Python Advance Assignment 5

1. Explain super() in the context of inheritance.

Python super()

The super() builtin returns a proxy object (temporary object of the superclass) that allows us to access methods of the base class.

Example

```
class Animal(object):

  def __init__(self, animal_type):

    print('Animal Type:', animal_type)

class Mammal(Animal):

  def __init__(self):

    # call superclass

    super().__init__('Mammal')

    print('Mammals give birth directly')

dog = Mammal()

# Output: Animal Type: Mammal

#         Mammals give birth directly
```

Use of super()

In Python, super() has two major use cases: Allows us to avoid using the base class name explicitly, Working with Multiple Inheritance

Example 1: super() with Single Inheritance

In the case of single inheritance, we use super() to refer to the base class.

```
class Mammal(object):

  def __init__(self, mammalName):

    print(mammalName, 'is a warm-blooded animal.')
```

```
class Dog(Mammal):

  def __init__(self):

    print('Dog has four legs.')

    super().__init__('Dog')

d1 = Dog()

# Output

# Dog has four legs.

# Dog is a warm-blooded animal.
```

Here, we called the __init__() method of the Mammal class (from the Dog class) using code super().__init__('Dog') instead of Mammal.__init__(self, 'Dog')

Since we do not need to specify the name of the base class when we call its members, we can easily change the base class name (if we need to).

```
# changing base class to CanidaeFamily

class Dog(CanidaeFamily):

  def __init__(self):

    print('Dog has four legs.')

    # no need to change this

    super().__init__('Dog')
```

The super() builtin returns a proxy object, a substitute object that can call methods of the base class via delegation. This is called indirection (ability to reference base object with super())

Since the indirection is computed at the runtime, we can use different base classes at different times (if we need to).

Example 2: super() with Multiple Inheritance

```
class Animal:
```

```
  def __init__(self, Animal):
    print(Animal, 'is an animal.');
class Mammal(Animal):
  def __init__(self, mammalName):
    print(mammalName, 'is a warm-blooded animal.')
    super().__init__(mammalName)
class NonWingedMammal(Mammal):
  def __init__(self, NonWingedMammal):
    print(NonWingedMammal, "can't fly.")
    super().__init__(NonWingedMammal)
class NonMarineMammal(Mammal):
  def __init__(self, NonMarineMammal):
    print(NonMarineMammal, "can't swim.")
    super().__init__(NonMarineMammal)
class Dog(NonMarineMammal, NonWingedMammal):
  def __init__(self):
    print('Dog has 4 legs.');
    super().__init__('Dog')
d = Dog()
print('')
bat = NonMarineMammal('Bat')
#Output
#Dog has 4 legs.
#Dog can't swim.
#Dog can't fly.
```

#Dog is a warm-blooded animal.

#Dog is an animal.

#Bat can't swim.

#Bat is a warm-blooded animal.

#Bat is an animal.

2. Describe the file-handling system.

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character.

Working of open() function : Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function open() but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

f = open(filename, mode)

Where the following mode is supported:

r: open an existing file for a read operation.

w: open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.

a: open an existing file for append operation. It won't override existing data.

r+: To read and write data into the file. The previous data in the file will be overridden.

w+: To write and read data. It will override existing data.

a+: To append and read data from the file. It won't override existing data.

Eg. a file named "test", will be opened with the reading mode & print every line one by one in the file.

file = open('test.txt', 'r')

```
for each in file:
    print (each)
```

The open command will open the file in the read mode and the for loop will print each line present in the file.

Working of read() mode: There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use file.read(). The full code would work like this:

```
# Python code to illustrate read() mode
file = open("file.txt", "r")
print(file.read())
```

Another way to read a file is to call a certain number of characters like in the following code the interpreter will read the first five characters of stored data and return it as a string:

```
# Python code to illustrate read() mode character wise
file = open("file.txt", "r")
print (file.read(5))
```

Creating a file using write() mode: Let's see how to create a file and how to write mode works, so in order to manipulate the file,

```
# Python code to create a file
file = open('test.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()
```

The close() command terminates all the resources in use and frees the system of this particular program.

Working of append() mode: Let us see how the append mode works:

```
# Python code to illustrate append() mode

file = open('geek.txt', 'a')

file.write("This will add this line")

file.close()
```

There are also various other commands in file handling that is used to handle various tasks like:

rstrip(): This function strips each line of a file off spaces from the right-hand side.

lstrip(): This function strips each line of a file off spaces from the left-hand side.

It is designed to provide much cleaner syntax and exception handling when you are working with code. That explains why it's good practice to use them with a statement where applicable. This is helpful because using this method any files opened will be closed automatically after one is done, so auto-cleanup.

Eg.
```
# Python code to illustrate with()

with open("file.txt") as file:

    data = file.read()

# do something with data
```

Using write along with the with() function: We can also use the write function along with the with() function:

```
# Python code to illustrate with() alongwith write()

with open("file.txt", "w") as f:

    f.write("Hello World!!!")
```

split() using file handling: We can also split lines using file handling in Python. This splits the variable when space is encountered. You can also split using any characters as we wish. Here is the code:

```
# Python code to illustrate split() function

with open("file.text", "r") as file:
```

```
data = file.readlines()
for line in data:
    word = line.split()
    print (word)
```

3. In Python, explain multiple inheritance.

Multiple inheritance is a feature of object-oriented programming in Python that allows a class to inherit from multiple parent classes. This means that a child class can inherit attributes and methods from more than one parent class, which can be useful in situations where a subclass needs to combine the behavior of two or more classes.

In multiple inheritance, the order of the parent classes specified in the subclass definition is important, as it determines the order in which the parent classes are searched for attributes and methods. This is known as the Method Resolution Order (MRO), which can be accessed using the mro() method.

However, multiple inheritance can also lead to complexities and potential conflicts, such as when two parent classes have methods with the same name. Therefore, it is important to use multiple inheritance judiciously and carefully consider the design of the class hierarchy.

4. Write the MySQL query syntax for INSERT, UPDATE, and DROP.

MySQL create a new table:

```
mysql> CREATE TABLE Books(Id INTEGER PRIMARY KEY, Title VARCHAR(100), Author VARCHAR(60));
```

MySQL INSERT statement:

```
mysql> INSERT INTO Books(Id, Title, Author) VALUES(1, 'War and Peace', 'Leo Tolstoy');
mysql> INSERT INTO Books(Title, Author) VALUES ('The Brothers Karamazov', 'Fyodor Dostoyevsky');
```

Priting table: `mysql> SELECT * FROM Books;`

MySQL UPDATE: The UPDATE statement is used to change the value of columns in selected rows of a table.

```
mysql> UPDATE Books SET Author='Lev Nikolayevich Tolstoy'
    -> WHERE Id=1;
```

MySQL DROP: drops the existing table

```
mysql> DROP TABLE Books;
```

5. Describe MongoDB's features.

Features of MongoDB

1. Support ad hoc queries: In MongoDB, you can search by field, range query and it also supports regular expression searches.

2. Indexing: You can index any field in a document.

3. Replication: MongoDB supports Master Slave replication. A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

4. Duplication of data: MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

5. Load balancing: It has an automatic load balancing configuration because of data placed in shards.

6. Supports map reduce and aggregation tools.

7. Uses JavaScript instead of Procedures.

8. It is a schema-less database written in C++.

9. Provides high performance.

10. Stores files of any size easily without complicating your stack.

11. Easy to administer in the case of failures.

12. It also supports:

JSON data model with dynamic schemas

Auto-sharding for horizontal scalability

Built in replication for high availability