Section 1: Python Basics & Control Flow

Q1. Print all odd numbers between 10 and 50:

```
for i in range(11, 50, 2):
    print(i)
```

Q2. Function to check for a leap year:

```
def is_leap_year(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

Q3. Count how many times the letter 'a' appears in a string:

```
def count_a(text):
    return text.lower().count('a')
```

Section 2: Collections (Lists, Tuples, Sets, Dicts)

```
keys = ['a', 'b', 'c']
values = [100, 200, 300]
my_dict = dict(zip(keys, values))
```

Q5. Salary analysis:

```
salaries = [50000, 60000, 55000, 70000, 52000]
max_salary = max(salaries)
avg_salary = sum(salaries) / len(salaries)
above_avg = [s for s in salaries if s > avg_salary]
sorted_desc = sorted(salaries, reverse=True)
```

Q6. Set operations:

```
a = [1, 2, 3, 4]
b = [3, 4, 5, 6]
set_a = set(a)
set_b = set(b)
difference = set_a.difference(set_b)
```

Section 3: Functions & Classes

Q7. `Employee` class:

```
class Employee:
    def __init__(self, name, department, salary):
        self.name = name
        self.department = department
        self.salary = salary

    def display(self):
        print(f"{self.name}, {self.department}, {self.salary}")

    def is_high_earner(self):
        return self.salary > 60000
```

Q8. `Project` class that inherits `Employee`:

```
class Project(Employee):
    def __init__(self, name, department, salary, project_name,
hours_allocated):
        super().__init__(name, department, salary)
        self.project_name = project_name
        self.hours_allocated = hours_allocated
```

Q9. Instantiate and check high earners:

```
e1 = Employee("Ali", "HR", 50000)
```

```
e2 = Employee("Neha", "IT", 60000)
e3 = Employee("Sara", "IT", 70000)

print(e1.is_high_earner())
print(e2.is_high_earner())
print(e3.is_high_earner())
```

Section 4: File Handling

Q10. Write names of IT employees to a file:

```
import pandas as pd
df = pd.read_csv("employees.csv")
df[df["Department"] == "IT"]["Name"].to_csv("it_employees.txt",
index=False, header=False)
```

Q11. Count words in a file:

```
with open("sample.txt", "r") as f:
    text = f.read()
    word_count = len(text.split())
```

Section 5: Exception Handling

Q12. Square a number with error handling:

```
try:
    num = int(input("Enter a number: "))
    print("Square:", num ** 2)
except ValueError:
    print("Invalid input. Please enter a number.")
```

Q13. Handle `ZeroDivisionError`:

```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Cannot divide by zero"
```

Section 6: Pandas – Reading & Exploring CSVs

Q14. Load CSVs:

```
import pandas as pd
employees = pd.read_csv("employees.csv")
projects = pd.read_csv("projects.csv")
```

Q15. Data exploration:

```
print(employees.head(2))
print(employees["Department"].unique())
print(employees.groupby("Department")["Salary"].mean())
```

Q16. Add `TenureInYears`:

```
from datetime import datetime
current_year = datetime.now().year
employees["JoiningDate"] = pd.to_datetime(employees["JoiningDate"])
employees["TenureInYears"] = current_year -
employees["JoiningDate"].dt.year
```

Section 7: Data Filtering, Aggregation, and Sorting

Q17. Filter IT employees with salary > 60000:

```
filtered = employees[(employees["Department"] == "IT") &
(employees["Salary"] > 60000)]
```

Q18. Group by Department:

```
grouped = employees.groupby("Department").agg(
    EmployeeCount=("EmployeeID", "count"),
    TotalSalary=("Salary", "sum"),
    AvgSalary=("Salary", "mean")
)
```

Q19. Sort by salary descending:

```
sorted_emps = employees.sort_values(by="Salary", ascending=False)
```

Section 8: Joins & Merging

Q20. Merge datasets on `EmployeeID`:

```
merged = pd.merge(employees, projects, on="EmployeeID", how="inner")
```

Q21. List employees with no projects:

```
merged_left = pd.merge(employees, projects, on="EmployeeID",
how="left")
no_projects = merged_left[merged_left["ProjectID"].isna()]
```

Q22. Add `TotalCost` column:

```
merged["TotalCost"] = merged["HoursAllocated"] * (merged["Salary"] /
160)
```