Python Review

Sam Scott, Mohawk College, 2019.

Getting Python

For the material in this workshop, you need Python 3 with the numpy, sklearn, and matplotlib libraries. The easiest way to get all this is to install the Anaconda distribution of Python 3:

https://www.anaconda.com/distribution/#download-section

You should also use a decent IDE. There are lots of options, but I recommend Pyzo, a lightweight IDE with lots of functionality that is specifically designed to work with Anaconda.

https://pyzo.org/start.html

If you install Anaconda, then run Pyzo it will probably find your Anaconda installation and give you a one-click solution to configure it. If not, you can "Edit Shell Configurations" to point to your installation of Anaconda.

Python 3 vs. Python 2

We are using Python 3, which is not backwards compatible with Python 2. But lots of people still use Python 2, so code you find on the internet might not always work the way you expect in Python 3.

Types

Python is dynamically-typed. No need to declare variables, and a variable can take a value of any type.

Important types include `int`, `float`, `str`, `bool`, `list` and `dict`.

Use the `type` function to find out the type of a value.

Convert between types using the type name function (e.g. `int("34")` returns `34`).

Syntax

No semicolons. Code blocks are indented instead of being enclosed in braces { }.

Use a **colon (:)** before a block.

Use **`and`**, **`or`**, and **`not`** instead of `&&`, `||`, and `!`.

Use **#** for a comment. Use **##** for an underlined comment in Pyzo.

Special Operators

```
2 ** 10                 # means 2 to the power 10
"hi " + "there"         # string concatenation
"hi" * 5                # evaluates to "hihihihihi "
```

Input and Output

```
print(item1, item2, item3…, sep="", end="")
     # sep="" is optional. Otherwise spaces between the items.
     # end="" is optional. Otherwise carriage return at the end.

x = input()                  # get a string input from the keyboard

y = float( input() )         # get an input and convert to float
```

Control Flow

Selection (if)

```
if boolean_expression:
     indented block of code

elif boolean_expression:               # optional "else if"
     indented block of code

else:                                  # optional else
     indented block of code
```

Repetition (while)

```
while boolean_expression:
     indented block of code
```

Repetition (for)

```
for variable in range(min, max, step):           # step is optional
     indented block of code

for variable in list                             # for-each
     indented block of code
```

Functions

```
def function_name(parameter_list): # no type declarations
     indented block of code
     return value                      # Optional – default is None


variable = function_name(argument_list) # calls the function
```

Python Lists

This is a quick primer on lists in Python. Familiarity with Python is assumed. For more details on lists and on Python syntax in general, the free textbook *Think Python* is a good resource.

List Literals

Literals are values that are typed directly into a program (like `32` or `"hello, world"`). List literals are comma separated lists of values or expressions enclosed in square brackets.

`a = ["fda", 45, "23"]`	Creates a list and assigns it to `a`
`a = [b, c, d, b+c+d]`	Creates a list with values from variables `b`, `c` and `d`
`a = []`	Creates an empty list

List Operators

Operators are symbols or keywords in the language that change or combine the contents of variables. Here are some important list operators:

`a + b`	Concatenates `a` and `b` into a new list
`a * n`	Creates a new list by repeating (`n` times) the contents of the list `a`
`a[i]`	Accesses element `i` of list `a` for assignment or use in expressions
`a[i:j]`	Returns a copy of a slice of list `a` (both `i` and `j` are optional)
`e in a`	Returns `True` if `e` is an element of `a`
`e not in a`	Returns `True` if `e` is not an element of `a`
`del a[i]`	Removes the selected element from `a` (also works with slices)

List Functions

The following are built in global functions that operate on lists.

`len(a)`	Returns the length of `a`
`sum(a)`	Returns the sum of the elements in `a`
`min(a)`	Returns the minimum element in `a`
`max(a)`	Returns the maximum element in `a`
`sorted(a)`	Returns a new sorted list with the same contents as `a`
`sorted(a, reverse=True)`	Sorts in descending order

List Methods

In Python, lists are objects with a number of built-in methods.

`a.append(e)`	Adds element `e` to the end of `a`
`a.extend(b)`	Adds all elements from list `b` to the end of `a`
`a.remove(e)`	Deletes the first element that is equal to `e` (using ==)
`a.insert(i, e)`	Inserts `e` into `a` at index `i`
`a.index(e)`	Returns the index of the first element equal to `e` (Raises a `ValueError` exception if `e` is not in the list)
`a.count(e)`	Returns a count of the number of times `e` appears in `a`

Processing Lists

To process a list is to “visit” (print, change, etc.) every element of the list.

In Python the `for` loop is actually a for-each loop. You can use it like a regular for loop by using the `range` function to create a numerical range

- `range(10)` creates the range `{0…9}`
- `range(5, 10)` creates the range `{5…9}`
- `range (1, 11, 2)` creates the range `{1,3,5,7,9}`
- The `reversed` function will reverse a range

Use Loop 1 for visiting in order without changing. Loop 2 is for visiting and changing. Loop 3 is for visiting in reverse order.

1

```
for e in l
   visit e
```

2

```
for i in range(len(l))
   visit l[i]
```

3

```
for i in reversed(range(len(i)))
   visit l[i]
```

Numpy Arrays

What is Numpy?

Numpy is a Python library that allows you to use the Array Programming paradigm that is popular in machine learning and data science and embodied in languages like R and MatLab. Array Programming languages provide powerful and efficient built-in operations for arrays.

In Array Programming, you represent structured data using sets of related (parallel) arrays instead of using objects. You lose many of the benefits of the OOP paradigm, but you gain flexibility and efficiency.

Importing Numpy

`import numpy as np` ← everyone uses "np" instead of "numpy"

Creating Numpy Arrays

Arrays from Python Lists

`np.array(list)` ← creates array from a Python list

Arrays of 0's and 1's

```
np.zeros(size)        np.zeros( (rows, cols) )
np.ones(size)         np.ones( (rows, cols) )
```
← arrays of zeroes
← arrays of ones

Note: Array size can be an integer for a 1D array or a list or tuple for a multi-dimensional array.

Arrays of Random Numbers

```
np.random.randint(min, max, size)
np.random.random(size)
```
← list of random ints
← list of random doubles (0 to 1)

```
np.random.randint(min,max,(rows,cols))
np.random.random( (rows,cols) )
```
← 2D array of random ints
← 2D array of random doubles

Arrays of Number Ranges

`np.arange(10)` = array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

`np.arange(1,11)` = array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

`np.arange(2, 11, 2)` = array([2, 4, 6, 8])

Array Arithmetic

`a = a + b`	if `b` is a scalar, adds b to every element of `a`
	if `b` is an array of same shape as `a`, adds element by element
	if `a` is 2D and `b` is the shape of a single row of `a`, broadcasts `+ b` across `a` (i.e. adds `b` element by element to each row of `a`).
	Works for `+`, `−`, `*`, `/`, etc.

Array Indexing and Slicing

1D arrays work just like python lists

2D arrays: `x[1,6]` `x[1,:]` = all of row 1 `x[1,2:5]` = slice of row 1

`x[:,1]` = all of column 1 `x[2:5,1]` = slice of column 1

`x[2:5, 2:5]` = rectangular “slice” of `x`

`x[ [list of indices] ]`	← pulls out selected indices
`x[ [True, False, True, True, False] ]`	← pulls out values where True

Summarizing Array Data

`a.min()` `a.max()` `a.mean()` `a.sum()`

`a.shape`	← Size of array. Use `a.shape[0]` for rows, `a.shape[1]` for columns
`a.min(axis=0)`	← min of each column
`a.mean(axis=1)`	← mean of each row

Sorting and Shuffling Array Data

`a.sort()`	← sort in ascending order (sorts each row if a 2D array)
`a.sort(axis=0)`	← sort each column
`a = a[::-1]`	← slicing trick for reversing a 1D array
`np.random.shuffle(a)`	← shuffles an array

Comparisons with Arrays

`x > 40`	← returns a Boolean array, one value per element of x
`x > y`	← if the arrays are the same size, returns a Boolean array with element by element comparisons
`x[ x>40 ]`	← returns new array of items from x that are > 40

Related (Parallel) Arrays

Related arrays are related to one another by index. For example, here are two arrays holding player names and scores:

```
    names = np.array(["Sam", "Anne", "Max", "Rosa"])

    scores = np.array([100, 234, 1000, 10])
```

In this example, `scores[i]` holds the score that goes with `names[i]`.

Finding Things in Related Arrays

Use the comparison operator with one array to create an array of Booleans that will select from the related array.

```
scores[ names == "Max"]   ← array with Max's score

names[ scores >= 100]     ← array with names of high scorers
```

The "arg" Methods

The `argmin`, `argmax`, and `argsort` methods in Numpy work the same way as their non-"arg" counterparts, but they return the "arguments" (i.e. the indexes) rather than the values.

```
scores.argmax()               ← the index of the maximum score

names[ scores.argmax() ]      ← the name of the highest scoring player

scores[ names.argmin() ]      ← the score of the first  player in alphabetical order

scores.argsort()              ← a list of indices representing a sort of scores.

scores.argsort()[::-1]        ← Use the step value of a slice to reverse the sort.

names[ scores.argsort() ]              ← a list of names sorted by score

names[ scores.argsort()[::-1]]         ← high score first

names[ scores.argsort()[::-1]][:2]     ← top 2 scorers
```

Shuffling Related Arrays

```
i = np.arange( names.shape[0] )        ← array of indices

np.random.shuffle( i )                 ← shuffle the indices

names = names[i]                       ← rearrange the related arrays

scores = scores[i]
```

Numpy Exercises

Basic NumPy

1. Write a program that creates a 100 x 100 array of random numbers in the range -1,000,000 to 1,000,000. Print the maximum, minimum, and average value in the array.

2. Create a new array that contains the sum of each row of the array from exercise 1 (use parameter axis=__ to sum just the rows or just the columns). Print the maximum, minimum, and average of these sums.

3. Combining the arrays from part 1 and part 2, print the range of values in the row with the largest sum.

Numpy Related Arrays

4. Create a pair of related arrays. One stores 20 randomly determined x coordinates (range -100 to 100) and the other stores 20 corresponding y coordinates (range 100 to 200). Sort both arrays in ascending order of y coordinate. Check the arrays to make sure the same x's are still paired with the same y's.

5. Using the same set of arrays from exercise 4, create a third related array that contains the distance of each point from the point (100, 100) (i.e. the square root of $(x\text{-}100)^2 + (y\text{-}100)^2$).

6. Using the same arrays from exercise 5, create a new pair of arrays containing only the x and y values for coordinates which are within 100 units distance from (100,100). Check to make sure that the x and y values are still paired correctly.

More NumPy Fun!

The following exercises are selected from “100 Numpy Exercises” by Nicolas P. Rougier. Most can be solved in 1 or two lines. Those marked * require methods that we did not cover in class (you’ll have to look them up!)

For sample solutions, go to https://github.com/rougier/numpy-100/ and open the file “100 Numpy exercises.md”.

1. Import the numpy package under the name np
3. Create a null (i.e. all zeroes) vector of size 10
6. Create a null (i.e. all zeroes) vector of size 10 but the fifth value which is 1
7. Create a vector[1] with values ranging from 10 to 49
9. Create a 3x3 matrix[2] with values ranging from 0 to 8
10. Find indices of non-zero elements from [1,2,0,0,4,0] *
12. Create a 3x3x3 array with random values
13. Create a 10x10 array with random values and find the minimum and maximum values
14. Create a random vector of size 30 and find the mean value
15. Create a 2d array with 1 on the border and 0 inside (Hint: Use slices)
20. Consider a (6,7,8) shape array, what is the index (x,y,z) of the 100th element?
21. Create a checkerboard 8x8 matrix using the tile function *
25. Given a 1D array, negate all elements which are between 3 and 8, in place.
40. Create a random vector (i.e. 1D array) of size 10 and sort it
45. Create random vector of size 10 and replace the maximum value by 0
54. Read a file with the following contents into an array.

```
1, 2, 3, 4, 5
6,  ,  , 7, 8
 ,  , 9,10,11
```

59. Sort an array by the nth column.
72. Swap the first two rows of an array.
89. Retrieve the n largest values in an array

[1] A vector is a one-dimensional array.
[2] A matrix is a two-dimensional array.