Mighty Modules

October 13, 2021

1 Mighty Modules

1.1 Learning objectives

- What is a module?
- How do you import a module?
- How do you call a function in a module that you have imported?
- What is the difference between running a program as a script versus using a program as a module?
- How do you execute statements only when a module is run as a script (and *not* when it is used as a module)

2 What is a module?

A file that contains a collection of related functions

You can write your own modules (this week!) or use modules other people have written (next week!)

You wrote `caesar_cipher.py`:

```python
def encrypt(text, shift):
    encrypted = ''
    for character in text:
        encrypted += chr(ord(character) + shift)
    return encrypted

def decrypt(text, shift):
    return encrypt(text, -shift)
```

Some other people wrote `math` (`mathmodule.c`)

3 How do you import a module?

Use an `import` statement: `import <NAME-OF-MODULE>`

[1]:
```python
import caesar_cipher

print(type(caesar_cipher))
```

```
<class 'module'>
```

[2]:
```python
import math

print(type(math))
```

```
<class 'module'>
```

4 How do you call a function in a module that you have imported?

Use dot notation–specify the name of the module and the name of the function, separated by a dot: `<NAME-OF-MODULE>.<NAME-OF-FUNCTION>`

[3]:
```python
import caesar_cipher

print(caesar_cipher.encrypt('cheer', 7))
print(caesar_cipher.decrypt('hal', -1))
```

```
jolly
ibm
```

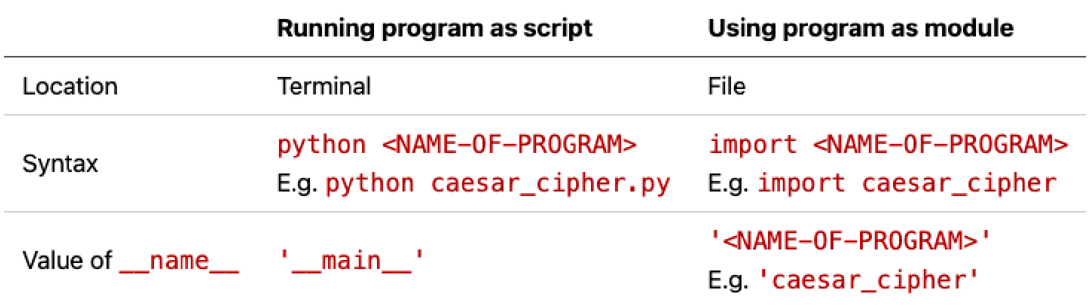
[4]:
```python
import math

print(math.sqrt(64))
print(math.factorial(3))
```

```
8.0
6
```

5 What is the difference between running a program as a script versus using a program as a module?

1. Location–Where does this happen?
2. Syntax–What do I need to type to make this happen?
3. Value of `__name__`–What is the value of `__name__` within the program when this happens?



	Running program as script	**Using program as module**
Location	Terminal	File
Syntax	`python <NAME-OF-PROGRAM>` E.g. `python caesar_cipher.py`	`import <NAME-OF-PROGRAM>` E.g. `import caesar_cipher`
Value of `__name__`	`'__main__'`	`'<NAME-OF-PROGRAM>'` E.g. `'caesar_cipher'`

```
caesar_cipher.py
```

```
def encrypt(text, shift):
    encrypted = ''
    for character in text:
        encrypted += chr(ord(character) + shift)
    return encrypted

def decrypt(text, shift):
    return encrypt(text, -shift)

# Print value of __name__
print(f'__name__: {__name__}')
# Test encrypt and decrypt
print(encrypt('cheer', 7))
print(decrypt('hal', -1))
```

Running `caesar_cipher.py` as script in terminal:

```
$ python caesar_cipher.py
__name__: __main__
jolly
ibm
```

Using `caesar_cipher.py` as module in `secret_message.py`:

```
import caesar_cipher

secret_message = 'Hevr0$}sy$ger$vieh$qi$rs{%'
shift = 4

print(caesar_cipher.decrypt(secret_message, shift))
```

Running `secret_message.py` as script in terminal:

```
$ python secret_message.py
__name__: caesar_cipher
jolly
ibm
Darn, you can read me now!
```

But… we only had one print statement in `secret_message.py`!

`caesar_cipher.py`

```
...

# Print value of __name__
print(f'__name__: {__name__}')
# Test encrypt and decrypt
print(encrypt('cheer', 7))
print(decrypt('hal', -1))
```

What if we don't want to see

```
__name__: caesar_cipher
jolly
ibm
```

when we use `caesar_cipher` as a module?

6 How do you execute statements only when a module is run as a script (and *not* when it is used as a module)?

Think about the value of `__name__` within `caesar_cipher.py` when it is run as a script versus used as a module.

What statement could you add to `caesae_cipher.py` to run the print statements only when `__name__` has a certain value?

`caesar_cipher.py`

```python
...

if __name__ == '__main__':
    # Print value of __name__
    print(f'__name__: {__name__}')
    # Test encrypt and decrypt
    print(encrypt('cheer', 7))
    print(decrypt('hal', -1))
```

Running `caesar_cipher.py` as script in terminal:

```
$ python caesar_cipher.py
__name__: __main__
jolly
ibm
```

Using `caesar_cipher.py` as module in `secret_message.py`:

```python
import caesar_cipher

secret_message = 'Hevr0$}sy$ger$vieh$qi$rs{%'
shift = 4

print(caesar_cipher.decrypt(secret_message, shift))
```

Running `secret_message.py` as script in terminal:

```
$ python secret_message.py
Darn, you can read me now!
```