Lecture 6 Class and Modules

A possibly overlooked point: Modules and Class in Python share many similaries at the basic level. They both define some names (attributes) and functions (methods) for the convenience of users -- and the codes to call them are also similar. Of course, Class also serves as the blue prints to generate instances, and supports more advanced functions such as Inheritance.

Class and Instance

Simple Example of Vector

Let's first define the simplest class in Python

```
In [ ]:  class VectorV0:
             '''The simplest class in python'''  # this is the document string

             pass
```

and create two instances `v1` and `v2`

```
In [ ]:  v1 = VectorV0()   # note the parentheses here
         print(id(v1))
         v2 = VectorV0()
         id(v2)
```

Now `v1` and `v2` are the objects in Python

```
In [ ]:  type(v1)
```

```
In [ ]:  dir(v1)
```

We can manually assign the attributes to instance `v1` and `v2`

```
In [ ]:  v1.x = 1.0
         v1.y = 2.0
         v2.x = 2.0
         v2.y = 3.0
```

```
In [ ]:  dir(v1)
```

We don't want to create the instance or define the coordinates seperately. Can we do these in one step, when initializing the instance?

```
In [ ]:  class VectorV1:
             '''define the vector'''  # this is the document string
             dim = 2    # this is the attribute in class
             def __init__(self, x=0.0, y=0.0):  # any method in Class requires the first parameter to be self!
                 self.x = x
                 self.y = y
```

```
In [ ]:  v1 = VectorV1(1.0,2.0)
```

```
In [ ]: dir(v1)
```

```
In [ ]: print(v1.dim)
        print(v1.x)
        print(v1.y)
```

Btw, there is nothing mysterious about the `__init__` : you can just assume it is a function (method) stored in v1, and you can always call it if you like!

When you write `v1.__init__()` , you can equivalently think that you are calling a function with "ugly function name" `__init__` , and the parameter is `v1` (self), i.e. you are writing `__init__(v1)` . It is just a function updating the attributes of instance objects!

More generally, for the method `method(self, params)` you can call it by `self.method(params)` .

```
In [ ]: print(v1.x)
        print(id(v1))
        y = v1.__init__()
        print(v1.x)
        print(id(v1))
        print(y)
```

`v1` is just like a mutable object, and the "function" `__init__( )` just change `v1` in place!

Now we move on to update our vector class by defining more functions. Since you may not like ugly names here with dunder, let's just begin with normal function names.

```
In [ ]: class VectorV2:
            '''define the vector'''   # this is the document string
            dim = 2    # this is the attribute

            def __init__(self, x=0.0, y=0.0):   # any method in Class requires the first param
        eter to be self!
                '''initialize the vector by providing x and y coordinate'''
                self.x = x
                self.y = y

            def norm(self):
                '''calculate the norm of vector'''
                return math.sqrt(self.x**2+self.y**2)

            def vector_sum(self, other):
                '''calculate the vector sum of two vectors'''
                return VectorV2(self.x + other.x, self.y + other.y)

            def show_coordinate(self):
                '''display the coordinates of the vector'''
                return 'Vector(%r, %r)' % (self.x, self.y)
```

```
In [ ]: help(VectorV2)
```

```
In [ ]: import math
        v1 = VectorV2(1.0,2.0)
        v2 = VectorV2(2.0,3.0)
```

```
In [ ]: v1.norm()
```

```
In [ ]: v3 = v1.vector_sum(v2)
        v3.show_coordinate()
```

```
In [ ]: v1+v2 # will it work?
```

```
In [ ]: print(v3)
```

Something that we are still not satisfied:

- By typing v3 or using `print()` in the code, we cannot show its coordinates directly
- We cannot use the `+` operator to calculate the vector sum

Special (Magic) Methods

Here's the magic: by merely changing the function name, we can realize our goal!

```
In [ ]: class VectorV3:
            '''define the vector'''  # this is the document string
            dim = 2   # this is the attribute

            def __init__(self, x=0.0, y=0.0):  # any method in Class requires the first param
        eter to be self!
                '''initialize the vector by providing x and y coordinate'''
                self.x = x
                self.y = y

            def norm(self):
                '''calculate the norm of vector'''
                return math.sqrt(self.x**2+self.y**2)

            def __add__ (self, other):
                '''calculate the vector sum of two vectors'''
                return VectorV3(self.x + other.x, self.y + other.y)

            def __repr__(self):    #special method of string representation
                '''display the coordinates of the vector'''
                return 'Vector(%r, %r)' % (self.x, self.y)
```

```
In [ ]: help(VectorV3)
```

```
In [ ]: v1 = VectorV3(1.0,2.0)
        v2 = VectorV3(2.0,3.0)
```

```
In [ ]: v3 = v1.__add__(v2) # just call special methods as ordinary methods
        v3.__repr__()
```

```
In [ ]: v1 +v2 # here is the point of using special methods!
```

```
In [ ]: v3
```

Special methods are just like VIP admissions to take full use of the built-in operators in Python. With other special methods, you can even get elements by index `v3[0]`, or iterate through the object you created. For more advanced usage, you can see here (https://rszalski.github.io/magicmethods/).

Inheritance

Now we want to add another scalar production method to Vector, but we're tired of rewriting all the other methods. A good way is to create new Class VectorV4 (Child Class) by inheriting from VectorV3 (Parent Class) that we have already defined.

```
In [ ]:  class VectorV4(VectorV3): # Note the class VectorV3 in parentheses here
             '''define the vector'''  # this is the document string
             def __mul__(self, scalar):
                 '''calculate the scalar product'''
                 return VectorV4(self.x * scalar, self.y * scalar)
```

```
In [ ]:  help(VectorV4)
```

```
In [ ]:  v1 = VectorV4(1.0,2.0)
         v2 = VectorV4(2.0,3.0)
```

```
In [ ]:  v1+v2
```

```
In [ ]:  v1*2
```

Modules and Packages

In Python, Functions (plus Classes, Variables) are contained in Modules, and Modules are organized in directories of Packages. In fact, Modules are also objects in Python!

Now we have the `Vector.py` file in the folder. When we import the module, the interpreter will create a name `Vector` pointing to the module object. The functions/classes/variables defined in the module can be called with `Vector.xxx`.

Of course, the (annoying) rules of object assignment (be careful about changing mutable objects even in modules) in Python still applies, but we won't go deep in this course.

```
In [ ]:  import Vector
         print(type(Vector))
         dir(Vector) # 'attributes' (namespace) in the module Vector -- note the variables/fun
         ctions we have defined above are here!
```

```
In [ ]:  Vector.string
```

```
In [ ]:  Vector.print_hello()
```

```
In [ ]:  v5 = Vector.VectorV5(1.0,2.0)
         v5
```

Other different ways to import module:

```
In [ ]:  import Vector as vc # create a name vc point to the module Vecotr.py -- good practic
         e, all the functions will start with vc. -- you know where they are from!
         vc.string
```

```
In [ ]:  from Vector import print_hello # may cause some name conflicts if write larger progra
         ms
         print_hello() # where does this print_hello come from ? it may take some time to figu
         re out...
```

```
In [ ]:  from Vector import * # bad choice! import everything -- may cause serious name confli
         cts
         string
```

To import the modules, you must ensure that they are in your system paths.

```
In [ ]:  import sys
         sys.path
```

```
In [ ]:  sys.modules.keys()  # check all the modules are currently imported in the kernel
```

We can import the `inspect` package and use `getsource` method to see the source codes of imported modules. Note that this does not work for [built-in functions (https://github.com/python/cpython)](https://github.com/python/cpython).

```
In [ ]:  import inspect
         lines = inspect.getsource(Vector.VectorV5)
         print(lines)
```

If we are interested in `numpy` -- in fact `numpy` is a package rather than modules. Package can contain many submodules -- for example, the module of [linalg (https://github.com/numpy/numpy/blob/master/numpy/linalg/linalg.py)](https://github.com/numpy/numpy/blob/master/numpy/linalg/linalg.py).

```
In [ ]:  import numpy as np
         [name for name in sys.modules.keys() if name.startswith('numpy')]  # check what module
         s in numpy package has been imported
```

```
In [ ]:  print(np)
         dir(np)  # actually has the functions in np.core
```

```
In [ ]:  print(inspect.getsource(np.sum))  # let's see the source code of sum function
```

```
In [ ]:  'eig' in dir(np)  # where is the eigen value/vector function?
```

```
In [ ]:  np.eig  # Won't work! Because eig is not defined in numpy (core) module!
```

```
In [ ]:  print(np.linalg)
         dir(np.linalg)  # let's check the names in linalg module
```

```
In [ ]:  help(np.linalg.eig)  # eig function is here!
```

You can view all the source code of [numpy (https://github.com/numpy/numpy)](https://github.com/numpy/numpy) on Github. Pay attention to the folder structures.

```
In [ ]:  from numpy import linalg  # another way to import linalg module from numpy package
         linalg  # now we create a name linalg to point to the linalg.py module
```

Beyond Basic Python: What's next?

- Knowledge and wisdom
- What we have not covered in basic python: other data types (dictionary, set, tuple), input/output, exceptions, -- consult [a byte of python (https://python.swaroopch.com/)](https://python.swaroopch.com/), or [programiz (https://www.programiz.com/python-programming)](https://www.programiz.com/python-programming)
- The systematic book ([for example,Python Cookbook (https://www.oreilly.com/library/view/python-cookbook-3rd/9781449357337/)](https://www.oreilly.com/library/view/python-cookbook-3rd/9781449357337/)) or course in computer science department (ICS-31,33)
- Practice!Practice!Practive! Useful websites such as [Leetcode (https://leetcode.com/)](https://leetcode.com/)