1.9 Module

A module is a importable python file and can be created by creating a file with extension as `.py`

We can import the objects present in the module.

In the below 👇 example, we are importing `hello` function from `greet` module (greet.py)

`greet.py`

```
"""Module to greet the user"""

import getpass


def hello():
    username: str = getpass.getuser().capitalize()
    print(f"Hello {username}. Have a great day :)")


if __name__ == "__main__":
    hello()
```

In [1]:
```
from greet import hello
```

In [2]:
```
hello()
```

```
Hello Root. Have a great day :)
```

let's have a look at the greet.py module. Well, we see the below `if` condition.

```
if __name__ == "__main__":
    hello()
```

But why do we we need to have it🤔? We can just call the `hello` function at the end as

```
hello()
```

Let's see the below👇 code to know why we use the first approach rather than the second.🙃

In [3]:
```
import greet
```

🔍 The above code doesn't greet you 😢

In [4]:
```
%run ./greet.py
```

```
Hello Root. Have a great day :)
```

But, this above code greets you😎.

The reason for this is, in the first snippet, we are importing a module called `greet`, so the actual code we are executing is in this REPL or Ipython shell.

Coming to second snippet, we are executing the `greet.py` directly.

Value of `__name__` would be "__main__" if we are executing a Python module directly. If we import a module(using the module indirectly) then value of `__name__` would be the relative path of the imported module. In the first example the `__name__` in the greet module would be "greet". As the "greet" is not equal to "__main__", that's the reason, we never went to the `if` condition when we imported greet module. 🙂