

Python: Intro - Filip Schouwenaars

Python: Intro - Filip Schouwenaars

Note as of July 2019

Note Taker: Paris Zhang

Chapters:

1. Python Basics
2. Python Lists
3. Functions and Packages
4. NumPy

Ch 1 - Python Basics

- Background:
 - Created by Guido Van Rossum
 - Text file - .py
- Variables and types
 - Case-sensitive
 - `type(): float, int, str, bool`

Example:

```
In [1]: type(bmi)
```

```
Out[2]: float
```

- Different types = different behavior

Example:

```
In [3]: 2 + 3
```

```
Out[4]: 5
```

```
In [5]: 'ab' + 'cd'
```

```
Out[6]: 'abcd'
```

Ch 2 - Python Lists

- List types: contains any type and different types

Example:

```
In [1]: fam = [1, 2, 3, 4]
```

```
In [11]: fam2 = [{"liz", 1.73},
                 ["emma", 1.68],
                 ["mom", 1.71],
                 ["dad", 1.89]]
```

```
In [12]: fam2
Out[12]: [['liz', 1.73], ['emma', 1.68],
          ['mom', 1.71], ['dad', 1.89]]
```

- Subsetting lists
 - Subset
 - *zero-based indexing (index starts with 0)
 - `fam[0]` - first item
 - `fam[1]` - second item
 - `fam[-1]` - last item

- Slicing

- Syntax:

- [start : end] - start is inclusive, end is exclusive

- fam[:4], fam[5:] - when leaving blank, means all.

- (1) fam[:4] = 1st (index 0) to 4th (index 3) elements;

- (2) fam[5:] = 6th (index 5) to the rest of elements;

- (3) fam[:] = select all elements;

- (4) fam[-4:] = last 4 elements

- Example:

```
In [7]: fam
Out[7]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
        0      1      2      3      4      5      6      7

In [8]: fam[3:5]
Out[8]: [1.68, 'mom']

In [9]: fam[1:4]
Out[9]: [1.73, 'emma', 1.68]

In [10]: fam[:4]
Out[10]: ['liz', 1.73, 'emma', 1.68]

In [11]: fam[5:]
Out[11]: [1.71, 'dad', 1.89]
```

- List manipulation

- Replace list elements

- In [5]: fam[0:2] = ["lisa", 1.74]

- Adding elements

- fam + ["me", 1.79]

- Removing elements

- del(fam[2])

- Inner working of lists (behind the scenes)

- If $y = x$, the change of y also changes x since they represent a reference to the original list;

- If $y = \text{list}(x)$ or $y = x[:]$, then change of y doesn't affect x .

Ch 3 - Functions and Packages

- Functions

- Basic functions

- max()

- round(number, ndigits) # decimal point is 0 by default

- help() # open up documentation

- Methods: functions that belong to objects

- String methods

- (str, float, list are called Python objects)

- str - capitalize(), replace()

```
In [7]: sister
Out[7]: 'liz'

In [8]: sister.capitalize()
Out[8]: 'Liz'
```

```
In [9]: sister.replace("z", "sa")
Out[9]: 'lisa'
```

- float - `bit_length()`, `conjugate()`

- list - `index()`, `count()`

list - `append()`, `remove()`, `reverse()` # will change the list they're called on.

- List methods

(To call the method, use ".")

- Examples

```
In [4]: fam
Out[4]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]

In [5]: fam.index("mom")
Out[5]: 4
```

Call method index() on fam

```
In [6]: fam.count(1.73)
Out[6]: 1
```

- More examples

```
In [15]: fam.append("me")

In [16]: fam
Out[16]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']

In [17]: fam.append(1.79)

In [18]: fam
Out[18]: ['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me', 1.79]
```

- Summary

- Functions

```
In [11]: type(fam)
```

```
Out[11]: list
```

- Methods: call functions on objects

```
In [12]: fam.index("dad")
```

```
Out[12]: 6
```

- Packages

- Packages for data science

- **NumPy**- to efficiently work with arrays

- **Matplotlib**- for data visualization

- **Scikit-learn** - for machine learning

- Install package

- Download `get-pip.py`

- Terminal

- `python3 get-pip.py`

- `pip3 install numpy`

- Example

```
In [1]: import numpy

In [2]: array([1, 2, 3])
NameError: name 'array' is not defined

In [3]: numpy.array([1, 2, 3])
Out[3]: array([1, 2, 3])

In [4]: import numpy as np

In [5]: np.array([1, 2, 3])
```

```
Out[5]: array([1, 2, 3])

In [6]: from numpy import array

In [7]: array([1, 2, 3])
Out[7]: array([1, 2, 3])
```

- Here, `from numpy import array` only imports the `array` function; this is called a selective import.
A courtesy note: it is a good practice to keep the standard way, i.e., using `numpy.array()` so that people reading the script can easily locate the package of a function.
- More example
`from scipy.linalg import inv as my_inv`

Ch 4 - NumPy

- Array calculation solution: NumPy arrays
 - Installation (terminal): `pip3 install numpy`
- Comparison

```
In [13]: height = [1.73, 1.68, 1.71, 1.89, 1.79]

In [14]: weight = [65.4, 59.2, 63.6, 88.4, 68.7]

In [15]: weight / height ** 2
TypeError: unsupported operand type(s) for **: 'list' and 'int'

In [16]: np_height = np.array(height)

In [17]: np_weight = np.array(weight)

In [18]: np_weight / np_height ** 2
Out[18]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

- Example

```
In [6]: import numpy as np
Element-wise calculations

In [7]: np_height = np.array(height)

In [8]: np_height
Out[8]: array([ 1.73,  1.68,  1.71,  1.89,  1.79])

In [9]: np_weight = np.array(weight)

In [10]: np_weight
Out[10]: array([ 65.4,  59.2,  63.6,  88.4,  68.7])

In [11]: bmi = np_weight / np_height ** 2

In [12]: bmi
Out[12]: array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])

= 65.5/1.73 ** 2
```

- NumPy remarks
 - NumPy arrays contain only one type

```
In [19]: np.array([1.0, "is", True])
Out[19]:
array(['1.0', 'is', 'True'],
      dtype='<U32')
```

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

```
In [20]: python_list = [1, 2, 3]

In [21]: numpy_array = np.array([1, 2, 3])
```

```
In [22]: python_list + python_list
Out[22]: [1, 2, 3, 1, 2, 3]
```

```
In [23]: numpy_array + numpy_array
Out[23]: array([2, 4, 6])
```

- 2D NumPy Arrays

- Type of NumPy arrays

```
In [4]: type(np_height)
```

```
Out[4]: numpy.ndarray # ndarray = N-dimensional array
```

- Shape of 2D NumPy arrays

```
In [6]: np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                          [65.4, 59.2, 63.6, 88.4, 68.7]])
```

```
In [7]: np_2d
Out[7]:
array([[ 1.73,   1.68,   1.71,   1.89,   1.79],
       [ 65.4,   59.2,   63.6,   88.4,   68.7]])
```

```
In [8]: np_2d.shape
Out[8]: (2, 5) 2 rows, 5 columns
```

```
In [9]: np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
                  [65.4, 59.2, 63.6, 88.4, "68.7"]])
Out[9]:
array([[ '1.73', '1.68', '1.71', '1.89', '1.79'],
       [ '65.4', '59.2', '63.6', '88.4', '68.7']],
      dtype='<U32') Single type!
```

- 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

```
In [10]: np_2d[0]
Out[10]: array([ 1.73,   1.68,   1.71,   1.89,   1.79])
```

```
In [11]: np_2d[0][2]
Out[11]: 1.71
```

```
In [12]: np_2d[0,2]
Out[12]: 1.71
```

```
In [13]: np_2d[:,1:3]
Out[13]:
array([[ 1.68,   1.71],
       [ 59.2,   63.6]])
```

```
In [14]: np_2d[1,:]
Out[14]: array([ 65.4,   59.2,   63.6,   88.4,   68.7])
```

- Example: both syntax returns "a" and "c"

```
# 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]
```

- NumPy: basic statistics

- np.mean()

- `np.median()`
- `np.corrcoef(np_city[:,0], np_city[:,1])`
- `np.std()`
- `np.sum()`
- `np.sort()`

- Generate data

Generate data

distribution mean distribution standard dev. number of samples

```
In [8]: height = np.round(np.random.normal(1.75, 0.20, 5000), 2)
In [9]: weight = np.round(np.random.normal(60.32, 15, 5000), 2)
In [10]: np_city = np.column_stack((height, weight))
```