MAY 30 PYTHON ASSESSMENT

Python Full-Spectrum Assessment (No Solutions)

Section 1: Python Basics & Control Flow

Q1. Write a Python program to print all odd numbers between 10 and 50.

```
for num in range(11,50):
    if num%2!=0:
        print(num, end=",")
```

Q2. Create a function that returns whether a given year is a leap year.

```
year = int(input("Enter a year: "))
if (year %4==0 and year %100!= 0) or (year %400== 0):
    print("leap year")
else:
    print("not a leap year")
```

Q3. Write a loop that counts how many times the letter

```
a=input("Enter a word: ")
print(a.lower().count("a"))
```

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

Q4. Create a dictionary from the following lists:

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

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

Q5. From a list of employee salaries, extract: The maximum salary, All salaries above average and A sorted version in descending order.

```
salaries = [96000,24525,10000,100000,26000]
max_salary = max(salaries)
avg_salary = sum(salaries) / len(salaries)
above_avg = [s for s in salaries if s > avg_salary]
sorted_salaries = sorted(salaries, reverse=True)

print("Max salary:", max_salary)
print("Salaries above average:", above_avg)
print("Sorted salaries (descending):", sorted_salaries)
```

Q6. 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]
set_a = set(a)
set_b = set(b)
print("Unique elements:", set_a,",", set_b)
print("Difference(a-b):", set_a-set_b)
print("Difference(b-a):", set_b-set_a)
```

Section 3: Functions & Classes

Q7. Write a class Employee with __init__ , display() , and An employee is a high earner if salary > 60000. is_high_earner() methods.

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

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

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

Q8. Create a class Project that inherits from Employee and adds project_name and hours_allocated .

```
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

    def display(self):
        super().display()
        print(f"Project: {self.project_name}, Hours Allocated: {self.hours_allocated}")
```

Q9. Instantiate 3 employees and print whether they are high earners.

```
emp= Project("Ali", "HR", 50000, "AI Chatbot","120")
emp1= Project("Neha", "IT", 60000, "ERP System", 200)
emp2= Project("Sara", "IT", 70000, "Payroll", 150)

for i in [emp, emp1, emp2]:
    i.display()
    if i.is_high_earner():
        print(f"{i.name} is a high earner\n")
    else:
        print(f"{i.name} is a low earner\n")
```

Section 4: File Handling

Q10. Write to a file the names of employees who belong to the 'IT' department.

```
import pandas as pd
df = pd.read_csv('employees.csv')
employees = df[df['Department'] == 'IT']['Name']
with open('emp.txt', 'w') as f:
    for i in employees:
        f.write(i + '\n')
```

Q11. Read from a text file and count the number of words.

```
def count_words(filename):
    with open(filename, 'r') as f:
        content = f.read()
        return len(content.split())
print("Word count:", count_words('it_employees.txt'))
```

Section 5: Exception Handling

Q12. 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 = float(input("Enter a number: "))
    print(f"Square: {num**2}")
except ValueError:
    print("Invalid input. Please enter a number.")
```

Q13. Handle a potential ZeroDivisionError in a division function.

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

print(ZeroDivisionError (2, 0))
```

Section 6: Pandas – Reading & Exploring CSVs

Q14. Load both employees.csv and projects.csv using Pandas.

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

Q15. Display: First 2 rows of employees Unique values in the Department column Average salary by department

```
print("First 2 rows:\n", employees.head(2))
print("\nUnique departments:", employees['Department'].unique())
print("\nAverage salary by department:\n", employees.groupby('Department')['Salary'].mean())
```

Q16. Add a column TenureInYears = current year - joining year.

```
from datetime import datetime

current_year = datetime.now().year
employees['JoiningYear'] = pd.to_datetime(employees['JoiningDate']).dt.year
employees['TenureInYears'] = current_year - employees['JoiningYear']
```

Section 7: Data Filtering, Aggregation, and Sorting

Q17. From employees.csv , filter all IT department employees with salary > 60000.

```
import pandas as pd
df = pd.read_csv('employees.csv')
filtered_it = df[(df['Department'] == 'IT') & (df['Salary'] > 60000)]
print(filtered_it,"\n")
```

Q18. Group by Department and get: Count of employees Total Salary Average Salary

```
grouped = df.groupby('Department').agg(
   Employeecount=('EmployeeID', 'count'),
   Totalsalary=('Salary', 'sum'),
   Averagesalary=('Salary', 'mean'))
print(grouped,"\n")
```

Q19. Sort all employees by salary in descending order.

```
sorted_df = df.sort_values(by='Salary', ascending=False)
print(sorted_df)
```

Section 8: Joins & Merging

Q20. Merge employees.csv and projects.csv on allocations. EmployeeID to show project.

```
emp = pd.read_csv('employees.csv')
pr = pd.read_csv('projects.csv')
merge = pd.merge(emp, pr, on='EmployeeID', how='inner')
print(merge)
```

Q21. List all employees who are not working on any project (left join logic).

```
merged_all = pd.merge(emp, pr, on='EmployeeID', how='left')
no_project = merged_all[merged_all['ProjectName'].isnull()]
print(no_project)
```

Q22. Add a derived column TotalCost = HoursAllocated * (Salary / 160) in the merged dataset.

```
merged_all['TotalCost'] = merged_all['HoursAllocated'] * (merged_all['Salary'] / 160)
print(merged_all[['EmployeeID', 'Name', 'ProjectName', 'HoursAllocated', 'Salary', 'TotalCost']])
```