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`

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.

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:**

```
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

S.No	Function	Generator
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.