Programming in Python

numpy arrays and math functions

Topics covered

1. Numpy arrays
2. Mathematical operators
3. Basic math functions
4. Exercises

numpy basics

- NumPy stands for Numerical Python
- Has many of the same functions as does the Math module, but Numpy is more extensive
- NumPy is commonly imported and aliased as np
  - import numpy as np

numpy arrays

- Python does not have a built-in array data type
- It does not have a module called ARRAY that has objects called arrays
- These arrays behave essentially like lists that are forced to all have the same data type for their elements
- NumPy has array objects that behave more like Fortran arrays
  - `import numpy as np`
  - `a = np.array([3, -5, 8])`
  - `print(a)`

Array indexes

- Array indices begin at 0
  - Different from Fortran
- For 2-D arrays the first index is the row, and the second index is the column
  - `a = np.array([[3, -5, 8], [-7, 6, 9]])`
  - `print(a)`
- A colon is used to specify a range if indices
  - Form is m:n
  - This refers to indices from m to n-1
    - `a = np.array([3, -5, 8])`
    - `a[2:6]`
- All elements from the beginning of the array to element to n can be accessed by :n+1

Creating arrays

- Array with all 0
  - ```
    a = np.zeros((5,3), dtype = np.float64)
    a
        array([[0., 0., 0.],
                              [0., 0., 0.],
                              [0., 0., 0.],
                              [0., 0., 0.],
                              [0., 0., 0.]])
    ```
- Array with all 1
  - ```
    a = np.ones((5,3), dtype = np.float64)
    ```
- With other numbers
  - ```
    b = a*4
    ```
- Empty array
  - ```
    a = np.empty((5,3),dtype = np.float64)
    ```
- Check it!

Creating arrays with numpy.arange()

```
a = np.arange(0, 10)
a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
a = np.arange(0, 10.0)
a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
a = np.arange(0, 10, 0.5)
a
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6.
,6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5])
```

Creating arrays with numpy.linspace()

```
a = np.linspace(-5, 5, 20)
a
array([-5., -4.47368421, -3.94736842, -3.42105263, -2.89473684,
       -2.36842105, -1.84210526, -1.31578947, -0.78947368, -0.26315789,
        0.26315789, 0.78947368, 1.31578947, 1.84210526, 2.36842105,
 2.89473684, 3.42105263, 3.94736842, 4.47368421, 5. ])
```

- See also logspace()

The where() function

```
a = np.linspace(-5, 5, 20)
a
array([-5. , -4.47368421, -3.94736842, -3.42105263, -2.89473684,
 -2.36842105, -1.84210526, -1.31578947, -0.78947368, -0.26315789,
        0.26315789, 0.78947368, 1.31578947, 1.84210526, 2.36842105,
        2.89473684, 3.42105263, 3.94736842, 4.47368421, 5. ])
result = np.where(a < 0)
result
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),)
type(result)
        <class 'tuple'>
a[result]
        array([-5., -4.47368421, -3.94736842, -3.42105263, -2.89473684,
        -2.36842105, -1.84210526, -1.31578947, -0.78947368, -0.26315789])
```

numpy exercise 1

- Create a numpy array of 1.000.000 elements made as follows:
  - the elements in even position are 0
  - the elements in odd position are 1

numpy exercise 1: (one) solution

```
import numpy as np
a = np.zeros(1000000)
a[::2] = 1
```

several others are possible...

Saving NumPy Arrays

- NumPy provides its own functions to read and write arrays to binary files

  This is accomplished with either
  - np.save() function, which writes a single array to a NumPy .npy file
  - np.savez() function, which archives several arrays into a NumPy .npz file

Example

```
import numpy as np
a = np.arange(0, 100)*0.5
b = np.arange(-100, 0)*0.5
np.save('a-file', a)
np.save('b-file', b)
np.savez('ab-file', a=a, b=b)
```

- Creates three files:
  - a-file.npy which contains the values for a
  - b-file.npy which contains the values for b
  - ab-file.npz which is an archive file containing both the a and b values

Loading from numpy files

- To retrieve the values from the .npy files we use the np.load() function
  - `a = np.load('a-file.npy')`
  - `b = np.load('b-file.npy')`
- To retrieve the values from the .npz files we also use the np.load() function to load all the data into a dictionary that contains the archived arrays
  - `z = np.load('ab-file.npz')`
  - `a = z['a']`
  - `b = z['b']`
- To find the names of the arrays used in the dictionary, use the *files* attribute of the dictionary
  - `z.files`

    `['a', 'b']`

Basic math operators

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `//` truncating division (`16.3 // 5.2` => `3.0`)
- `**` power
- `%` modulo (`5 % 3` => `2` , `5.2 % 3` => `2.2`)

Comparison operators

- `<` less than
- `>` greater than
- `==` equal to
- `!=` not equal to
- `>=` greater than or equal to
- `<=` less than or equal to

Augmented assignments

- The augmented assignment operators are shorthand operators and take the form
  - `x += y`, which is the same as `x = x + y`
  - `x += 1`, which is the same as `x = x + 1`
- This works not only with addition (+), but also with subtraction, multiplication, division, truncated division, and powers

Math functions

- Python has a limited number of built-in mathematical functions
  - abs(x) absolute value of x
  - divmod(x,y) returns a tuple with (x // y, x % y)
  - pow(x,y) same as x**y
  - round(x, [m]) rounds x to nearest integer value, unless optional integer m is given, in which case it round to nearest multiple of 10^{-m}

numpy functions

- The NumPy module contains many of the same mathematical functions as the math module, and is the preferred module to use for math
- NumPy must be imported before use, either as
- `import numpy`
- or
- `import numpy as np`

numpy trig functions

- arccos(x) arc cosine
- arccosh(x) inverse cosh
- arcsin(x) arc sin
- arcsinh(x) inverse sinh
- arctan(x) arc tangent
- arctan2(y,x) arc tangent of y/x.
- arctanh(x) inverse tanh
- cos(x) cosine of x
- cosh(x) hyperbolic cosine
- degrees(x) converts x to degrees from radians
- radians(x) converts x to radians from degrees
- sin(x) sin of x
- sinh(x) hyperbolic sine
- tan(x) tangent of x
- tanh(x) hyperbolic tangent

Note: All angles are in radians unless otherwise specified!

numpy useful numeric functions

- ceil(x) the smallest integer >= to x
- copysign(x,y) returns x with the same sign as y
- floor(x) the largest integer >= x
- fabs(x) absolute value
- trunc(x) truncates x to integer
- fmod(x,y) like x % y
  - Use fmod(x,y) if either x or y are floating point values
  - Use x % y if both x and y are integers.
- modf(x) breaks x into integer and fractional parts
  - np.modf(89.4357) => (0.4356999999999971, 89.0)
- isinf(x) returns True if x is positive or negative infinity
- isnan(x) returns True if x is not a number (NaN)

Loading modules

- All of methods and attributes of a module can be loaded by simply using the import command
- We then access the functions and constants by prefacing them with numpy
  - ```
    import numpy
    numpy.cos(0.5)
        0.87758256189037276
    ```
- We can use an alias when importing a module
  - Avoids having to repeatedly typing long module names
  - ```
    import numpy as np
    np.cos(0.5)
        0.87758256189037276
    ```
- We can import individual functions or constants from modules
  - They can also be aliased on import
  - ```
    from numpy import cos as npcos
    ```
    ```

Don’t Import All Functions Using *

- We can import every function and constant and then not have to preface them with the module name
- However, you should avoid this!!
- Can cause confusion if multiple modules have functions with the same name

numpy exercise 2

- Write a program that reads a csv (comma separated values) file containing the following information regarding the students of a MS course
  - ID, AGE, ID_Registration, Num_exams
- calculates the average number of exams of the students
- calculates the average number of exams of the students who are older than a given age

Notes: I can provide you a sample csv file, but you should look for functions to import the csv

numpy exercise 2: solution

```
import numpy as np
thres = 40
# LET'S READ THE FILE
csv_file = open("dataset1.csv", "r")
lines = csv_file.readlines()[1:] # Ignore the first line of the file
students = np.empty([len(lines), 2])
i = 0
for line in lines:
 line = line.rstrip('\n')
 dataItem = line.split(',')
 age = int(dataItem[1])
 exam = int(dataItem[3])
 students[i,:] = [age, exam]
 i += 1
csv_file.close()
```

numpy exercise 2: solution

```
#LET'S CALCULATE THE MEAN VALUES
meanAll = 0
meanFilt = 0
j = 0
for i in range(len(students)):
 meanAll += students[i, 1]
 if (students[i, 0] > thres):
    meanFilt += students[i, 1]
    j += 1
meanAll /= (i + 1)
meanFilt /= j
print ("The average on all is: ", meanAll)
print ("The average on students older than ", thres,
 " is: ", meanFilt)
```