

Create a module

```
NA = 6.02e-23 # module-level constant
class Class_1():
    """
    #do cst rings, output of help(module)
    # id PEP257 chapitre 15
    """

    def __init__(self, function_1, function_2, data):
        self.function_1 = function_1
        self.function_2 = function_2
        self.data = xxx

    def function_1(self):
        blabla
        return var

    def function_2(self):
        print(data)

    def update(self, add_var):
        #Modifying an Attribute's Value Through a Method
        self.data += add_var
class Class_2(Class_1): # inherit from Animal
    """
    comment for the class2
    blabla
    """
    PI = 3.14 # constant with uppercase letters
    # invariable during program execution
    def __init__(self, make, model, year):
        super(Class_1, self).__init__(function_1, function_2, data)
        # call the superclass constructor
        blabla
    def function_3(self):
        """ doc strings under method """
        blabla
    def function_4(self):
        blabla
```

The `__init__()` method takes in these parameters and stores them in the attributes that will be associated with instances made from this class.

import a module

```
import module as alias
# import a entire module and give it an alias
from module import class
# from a module import a class
# it should prepare at first a file __init__.py
# at the directory
from module.class import method
# from a class of module import a method
(function)
```

init.py

An `init.py` file is a special file that is used to indicate that a directory contains a Python package. A Python package is a collection of modules that can be imported and used together. An `init.py` file can also contain code that is executed when the package or a module in the package is imported. This can be useful for initializing package-level data or performing other tasks.

Before Python 3.3, an `init.py` file was required for every package directory. Otherwise, Python would not recognize the directory as a package and would not be able to import modules from it. This was done to prevent directories with common names, such as `string`, from unintentionally hiding valid modules that occur later on the module search path.

However, since Python 3.3, an `init.py` file is no longer mandatory for defining a package. Python can also recognize namespace packages, which are packages that do not have an `init.py` file and can span multiple directories. Namespace packages allow multiple portions of a package to be distributed and installed separately, and then merged together at runtime.

create an environment path

```
Linux & MacOS in file bash (~.bashrc)
export PYTHONPATH=$PYTHONPATH:/chemin/vers/module
then
echo $PYTHONPATH
source ~/.bashrc
Windows in shell PowerShell
$env:PYTHONPATH += ";C:\chemin\vers\module\"
then
echo $env:PYTHONPATH
```

