Session 1b:

An introduction to Python

Andreas Bjerre-Nielsen

Agenda

1. On Data Science
2. Python an overview:
    - The what, why and how
    - Some advice
    - Scripting and Jupyter
3. The Python language
    - Fundamental data types and debugging
    - Operators and control flow
    - Containers and loops
    - Reusable code

The Academic Quarter

- 9 means 9.15,
- 13 means 13.15 (i.e. 1.15pm)

Why data science

Some trends

- **Data** is increasingly available
- Improved **algorithms** and methods for computation
- Faster and bigger **computers**

Some big successes

- Image and text recognition (e.g. self-driving cars, Google Translate)
- Artificial intelligence (play computer games, poker, chess, game of go)
- Smart services from Silicon valley (virtual assistants, recommendation etc.)

Consequence > data scientists have HIGHEST entry wages

Beyond data science

The skills and ideas of data science are spreading beyond

- machine learning is now gaining traction in
    - statistics
    - theoretical economic modelling
- smart, free tools for working with
    - small and big data on structured (tabular) data
    - unstructured data sources from image, text and social media

Takeaway > data science is useful broadly.

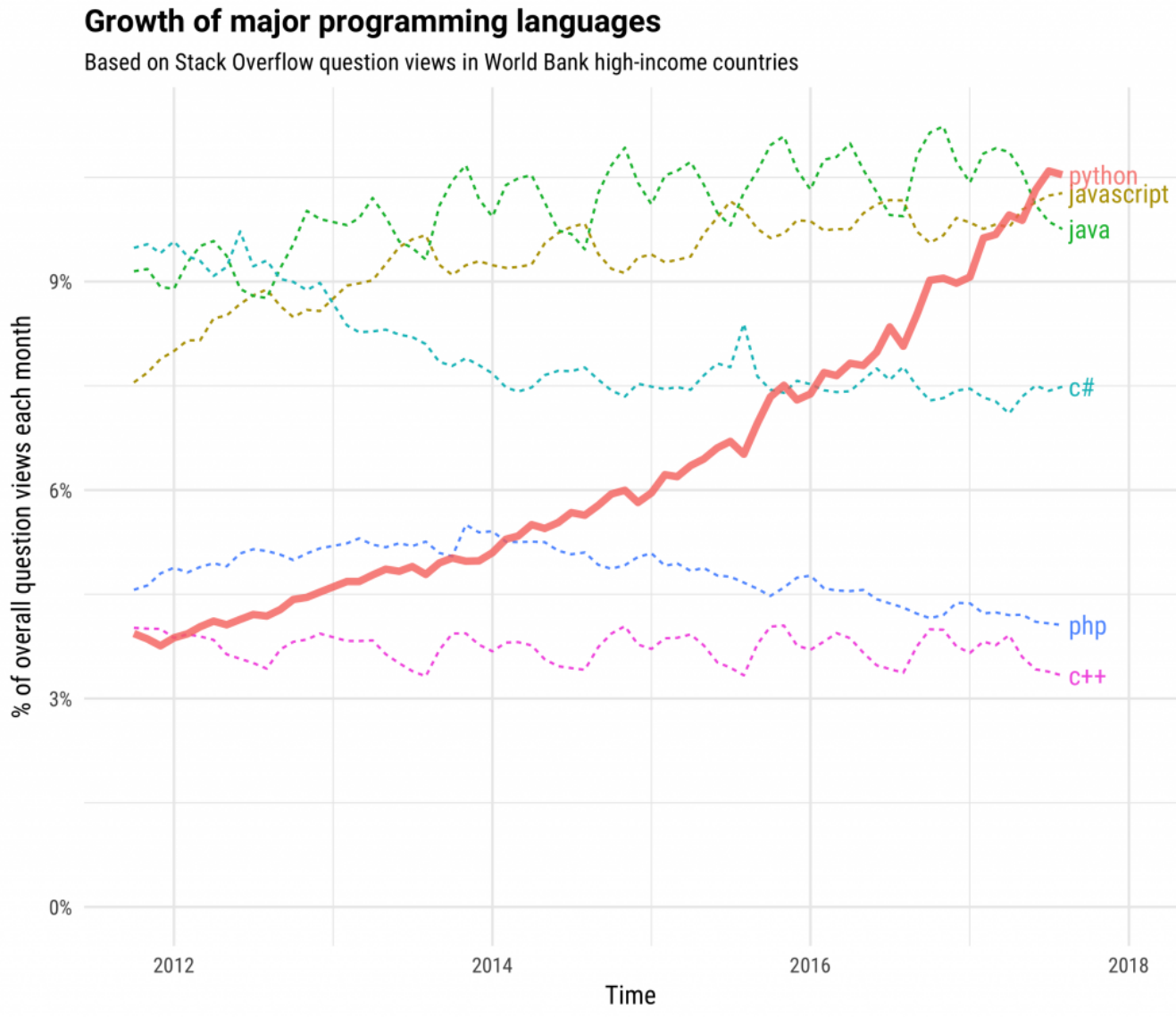
Introducing Python

Introduction

What is Python useful for?

- 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)



Introduction (3)

What is Python?

A multiparadigm, general purpose programming language.

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

Why Python?

Python's main objective is to make programming more ***effortless***.

- This is done by making syntax intuitive.
- A side effect: programming can be fun
- Downside: not the fastest (solved with packages)

Introduction (4)

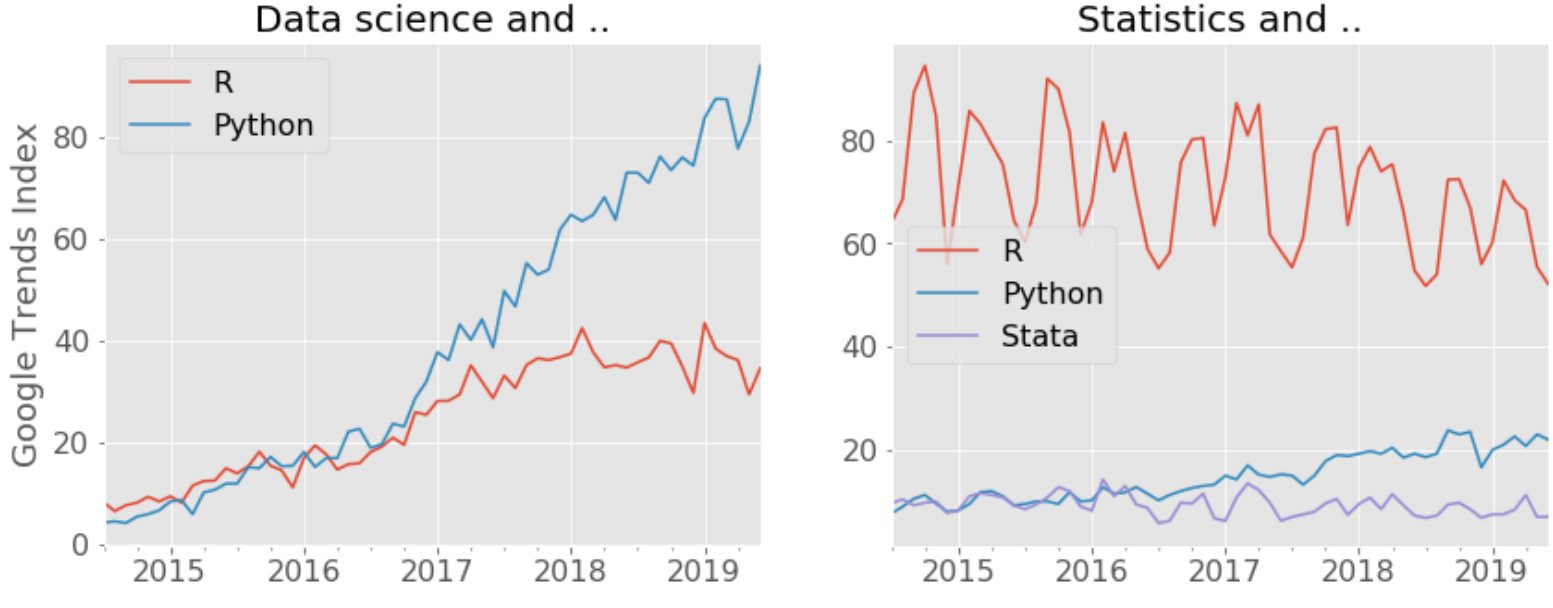
Is Python the most popular for statistics and data science?

There are other good languages, e.g. R, Stata or SAS, why not use them?

- Python has the best data science packages.
- And it is also being used increasingly in statistics.

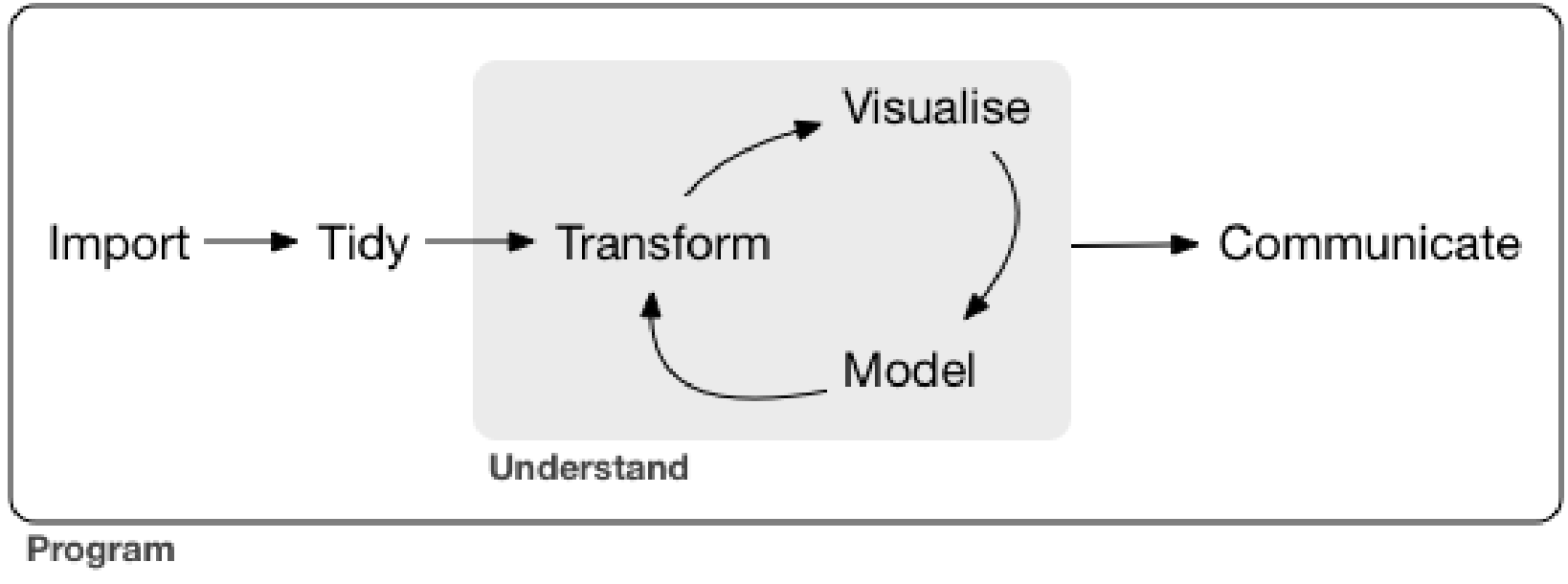
In [4]:
```
f_py_trend
```

Out[4]:



Introduction (5)

How does data science work?



Introduction (6)

What are we going to learn?

Course competencies:

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

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, anthropologit, 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).

4: You raise an issue in our Github repo (https://github.com/abjer/sds/issues) or 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 notebook?

- great for writing.
    - markdown, equations and direct visual output;
- interactive allows keeping, changing 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 = 26

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 cell (`a` above, `b` below) and removing cells (press `d` twice)

More info:

- For tips see blog post (https:abjer.github.io/sds2019/post/jupyter) or see list Jupyter keyboard shortcuts in menu (top): `Help > Keyboard Shortcuts`.
- General resources in documentation and tutorial available here (http://jupyter.readthedocs.io/en/latest/).

The Python language

We begin with a quiz..

Fundamental data types

Data types (1)

Recall the four fundamental data types: `int`, `float`, `str` and `bool`.

- Sometimes know as elementary, primitive or basic.

Some data types we can change between, e.g. between `float` and `int`.

In [ ]:
```
int(1.6) # integer conversion always rounds down, i.e. floor
```

In [ ]:
```
float(int(1.6)) # it does not retake its former value
```

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

Data types (2)

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 [ ]:
```
(1.5).as_integer_ratio()
```

Debugging

Debugging (1)

Code fails all the time!

In [ ]:
```
A='I am a string'
int(A)
print(A)
```

Debugging (2)

How do you fix code errors?

Look at the error message:

1. **Where** is the error? I.e. what linenumber (and which function).
    - Inspect the elements from the failure before the error occurs.
        - Note: if you use a function you may want to try printing elements
    - Try replacing the objects in the line.
2. **What** goes wrong? Examples:
    - `SyntaxError`: spelling error; `ValueError`: datatype mismatch.
    - Hint: reread it several times, search on Google if you do not understand the error.

Debugging (3)

Exercise: investigate the error we incurred

- 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 and control flow

Operators

What computations can python do?

- Numeric operators: Output a numeric value from numeric input.
    - +; *; -; /.
- Comparison operators: Output a boolean value, `True` or `False`
    - ==; != (equal, not equal - input from most object types)
    - >; <. (greater, smaller - input from numeric)
- Logical operators: Output a boolean value from boolean input.
    - `and` / &; `or` / |; `not` / !

Operators (2)

How can we test an expression in Python?

We can check the validity of a statement using comparison operations:

In [ ]:
```
3 == (2 + 1) # other ops: >, !=, >=
```

And apply logical operations:

In [ ]:
```
True | False
```

Control flow

How can we activate code based on data?

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

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

```
if condition:
    (CODE BLOCK LINE 1)
    (CODE BLOCK LINE 2)
    ...
```

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

Control flow (2)

We can use comparison and logical operators directly as they output boolean values.

In [ ]:
```
if 4 == 4:
    print ("I'm being executed, yay!")
else:
    print ("Oh no, I'm not being executed!")
```

Control flow (3)

We can make deep control flow structures:

In [ ]:

```
A = 11
if  A>=0:
    if A==0:
        print ("I'm exactly zero!")

    elif A<10:
        print ("I'm small but positive!")

    else:
        print ("I'm large and positive!")
else:
    print ("Oh shoot, I'm negative!")
```

Containers and loops

Containers

How do we store multiple objects?

- We put objects into containers. (Like a bag)
- An example is a `list` where we can add and remove objects

What are they useful for?

- We can use them to compute statistics (max, min, mean)

Sequential containers

Which data types are ordered?

- Sequential containers are ordered from first to last.
- They can be accessed using their element using integer position/order.
    - Done with square bracket syntax `[ ]`
    - Note **first element is 0, and last is n-1!**
    - One exception are iterators (`iter`) which are incredibly fast.

Sequential containers (2)

Which containers are sequential?

- `list` which we can modify (**mutable**).
    - useful to collect data on the go
- `tuple` which is after initial assignment (**immutable**)
    - tuples are faster as they can do less things
- `array`
    - which is mutable in content (i.e. we can change elements
    - but immutable in size
    - great for data analysis

Lists

A list can be modified (mutated) by methods, e.g.

- We can append objects to it and remove `remove` them again.
- We can use operations like + and *.

In [ ]:
```
list_1 = ['A', 'B']
list_2 = ['C', 'D']
list_1 + list_2
```

Non-sequential types

Are there any non-sequential containers?

- A dictionary (`dict`) which are accessed by keys (immutable objects).
    - Focus of tomorrow.
- A `set` where elements are
    - unique (no duplicates)
    - not ordered
    - disadvantage: cannot access specific elements!

Loops

For loops

Why are containers so powerful?

We can iterate over elements in a container - this creates a *finite* loop, called the `for` loop.

Example - try the following code:

In [ ]:
```
A = []
for i in range(4):
    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?

The one line loop

What is the fastest way to write a loop?

Using list comprehension (also work for containers):

In [ ]:
```
A = [i**2 for i in range(4)]
print(A)
```

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):
    L.append(i*2)
    i += 1
print(L)
```

Applications

- Can be applied in scraping, model which converges, etc.
- Make server process that keeps running

Reusable code

Why do we reuse code?

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

Functions

What procedures have we seen?

How can we make a reusable procedure?

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

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

squared_plus_1(2)
```

Class

Where do objects come from?

- They come from Python `class` which broadly define what they are.
- E.g. a chair has certain
    - attributes, e.g. kind legs, whether or not have back, armrests
    - methods e.g. have a person seated

Modules (1)

Do I have to program everything myself?

- Nope, you can load other people's stuff (https://imgur.com/gallery/ta0DGi3).
- This is how we overcome some of Python's limitations of speed etc.

Modules (2)

How the hell can I do that?

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]

arr = np.array(table)
arr
```

Modules (3)

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
```

Modules (4)

Why are numpy arrays smart?

Crazy fast and compact notation.

- E.g. we can use the same numeric and boolean operations:

In [ ]:
```
arr + 5
```

Final remarks

Quiz time, again..

Summary

In this lecture we learned how to use:

- Python motivation
- Python scripts and Jupyter
- Fundamentals of Python, including
    - Fundamental data types and debugging
    - Operators and control flow
    - Containers and loops
    - Reusable code

The end

Return to agenda