MODULES and PACKAGES

softserve

Difference between Module, Package, Library, Framework

A module is a file containing Python definitions and statements.

The file name is the module name with the suffix **.py** appended.

Within a module, the module's name (as a string) is available as the value of the global variable __name__. Modules have private symbol tables.

Package is namespace which contains multiple package/modules.

Library is collection of various packages. There is no difference between package and python library conceptually.

Framework is a collection of various libraries which architects the code flow.

Difference between Module, Package, Library, Framework

A module is a file containing Python definitions and statements.

The file name is the module name with the suffix **.py** appended.

Within a module, the module's name (as a string) is available as the value of the global variable __name__. Modules have private symbol tables.

Package is namespace which contains multiple package/modules.

Library is collection of various packages. There is no difference between package

and python library conceptually.

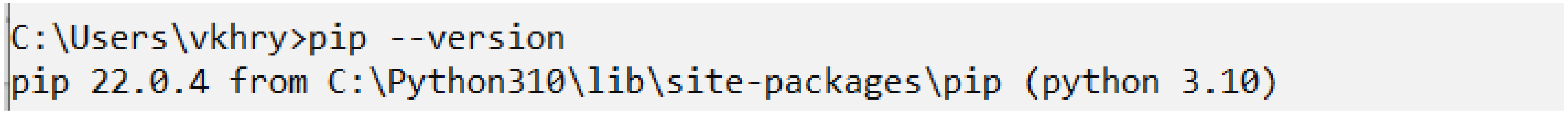
Framework is a collection of various libraries which architects the code flow.

What is PIP?

PIP is a package manager for Python packages, or modules if you like.

Note: ***If you have Python version 3.4 or later, PIP is included by default.***

Check if PIP is Installed



```
C:\Users\vkhry>pip --version
pip 22.0.4 from C:\Python310\lib\site-packages\pip (python 3.10)
```

Install PIP

If you do not have PIP installed, you can download and install it from this page:

https://pypi.org/project/pip/

softserve

Import modules

The import statement:

```
>>> import <module_name_1>[,<module_name_2>, …, <module_name_N>]
```

The statement import <module_name> only places <module_name> in the caller’s symbol table. The objects that are defined in the module remain in the module’s private symbol table.

An alternate form of the import statement allows individual objects from the module to be imported directly into the caller’s symbol table:

```
>>> from <module_name> import <name(s)>
```

Note: because this form of import places the object names directly into the caller’s symbol table, any objects that already exist with the same name will be overwritten.

Import modules

It is even possible to indiscriminately import everything from a module at one fell swoop:

```
>>> from <module_name> import *
```

This will place the names of all objects from <module_name> into the local symbol table, with the exception of any that begin with the underscore (_) character.

This **isn’t recommended** in large-scale production code. It’s a bit dangerous because you are entering names into the local symbol table en masse.

It is also possible to import individual objects but enter them into the local symbol table with alternate names:

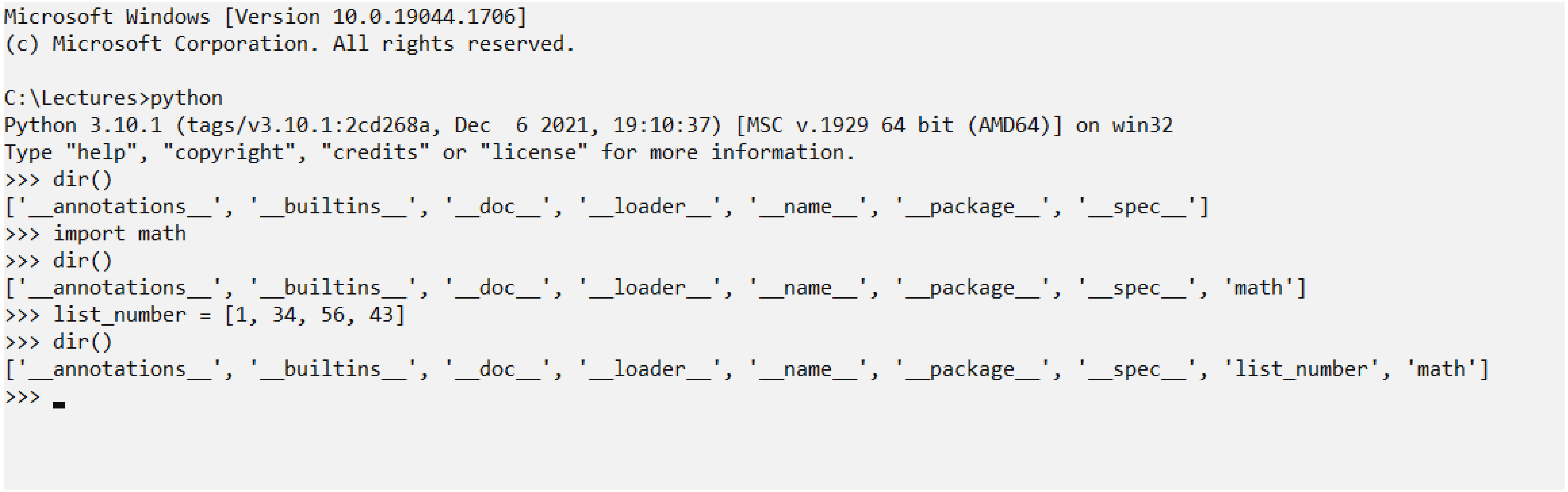
```
>>> from <module_name> import <name1> as <alt_name1>[, <name2> as <alt_name2>, …]
```

This makes it possible to place names directly into the local symbol table but avoid conflicts with previously existing names:

softserve

The dir() function

The built-in function dir() returns a list of defined names in a namespace. Without arguments, it produces an alphabetically sorted list of names in the current local symbol table:



```
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Lectures>python
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
>>> import math
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'math']
>>> list_number = [1, 34, 56, 43]
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'list_number', 'math']
>>> _
```

Modules

Create module fibo.py :

```
# Fibonacci numbers module fibo.py

def fib(n):          # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):          # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Import module fibo

```
>>> import fibo
```

Using the module name you can access the functions

```
>>> fibo.fib(1000)
 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
 [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
 'fibo'
```

softserve

Modules

If you intend to use a function often:

```
>>> fib = fibo.fib
>>> fib(500) 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import fib, fib2
>>> fib(500)
 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import *
>>> fib(500)
 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> import fibo as f
>>> f.fib(500)
 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
# Fibonacci numbers module fibo.py

def fib(n):          # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):         # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

softserve

Executing modules as scripts

When a **.py** file is imported as a module, Python sets the special variable **__name__** to the name of the module. However, if a file is run as a **standalone script**, **__name__** is set to the string **'__main__'**. Using this fact, you can discern which is the case at run-time and alter behavior accordingly:

```
def fib2(n):          # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

if (__name__ == '__main__'):
    print('Executing as standalone script')
```

softserve

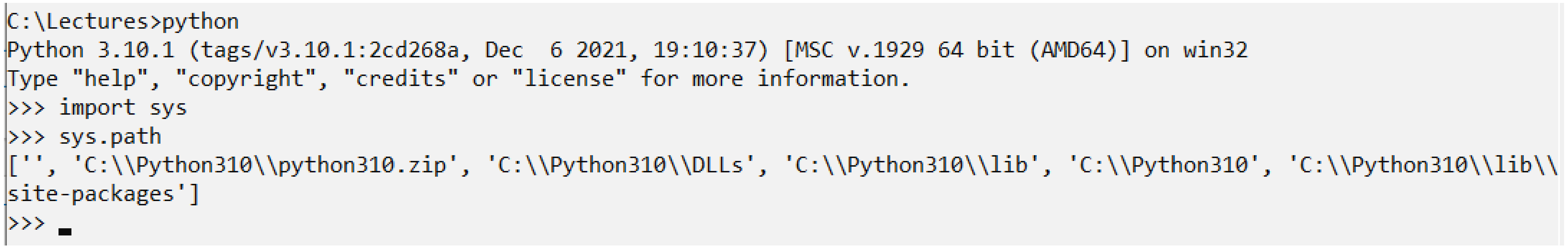
Module search path

current directory

list of directories specified in PYTHONPATH environment variable

uses installation-default if not defined, e.g., .:/usr/local/lib/python

uses sys.path



```
C:\Lectures>python
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', 'C:\\Python310\\python310.zip', 'C:\\Python310\\DLLs', 'C:\\Python310\\lib', 'C:\\Python310', 'C:\\Python310\\lib\\
site-packages']
>>> _
```

softserve

Structure of packages

Package is namespace which contains multiple package/modules. Each package in Python is a directory which **may contain** a special file called **__init__.py**. This file can be empty (Since Python 3.3, though, a folder without an __init__.py can be considered part of an implicit namespace package), and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

```
Customer/        # top level package
    __init__.py

    Salary/      # first subpackage
        __init__.py
        info.py
        exchange.py
        transactions.py

    Personal/   # second subpackage
        __init__.py
        position.py
        expirience.py
```

```
>>> import Customer.Personal.position
>>> Customer.Personal.position.personalposition()

>>> from Customer.Personal import position
>>> position.personalposition()

>>> from Customer.Salary import info
>>> info.print_info()

>>> from Customer.Salary.info import print_info
>>> print_info()

>>> from Customer.Salary.info import *
>>> print_info()
```

softserve

Python Packages

- Packages are a way of structuring Python’s module namespace by using “dotted module names”
- When importing the package, Python searches through the directories on sys.path looking for the package subdirectory.
- To import module C you can use:

  import A.B.C and use the fully qualified name **print(A.B.C.my_func())**

- To import module or function from the package use following:

  from A.B import C and use C only by its name (without package prefix) **print(C.my_func())**

Using packages in Python

```
fincalc /
   __init__.py
   simper.py
   compper.py
   annuity.py
```

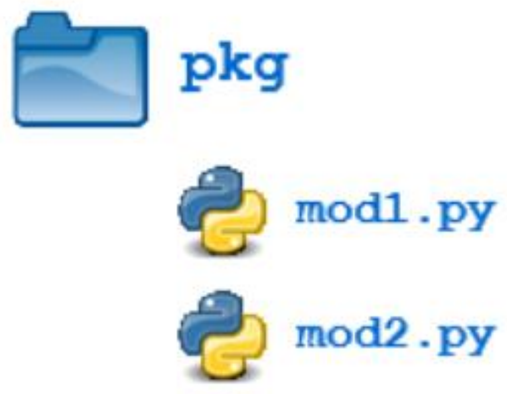
Module **__init.py__** may be empty, or may contain a variable **__all__**

```
__all__ = ["simper", "compper", "annuity"]
```

softserve

Packages

Packages allow for a hierarchical structuring of the module namespace using dot notation. In the same way that modules help avoid collisions between global variable names, packages help avoid collisions between module names.



mod1.py

```
def print_info():
    print(“[mod1] print_info()”)

class Inform:
    pass
```

mod2.py

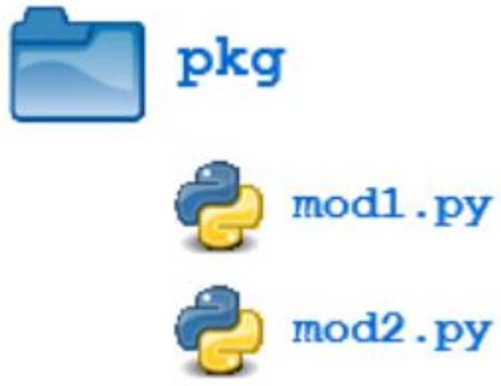
```
def user_info():
    print(“[mod2] about_info()”)

class User:
    pass
```

softserve

Packages

Given this structure, if the pkg directory resides in a location where it can be found (in one of the directories contained in sys.path), you can refer to the two modules with dot notation (pkg.mod1, pkg.mod2) and import them with the syntax



```
import  <module_name_1>, [<module_name_2>]
```

```
>>> import pkg.mod1, pkg.mod2
>>> pkg.mod1.print_info() #[mod1] print_info()
>>> x = pkg.mod2.user_info()
```

softserve

TO
THE
FUTURE
softserve