You have already learned about methods and how they are just functions that belong to a class. They define the behavior that an object of the class can perform. Special operators are specific symbols or keywords that are built-in and provide special behavior when used with certain data types or objects. In your class, you can define methods to implement or override the standard behavior of Python operators, thus creating methods as special operators.

In this reading you will learn about the different types of special operators, how to override the standard operators and embed them in your code, and see examples along the way.

## ****Different types of special operators****

Python supports a variety of different operators that you can use in your code to make life easier for you. Some of the more common operators are:

* Arithmetic operators. These include **+** (addition), **-** (subtraction), **\*** (multiplication), **/** (division), and **\*\*** (exponentiation).
* Comparison operators. These include **==** (equality), **!=** (inequality), **<** (less than), and  **>=** (greater than or equal to)
* Logical operators. These include **and**, **or**, and **not**.
* Assignment operators. These include **=** (simple assignment), **+=** (addition assignment), and  **%=** (modulo assignment)

**Note:** This is not an all-inclusive list, but different examples of common operators that you would use in Python.

## ****Performing special operations****

Every special operator has a corresponding **dunder method** that implements the operation. In Python, you denote a dunder method by placing double underscores at the beginning and end of the name; in fact, the term “dunder” comes from this use of **d**ouble **unders**cores. You can change how an operator behaves with an instance of your object by overriding the implementation. Let’s look at an example:

**class Triangle:**

**def \_\_init\_\_(self, base, height):**

**self.base = base**

**self.height = height**

In this example, the **Triangle** class has a method **\_\_init\_\_()**which is called a constructor and is used to initialize the object’s attributes.

**def area(self):**

**return 0.5 \* self.base \* self.height**

This part of the code, area(self) method, computes the area of the triangle based on its height and base length.

**def \_\_add\_\_(self, other):**

**return self.area() + other.area()**

This method overrides the + operator to "add" two triangles together.

**triangle1 = Triangle(10, 5)**

**triangle2 = Triangle(6, 8)**

**print("The area of triangle 1 is", triangle1.area())**

**print("The area of triangle 2 is", triangle2.area())**

**print("The area of both triangles is", triangle1 + triangle2)**

The output of this problem is:

**The area of triangle 1 is 25.0**

**The area of triangle 2 is 24.0**

**The area of both triangles is 49.0**

For a full list of operators and the method names you can use to override their behavior, view this resource:

* [Mapping operators to functions](https://docs.python.org/3/library/operator.html#mapping-operators-to-functions)

## ****Key takeaways****

Python allows you to override or implement standard operations in your code to make your code cleaner for yourself and others to read. Being able to override certain behaviors allows you to control the output of your code and provides flexibility in how you write code.