Basic Topics

Operators:

Difference between `is` and `==`.

S.No	`is`	`==`
1.	`is` is a identity operator.	`==` is a relational operator.
2.	`is` compares the memory address of the two operands.	`==` compares the value of the two operands.
3.	For Example `a is b`	For Example `a == b`

Copy:

Difference between deep copy and shallow copy:

S.No	Shallow Copy	Deep Copy
1.	The shallow copy creates the copy of the orginal object but does not create the copy of inner object. Instead it refers the same object address	The deep copy creates the exact copy of the object including the inner object
2.	If we change somthing using copied reference it will affect the orginal object also.	If we change something in the duplicate object it will not affect the orginal object.
3.	The shollow copy can be achived by `slicing` and using `copy()` in copy module.	The deep copy can be achived by `deepcopy()` in copy module

OOPS:

General Parameters:

self:

  self is a parameter of class which points the memory address of the object created for the specific class. self is a mandatory parameter for the object attribute and instance method.

cls:

  cls is a parameter which points the memory address of the class where we can create object for that class. cls is a mandatory parameter for the class attribute and the class method.

Difference between `self` and `cls`:

S.No	self	cls
1.	self is a parameter which points the memory address of the object.	cls is a parameter which points the memory address of the class.
2.	Attribute in a self changes object to object of the same class.	Attribute in cls will not change according to object to object in same class.
3.	self is mandatory parameter for instance attribute and method.	cls is mandatory parameter for the class attribute and method.

Difference between `@staticmethod` and `@classmethod`:

S.No	`@staticmethod`	`@classmethod`
1.	It is a decorator which adds extra functinality to method and makes those methods as static method.	It is a decorator which adds extra functionality to the method to make it as class method.
2.	It does not take any mandatory parameter.	It takes a mandatory parameter `cls`
3.	These methods are independent of class	These methods depends on class
4.	These method will not change anything in class	These method can change the class attributes.

Inheritance:

The process of inherting the property of one class to the another class is known as inheritance. The class which provide the property is known as super or parent class. The class which takes those property is known as sub or child class.

Polymorphism:

The process of making the same methods to behave differently according to the instance created or according to the arguments type or number of arguments is known as polymorphism. They are two types of polymorphism they are:

- Compile time polymorphism
- Runtime Polymorphism

Compile time polymorphism:

The process of creating method with same signature but different argument length or argument data type is known as method overloading. In python method overloading is not possible because two methods cannot have same identifier. But it can be clarrified by overloading the operators. For this we can use default arguments or Variable length arguments.

```python
def func(*args):
    # code
```

Run time polymorphism:

The method which has the same method signature of the parent method and it is compiled during the runtime is known as runtime polymorphism. The best example is overriding. The parent method which is implemented to child method can be overridden and change the implementation of parent class in the child class with same method name. It is possible in python.

```python
class A:
    def method():
        # code

class B:
    def method():
        #code
```

Advance Topic

Decorators:

Decorators are the function which give extra functionality to other function. The decorators can be used by by `@identifier`. To create a decorators we should follow some protocol. They are:

- The decorator function should have a mandatory argument.
- It should have a inner function.
- The decorator should return the inner function address.
- The number of arguments present in inner function should be equal to the function which needs the extra functionality. **For Example:**

```python
def outer(args):
    def inner():
        # write the code
        args()
        # write the code
    return inner


@outer
def func():
    # write a code
```

Control flow in Decorators:

1. Initially python loads the outer function and will not execute that function.
2. At the line of @outer the outer function will be executed.
3. It takes the address of the function as arguments and return the address of inner function.
4. At the args we have the address of func. At the place of func we have inner function address.
5. Whenever we call func we will be executing inner function which adds the extra functionality.

Genrators:

Generators are the iterables like list and tuple unlike they don't store all data at once. It gives the result during the runtime. By using yeild keyword we can convert a normal function into a generators. By using generators we can avoid of storing large dataset instead we can fetch one by one on fly which improves the memory management. For Example:

```
def square(num):
    for i in range(num):
        yield i * i
```

Difference between function and generator:

S.No	Function	Generator
1.	The regular function are the function which will be executed once whenever it is called.	Generators are the function which iters the value in fly.
2.	It returns whole collection data type at once.	It iters the value one by one and return it
3.	It occupies more memory	It occupies very less memory
4.	return keyword is used to written the function whereever it is called.	yield will not return the control flow fully to program instead it iter the value one by one.