NumPy

BILD 62

Objectives for today

- Install and import packages for Python
- Create NumPy arrays
- Execute methods & access attributes of arrays

Python supports **modular programming** in multiple ways.

Functions and **classes** are examples of tools for low-level modular programming.

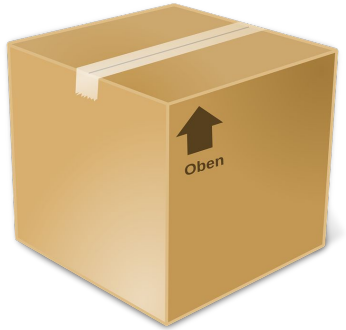
Python **modules** are a higher-level modular programming construct, where we can collect related variables, functions and classes in a module.

Modules are often bundled up into **packages**.

Packages in Python

Python's standard library works for some purposes, but there are many very useful packages for additional purposes:

- **numpy** (http://numpy.scipy.org): numerical Python
- **scipy** (http://www.scipy.org): scientific Python; built on numpy
- **matplotlib** (http://www.matplotlib.org) graphics library



Installing packages & importing modules

To install packages, use

```
$ pip install PACKAGE
```

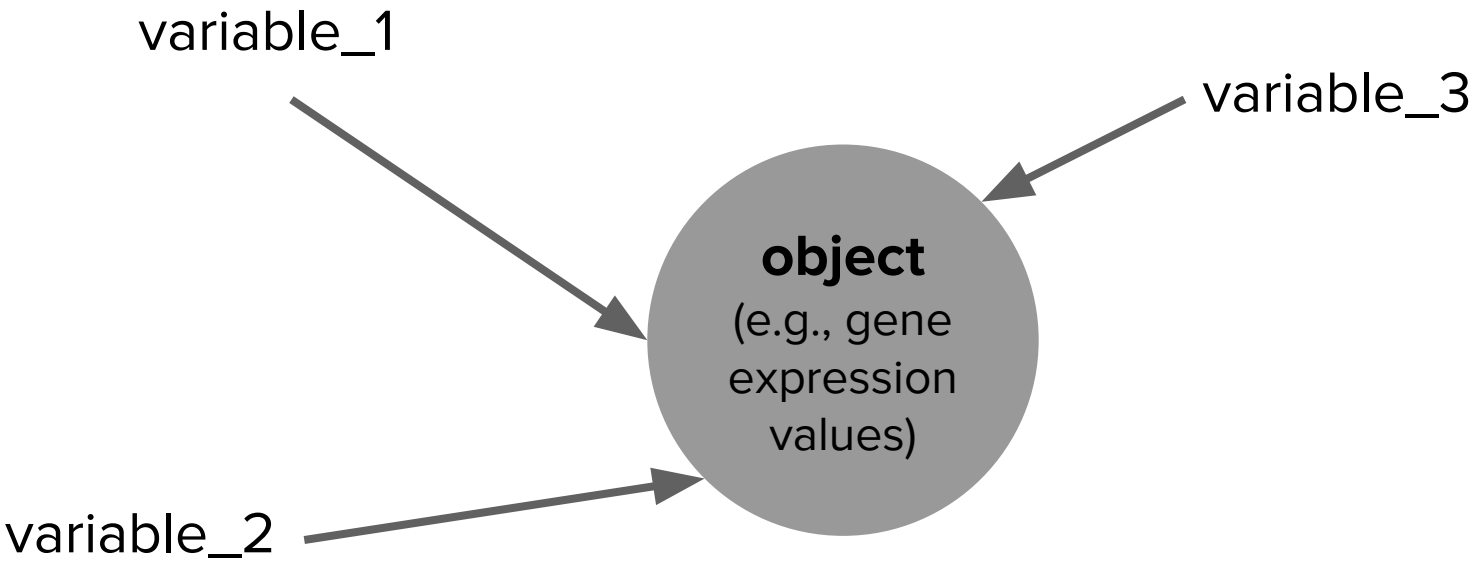
We typically won't need to do this in the DataHub, because many packages have been installed into our container. However, you *may* need to do this for local notebook operation.

You can then import modules from the package with

```
>>> from PACKAGE import MODULE
```

to see all of the modules available, use

```
>>> print(dir(MODULE))
```



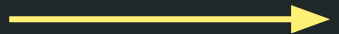
Object-oriented programming

This is how Python containers typically work, but saving all of our data in lists isn't great for performance or memory.
NumPy is a tool for computing with big arrays, and is much more efficient.*

* for details, see this breakdown

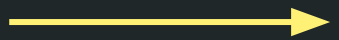
We’re learning how to deal with more and more complex data

```
data_point = 8.02
```



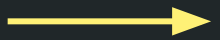
single variable (int, float, string)

```
data = [8.38,3.34,6.35]
```



data structure (list, tuple, dictionary)

```
big_data = [data_1,data_2,...]
```



array or **dataframe**

NumPy is the fundamental package for scientific computing with Python

- A numpy **array** is a grid of values which are all the same type (they're **homogenous**)
- Useful attributes:
  - **ndim** = # of dimensions
  - **shape** = a tuple of integers giving the size of the array along each dimension
  - **dtype** = type of data

Numpy Arrays

my_array = 1D array

3	2	4	1

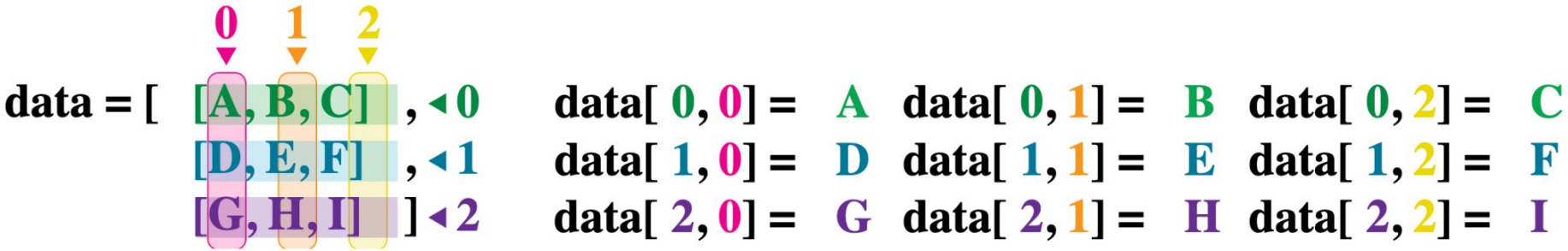
```
my_array[0] = 3

my_array.ndim = 1

my_array.shape = (4,)

my_array.size = 4
```

2D array

3	2	4	1
1	2	5	3

how to index 2D NumPy arrays

```
my_array[1,3] = 3

my_array.ndim = 2

my_array.shape = (2,4)

my_array.size = 8
```


0
1
2
data = [
[A, B, C] , ◂ 0
[D, E, F] , ◂ 1
[G, H, I]] ◂ 2
data[0, 0] = A
data[0, 1] = B
data[0, 2] = C
data[1, 0] = D
data[1, 1] = E
data[1, 2] = F
data[2, 0] = G
data[2, 1] = H
data[2, 2] = I

Slicing & indexing NumPy arrays works *almost* the same as with Python lists

However, be aware that if you slice an array, it changes the original array.

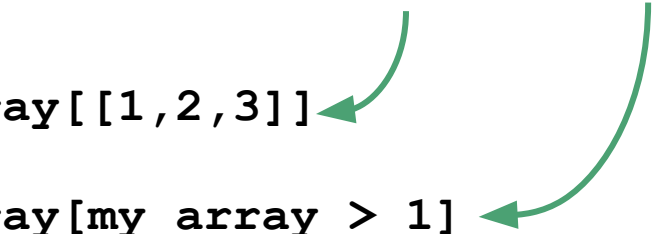
If you need to copy, you need to explicitly do:

```
v3 = v[2:4].copy()
```

In this case, we would not change original array (v).

```
In [33]: v = np.random.random((5,4))
         v

Out[33]: array([[0.70782755, 0.1080363 , 0.63931318, 0.30594658],
                [0.23089631, 0.58842692, 0.03879193, 0.56396161],
                [0.92250973, 0.54564224, 0.89690301, 0.76679512],
                [0.83668402, 0.18075749, 0.54652922, 0.03487156],
                [0.48236452, 0.77258043, 0.61857768, 0.66614441]])

In [35]: v2 = v[2:4]
         v2

Out[35]: array([[0.92250973, 0.54564224, 0.89690301, 0.76679512],
                [0.83668402, 0.18075749, 0.54652922, 0.03487156]])

In [37]: v2[1,3] = 2

In [38]: v

Out[38]: array([[0.70782755, 0.1080363 , 0.63931318, 0.30594658],
                [0.23089631, 0.58842692, 0.03879193, 0.56396161],
                [0.92250973, 0.54564224, 0.89690301, 0.76679512],
                [0.83668402, 0.18075749, 0.54652922, 2.        ],
                [0.48236452, 0.77258043, 0.61857768, 0.66614441]])
```

You can also use lists & Booleans to index NumPy arrays



```
my_array[[1,2,3]]

my_array[my_array > 1]
```

We can also use this to selectively operate on values in the array that meet our criteria:

```
my_array[my_array > 1] = my_array[my_array > 1] * 2
```

Useful NumPy functions

`np.zeros()`

`np.empty()`

`np.linspace()`

`np.arange()`

`np.reshape()`

`np.random.random()`

`np.vstack()`

`np.hstack()`

`np.save()`

`np.load()`

See here for a useful Numpy overview.

Resources

Numerical & Scientific Computing with Python: Introduction into NumPy

Lecture-2-Numpy.ipynb

Analyzing Patient Data – Programming with Python