Session 1b:

An introduction to Python

Andreas Bjerre-Nielsen

Agenda

Introducing Python

Introduction

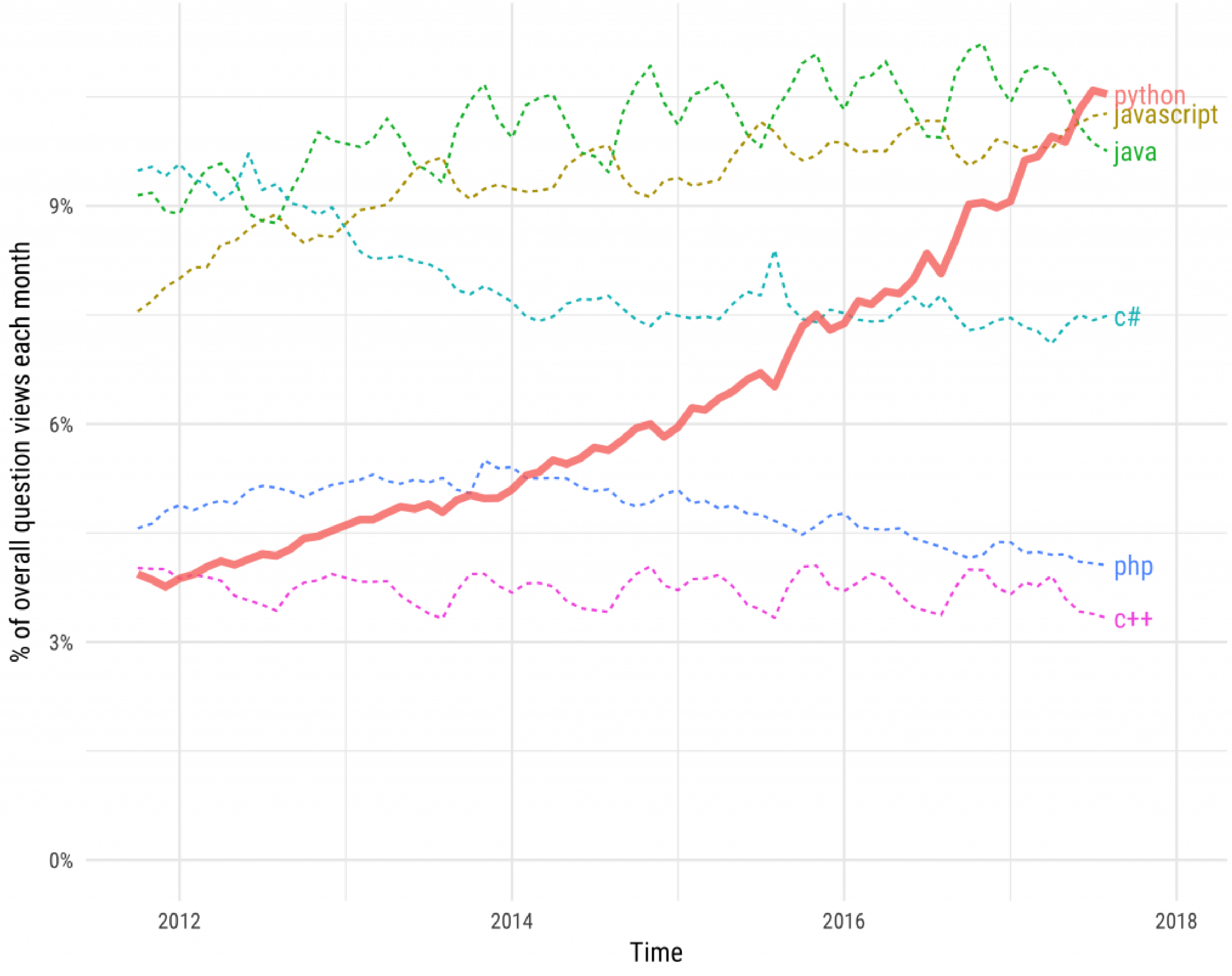
Why do we use Python?

- It can do "anything" and used everywhere (https://www.python.org/about/success/)
    - High-tech manufacturing
    - Space shuttles
    - Large servers
- Python has incredible resources for machine learning, big data, visualizations.

Introduction (2)

Growth of major programming languages

Based on Stack Overflow question views in World Bank high-income countries



Introduction (3)

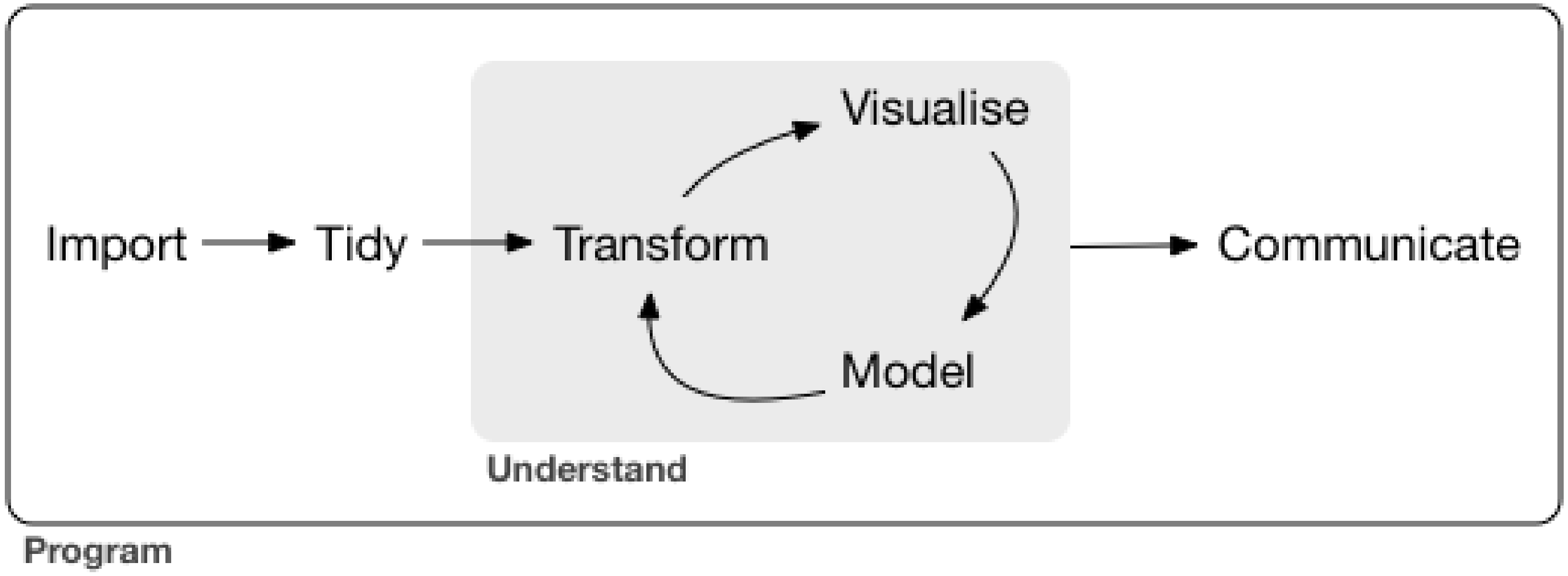
What is Python?

A general purpose programming language.

- Can do everything you can imagine a computer can do.
- E.g. manage databases, advanced computation, web etc.

Introduction (4)

How does data science work?



Introduction (5)

What are we going to learn?

Course competencies:

- Import: scraping and data IO
- Tidy / transform: data structuring and text
- Visualize: plotting
- Model: machine learning
- Communicate: markdown

Help and advice

Learning how to code (1)

This course.. ain't easy..

Why would you go through this pain? You choose one of two paths.

i. You move on, you forget some or most of the material.

ii. You are lit and your life has changed.

- You may return to become a better sociologist, economist etc.
- Or, you may continue along the new track of data science.
- In any case, you keep learning and expanding your programming skills.

Learning how to code (2)

Hadley Wickham

> *The bad news is that when ever you learn a new skill you’re going to suck. It’s going to be frustrating. The good news is that is typical and happens to everyone and it is only temporary. You can’t go from knowing nothing to becoming an expert without going through a period of great frustration and great suckiness.*

Learning how to code (3)

Kosuke Imai

One can learn data analysis only by doing, not by reading.

Learning how to code (4)

Practical advice

- Do not use the console, write scripts or preferably notebooks instead
- Be lazy: resuse code and write reusable code (functions)
- Think before you code
- Code is a medium of communication
    1. Between you and the computer
    2. Between you and other people (or future you)

Learning how to code (5)

How do we participate optimally?

- Practice, practice and more practice.
- Try everything on your own computer
- Type the code in yourself,
    - Word-by-word, line-by-line.
    - DO NOT copy-paste.

Guide on getting help

Whenever you have a question you do as follows:

1: You ask other people in your group.

2: You ask the neighboring groups.

3: You search on Google (more advice will follow); then ask us.

4: You ask us.

The python shell and scripts

Python interpreter (1)

Shell access

The fundamental way of accessing Python is from your shell by typing *`python`*.

Everyone should be able to run the following commands and reproduce the output.

```
>>> print ('hello my friend')
hello my friend
```

```
>>> 4*5
20
```

Python interpreter (2)

Python scripts

The power of the interpreter is that it can be used to execute Python scripts.

What is a script?

These are programs containing code blocks.

Python interpreter (3)

Everyone should be able to make a text file called *`test.py`* in their current folder. The file should contain the following two lines:

```
print ('Line 1')
print ('Line 2')
```

Try executing the test file from the shell by typing:

```
python test.py
```

This should yield the following output:

```
Line 1
Line 2
```

The Jupyter framework

Jupyter (1)

What is Jupyter Notebook?

- Jupyter provides an interactive and visual platform for working with data.
- It is an abbreviation of Julia, Python, and R.

Why Jupyter?

- great for writing.
    - markdown, equations and direct visual output;
- interactive allows keeping data etc.
- many tools (e.g. create this slideshow)

Jupyter (2)

How do we create a Jupyter Notebook?

We start Jupyter Notebook by typing `jupyter notebook` in the shell.

Try making a new notebook:

- click the button `New` in the upper right corner
- clicking on `Python 3`.

Jupyter (3)

How do we interact with Jupyter?

Jupyter works by having cells in which you can put code. The active cell has a colored bar next to it.

A cell is *`edit mode`* when there is a *green* bar to the left. To activate *`edit mode`* click on a cell.

A cell is in *`command mode`* when the bar to the left is *blue*.

Jupyter (4)

How do we add and execute cude?

Go into edit mode - add the following:

```
In [ ]: A = 11
        B = 15
        A+B
```

Click the ► to run the code in the cell. What happens if we change A+B to A*B?

Jupyter (5)

How can we add cells to our notebook?

Try creating a new cell by clicking the **+** symbol.

Jupyter (6)

Most relevant keyboard short cuts

editing and executing cells

- enter edit mode: click inside the cell or press `ENTR`
- exit edit mode: click outside cell or press `ESC`.
- executing code within a cell is `SHFT+ENTR` or `CTRL+ENTR` (not same!)

adding removing cells (command mode only)

- delete a cell:`d`,`d` (press `d` twice)
- add cell: `a` above, `b` below

See all Jupyter keyboard shortcuts in menu (top): `Help > Keyboard Shortcuts`, or press `H` in command mode.

Jupyter (7)

What if I need more information?

Further resources can be found in the documentation and tutorial available here (http://jupyter.readthedocs.io/en/latest/).

The Python language

Fundamental data types

Data types (1)

What are the most basic data types in Python?

Python supports many different data types. The four most basic are as follows.

- numbers come in two flavors
  - floating point numbers (`float`), e.g.: 3.14, 0.011
  - integers (`int`), e.g.: 1, 3, 8
- strings such as 'e', 'Wow', 'dO' (`str`);
- boolean (`bool`) which is either `True` or `False`.

Data types (2)

We can store data as a variable X with one equal symbol, i.e. '='

Try creating a variable A which equals 1.5 by typing

```
In [ ]: A = 1.5
```

Convert A to integer by typing:

```
In [ ]: int(A) # rounds down, i.e. floor
```

We can do the same for converting to `float`, `str`, `bool`. Note some conversion are not allowed.

Data types (3)

What is an object in Python?

- A thing, anything - everything is an object.

Why use objects?

- Easy manipulable, powerful methods and flexible attributes.
- We can make complex objects, e.g. estimation methods quite easy.
- Example of a float method:

```
In [ ]: A.as_integer_ratio()
```

Print and debug

Basic printing (1)

An essential procedure in Python is `print`.

Try executing some code:

```
In [ ]: my_str = 'I can do any THING in Python !#'
        print(my_str)
```

```
In [ ]: my_var1 = 33
        my_var2 = 2
        print(my_var1, my_var2)
```

Basic printing (2)

Why do we print?

- Can inspect values.
- Espcially intermediate output (within function, .

Debugging (1)

What happens if my code has errors?

Try executing the following code block:

```
In [ ]: float('a')
```

Interpretation: The output message tells us that the string `'a'` cannot be converted to a floating number, which makes sense.

Debugging (2)

What if I don't understand the error?

Ask for help - search Google:

- Look at the answers in this stackoverflow post: https://stackoverflow.com/questions/8420143 (https://stackoverflow.com/questions/8420143).
- An explanation by Blender:

> *Somewhere in your text file, a line has the word `id` in it, which can't really be converted to a number.*

Operators

Operators (1)

What computations can python do?

An operator in Python manipulates various data types.

We have seen addition +. Other basic numeric operators:

- multiplication *;
- subtraction -;
- division /;
- power, **.

Operators (2)

How can we test an expression in Python?

We can check the validity of a statement - using the equal operator, **==**, or not equal operator **!=**.

```
In [ ]: #3 == (2 + 1)
        #20 == 4 * 5
        21 == 4 * 5
```

We also do numeric comparisons, greater than >, greater than or equal >= etc.

Operators (3)

How can we manipulate boolean values?

Combining boolean values can be done with using:

- the **and** operator - equivalent to **&**
- the **or** operator - equivalent to **|**

Let's try this!

In [ ]:
```
print(True | False)
#print(True & False)
```

Operators (4)

What other things can we do?

We can negate/reverse the statement with the `not` operator:

```
In [ ]: not (True and True)
```

Conditional logic

Conditional execution of code.

Conditionals (1)

How can we activate code based on data?

A conditional execution of code, if a statement is true then active code.

In Python the syntax is easy with the `if` syntex:

In [ ]:
```
if statement:
    code
```

- statement is either a variable or an expression
- if statement is `True` then execute a code block

Conditionals (2)

Examples using the equal and unequal operators:

```
In [ ]: if 4 == 4:
            print ("I'm being executed, yay!")
```

```
In [ ]: my_statement = (5 == 4)
        if my_statement:
            print ("I'm not being executed.!")
```

Conditionals (3)

If the statement in our condition is false then we can execute other code with the `else` statement:

In [ ]:
```
if statement:
    code
else:
    alternative code
```

Try running the following command.

In [ ]:
```
if (5 == 4):
    print ('true!')
else:
    print ('false..')
```

Conditionals (4)

Quiz: What differentiates a code block in conditional statements?

By indenting the line with four whitespaces, example if A then B:

```
In [ ]: if A:
            B
```

Containers

Lists (1)

How can we make a containter which is sequential and accessed by integers?

- One answer is the **list** data type. A list is a container for data. In terms of math it is an ordered array / vector.

A list named 'B' can contain the sequence of arbitrary elements 1,2,3,5. Try constructing this sequence as:

```
In [ ]: B = [1,2,3,5]
```

Lists (2)

Why are lists so useful?

1) Lists can be manipulated in multiple ways.

- adding new elements with `append`, removing elements with `remove`
- changing objects contained with specific indices

2) We can do all sort of things with lists:

- we can make basic computation with lists, e.g. `max`, `min`
- we can sort the order of lists with `sorted`
- we can check if a certain object is in the list

Lists (3)

Quiz: can a list be empty, i.e. have no elements?

Yes, empty lists are useful as placeholders.

Other containers

There are many useful concepts for containers that we recommend looking at:

- list comprehension (http://www.python-course.eu/python3_list_comprehension.php) for fast to make one-line for loops;
- tuples, which are index based containers that are *immutable*;
- generators, that are ordered containers without indices.

Loops

A `loop` is to run a repeated process.

For loops

Why are lists so powerful?

Lists can be used to iterate over its elements - this created a *finite* loop, called the `for` loop.

Example - try the following code:

```
In [ ]: A = []

        for i in B:
            i_squared = i**2
            A.append(i_squared)

        print(A)
```

For loops are smart when: iterating over files in a directory; iterating over specific set of columns.

How does Python know where the code associated with inside of the loop begins?

While loops

Can we make a loop without specifying the end?

Yes, this is called a `while` loop. Example - try the following code:

```
In [ ]: i = 0
        L = []
        while (i<3): # i<3 is condition for while loop to keep going
            L.append(i*2) # add i times 2 to L
            i += 1 # add 1 to i
        print(L)
```

Why is this smart?

- Can be applied in scraping, make processes that keeps running, model which converges, etc.

Reusable code

Why do we reuse code?

- To save time.
- To learn from other or 'borrow' their code.

Functions (1)

What procedures have we seen?

How can we make a reusable procedure?

- We make a Python function with the `def` syntax. Try this:

```
In [ ]: def squared_plus_1(x): # takes input x
            x_sq = x**2 # x squared
            return x_sq + 1

        squared_plus_1(3.1)
```

Functions (2)

A function can also be used to print:

```
In [ ]: def print_cubic(x):
            print(x**3)
```

```
In [ ]: print_cubic(2)
```

Class (1)

How can we make objects with specific attributes and methods?

We define a `class`, i.e. class of objects. A `class` can be used to generate an `instance` of the class.

Classes are useful when we want to build useful tools.

- e.g. for estimation procedures, downloading data

Class (2)

What could a class for chairs look like?

- Chairs have three attributes:
  - A number of legs between 0, 1, 3 or 4 (integer).
  - May have a backrest or not (boolean).
  - May have two armrests or not (boolean).
- Chairs have two functions/methods:
  - We may move the chair to a new location, e.g. change geo-coordinates.
  - We may sit a person if unseated, and remove a person if seated.

Class (3)

What characterizes a Python class?

A `class` in Python are a collection of attributes and methods. We can make objects that are an `instance` of the class which inherit:

- the class' attributes, which are variables associated with the object;
- the class' methods, which are functions that can be applied on the object.

How relevant is this?

- it is useful for any user of Python, provides understanding how everything is built
- allows you to make powerful tools for yourself and make code for others

Modules (1)

How can we load resources made by others in Python?

We load a module. Try importing numpy:

In [ ]:
```
import numpy as np
```

Let's create an `array` with numpy.

In [ ]:
```
row1 = [1,2]
row2 = [3,4]
table = [row1, row2]

np.array(table)
```

Modules (2)

What is a numpy array?

An n-dimensional array with certain available methods. In 2-d it is a matrix, in 3-d it is a tensor.

Objects can have useful attributes and methods, that are built-in. These are accessed using "`.`"

Example, an array can be transposed as follows:

In [ ]:
```
arr.T
```

Final remarks

Summary

In this lecture we learned how to use:

- Python scripts and Jupyter
- Fundamental data types: numeric, string and boolean
- Operators: numerical and logical
- Conditional logic
- Containers with indices
- Loops: for and while
- Reuseable code: functions, classes and modules

More useful material

Due to the scope of the course we cannot cover everything.

We recommend more core Python skills. Consider the following:

- "python-course.eu (http://www.python-course.eu/python3_course.php)" has some good free material;
- "Learn Python the Hard Way" is a great resource but is not free anymore

The end

Return to agenda