

Operators:

Difference between `is` and `==`.

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

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	<code>self</code>	<code>cls</code>
1.	<code>self</code> is a parameter which points the memory address of the object.	<code>cls</code> is a parameter which points the memory address of the class.
2.	Attribute in a <code>self</code> changes object to object of the same class.	Attribute in <code>cls</code> will not change according to object to object in same class.
3.	<code>self</code> is mandatory parameter for instance attribute and method.	<code>cls</code> is mandatory parameter for the class attribute and method.

Decorators:

Decorators are the function which give extra functionality to other function. The decorators can be used 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.