Advanced Python - Notes

Data Structures

List

- **Definition**: A list is a mutable, ordered collection of items. A Python List can be easily identified by square brackets [ ]. Lists are used to store the data items where each data item is separated by a comma (,). A Python List can have data items of any data type, be it an integer type or a boolean type.

- **List Operations**:
```
fruits = ['apple', 'banana', 'cherry']
```
```
print(fruits[1])             # banana

fruits.append('orange')

print(fruits)                # ['apple', 'banana', 'cherry', 'orange']

more_fruits = ['mango', 'grapes', 'cherry']

fruits.extend(more_fruits)

print(fruits) # ['apple', 'banana', 'cherry', 'orange', 'mango', 'grapes', 'cherry']

print(fruits.count('cherry')) # 2

fruits.insert(1, "Black berry")

print(fruits) # ['apple', 'Black berry', 'banana', 'cherry', 'orange', 'mango', 'grapes', 'cherry']

fruits.remove('cherry')

print(fruits) # ['apple', 'Black berry', 'banana', 'orange', 'mango', 'grapes', 'cherry']

fruits.pop(3)
```

```
print(fruits) # ['apple', 'Black berry', 'banana', 'mango',
'grapes', 'cherry']

fruits.sort()

print(fruits) # ['apple', 'Black berry', 'banana', 'cherry',
'grapes', 'mango']

fruits.reverse()

print(fruits) # ['mango', 'grapes', 'cherry', 'banana', 'Black
berry', 'apple']
```

- **List Comprehension**:
  ```
  squares = [x**2 for x in range(5)]

  print(squares)  # [0, 1, 4, 9, 16]
  ```

Class Activity:
Write a program to create a list of even numbers between 1 to 50 using list comprehension.

Tuple

- **Definition**: Tuples are Ordered, Immutable collections of Data, and are the default data type in Python. Immutable means that they can't be modified and items can't be added or deleted. The data is enclosed in parenthesis ().

- **Code Example**:
  ```
  t = (10, 20, 30, 40, 50)


  print(t[1])  # 20

  print(t[0:3]) # (10, 20, 30)



  t1 = (1, 2, 3)
  ```

```
a,b,c= t1

print(a,b,c) # 1 2 3
```

Class Activity:
Create a tuple with 4 numbers. Unpack them into variables and calculate their sum.

Set

- **Definition**: A set is an unordered collection of unique items. Sets are unordered, mutable collections of data, which can't contain repeated values. They can take different data types, and are enclosed by curly braces {}.

- **Code Example**:

```
s1 = {1, 2, 3}

s2 = {3, 4, 5}

print(s1 | s2)  # Union: {1, 2, 3, 4, 5}

print(s1 & s2)  # Intersection: {3}



Genres= {"Fiction", "NonFiction", "Drama", "Poetry"}

print(Genres) # {'Fiction', 'Poetry', 'NonFiction', 'Drama'}

Genres.add("Folktale")

print(Genres) # {'Fiction', 'NonFiction', 'Poetry', 'Folktale',
'Drama'}

#Adding Multiple Items



Genres.update(['Horror','Distopian','SciFi'])
```

```
print(Genres) # {'Fiction', 'NonFiction', 'Poetry', 'SciFi',
'Folktale', 'Distopian', 'Horror', 'Drama'}



Genres.remove("Poetry")       # .remove() deletes an item, and
raises an Error if it doesn't already exist in the set

print(Genres) # {'Fiction', 'NonFiction', 'SciFi', 'Folktale',
'Distopian', 'Horror', 'Drama'}

Genres.discard("Thriller")    # .discard() deletes an item in a set,
but in case the item doesn't exist, it doesn't return an error

print(Genres) # {'Fiction', 'NonFiction', 'SciFi', 'Folktale',
'Distopian', 'Horror', 'Drama'}
```

Class Activity:
Write a program to input two sets from the user and display common elements.

Dictionary

- **Definition**: A dictionary is a collection of key-value pairs. Dictionaries are unordered, Mutable collections of data, where the data is stored in the form of key:value pairs. The values can be accessed directly if their keys are known, without having to iterate over. The data is enclosed in curly brackets. The values can be mutable and repetitive, but keys have to be unique and immutable.

- **Code Examples**:
```
Employee= {

   'Id':1,

   'Name':'Tom',

   'Age':30,

   'Education':'Masters',
```

```python
   'Department':'IT',

   'Salary':100000

}

print(Employee) # {'Id': 1, 'Name': 'Tom', 'Age': 30, 'Education': 'Masters', 'Department': 'IT', 'Salary': 100000}

print(Employee['Name']) # Tom

#Finding keys using 'IN' Operator

print('Education' in Employee)  #output: True

#Viewing Keys

print(Employee.keys())  #output: dict_keys(['Id', 'Name', 'Age', 'Education', 'Department', 'Salary'])

#Viewing Values

print(Employee.values())   #output: dict_values([1, 'Tom', 30, 'Masters', 'IT', 100000])

Employee['Bonus']='Yes'    # Syntax: dictionary['newkey']='value'

print(Employee) # {'Id': 1, 'Name': 'Tom', 'Age': 30, 'Education': 'Masters', 'Department': 'IT', 'Salary': 100000, 'Bonus': 'Yes'}

del Employee['Bonus']   #del function

print(Employee) # {'Id': 1, 'Name': 'Tom', 'Age': 30, 'Education': 'Masters', 'Department': 'IT', 'Salary': 100000}



Employee.pop('Salary')  # .pop()

print(Employee) # {'Id': 1, 'Name': 'Tom', 'Age': 30, 'Education': 'Masters', 'Department': 'IT'}
```

```
Employee.popitem()       # .popitem() deletes the last element of the
dictionary

print(Employee) # {'Id': 1, 'Name': 'Tom', 'Age': 30, 'Education':
'Masters'}



#output: {'Id': 1, 'Name': 'Tom', 'Age': 30, 'Education': 'Masters'}

Employee.clear()

print(Employee)  #output: {}



del Employee
```

Class Activity:
Create a dictionary for 3 students with names as keys and a nested dictionary of subjects and marks as values.

Lambda Functions

- **Definition**: A lambda function is a small function containing a single expression written in a single line. Lambda functions can also act as anonymous functions where they don't require function name or identifier. These are very helpful when we have to perform small tasks with less code.

```
square = lambda x: x**2
print(square(5))  # 25
```

Class Activity:
Write a lambda function to find the maximum of two numbers.

Map, Filter, Reduce

- **Map**:
  ```
  nums = [1, 2, 3, 4]

  squares = list(map(lambda x: x**2, nums))
  print(squares)  # [1, 4, 9, 16]
  ```

- **Filter**:
  ```
  nums = [1, 2, 3, 4]

  even = list(filter(lambda x: x % 2 == 0, nums))

  print(even)  # [2, 4]
  ```

- **Reduce**:
  ```
  from functools import reduce

  nums = [1, 2, 3, 4, 5]

  sum_all = reduce(lambda x, y: x + y, nums)

  print(sum_all)  # 15
  ```

Class Activity:
Use `map()` to convert a list of temperatures in Celsius to Fahrenheit.

🔹 OOP Basics

- **Definition**: OOPs stands for Object-Oriented Programming. It is a programming paradigm that focuses on the use of objects and classes to create programs. An object is a group of interrelated variables and functions. These variables are often referred to as properties of the object, and functions are referred to as the behavior of the objects. These objects provide a better and clear structure for the program. Main principles of OOPs in Python are abstraction, encapsulation, inheritance, and polymorphism.

Classes & Objects

```
class Car:
    def __init__(self, brand, year):
```

```
        self.brand = brand
        self.year = year

    def show(self):
        print(f"{self.brand} - {self.year}")

c1 = Car("Toyota", 2020)
c1.show()
```

Class Activity:
Create a `Student` class with attributes `name` and `marks`. Add a method to print student details.
