Modules and Packages

UBCO Master of Data Science – DATA 533

Today’s Class

Python modules and packages

- Python OOP (L1-2)
- **Modules and Packages (L3)**
- Collaborative version control (L4)
- Testing, Error and Exception and CI (L5-7)
- Publishing packages (L8)

Design an application
in collaboration

Making importable modules and packages

- Using the `import` statement
- Install other people's packages and modules

Recap: Object-Oriented Programming

Every piece of data in Python is an ***object***.

We make our own types that can also give rise to instances (objects).

Classes are organized in a hierarchy.

Derived (child) classes inherit attributes from their base (parent) class.

Example

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Point(3, 4)
print('Distance: ',p.distance_from_origin())
```

#3**2=3^2=9

#4**2=4^2=16

#25**0.5=5

Namespace and the . Operator

Name (also called identifier) is simply a name given to objects

- When using `a = 10`, `10` is an object stored in memory and `a` is the name we associate it with

A ***namespace*** is a collection of names.

The `.` is used to access the namespace of an object.

Each instance of an object defines a new namespace.

Every object in Python has an attribute denoted by `__dict__`.

This contains all the attributes defined for the object itself.

Example

```
class Person:
    def __init__(self, name, age):
        self.name = name  # (default) public attribute
        self.age = age    # (default) public attribute

p1 = Person('Alex', 10)
p2 = Person('Adam', 20)

print(p1.__dict__)
print(p2.__dict__)
print(Person.__dict__)
```

Example

```
print(p1.__dict__)
```

- `{'name': 'Alex', 'age': 10}`

```
print(p2.__dict__)
```

- `{'name': 'Adam', 'age': 20}`

```
print(Person.__dict__)
```

-
  ```
  {'__module__': '__main__', '__init__': <function Person.__init__ at
  0x0000015D71AE8730>, '__dict__': <attribute '__dict__' of 'Person'
  objects>, '__weakref__': <attribute '__weakref__' of 'Person'
  objects>, '__doc__': None}
  ```

```
p1.__dict__['name'] = 'Khalad'

print(p1.name)
```

Output: Khalad

The . Operator Question

Question: What is the output of the following program?

```
class Company:
    def __init__(self, name, location):
        self.name = name
        self.location = location

comp = Company('TimeTrex','Vancouver')
comp.location = 'Kelowna'
print(comp.__dict__)
```

A) {'TimeTrex', ‘Vancouver’}

B) {'TimeTrex', 'Kelowna’}

C) {'name': 'TimeTrex', 'location': ‘Vancouver’}

D) {'name': 'TimeTrex', 'location': 'Kelowna’}

E) The program has an error

Modular Programming

Breaking a large programming task into separate, smaller, more manageable subtasks or module

Modules can then used together like building blocks to create a larger application.

Advantages:

- Simplicity: focuses on one relatively small portion of the problem
- Maintainability: less interdependent modules are easy to maintain
- Reusability: Functionality defined in a single module can be easily reused

Python Modules

A ***module*** is a file containing Python definitions and statements to perform a specific task.

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

Modules are imported using the **`import`** command:

```
import modulename
```

Module names should be all lower case

Module Example

```
# File myperson.py
class Person:
    def __init__(self, name, age):
        self.name = name  # (default) public attribute
        self.age = age    # (default) public attribute

    def display(self):
        print("Name:", self.name, "Age:",self.age)
```

To find the current directory you are in:

```
import os          #useful operating system functions
print(os.getcwd()) #getcwd() returns current working directory
```

Module Example

In Jupyter Notebook:

```
import myperson


p1 = myperson.Person('Alex',10)
p1.display()
```

```
Output:

Name: Alex Age: 10
```

Another Example

```
# File myfunc.py
def double(n):
    return n*2

def triple(n):
    return n*3
```

```
# In Jupyter Notebook
import myfunc

print(myfunc.double(10))
print(myfunc.triple(10))
```

```
Output:

20
30
```

Note: `dir()` used to define which names a module defines. Command: `dir(myfunc)`

```
Output: […,'double', 'triple']
```

Recall: DATA541 Interactive Visualization

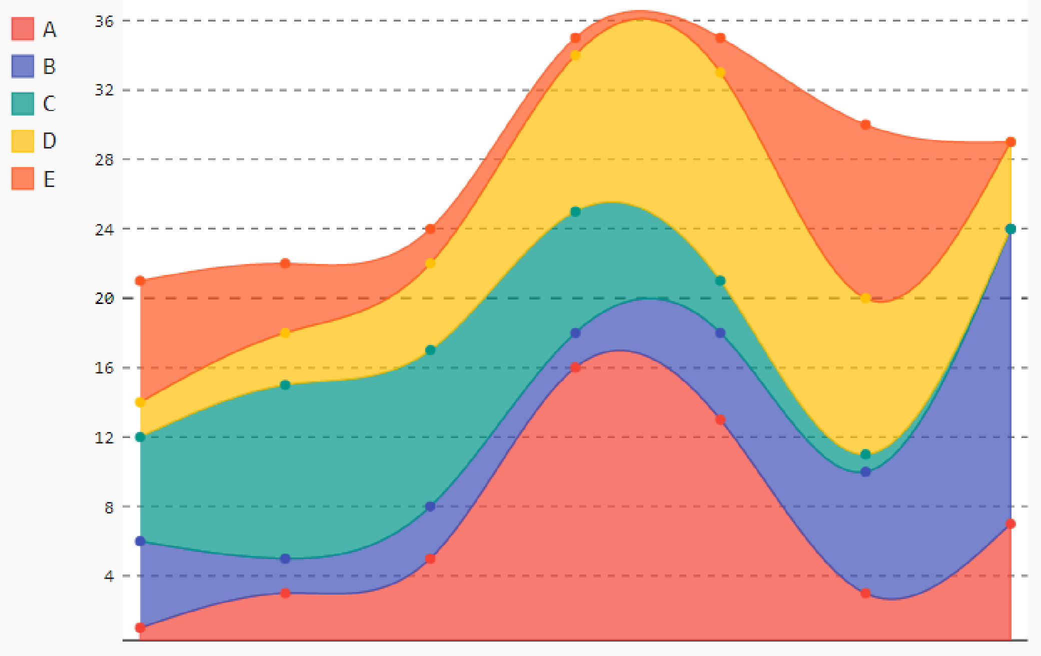
Pygal

- http://www.pygal.org/

Bokeh

- https://bokeh.pydata.org/

```
import pygal
```



Useful Python Modules

Useful modules: https://docs.python.org/3/py-modindex.html

`math` — Mathematical functions

```
# import standard math module
import math

# use math.pi to get value of pi
print("The value of pi is", math.pi)

#Return the sine of x radians
math.sin(x)

#Return x raised to the power y (2^3=8)
math.pow(2, 3)
```

Python import with Renaming

Define an alias for an imported module

```
import myfunc as fn

print(fn.double(10))

print(fn.triple(10))
```

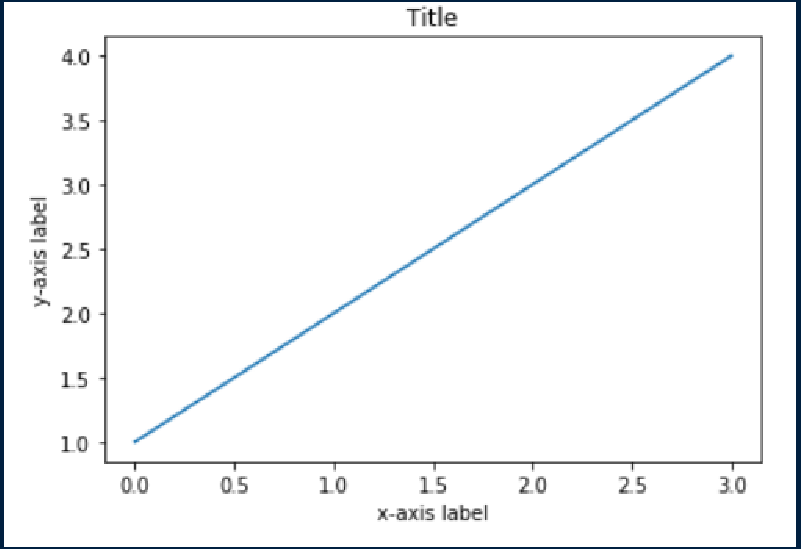
You can import multiple modules in one line

```
import os, myplot
```

pyplot Example

Creating a chart with pyplot.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.title('Title')
plt.xlabel('x-axis label')
plt.ylabel('y-axis label')
plt.show()
```



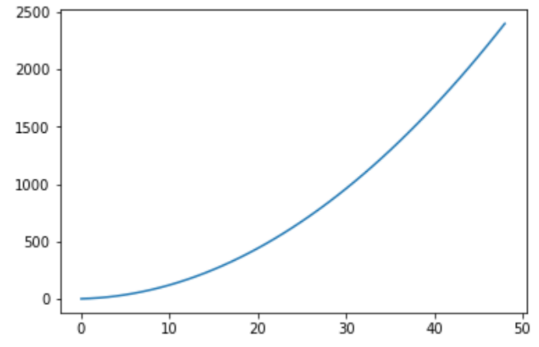
myplot.py Example

Let's make a module to take care of steps of plotting (**myplot.py**).

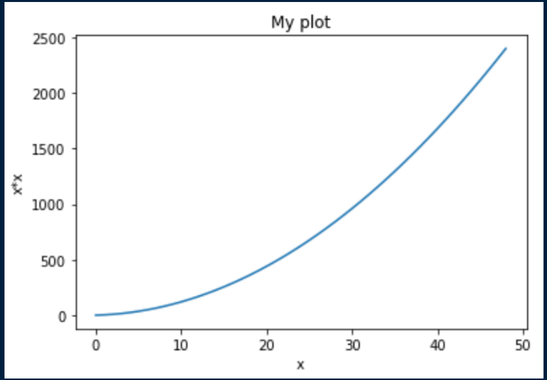
```
import matplotlib.pyplot as plt

def lineplot(y, title=None, xlabel=None, ylabel=None):
    plt.plot(y)
    if xlabel:
        plt.xlabel(xlabel)
    if ylabel:
        plt.ylabel(ylabel)
    if title:
        plt.title(title)
    plt.show()

print("Inside myplot.py")
```

myplot.py Example

```
import myplot

myplot.lineplot([x*x for x in range(1, 50)])
```



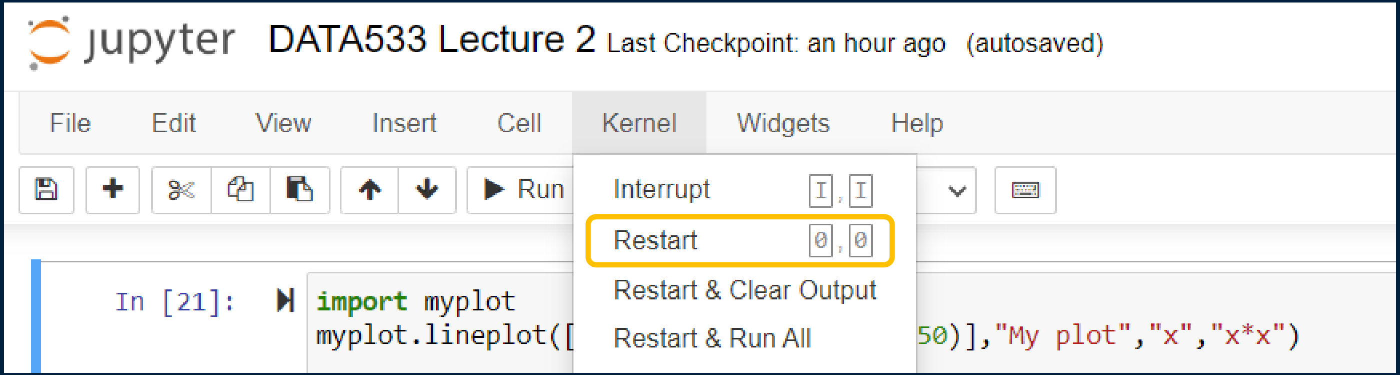
```
import myplot

myplot.lineplot([x*x for x in range(1,
50)],"My plot","x","x*x")
```



Technical Note

In Jupyter notebook, if you make a change in a module (i.e., .py file) that you are importing, you have to restart the kernel.

Otherwise, you will not see the change in the notebook.



```
import myplot
myplot.lineplot([ … 50)],"My plot","x","x*x")
```

The `import` Statement: Behind The Scenes

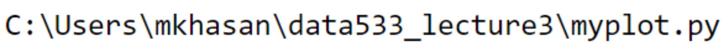
1. Python looks for a module in your current directory first.
2. If it does not find the module, it looks in directories contained in the PYTHONPATH environment variable
3. An installation-dependent list of directories configured at the time Python is installed

You can get the other folder list using the following commands:

```
import sys
print(sys.path)
```

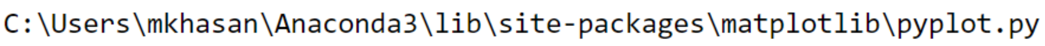
To see the module location:

```
print(myplot.__file__)
```



```
C:\Users\mkhasan\data533_lecture3\myplot.py
```

```
print(plt.__file__)
```



```
C:\Users\mkhasan\Anaconda3\lib\site-packages\matplotlib\pyplot.py
```

Module Question

Question: How many of the following statements are TRUE?

1) Module names should be all lower case by convension.

2) A module can have many methods/functions.

3) Modules are imported using the **`import`** command

4) A module can import another module.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Module Question

Question: What is the output of the following program?

#hierarchy.py file

```
class Parent:
    def __init__(self, param):
        self.v1 = param

class Child(Parent):
    def __init__(self, param):
        Parent.__init__(self,param)
        self.v2 = param
```

# In Jupyter Notebook

```
import hierarchy

obj = hierarchy.Child(5)
print(obj.v1," ",obj.v2)
```

A) None None **B)** None 5 **C)** 5 None **D)** 5 5 **E)** Error is generated by program

Try it: Creating Module

Question: Create a module called `mathfunc` that includes functions to return values from addition `(add)`, subtraction `(sub)`, multiplication `(mul)` and division `(div)` of two numbers.

Now write a python program that imports the `mathfunc` module and calls the functions to perform addition, subtraction, multiplication and division operations.

Sample test code

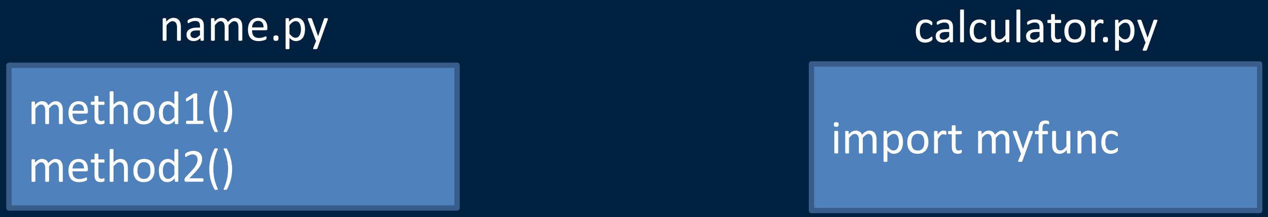
```
print(mathfunc.add(20,10))
print(mathfunc.sub(20,10))
print(mathfunc.mul(20,10))
print(mathfunc.div(20,10))
```

`__name__` Attribute

Before executing code, Python interpreter reads source file and assign few special variables/global variables.

If the python interpreter is running that module (the source file) as the main program, it sets the special `__name__` variable to have a value `__main__`.

If this file is being imported from another module, `__name__` will be set to the module's name.



name.py

method1()
method2()

calculator.py

import myfunc

__name__ Attribute Example

```
#name.py
if __name__ == "__main__":
    print("Executed directly")
else:
    print("Executed when imported")
```

```
Output:

Executed directly
```

```
# In Jupyter Notebook
import name
```

```
Output:

Executed when imported
```

Useful Tips on Importing

You can import specific definitions from a module using the `from` statement.

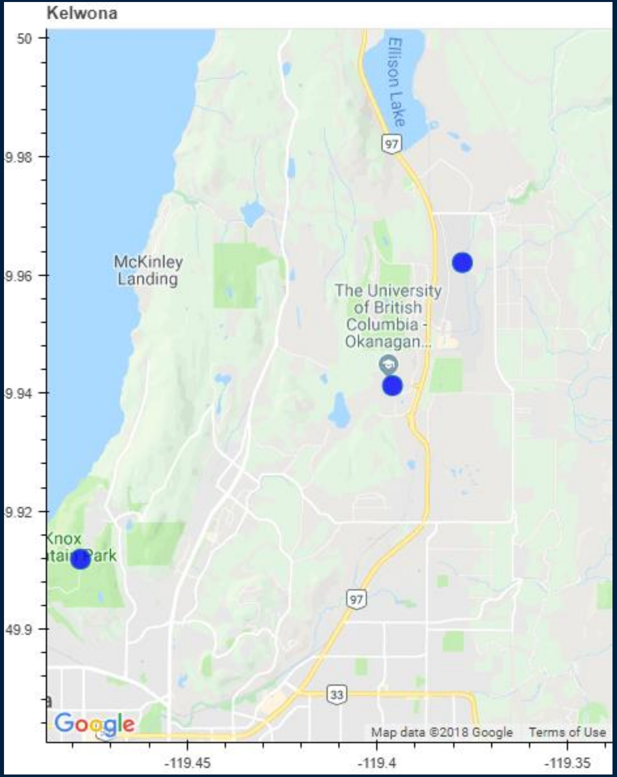
```
from myfunc import double
print(double(10))
print(triple(10))
Output:
```

You can also import all definitions from a module into the current namespace using the * character.

```
from myfunc import *
```

Recall: DATA541 Interactive Visualization

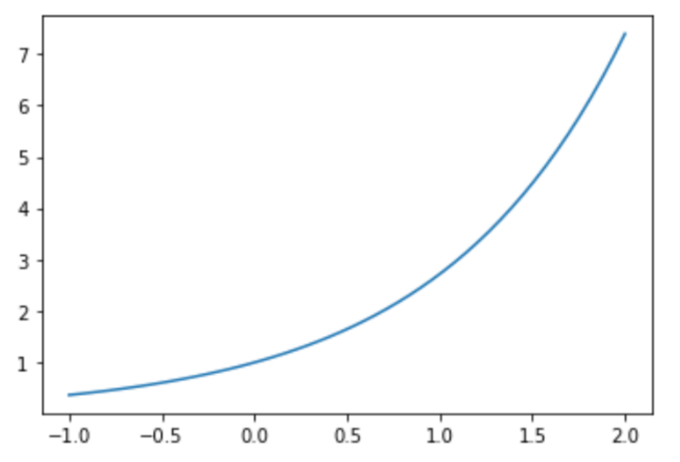
```
from bokeh.plotting import gmap
```



Local vs. Higher-Level Namespace

Local namespace always takes priority over higher level namespaces.

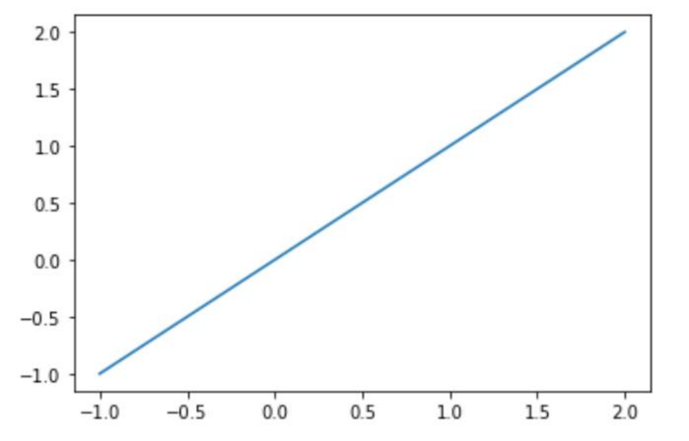
```
from numpy import exp, linspace
import matplotlib.pyplot as plt


x = linspace(-1, 2, 50)
y = exp(x)
plt.figure()
plt.plot(x, y)
plt.show()
```



Local vs. Higher-Level Namespace

As we define our work `exp()` function in the local/current module, the local module’s function will be used.

```
from numpy import exp, linspace
import matplotlib.pyplot as plt
def exp(x):
    return x
x = linspace(-1, 2, 50)
y = exp(x)
plt.figure()
plt.plot(x, y)
plt.show()
```



Style Tips

Import statements are typically placed in the following order:

1. Standard python modules (e.g., `os`)
2. Third party modules (e.g., `matplotlib`)
3. Your own modules. Separated by a line break.

This is not mandatory, but it is a good style

```
import os
import random
import matplotlib

import myplot
```

Python Packages

A Python ***package*** is a collection of modules.

Packages are used to group modules that perform some similar functions.

A module in a package is referenced by `package.module`

Packages may have sub packages.

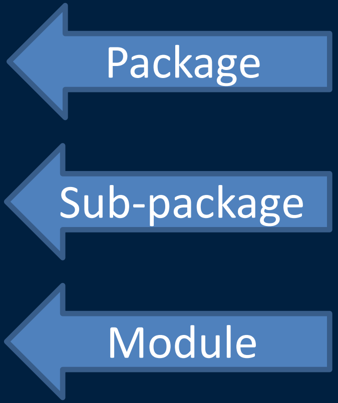
```
 import package.subpackage.modulename
```

Package names should be all lower case

File Structure

File structure for plotting 1D, 2D, and 3D plots.

```
myplotlib
    __init__.py
    plot1D
        __init__.py
        line.py
    plot2D
        __init__.py
        scatter.py
    plot3D
        __init__.py
        scatter3d.py
```



Package

Sub-package

Module

File Structure

Python recognizes packages by looking for the `__init__.py` file inside a folder.

The corresponding folder name is the package/sub-package name.

Typically the `__init__.py` file is left blank

However, `__init__.py` can execute initialization code for a package

Source: https://docs.python.org/3/tutorial/modules.html

Car Package Example

```
/cars
    | - __init__.py file
    | - audicars.py
    | - nissancars.py
```

Car Package Example

```
# audicars.py file
class Audi:

    def __init__(self):
        self.models = ['a6', 'a8', 'a3']

    def display(self):
        print('Audi car models:')
        for model in self.models:
            print('%s ' % model)
```

Car Package Example

```
# nissancars.py file
class Nissan:
    def __init__(self):
        self.models = ['sentra', 'altima', 'rogue']

    def display(self):
        print('Nissan car models:')
        for model in self.models:
            print('%s ' % model)

# Empty __init__.py file
```

Car Package Example

```
from cars import audicars
from cars import nissancars

a1 = audicars.Audi()
a1.display()

n1 = nissancars.Nissan()
n1.display()
```

Output:

```
Audi car models: a6 a8 a3
Nissan car models: sentra altima rogue
```

Another Example

```
/package
   |-subpackage1
   |   |-__init__.py
   |   |-src
   |   |   |-__init__.py
   |   |   |-functions.py
   |-subpackage2
   |   |-__init__.py
   |   |-src
   |   |   |-__init__.py
   |   |   |-mathfunctions.py
```

```python
def add(a,b):
    return a+b


def subtract(a,b):
    return a-b
```

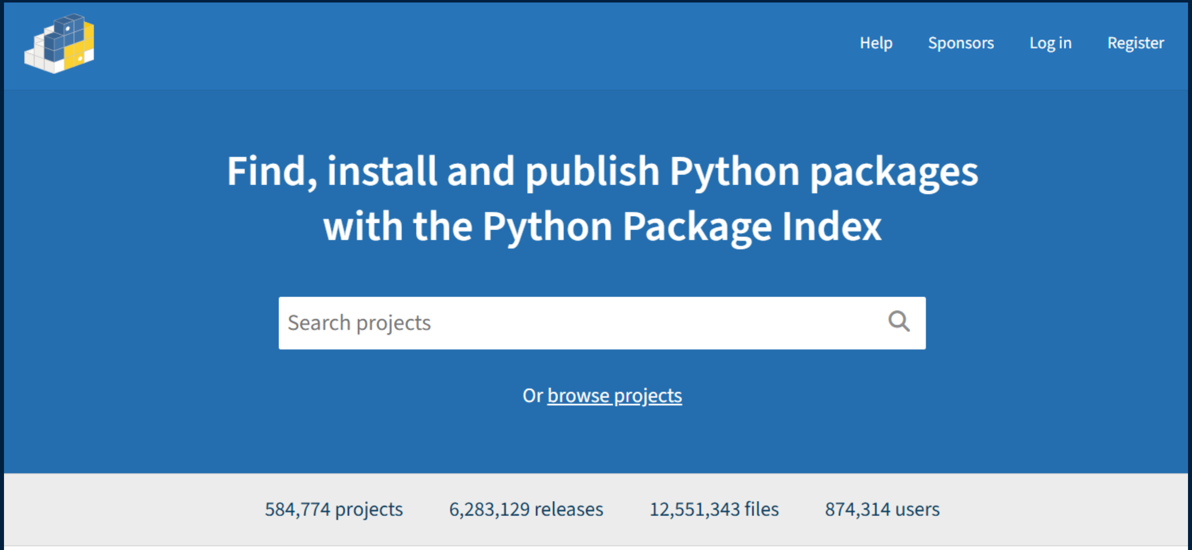
Another Example

```
import package.subpackage1.src.functions as fn


print(fn.add(20,10))
print(fn.subtract(20,10))




from package.subpackage1.src import functions


print(functions.add(20,10))
print(functions.subtract(20,10))
```

Install External Packages

Python has a huge community of people developing pieces of python code (modules and packages) that we can import and use.

The Python Package Index: https://pypi.org/



Find, install and publish Python packages with the Python Package Index

Or browse projects

584,774 projects 6,283,129 releases 12,551,343 files 874,314 users

Install External Packages

`pip` and `conda` commands search online for packages, download and place them in the right folder for you.

`pip` searches in the Python Package Index (PyPi, https://pypi.org/) which is a repository that holds python packages.

`conda` has its own package repository

To install a package, go to terminal (or command line) and type:

```
pip install [package name]

or

conda install [package name]
```

Package Question

Question: Assume that we have the following directory structure. To import the `mathfunc` module, we need to use

A) `import pkg.subpkg1.subpkg2`

B) `import subpkg1.subpkg2.mathfunc`

C) `import pkg.subpkg1.subpkg2.mathfunc`

D) `from pkg.subpkg1 import mathfunc`

E) None of the above

```
pkg/
    __init__.py
    subpkg1/
        __init__.py
        subpkg2/
            __init__.py
            mathfunc.py
```

Package Question

Question: How many of the following statements are TRUE?

1) A Python package is a collection of modules.

2) A module in a sub-package can be accessed by

`package.subpackage.modulename`

3) A higher-level module's function gets precedence over a local module's function

4) The `__init__.py` files can be used for execution of package initialization code

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Creating Package

Question: Create a package using the following hierarchy.

```
mypackage/
    __init__.py
    myfunc.py
    mysubpackage/
        __init__.py
        mathfunc.py
```

Now write a python program that uses functions (e.g., `double`, `add`) from the package to perform basic math operations.

Objectives

- Know about Python modules and packages
- Know how the `import` statement works
- Be able to write own "importable" packages and modules
- Know how to access other people's packages and modules

THE UNIVERSITY OF BRITISH COLUMBIA