# Module Class

## What is it

- A class the performs a series of related operations
- Most Python Packages that you use are module classes as they perform specific operations
- Allow us to carry operations between different parts of our code.

## Implementation

- Methods are the most important part of module classes
- Attributes are used to store data that is commonly used between the different methods
- Must be stateless
    - Any one method call should not affect the next method call
    - Every call should give me the same result (depending on the parameters I pass and initialization of the object)
- Should only take in parameters and return results
    - No print statements
    - No input functions

## Uses

- Allows us to write code once, and use it in different places
    - We can use the same code within a single codebase
    - We can take the module and us it in different projects

## Benefits

- When using abstraction, allows use to change the module we are working with without haveing to change any of the implementation code.
- Allows for easier code reuse

# Creating a module class

- Methods are the most important thing
- Attributes store commonly referenced information
- in most cases, attributes should not be changed during the life time of the object.

1. Basic Method class

*We are creating a class that reads and writes from a file, most basic form would be to just create the methods and let users pass the path they are working with *

```
# Basic file handling Module
class FileHandler():

    def write(self, path: str, data: str):
        with open(path, 'w') as f:
            f.write(data)

    def read(self, path: str):
        output = None

        try:
            with open(path, 'r') as f:
                output = f.read()
        except FileExistsError:
            # Log error
            ...

        return output

# Initialize class
file = FileHandler()

# Write to file
file.write("my_file", "Hello World")

# Read from file
output = file.read("my_file")

print(output)
```

```
 Hello World
```

2. If for all usecases, the application will be used the same file, we can create an attribute for the file path.

*We have made the attribute private so that it does not get changed between method calls*

```
# Basic file handling Module
class FileHandler():

    def __init__(self, path: str):
        self.__path = path

    def write(self, data: str):
        with open(self.__path, 'w') as f:
            f.write(data)

    def read(self):
        output = None

        try:
            with open(self.__path, 'r') as f:
                output = f.read()
        except FileExistsError:
            # Log error
            ...

        return output

# Initialize the file object
file = FileHandler('my_file.txt')

# Write to the file without having to pass the file name
file.write("Hello there")

# Read from file
output = file.read()
print(output)
```

```
 Hello there
```

3. Using private methods to perform operations that the user doesn't need to know about

*We will be instead of handling the error, we will check if the file exists before reading from the file*

```
import os

class FileHandler():

    def __init__(self, path: str):
        self.__path = path

        # Create the file if it does not exist
        if (not self.__file_exists()):
            self.write("")

    def write(self, data: str):
        with open(self.__path, 'w') as f:
            f.write(data)
```