Photo by Caspar Camille Rubin on Unsplash

Understanding if `__name__ == "__main__"` in Python

Dunder variable __name__ behind the scenes!!

Rachit Tayal Follow

Nov 21, 2019 · 3 min read ★

Using the *if __name__ == "__main__"* statement is quite regular in Python files. However, it may seem confusing at times. This aim of this article is to uncover the behaviour of the statement and further discusses on where to use it. So let's get started!

A Python module is a file that has a `.py` extension. All we need to do is create a file that contains legitimate Python code and give the file a name with `.py` extension. A module can be imported to other modules or run directly via command line.

The *__name__* is a special built-in variable which evaluates to the name of the current module. However, if a module is being run directly (from command line), then *__name__* instead is set to the string "*__main__*".

Let's put together this little code example for understanding. Suppose we create two modules, *foo* and *bar* with the following code:

```
# foo.py

import bar
print("foo.__name__ set to ", __name__)
```

And the *bar* module:

```
# bar.py

print("bar.__name__ set to ", __name__)
```

On invoking the *bar* module from command line, its *__name__* attribute will be set to "*__main__*":

```
python bar.py
>>>
bar.__name__ set to __main__
```

However, on invoking the *foo* module in a similar fashion, the bar's *__name__* attribute will be set equivalent to it's module name i.e *bar*:

```
python foo.py
>>>
bar.__name__ set to bar
foo.__name__ set to __main__
```

Therefore, we can test whether our module is being run directly or being imported using the following check:

```
if __name__ == "__main__":
    ...
```

In this way, modules can look at their own *__name__* value to determine for themselves how they are being used, whether as support for another program or as the main application executed from the command line.

When a module is being imported into another module, its various function and class definitions will be imported and its top-level code will be executed. To illustrate this, let's consider the following two modules:

```
# module person.py

def creds():
    name = "Sarah"
    age = 26
    print("person details {}, {}".format(name, age))

print("top-level in person module")

if __name__ == "__main__":
    print("person mod is run directly")
else:
    print("person mod is imported into another module")

# module utility.py
import person
```

```
person.creds()

if __name__ == "__main__":
    print("utility mod is run directly")
else:
    print("utility mod is imported into another module")
```

On invoking the *person* module directly, *__name__* attribute will be set to *__main__*, hence we'll see the following output:

```
python person.py
>>>
top-level in person module
person mod is run directly
```

Whereas, when *utility.py* module is executed directly, *__name__* attribute for *person* module will be set to "*person*" itself, while *__name__* of the *utility* module will be set to *__main__*. Therefore, we'll get the following output:

```
python utility.py
>>>
top-level in person module
person mod is imported into another module
person details Sarah, 26
utility mod is run directly
```

Why it's designed like this?

We might naturally wonder why it's designed the way it is. Well, sometimes we want to write a `.py` file that can be both used by other programs and/or modules as a module, and can also be run as the main program itself.

This behaviour comes in handy for quick developing and testing our code. It also helps in debugging since it allows us to have a script mode where we can run unit tests directly.

Besides this, it's elegant that running a module in Python directly is just setting up a single variable.

Conclusions:

- Every module in Python has a special attribute called *__name__*. The value of *__name__* attribute is set to *"__main__"* when module is run as main program. Otherwise, the value of *__name__* is set to contain the name of the module.
- We use *if* ***__name__ == "__main__"*** block to prevent (certain) code from being run when the module is imported.

Python Programming