Python Basics & Control Flow

1. Write a Python program to print all odd numbers between 10 and 50.

```
for i in range(11, 50, 2):
    print(f"Odd Numbers: {i}")
```

2. Create a function that returns whether a given year is a leap year.

```
def leepYear(x):
    if (x % 4 == 0 and x % 100 != 0) or (x % 400 == 0):
        return f"{x} is leap year"
    else:
        return f"{x} is not a leap year"

print(leepYear(2001))
```

3. Write a loop that counts how many times the letter a appears in a given string.

```
counter = {}
word = "Spiderman - Great power comes great responsibilities."

for i in word:
    if i not in counter:
        counter[i] = 1
    else:
        counter[i] += 1

print(counter)
```

Collections (Lists, Tuples, Sets, Dicts)

4. Create a dictionary from the following lists:

keys = ['a', 'b', 'c']

values = [100, 200, 300]

```
keys = ["a", "b", "c"]
values = [100, 200, 300]

dictionary = dict(zip(keys, values))
print(dictionary)
```

5. From a list of employee salaries, extract:

The maximum salary

All salaries above average

A sorted version in descending order

```
salary = [50_000, 60_000, 55_000, 70_000, 52_000]

maximumSalary = max(salary)
averageSalary = sum(salary) / len(salary)

aboveAverage = [i for i in salary if i > averageSalary]

sortedSalary = sorted(salary, reverse=True)

print(f"Max Salary: {maximumSalary}")
print(f"Above Average Salary: {aboveAverage}")
print(f"Descending order Salary: {sortedSalary}")
```

6. Create a set from a list and remove duplicates. Show the difference between two

sets:

a = [1, 2, 3, 4]

b = [3, 4, 5, 6]

```
a = [1, 2, 3, 4]
b = [3, 4, 5, 6]

setCombined = set(a + b)
setA = set(a)
setB = set(b)

print(f"Removed dupicates: {setCombined}")
print(f"Difference : {setA.difference(setB)} and
{setB.difference(setA)}")
```

Functions & Classes

7. Write a class Employee with __init__ , display() , and is_high_earner() methods.

An employee is a high earner if salary > 60000.

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

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

    def is_high_earner(self):
        if self.salary > 60_000:
            print(f"{self.name} is high earner")
        else:
```

```
            print(f"{self.name} is not a high earner")

e1 = Employee("Tharun", 45_000)
e1.display()
e1.is_high_earner()
```

8. Create a class Project that inherits from Employee and adds project_name and hours_allocated .

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

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}, Project
Name: {self.project_name}, Hours: {self.hours_allocated}")

p1 = Project("Tharun", 45_000, "Yolo detection", 4.5)
p1.display()
p1.is_high_earner()
```

9. Instantiate 3 employees and print whether they are high earners.

```
e1 = Employee("Tharun", 45_000)
e2 = Employee("Suriya", 89_000)
e3 = Employee("Ajith Kumar", 1_30_000)

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

File Handling

10. Write to a file the names of employees who belong to the 'IT' department.

```
employees = [
    {"name": "Ali", "department": "HR"},
    {"name": "Neha", "department": "IT"},
    {"name": "Ravi", "department": "Finance"},
    {"name": "Sara", "department": "IT"},
    {"name": "Vikram", "department": "HR"}
]

with open(r"IT.txt", "w") as f:
    for i in employees:
```

```
        if i["department"] == "IT":
            f.write(i["name"] + "\n")
```

11. Read from a text file and count the number of words.

```
with open(r" dialogue.txt", "r") as f:
    content = f.read()
    count = len(content.split(" "))

print(f"Number of words: {count}")
```

Exception Handling

12. Write a program that accepts a number from the user and prints the square. Handle the case when input is not a number.

```
try:
    num = int(input("Enter a number: "))

    if not isinstance(num, (int, float)):
        raise ValueError
    print(num ** 2)

except ValueError as e:
    print(f"Error: {e}")
```

13. Handle a potential ZeroDivisionError in a division function.

```
def division(x, y):
    if y == 0:
        raise ZeroDivisionError("Division by zero!!")
    return x / y

try:
    x = int(input("Enter X: "))
    y = int(input("Enter Y: "))
    div = division(x, y)
    print(div)

except ZeroDivisionError as e:
    print(f"Error: {e}")
```

Pandas – Reading & Exploring CSVs

14. Load both employees.csv and projects.csv using Pandas.

```
import pandas as pd

dfEmp = pd.read_csv(r"employees.csv")
dfPro = pd.read_csv(r"projects.csv")
```

15. Display:

First 2 rows of employees

Unique values in the Department column

Average salary by department

```
print(dfEmp.head(2))
print(dfPro.head(2))

print(dfEmp["Department"].value_counts())
print(dfEmp.groupby(by="Department").agg({"Salary": "median"}))
```

16. Add a column TenureInYears = current year - joining year.

```
dfEmp["JoiningDate"] = pd.to_datetime(dfEmp["JoiningDate"])
dfEmp["TenureInYears"] = pd.Timestamp.now().year -
dfEmp["JoiningDate"].dt.year

print(dfEmp)
```

Data Filtering, Aggregation, and Sorting

17. From employees.csv , filter all IT department employees with salary > 60000.

```
deptIT = dfEmp.query("Department == 'IT' & Salary >= 60000")
print(deptIT)
```

18. Group by Department and get:

Count of employees

Total Salary

```
Average SalarydfEmpFiltered = dfEmp.groupby(by="Department").agg(
                              employeeCount = ("Name", "count"),
                              totalSalary = ("Salary", "sum"),
                              averageSalary = ("Salary", "median")
                                                  )
```

```
print(dfEmpFiltered)
```

19. Sort all employees by salary in descending order.

```
print(dfEmp.sort_values(by="Salary", ascending=False))
```

Joins & Merging

20. Merge employees.csv and projects.csv on EmployeeID to show project allocations.

```
merged = dfEmp.merge(dfPro, on="EmployeeID", how="inner")
print(merged[["Name", "Department", "ProjectName",
"HoursAllocated"]])
```

21. List all employees who are not working on any project (left join logic).

```
leftJoin = dfEmp.merge(right=dfPro, on="EmployeeID", how="left")
filteredLeftJoin = leftJoin[leftJoin["ProjectID"].isnull()]
print(filteredLeftJoin)
```

22. Add a derived column TotalCost = HoursAllocated * (Salary / 160) in the merged dataset.

```
merged = dfEmp.merge(dfPro, on="EmployeeID", how="inner")
merged["TotalCost"] = merged["HoursAllocated"] * (merged["Salary"] /
160)

print(merged)
```