Python for Data Analysis lecture 1: Python Language Basics

SULTAN ALPAR, PHD, ASSOCIATE PROFESSOR

S.ALPAR@IITU.EDU.KZ

What is this course about?

What Kinds of Data?

Why Python for Data Analysis?

Why Python for Data Analysis?

In the last 10 years, Python has gone from a bleeding-edge or “at your own risk” scientific computing language to one of the most important languages for data science, machine learning, and general software development in academia and industry.

Essential Python Libraries

NumPy

Pandas

Matplotlib

SciPy

NumPy

Short for Numerical Python

Contains:

- A fast and efficient multidimensional array object ndarray
- Functions for performing element-wise computations with arrays or mathematical operations between arrays
- Tools for reading and writing array-based datasets to disk
- Linear algebra operations, Fourier transform, and random number generation
- A mature C API to enable Python extensions and native C or C++ code to access NumPy's data structures and computational facilities

pandas

Provides high-level data structures and functions designed to make working with structured or tabular data fast, easy and expressive.

pandas blends the high-performance, array-computing ideas of NumPy with the flexible data manipulation capabilities of spreadsheets and relational databases.

- Data structures with labeled axes supporting automatic or explicit data alignment —this prevents common errors resulting from misaligned data and working with differently indexed data coming from different sources
- Integrated time series functionality
- The same data structures handle both time series data and non–time series data
- Arithmetic operations and reductions that preserve metadata
- Flexible handling of missing data
- Merge and other relational operations found in popular databases (SQL-based, for example)

matplotlib

matplotlib is the most popular Python library for producing plots and other two-dimensional data visualizations

While there are other visualization libraries available to Python programmers, matplotlib is the most widely used and as such has generally good integration with the rest of the ecosystem.

SciPy

SciPy is a collection of packages addressing a number of different standard problem domains in scientific computing. Here is a sampling of the packages included:

command	description
scipy.integrate	Numerical integration routines and differential equation solvers
scipy.linalg	Linear algebra routines and matrix decompositions extending beyond those provided in numpy.linalg
scipy.optimize	Function optimizers (minimizers) and root finding algorithms
scipy.signal	Signal processing tools
scipy.sparse	sparse Sparse matrices and sparse linear system solvers
scipy.special	Wrapper around SPECFUN, a Fortran library implementing many common mathematical functions, such as the gamma function
scipy.stats	Standard continuous and discrete probability distributions (density functions, samplers, continuous distribution functions), various statistical tests, and more descriptive statistics

statsmodels

statsmodels is a statistical analysis package, contains algorithms for classical statistics and econometrics.

This includes such submodules as:

- Regression models: Linear regression, generalized linear models, robust linear models, linear mixed effects models, etc.
- Analysis of variance (ANOVA)
- Time series analysis: AR, ARMA, ARIMA, VAR, and other models
- Nonparametric methods: Kernel density estimation, kernel regression
- Visualization of statistical model results

Python Language Basics

Introspection (?)

```
def add_numbers(a, b):
    """
    Add two numbers together

    Returns
    -------
    the_sum : type of arguments
    """
    return a + b
```

```
In [11]: add_numbers?
Signature: add_numbers(a, b)
Docstring:
Add two numbers together

Returns
-------
the_sum : type of arguments
File:      <ipython-input-9-6a548a216e27>
Type:      function
```

Magic Commands

Command	Description
`%quickref`	Display the IPython Quick Reference Card
`%magic`	Display detailed documentation for all of the available magic commands
`%debug`	Enter the interactive debugger at the bottom of the last exception traceback
`%hist`	Print command input (and optionally output) history
`%pdb`	Automatically enter debugger after any exception
`%paste`	Execute preformatted Python code from clipboard
`%cpaste`	Open a special prompt for manually pasting Python code to be executed
`%reset`	Delete all variables/names defined in interactive namespace
`%page` *`OBJECT`*	Pretty-print the object and display it through a pager
`%run` *`script.py`*	Run a Python script inside IPython
`%prun` *`statement`*	Execute *`statement`* with `cProfile` and report the profiler output
`%time` *`statement`*	Report the execution time of a single statement
`%timeit` *`statement`*	Run a statement multiple times to compute an ensemble average execution time; useful for timing code with very short execution time
`%who, %who_ls, %whos`	Display variables defined in interactive namespace, with varying levels of information/ verbosity
`%xdel` *`variable`*	Delete a variable and attempt to clear any references to the object in the IPython internals

Python language Basics

- Python uses whitespace to structure code instead of using braces.
- A colon denotes the start of an indented code block after which all of the code must be indented by the same amount until the end of the block.
- Python statements do not need to be terminated by semicolons. Semicolons can be used, however, to separate multiple statements on a single line.
- Everything is an object
- When you pass objects as arguments to a function, new local variables are created referencing the original objects without any copying.

Binary Operators

Operation	Description
a + b	Add a and b
a - b	Subtract b from a
a * b	Multiply a by b
a / b	Divide a by b
a // b	Floor-divide a by b, dropping any fractional remainder
a ** b	Raise a to the b power
a & b	True if both a and b are True; for integers, take the bitwise AND
a | b	True if either a or b is True; for integers, take the bitwise OR
a ^ b	For booleans, True if a or b is True, but not both; for integers, take the bitwise EXCLUSIVE-OR

Mutable and immutable objects

Most objects in Python, such as lists, dicts, NumPy arrays, and most user-defined types (classes), are mutable. This means that the object or values that they contain can be modified

```
In [43]: a_list = ['foo', 2, [4, 5]]

In [44]: a_list[2] = (3, 4)

In [45]: a_list
Out[45]: ['foo', 2, (3, 4)]
```

Others, like strings and tuples,

are immutable

Scalar Types

Type	Description
`None`	The Python "null" value (only one instance of the `None` object exists)
`str`	String type; holds Unicode (UTF-8 encoded) strings
`bytes`	Raw ASCII bytes (or Unicode encoded as bytes)
`float`	Double-precision (64-bit) floating-point number (note there is no separate `double` type)
`bool`	A `True` or `False` value
`int`	Arbitrary precision signed integer

Numeric types

Integer division not resulting in a whole number will always yield a floating-point number:

```
In [52]: 3 / 2
Out[52]: 1.5
```

To get C-style integer division (which drops the fractional part if the result is not a whole number), use the floor division operator //:

```
In [53]: 3 // 2
Out[53]: 1
```

Strings

You can write string literals using either single quotes ' or double quotes “

For multiline strings with line breaks, you can use triple quotes, either ''' or """“

Python strings are immutable; you cannot modify a string

Many Python objects can be converted to a string using the str function

The backslash character \ is an escape character, meaning that it is used to specify special characters like newline \n or Unicode characters

Adding two strings together concatenates them and produces a new string

None

None is the Python null value type. If a function does not explicitly return a value, it implicitly returns None:

```
In [97]: a = None

In [98]: a is None
Out[98]: True

In [99]: b = 5

In [100]: b is not None
Out[100]: True
```

Control Flow

If, elif, and else

```
if x < 0:
    print('It's negative')
elif x == 0:
    print('Equal to zero')
elif 0 < x < 5:
    print('Positive but smaller than 5')
else:
    print('Positive and larger than or equal to 5')
```

Loops

for loops

```
for value in collection:
    # do something with value
```

while loops

A while loop specifies a condition and a block of code that is to be executed until the condition evaluates to False or the loop is explicitly ended with break

Pass

n. It can be used in blocks where no action is to be taken (or as a placeholder for code not yet implemented); it is only required because Python uses whitespace to delimit blocks

Range

The range function returns an iterator that yields a sequence of evenly spaced integers

Ternary expressions

A ternary expression in Python allows you to combine an if-else block that pro- duces a value into a single line or expression. The syntax for this in Python is:

```
value = true-expr if condition else false-expr
```

Here, true-expr and false-expr can be any Python expressions. It has the identical effect as the more verbose:

```
if condition:
    value = true-expr
else:
    value = false-expr
```