NumPy

INTRODUCTION TO PYTHON

Hoai Thuan TRAN
Gia Dinh University

Lists Recap

- Powerful
- Collection of values
- Hold different types
- Change, add, remove
- Need for Data Science
  - Mathematical operations over collections
  - Speed

Illustration

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
height
```

```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

```
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight
```

```
[65.4, 59.2, 63.6, 88.4, 68.7]
```

```
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Solution: NumPy

- Numeric Python
- Alternative to Python List: NumPy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
  - In the terminal: `pip3 install numpy`

NumPy

```
import numpy as np
np_height = np.array(height)
np_height
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

```
np_weight = np.array(weight)
np_weight
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
bmi = np_weight / np_height ** 2
bmi
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

Comparison

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

```
np_height = np.array(height)
np_weight = np.array(weight)
np_weight / np_height ** 2
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

NumPy: remarks

```
np.array([1.0, "is", True])
```

```
array(['1.0', 'is', 'True'], dtype='<U32')
```

- NumPy arrays: contain only one type

NumPy: remarks

```
python_list = [1, 2, 3]
numpy_array = np.array([1, 2, 3])
```

```
python_list + python_list
```

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

```
numpy_array + numpy_array
```

```
array([2, 4, 6])
```

- Different types: different behavior!

NumPy Subsetting

```
bmi
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

```
bmi[1]
```

```
20.975
```

```
bmi > 23
```

```
array([False, False, False,  True, False])
```

```
bmi[bmi > 23]
```

```
array([24.7473475])
```

Let's practice!

INTRODUCTION TO PYTHON

Baseball players' height

You are a huge baseball fan. You decide to call the MLB (Major League Baseball) and ask around for some more statistics on the height of the main players. They pass along data on 20 players, which is stored as a regular Python list: height_in. The height is expressed in inches.

Instructions

- Create a numpy array from height_in. Name this new array np_height_in. Print out np_height_in.
- Multiply np_height_in with 0.0254 to convert all height measurements from inches to meters. Store the new values in a new array, np_height_m. Print out np_height_m

```
# Import numpy
import numpy as np

height_in = [74, 74, 72, 72, 73, 69, 69, 71, 76, 71,
             73, 73, 74, 74, 69, 70, 73, 75, 78, 79]

# Create a numpy array from height_in: np_height_in

# Print out np_height_in

# Convert np_height_in to m: np_height_m

# Print np_height_m
```

Baseball player's BMI

The MLB also offers to let you analyze their weight data. The weight is available as weight_lb (weight in pounds).

It's now possible to calculate the BMI of each baseball player.

Instructions

- Create a numpy array from the weight_lb list with the correct units. Multiply by `0.453592` to go from pounds to kilograms. Store the resulting as np_weight_kg.
- Use np_height_m and np_weight_kg to calculate the BMI.
- Save the resulting numpy array as bmi

```
weight_lb = [180, 215, 210, 210, 188, 176, 209,
             200, 231, 180, 188, 180, 185, 160,
             180, 185, 189, 185, 219, 230]

# Create array from height_in with metric units:
np_height_m


# Create array from weight_lb with metric units:
np_weight_kg


# Calculate the BMI: bmi


# Print out bmi
```

Lightweight baseball players

For **numpy** specifically, you can also use boolean **numpy** arrays:

```
high = y > 5
y[high]
```

Instructions

- Create a boolean **numpy** array: the element of the array should be True if the corresponding baseball player's BMI is below 21. You can use the **<** operator for this. Name the array light.
- Print the array light.
- Print out a **numpy** array with the BMIs of all baseball players whose BMI is below 21.
  Use light inside square brackets to do a selection on the bmi array

```
# Calculate the BMI: bmi
np_height_m = np.array(height_in) * 0.0254
np_weight_kg = np.array(weight_lb) * 0.453592
bmi = np_weight_kg / np_height_m ** 2

# Create the light array


# Print out light


# Print out BMIs of all baseball players
# whose BMI is below 21
```

NumPy Side Effects

numpy arrays cannot contain elements with different types. If you try to build such a list, some of the elements' types are changed to end up with a homogeneous list. This is known as *type coercion*.

The typical arithmetic operators, such as `+`, `-`, `*` and `/` have a different meaning for regular Python lists and numpy arrays.

Instructions

Have a look at this line of code:

```
np.array([True, 1, 2]) + np.array([3, 4, False])
```

Can you tell which code is exactly the same object?

- `np.array([True, 1, 2, 3, 4, False])`
- `np.array([4, 3, 0]) + np.array([0, 2, 2])`
- `np.array([1, 1, 2]) + np.array([3, 4, -1])`

Subsetting NumPy Arrays

Using the square bracket notation on lists or arrays works exactly the same. Try the following code:

```
x = ["a", "b", "c"]
x[1]

np_x = np.array(x)
np_x[1]
```

Instructions

- Subset np_weight_lb by printing out the element at index 5.
- Print out a sub-array of np_height_in that contains the elements at index 10 up to **and including** index 15.

```
# Store weight and height lists as numpy arrays
np_weight_lb = np.array(weight_lb)
np_height_in = np.array(height_in)

# Print out the weight at index 5


# Print out sub-array of np_height_in:
# index 10 up to and including index 15
```

2D NumPy Arrays

INTRODUCTION TO PYTHON

Hoai Thuan TRAN
Gia Dinh University

Type of NumPy Arrays

```
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
type(np_height)
```

```
numpy.ndarray
```

```
type(np_weight)
```

```
numpy.ndarray
```

2D NumPy Arrays

```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, 68.7]])
np_2d
```

```
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],
       [65.4 , 59.2 , 63.6 , 88.4 , 68.7 ]])
```

```
np_2d.shape
```

```
(2, 5) # 2 rows, 5 columns
```

```
np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
          [65.4, 59.2, 63.6, 88.4, "68.7"]])
```

```
array([['1.73', '1.68', '1.71', '1.89', '1.79'],
       ['65.4', '59.2', '63.6', '88.4', '68.7']], dtype='<U32')
```

Subsetting

```
          0       1       2       3       4

array([[  1.73,   1.68,   1.71,   1.89,   1.79],      0
       [  65.4,   59.2,   63.6,   88.4,   68.7]])     1
```

```
np_2d[0]
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

Subsetting

```
          0       1       2       3       4

array([[  1.73,   1.68,   1.71,   1.89,   1.79],      0
       [  65.4,   59.2,   63.6,   88.4,   68.7]])     1
```

```
np_2d[0][2]
```

```
1.71
```

```
np_2d[0, 2]
```

```
1.71
```

Subsetting

```
          0       1       2       3       4

array([[  1.73,   1.68,   1.71,   1.89,   1.79],      0
       [  65.4,   59.2,   63.6,   88.4,   68.7]])     1
```

```
np_2d[:, 1:3]
```

```
array([[ 1.68,  1.71],
       [59.2 ,  63.6 ]])
```

```
np_2d[1, :]
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

Let's practice!

INTRODUCTION TO PYTHON

Yours First 2D NumPy Array

In this exercise, baseball is a list of lists. The main list contains 4 elements. Each of these elements is a list containing the height and the weight of 4 baseball players.

Instructions

- Use np.array() to create a 2D numpy array from baseball. Name it np_baseball.
- Print out the type of np_baseball.
- Print out the shape attribute of np_baseball. Use **np_baseball.shape**

```
# Import numpy
import numpy as np

# Create baseball, a list of lists
baseball = [[180, 78.4],
            [215, 102.7],
            [210, 98.5],
            [188, 75.2]]

# Create a 2D numpy array: np_baseball


# Print out the type of np_baseball


# Print out the shape of np_baseball
```

Subsetting 2D NumPy Array

If your 2D numpy array has a regular structure, i.e. each row and column has a fixed number of values, complicated ways of subsetting become very easy. Have a look at the code below where the elements "a" and "c" are extracted from a list of lists.

```
# regular list of lists
x = [["a", "b"], ["c", "d"]]
[x[0][0], x[1][0]]

# numpy
import numpy as np
np_x = np.array(x)
np_x[:, 0]
```

Instructions

- Print out the 3th row of np_baseball.
- Make a new variable, np_weight_lb, containing the entire second column of np_baseball.
- Select the height (first column) of the 4th baseball player in np_baseball and print it out.

```
# Create np_baseball (2 cols)
np_baseball = np.array(baseball)

# Print out the 3th row of np_baseball


# Select the entire second column of np_baseball:
np_weight_lb


# Print out height of 4th player
```

2D Arithmetic

Execute the code below in your Python script and see if you understand:

```
import numpy as np
np_mat = np.array([[1, 2],
                   [3, 4],
                   [5, 6]])
print(np_mat * 2)
print(np_mat + np.array([10, 10]))
print(np_mat + np_mat)
```

Instructions

- Add np_baseball and np_updated and print out the result.
- Create a numpy array with two values: 0.0254, 0.453592. Name this array conversion.
- Multiply np_baseball with conversion and print out the result.

```
updated = [[5, 8.6],[-3, 2.7],
           [2, 2.5],[1, -1.2]]

# Create np_baseball and np_updated

# Print out addition of np_baseball and updated

# Create numpy array: conversion


# Print out product of np_baseball and conversion
```

NumPy: Basic Statistics

INTRODUCTION TO PYTHON

Hoai Thuan TRAN
Gia Dinh University

Data analysis

- Get to know your data
- Little data ->simply look at it
- Big data ->?

City-wide survey

```
import numpy as np
np_city = ... # Implementation left out
np_city
```

```
array([[1.64, 71.78],
       [1.37, 63.35],
       [1.6 , 55.09],
       ...,
       [2.04, 74.85],
       [2.04, 68.72],
       [2.01, 73.57]])
```

NumPy

```
np.mean(np_city[:, 0])
```

```
1.7472
```

```
np.median(np_city[:, 0])
```

```
1.75
```

NumPy

```
np.corrcoef(np_city[:, 0], np_city[:, 1])
```

```
array([[ 1.     , -0.01802],
       [-0.01803,  1.     ]])
```

```
np.std(np_city[:, 0])
```

```
0.1992
```

- sum(), sort(), ...
- Enforce single data type: speed!

Generate data

- Arguments for `np.random.normal()`
  - distribution mean
  - distribution standard deviation
  - number of samples

```
height = np.round(np.random.normal(1.75,  0.20, 5000), 2)

weight = np.round(np.random.normal(60.32,  15, 5000), 2)

np_city = np.column_stack((height, weight))
```

Let's practice!

INTRODUCTION TO PYTHON

Explore the baseball data

- Create numpy array np_height_in that is equal to first column of np_baseball.
- Print out the mean, the median, the standard deviation using np.std() of np_height_in.
- Use np.corrcoef() to store the correlation between the first and second column of np_baseball in corr.

```
# Create np_height_in from np_baseball

# Print mean height
avg = ...
print("Average: " + str(avg))

# Print median height.
med = ...
print("Median: " + str(med))

# Print out the standard deviation on height.
stddev = ...
print("Standard Deviation: " + str(stddev))

# Print out correlation between first and second column.
corr = ...
print("Correlation: " + str(corr))
```

Blend it all together

- Convert heights and positions, which are regular lists, to numpy arrays. Call them np_heights and np_positions.
- Extract all the heights of the goalkeepers. You can use a little trick here: use **np_positions == 'GK'** as an index for np_heights. Assign the result to gk_heights.
- Extract all the heights of all the other players. This time use **np_positions != 'GK'** as an index for np_heights. Assign the result to other_heights.
- Print out the median height of the attackers and other players using np.median().

```
# Import numpy
import numpy as np

heights   = [191,184,185,180,181,187,170,
             183,186,185,170,187,183,173,190]
positions = ['GK','M','A','D','M','D','M',
             'M','M','A','M','M','A','A','GK']
# Convert positions and heights to numpy arrays:
np_positions, np_heights

# Heights of the goalkeepers (GK): gk_heights

# Heights of the other players: other_heights

# Print out the median height of attackers (A).
print("Median height of attackers: " + ....)

# Print out the median height of other players.
print("Median height of other players: " + ...)
```