NumPy

BILD 62

Before we dive into new content, let's see if we can apply what we've learned to some *real* code documentation

`Class`

Template for objects

Defines properties for objects (**attributes**)

Defines behaviors for objects (**methods**)

`Object`

Instance of a class

Has attributes that are defined differently in each instance (using `__init__` method) or that are always inherited from the `Class`

```
8
9   class Words(Base):
10      """A class for collecting and analyzing words data for specified terms list(s).
11
12      Attributes
13      ----------
14      results : list of Articles
15          Results of 'Words' data for each search term.
16      labels : list of str
```

a

b

c

```
22      def __init__(self):
23          """Initialize LISC Words object."""
24
25
26
27          self.results = list()
28          self.meta_data = None
29
```

d

Possible matches:

1. Method that will execute whenever a class instance is created
2. Inherited class
3. Name of the class we are defining here
4. Attributes that will update when class is initialized

From https://github.com/lisc-tools/lisc/blob/c44af07492165f9a35b653b6aa1da1f397044593/lisc/objects/words.py

Feature Extraction

The **`EphysFeatureExtractor`** class calculates electrophysiology features from cell recordings. `extract_cell_features()` can be used to extract the precise feature values available in the Cell Types Database:

a

b

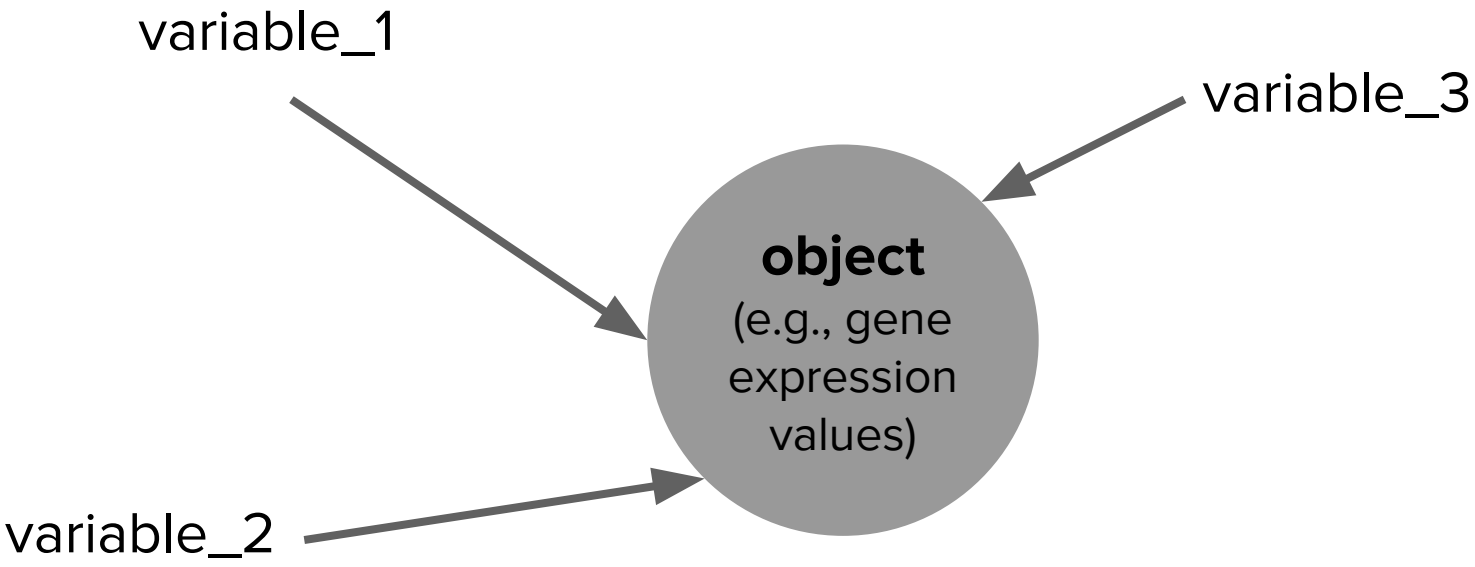
```
from allensdk.core.cell_types_cache import CellTypesCache
from allensdk.ephys.extract_cell_features import extract_cell_features
from collections import defaultdict

# initialize the cache
ctc = CellTypesCache(manifest_file='cell_types/manifest.json')

# pick a cell to analyze
specimen_id = 324257146

# download the ephys data and sweep metadata
data_set = ctc.get_ephys_data(specimen_id)
sweeps = ctc.get_ephys_sweeps(specimen_id)
```

c

From https://alleninstitute.github.io/AllenSDK/cell_types.html

Possible matches:

1. Modules we're importing
2. Executing method of class CellTypesCache
3. Instance of class CellTypesCache



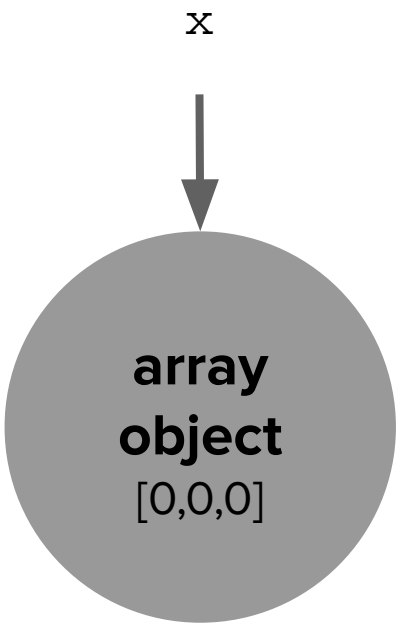
Object-oriented programming

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

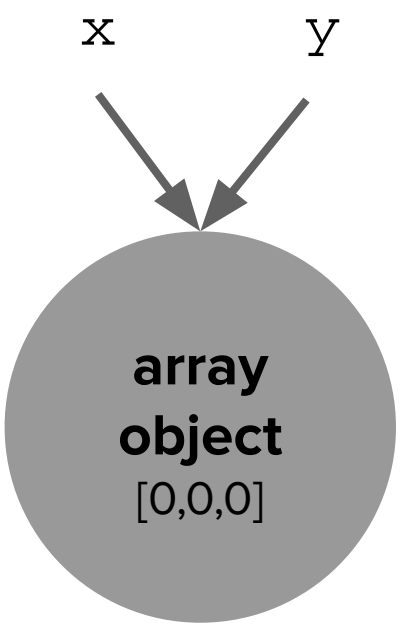
STEP 1.

```
x = np.zeros(3)
```



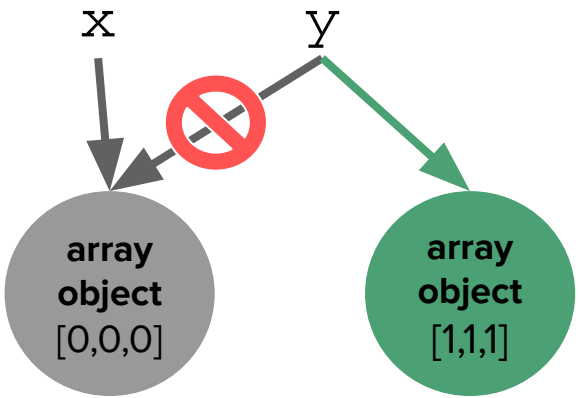
STEP 2.

```
y = x
```



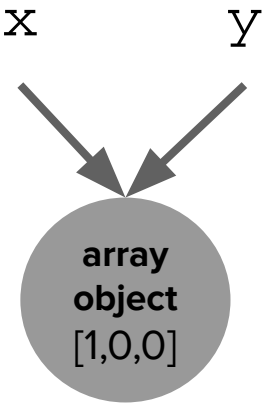
STEP 3a.

```
y = y+1
```



STEP 3b.

```
y[0] = y[0]+1
```



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.

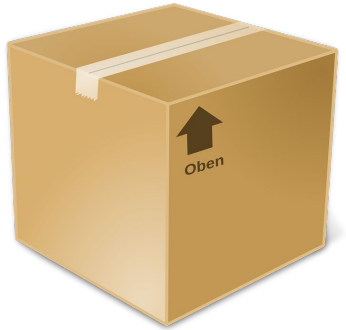
Python **modules** are a higher-level modular programming construct, where we can collect related variables, functions and classes in a module.

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

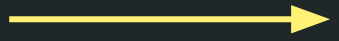
```
>>> from PACKAGE import MODULE
```

to see all of the modules available, use

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

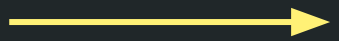
We're learning how to deal with more and more complex data

```
data_point = 8.02
```



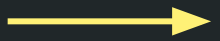
single variable (int, float, string)

```
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

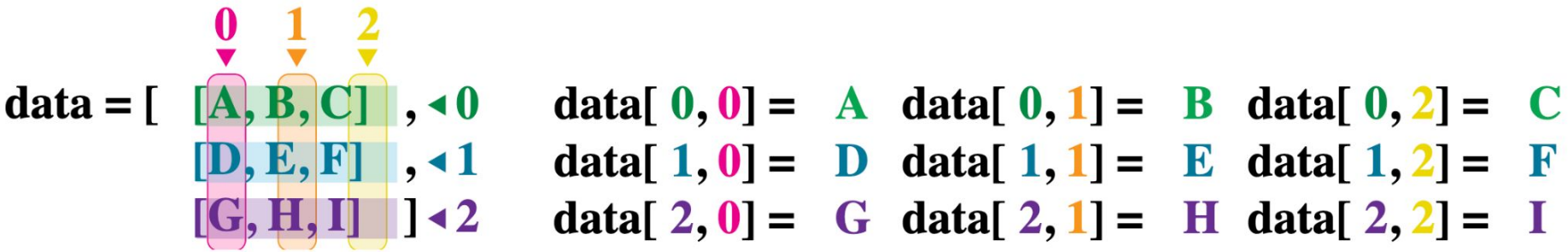
```
my_array[0] = 3

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
```



Indexing numpy arrays

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.**

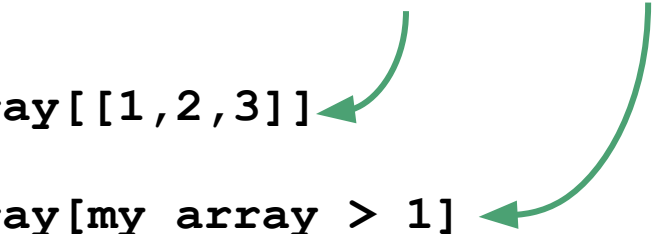
If you need to copy, you need to explicitly do:

```
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.

Key NumPy takeaways

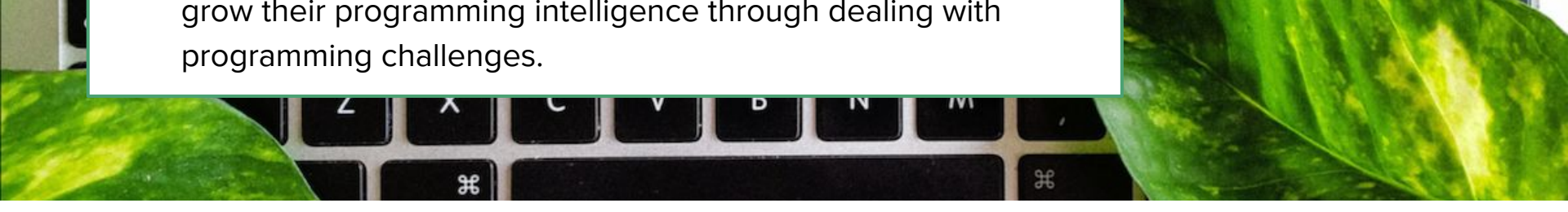
- Import a library into a program using `import libraryname`
- Use the NumPy library to work with arrays in Python.
- The expression array.shape gives the shape of an array.
- Use `array[x, y]` to select a single element from a 2D array.
- Array indices start at 0, not 1.
- Use `low:high` to specify a slice that includes the indices from low to high-1.
- Use `np.mean(array)`, `np.max(array)`, and `np.min(array)` to calculate simple statistics.
- Use `np.mean(array, axis=0)` or `np.mean(array, axis=1)` to calculate statistics across the specified axis.

Thinking back on mindset...

- Describe a time when you were learning something new other than programming (e.g., from school, at work or in everyday life) where you had to work really hard on a challenging task. Maybe you made a lot of mistakes, became extremely frustrated and wanted to give up, but with practice and perseverance you were able to succeed. Please be specific about the kinds of mistakes you made and how you overcame them.
- What advice would you give a beginning programmer in BILD 62 to help them cope with the challenge of learning to write and debug Python programs? Be sure to emphasize to them how to grow their programming intelligence through dealing with programming challenges.

Respond on Canvas for credit.

Your (anonymous) input will be shared with future classes!



Resources

Numerical & Scientific Computing with Python: Introduction into NumPy

Lecture-2-Numpy.ipynb

Analyzing Patient Data – Programming with Python