer capabilities?**
ii. **What is big data and what are the 4 V’s of Big Data?**

Ans (i):
Moore's Law is an observation made by Gordon Moore, co-founder of Intel, in 1965. It states that the number of transistors on a microchip doubles approximately every two years, while the cost of production is halved. This trend has historically led to exponential growth in computing power over time.

Relation to Computer Capabilities:

Moore’s Law directly impacts several aspects of computing:

1. **Performance Improvements**:
   - As transistor density increases, processors become faster and more efficient, improving computational capabilities.
   - This allows for more complex and powerful algorithms to be run in shorter times.
2. **Cost Reduction**:
   - The cost per transistor has decreased significantly, making computing power cheaper and more accessible.
3. **Energy Efficiency**:
   - Smaller transistors consume less power, making devices more energy-efficient.
4. **Miniaturization**:
   - The increase in transistor density enables the creation of smaller, more powerful devices, such as smartphones and IoT devices.
5. **Parallelism**:
   - Modern processors include more cores and support for parallel processing, leveraging increased transistor counts.

Ans (ii):
Big Data refers to extremely large datasets that are too complex, vast, or varied to be efficiently processed, stored, or analyzed using traditional data-processing tools. It encompasses structured, semi-structured, and unstructured data collected from various sources, including social media, sensors, transactions, and logs.

The 4 V's of Big Data are Volume, Velocity, Variety and Veracity.

(b) **From the following expressions, identify which statements produce a runtime error.** (2)

i. 10/3+7	**ii. 10//3+7**	**iii.10/(3+7)**
iv. 10/3-3	**v. 10/(3-3)**	**vi. 10//(3-3)**

Ans:

i. No runtime error.	ii. No runtime error.	iii. No runtime error.
iv. No runtime error.	v. Runtime error.	vi. Runtime error.

(c)	**Analyze the expression: 5 * (3 + 2) ** 2 - 10 / 2. Explain how each operator's precedence affects the final outcome.** **Ans:** 5 * (3 + 2) ** 2 - 10 / 2 = 5×(5)**2 – 10 / 2 = 5×25 - 10 / 2 = 125 – 5.0 = 120.0	(2)

2 (a)	Answer the following questions:
	i. Can you use a conditional expression in Python? Explain with an example.
	ii. What will happen if the user inputs "abc" when prompted for an integer using int(input('Enter an integer: '))? What type of error will be raised?
	
	Ans (i):
	Yes, Python supports **conditional expressions**, also known as **ternary operators**, which allow to write concise, one-line conditional statements.
	Example:
	`num = 5`
	`result = "Even" if num % 2 == 0 else "Odd"`
	`print(result)`
	
	Output:
	Odd
	
	Ans (ii):
	The ***input()*** function reads the user input as a string. The ***int()*** function tries to convert the string to an integer. Since **"abc"** is not a valid integer, the conversion fails and Python will raise a '**ValueError**'.
		(2)

(b)	**Write a Python script that inputs a five-digit integer from the user. Separate the number into its individual digits and print them, separated by a tab. For example, if the user types in the number 42339, the script should print:**
	
	4 2 3 3 9
	
	Ans:
	`num = input('Enter an integer: ')`
	`for digit in num:`
	`    print(digit, end='\t')`
		(2)

(c)	**Write the output of the following Python script:**
	count = 1
	while count <= 5:
	    print(count, end=' ')
	    count += 1
	    if count == 5:
	        continue
	    print('ITER', end=' ')
	print('out from the loop')
	
	Ans:
	1 ITER 2 ITER 3 ITER 4 5 ITER out from the loop
		(2)

3 (a)	**How can you calculate the mean, median, and mode of a list of numbers in Python? Write a program that takes a list of integers, computes these statistics, and prints the results.**
	
	Ans:
	import statistics
	numbers = list(map(int, input("Enter a list of integers separated by spaces: ").split()))
	
	mean = statistics.mean(numbers)
	median = statistics.median(numbers)
	mode = statistics.mode(numbers)
	
	print(f"Mean: {mean}")
	print(f"Median: {median}")
	print(f"Mode: {mode}")
		(2)

(b)	**Write a Python function that accepts an input parameter for the number of rows to be printed and generates a figure like this:**
	1
	2 1
	3 2 1
	4 3 2 1
	5 4 3 2 1
	
	Ans:
	def print_figure(rows):
	    for i in range(1, rows + 1):
	        for j in range(i, 0, -1):
	            print(j, end=" ")
	        print()
	
	num_rows = int(input("Enter the number of rows to print: "))
	print_figure(num_rows)
		(2)

(c)	**Write a Python function that accepts an integer and displays all of its smallest factors in increasing order. For example, if the input integer is 120, the output should be: 2, 2, 2, 3, 5.**
	
	Ans:
	`def find_factors(n):`
	`    divisor = 2`
	`    while n > 1:`
	`        if n % divisor == 0:  # If divisor is a factor`
	`            print(divisor, end=" ")  # Print the factor`
	`            n //= divisor  # Reduce n by dividing it by the divisor`
	`        else:`
	`            divisor += 1  # Move to the next divisor`
	
	`num = int(input("Enter an integer: "))`
	
	`if num > 0:`
	`    print("Smallest factors in increasing order:", end=" ")`
	`    find_factors(num)`
	`else:`
	`    print("Please enter a positive integer.")`
		(2)

4 (a)	**What is the output of the following code?**
	`def cube(x):`
	`    """Calculate the cube of x."""`
	`    x**3`
	`print('The cube of 2 is', cube(2))`
	
	Ans:
	The cube of 2 is None
		(2)

(b)	**Write a Python function to print all the perfect numbers between 1 and 100. A number is called perfect if the sum of its proper divisors (excluding the number itself) is equal to the number. For example, the divisors of 6 are 1, 2, and 3, and 1+2+3=6.**
	
	Ans:
	`def perfect_numbers(limit):`
	`    for num in range(2, limit + 1):  # Start from 2 as 1 is not a perfect number`
	`        divisors_sum = 0`
	`        for i in range(1, num):`
	`            if num % i == 0:`
	`                divisors_sum += i`
	
	`        if divisors_sum == num:`
	`            print(num)`
	
	`perfect_numbers(100)`
		(2)

(c) **Using Python, apply the concepts of map, filter, reduce, and lambda expressions to accomplish the following task:** (2)

Calculate the sum of the cubes of all odd numbers between 1 and 101 (inclusive of both).

Guidelines:

- **Use filter to select the odd numbers from the range.**
- **Use map to compute the cube of each odd number.**
- **Use reduce to sum the cubes together.**

Ans:

```
from functools import reduce

odd_numbers = filter(lambda x: x % 2 != 0, range(1, 102))

cubed_numbers = map(lambda x: x ** 3, odd_numbers)

sum_of_cubes = reduce(lambda x, y: x + y, cubed_numbers)

print("Sum of cubes of odd numbers between 1 and 101:", sum_of_cubes)
```

5 (a) **Mention two differences between lists and tuples.** (2)

Ans:

List	Tuple
1. Collection of elements separated by commas and enclosed in square brackets [].	1. Collection of elements separated by commas and enclosed in round brackets ().
2. Lists are Mutable.	2. Tuples are Immutable.

(b) **Write a Python program that returns a list of duplicate elements from an input list. For example, if the input list is [1, 2, 3, 6, 5, 2, 3, 6, 7, 8, 6, 4, 5], the output list should be [2, 3, 6, 5].** (2)

Ans:

```
l=[1, 2, 3, 6, 5, 2, 3, 6, 7, 8, 6, 4, 5]
repeatl=[i for i in l if l.count(i)>1]
newl=[]
for i in repeatl:
    if i not in newl:
        newl.append(i)
print(newl)
```

Output:

```
[2, 3, 6, 5]
```

(c) **What will be the output of the following code? Can you analyze how it works?** (2)

```
import copy
list_a = [1, 2, [3, 4]]
list_b = list_a.copy()
list_b[2][0] = 10
print("List A:", list_a)
print("List B:", list_b)
list_c = copy.deepcopy(list_a)
list_c[2][0] = 20
print("List A", list_a)
print("List C", list_c)
```

Ans:

```
List A: [1, 2, [10, 4]]

List B: [1, 2, [10, 4]]

List A [1, 2, [10, 4]]

List C [1, 2, [20, 4]]
```

list_b is a shallow copy of list_a.
So although the non-nested elements of list_a (shallow level elements) are copied properly in list_b, the nested elements are not copied. Thus, any changes in the nested elements are reflected in both list_a and list_b.

list_c is a deep copy of the modified list_a.
So the nested and non-nested elements of the modified list_a are copied. Thus, changes made to list_c remain confined to list_c only and are not reflected in list_a.