Modules, Packages and Classes

module namespace

- simpleclass_module.__dict__.keys( )
- simpleclass_module.__dict__['__name__']
- simpleclass_module.__dict__['__doc__']
- explore the other keys of the module
- notice the attributes of the module are listed towards the end
- explore the usage of the module attributes
- notice the documentation of the module, function, class, etc.
- try changing module and reloading using reload function of imp package

Using the module attributes and class attributes

- simpleclass_module.
- simpleclass_module.A_CONSTANT
- sc1 = simpleclass_module.SimpleClass()
- sc1.classAttribute1   (can be accessed through both instance and class objects)
- sc1.__class__
- sc1.method1( )
- SimpleClass.classAttribute   <— this will give an error, why?
- simpleclass_module.SimpleClass.classAttribute1
- simpleclass_module.__dict__.keys()
- simpleclass_module.SimpleClass.__dict__.keys()

Simplest of classes

- Class attributes
- Class methods
- Class and method documentation
- Class and instance object creation
- Class method and instance method (instance method has an extra first parameter typically named 'self', strictly speaking there is nothing like a class method as always an object is passed as the first param)

Accessing the methods on class and instance objects

- simpleclass_module.SimpleClass.method3(10)
- sc3 = simpleclass_module.SimpleClass()
- sc3.method3(10)   <— will give an error
- sc3.method3( )

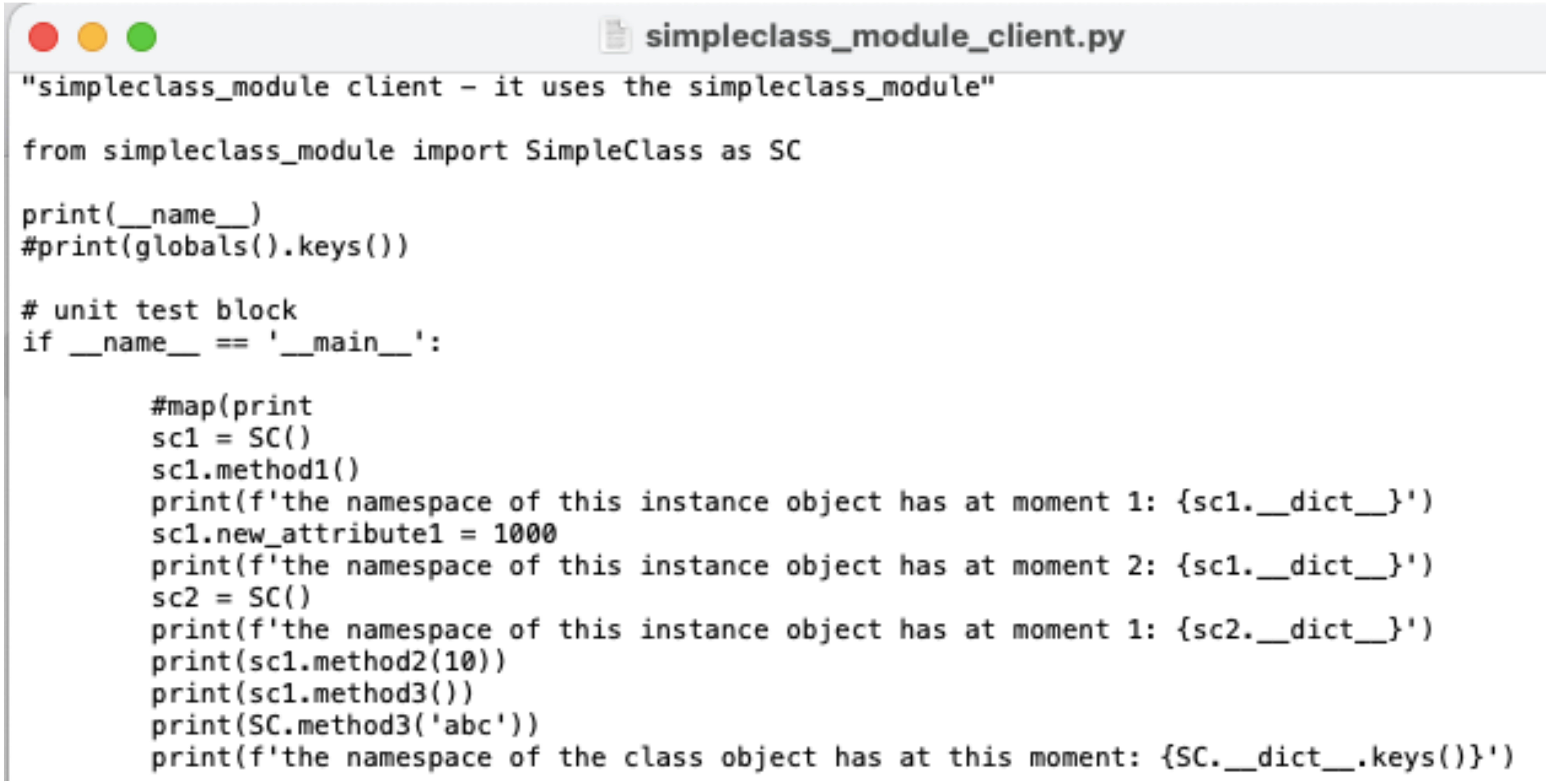
Using the module just created

- simpleclass_module.py
- simpleclas_module_client.py

Attributes can be added arbitrarily to class and instance objects as well as to modules as they are just namespaces

- import simpleclass_module
- simpleclass_module.SimpleClass.newAttribute2 = 'abc'
- simpleclass_module.SimpleClass.__dict__.keys()
- simpleclass_module.SimpleClass.newAttribute2
- sc1 = simpleclass_module.SimpleClass()
- sc1.newAttribute2
- sc1.newAttribute3 = 'completely new!'
- sc1.__dict__
- sc1.newAttribute3
- f1 = lambda x: str.upper(x)
- simpleclass_module.SimpleClass.newMethod = f1
- simpleclass_module.SimpleClass.__dict__.keys()
- simpleclass_module.SimpleClass.newMethod('king')
- sc1.newMethod( )   <- the new class method is available to the old object but it will given an error due to type mismatch which is as expected
- simpleclass_module.newModuleFunction = lambda x: x[:3]
- simpleclass_module.newModuleFunction('john')

More example of namespace concept



simpleclass_module_client.py

```python
"simpleclass_module client - it uses the simpleclass_module"

from simpleclass_module import SimpleClass as SC

print(__name__)
#print(globals().keys())

# unit test block
if __name__ == '__main__':

        #map(print
        sc1 = SC()
        sc1.method1()
        print(f'the namespace of this instance object has at moment 1: {sc1.__dict__}')
        sc1.new_attribute1 = 1000
        print(f'the namespace of this instance object has at moment 2: {sc1.__dict__}')
        sc2 = SC()
        print(f'the namespace of this instance object has at moment 1: {sc2.__dict__}')
        print(sc1.method2(10))
        print(sc1.method3())
        print(SC.method3('abc'))
        print(f'the namespace of the class object has at this moment: {SC.__dict__.keys()}')
```

Important

- Modules, classes and objects are just namespaces and you can add anything to them post facto. But such adhoc additions live only during the duration of the usage (process cycle).

- You can verify the namespace through the __dict__ attribute as it stores the contents of the particular namespace

- The anything are known as "attributes" and they can be variables, functions and classes.

- Functions are called as methods in case of classes.

- Classes differ from modules in terms of advanced OO features such as inheritance, constructor mechanisms, etc.

- Modules are really based on the software engineering concept of modularization of code.

Advanced classes

- More dunder methods such as __iter__, __next__, __str__ methods as well as other user defined methods
- Inheritance mechanism