if __name__ == '__main__':

Modules: When we create a file with Python statements and function definitions and save it with a .py extension (the default) we have created a module. This is what we have been doing when we write our programs in this class.

Sometimes we want to run the module as a stand-alone program (i.e. a "main program") and at other times we would like to "import" the module just to have access to some of the functions defined therein.

However, when we import a module not only do the functions get defined, but any executable statements, e.g. assignments, input/output, loops, etc. also get run. This might not be the behavior that we want. The if __name__ == '__main__': idiom helps us deal with this issue.

What is __name__?

This built in variable holds the name of the python script that is currently running.

As above, a Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized. (from geeksforgeeks.org)

We have used modules, for example when we "imported" the math functions: from math import sqrt.

Important: When we import a module, Python runs all the code there!

What does **if __name__ == '__main__':** mean and what is it used for?

It is used to control which code is executed when modules are imported.

How?

If a module is run as a top-level script the value of __name__ is '__main__', but if it's "imported" then the value of __name__ is the name of the imported module.

Example

Say we create a module of math functions that we would like to use both as a stand-alone script as well as a file to include in order to access some of its functions.

Here is the module:

```
# my_math.py

def add(x,y):
    return x+y

def mult(x,y):
    return x*y
print('This is the my_math module')
print('The value of __name__ is: ',__name__)
print(12345)
print('mmm', __name__=='__main__')
'''
if __name__== '__main__':
    print('This is the my_math module')
    print('The value of __name__ is: ',__name__)
    print(12345)
    print('mmm', __name__=='__main__')
'''
```

And here is a program that imports the above file to access some of its functions:

```
# test the import behavior of my_math.py
import my_math

print('In name tester, run at the top level __name__ is: ', __name__)
print(my_math.add(6,5))
```

If we run my_math.py directly (at the "top level") we get the following output:

```
This is the my_math module
The value of __name__ is:  __main__
12345
mmm True
>>>
```

And when we call it from the test program we get:

```
This is the my_math module
The value of __name__ is:  my_math
12345
mmm False
In name tester, run at the top level __name__ is:  __main__
11
```

which is not what we want; we only need access to the functions in my_math, we don't need all the other code there to run.

But, if we change my_math to use the **if __name__ == '__main__':** idiom:

```
# my_math.py
def add(x,y):
    return x+y
def mult(x,y):
    return x*y

if __name__== '__main__':
    print('This is the my_math module')
    print('The value of __name__ is: ',__name__)
    print(12345)
    print('mmm', __name__=='__main__')
```

we get:

```
In name tester, run at the top level __name__ is:  __main__
11
>>>
```

Another example – try this one:

1. `library.py`: This file contains a function and a conditional statement using `if __name__ == "__main__":`.

```
# library.py

def print_hello():
    print("Hello from library.py!")

if __name__ == "__main__":
    print("library.py is being run directly")
    print_hello()
else:
    print("library.py has been imported")
```

In this file, the function `print_hello` is defined to print a message. The `if __name__ == "__main__":` block checks if `library.py` is being run as the main program. If it is, it prints a message indicating this and calls `print_hello()`. If `library.py` is imported into another script, the `else` block executes, printing "library.py has been imported".

2. `main.py`: This file imports `library.py` and uses its `print_hello` function.

```
# main.py

import library

def main():
    print("This is main.py")
    library.print_hello()

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

Here, `main.py` imports `library.py` and calls the `print_hello` function from it. Because `library.py` is imported, the code under `if __name__ == "__main__":` in `library.py` does not execute, but the `else` part does, indicating that `library.py` has been imported.

When to Use `if __name__ == "__main__":`

1. Running Scripts Directly: Use this to ensure that certain code runs only when the script is executed directly (e.g., for tests or standalone applications).

2. Importing as a Module: Prevents certain code from executing when the script is imported as a module in another script.

3. Code Organization: Helps in separating executable code from definitions, making scripts clean and readable.

Let's run the above:
Running `python library.py` will output:

```
library.py is being run directly
Hello from library.py!
```

Running `python main.py` will output:

```
library.py has been imported
This is main.py
Hello from library.py!
```