Week 2:
Basic Python

1. **Resources**
2. **Syntax**
3. **Control Structures**
4. **Data Structures**
5. **Boilerplate Code**
6. **Outlook**

Resources

There are lots of resources on Python out there - use the ones that suit your learning style.

- The Official Python Tutorial: https://docs.python.org/3.7/tutorial/ (thorough)
- Google's Python Class: https://developers.google.com/edu/python (brief)
- Java Primer: https://lobster1234.github.io/2017/05/25/python-java-primer/ (minimal)
- The Official Python Docs: https://docs.python.org/3.7/library/ (everything)
- Course Recordings: https://scientificprogramminguos.github.io/lectures/ ("live" feeling)
- Any of the **397.000.000** Google search results for "Python tutorial"

… or use the following two lectures "Basic Python" and "Advanced Python"!

This Lecture

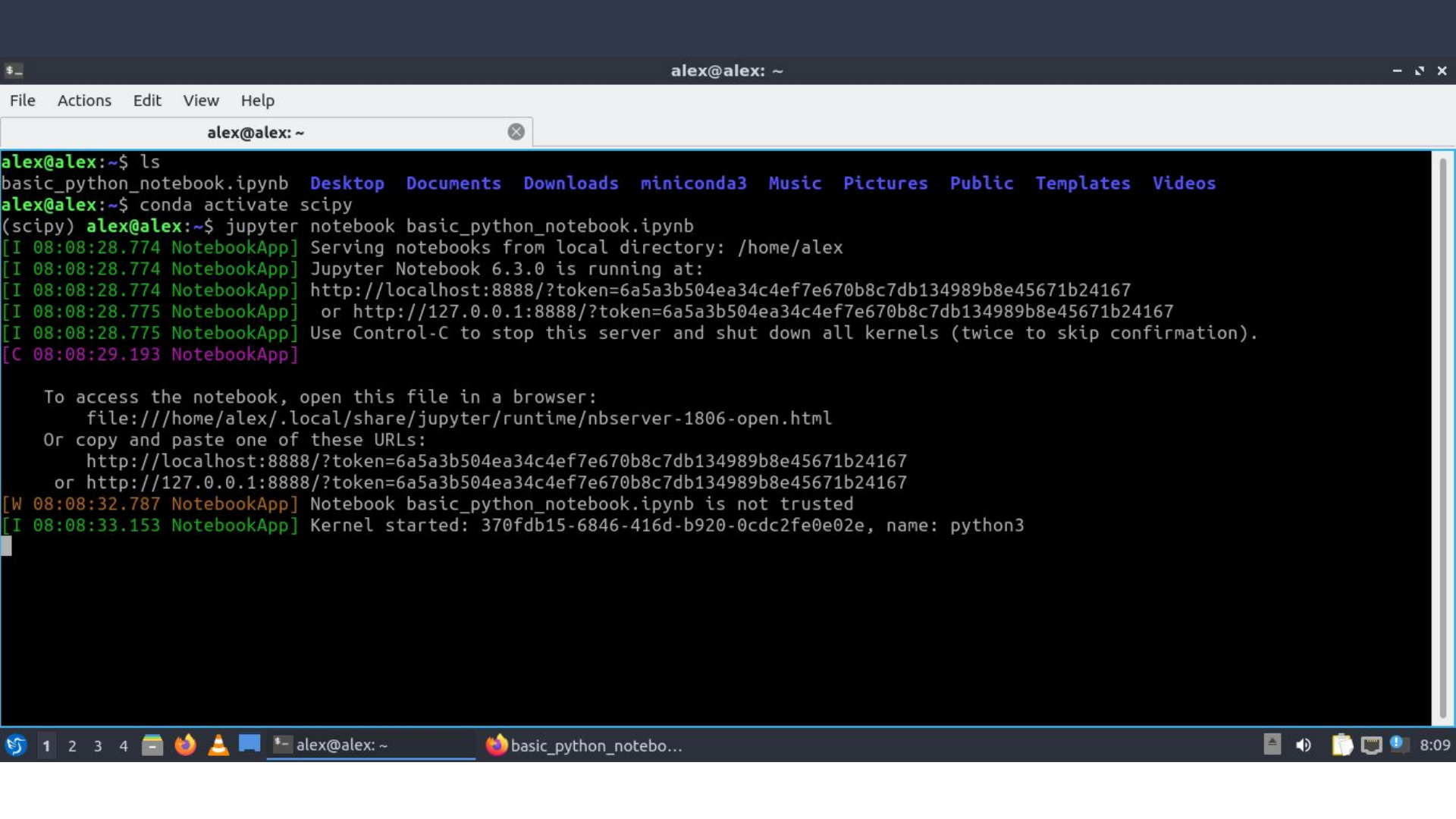
Short Video with Slides + **Interactive Jupyter Notebook**

basic_python_slides.pdf basic_python_notebook.ipynb

- Video **describes** and **explains**, notebook **shows with examples**

  → Use them at the same time or one after another
- **Open the Jupyter Notebook** in the terminal by running:

  (scipy) $ jupyter notebook basic_python_notebook.ipynb (will open a browser)
- If not installed: **Install jupyter** with **$ pip install jupyter**
- **Execute cells** in Jupyter Notebook with **Ctrl + Enter**


alex@alex: ~
File Actions Edit View Help
alex@alex: ~
alex@alex:~$ ls
basic_python_notebook.ipynb Desktop Documents Downloads miniconda3 Music Pictures Public Templates Videos
alex@alex:~$ conda activate scipy
(scipy) alex@alex:~$ jupyter notebook basic_python_notebook.ipynb
[I 08:08:28.774 NotebookApp] Serving notebooks from local directory: /home/alex
[I 08:08:28.774 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 08:08:28.774 NotebookApp] http://localhost:8888/?token=6a5a3b504ea34c4ef7e670b8c7db134989b8e45671b24167
[I 08:08:28.775 NotebookApp] or http://127.0.0.1:8888/?token=6a5a3b504ea34c4ef7e670b8c7db134989b8e45671b24167
[I 08:08:28.775 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 08:08:29.193 NotebookApp]
To access the notebook, open this file in a browser:
file:///home/alex/.local/share/jupyter/runtime/nbserver-1806-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=6a5a3b504ea34c4ef7e670b8c7db134989b8e45671b24167
or http://127.0.0.1:8888/?token=6a5a3b504ea34c4ef7e670b8c7db134989b8e45671b24167
[W 08:08:32.787 NotebookApp] Notebook basic_python_notebook.ipynb is not trusted
[I 08:08:33.153 NotebookApp] Kernel started: 370fdb15-6846-416d-b920-0cdc2fe0e02e, name: python3
1 2 3 4
alex@alex: ~
basic_python_notebo...
8:09

Basic Syntax

Variable assignment operator: **=** **(dynamically typed variables)**

Basic data types:

- **int:** Integer ranging from -2147483648 to 2147483647
- **bool:** Boolean that is either True or False
- **float:** Double precision floating point numbers such as -3.78 or 1239.9329
- **str:** Collection of characters written in single ‘ or double “ quotation marks
- **None:** Value indicating the absence of an object

Print type of a variable: **print(type(variable_name))**

Conversion: Call constructor by name, e.g. **a = float(“4.0”)**

Mathematical operators: ***** (multiply), **+** (add), **-** (subtract), **/** (divide), **%** (modulo)

Control Structures

Meaningful whitespace: Python separates instruction blocks by whitespace indentation!
→ That means **more readability**, but also **need for consistency**

Standard recommendation: **Configure** your editor **to indent 4 spaces** when you **press Tab**

Comparison operators: **==** (equals), **>** (greater than), **<** (less than), **>=**, **<=**, **!=**, **is, in**

Logical operators: **not**, **and**, **or**

Basic control structures:

- **Conditionals:** **if**-statement**, elif**-clauses**, else**-clause
- **Loops:** **for**-loop**, while**-loop**, break** instruction**, continue** instruction
- **Functions:** **def** for function definition, arguments in brackets **(), return**

Conditionals

In [3]:
```
a = 9
```

In [4]:
```
if a < 8:
    print("a is less than 8.")
elif a == 8:
    print("a is equal to  8.")
elif a == 9 or a == 10:
    print("a is equal to 9 or equal to 10.")
elif a > 8:
    print("a is greater than 10.")
else:
    print("This will never be executed.")
```

```
a is equal to 9 or equal to 10.
```

Conditions are evaluated top-to-bottom, left-to-right.

Loops

In [ ]:
```
a = 0

while a < 3:
    print(a)
    a = a + 1
```

In [ ]:
```
a = 0

while True:

    print(a)
    a = a + 1

    if a == 3:
        break
```

In [ ]:
```
for a in range(0, 3):
    print(a)
```

These loops all do the same thing:

They print first 0, then 1, then 2.

```
0
1
2
```

range() returns a sequence of numbers.

for variable in range(start, end) is a very common construct.

Functions

```
In [4]: def increment(my_int, increment_size=1):

            my_int = my_int + increment_size

            return my_int

        a = 5
        b = increment(a)
        c = increment(b, increment_size=3)

        print(a, b, c)
```

```
5 6 9
```

Functions must be defined before they are called.

Positional arguments like my_int must be given when called.

Keyword arguments like increment_size have default values that can be overwritten.

print() takes **arbitrarily many positional arguments**, which are **concatenated with spaces**

Data Structures

Basic data structures

- **list:** Ordered, mutable collection containing objects
- **tuple:** Ordered, immutable collection containing objects
- **set:** Unordered, mutable collection containing unique objects
- **dict:** Ordered*, mutable collection with key-value structure containing objects

```
In [ ]: my_list = [1, 2, 3, 4]
        my_tuple = (1, 2, 3, 4)
        my_set = {1, 2, 3, 4}
        my_dict = {1: 2, 3:4 }
```

* since Python 3.7

The following slides are stolen from the 2019 iteration of “Basic Programming in Python”

Lists

Creating Lists

There are many ways to create lists.

```
list_explicit = ["harry", "ron", "hermione"] # explicit
list_empty_explicit = []
list_empty_constructor = list() # constructor

list_from_range = list(range(100))

print(list_from_range) # [0, 1, 2, ..., 99]
```

Lists

Modifying Lists

```
hp_list = ["harry", "ron", "hermione"]
print(hp_list) # ["harry", "ron", "hermione"]

hp_list.append("luna")
print(hp_list) # ["harry", "ron", "hermione", "luna"]

hp_list.remove("harry")
print(hp_list) # ["ron", "hermione", "luna"]

hp_list[1] = "sirius"
print(hp_list) # ["ron", "sirius", "luna"]
```

Lists

Indexing Lists

1. Access items in the list like normal variables
2. Sytax: `my_list[index]`
3. Index must always be an `int` (e.g. 1, 4, 99)
4. First element has the index 0
5. Last element has the index -1

Lists

Indexing Lists

```
hp_list = ["harry", "ron", "hermione"]

print(hp_list[0]) # "harry"
print(hp_list[1]) # "ron"
print(hp_list[2]) # "hermione"

print(hp_list[-1]) # "hermione"
print(hp_list[-2]) # "ron"
print(hp_list[-3]) # "harry"
```

Lists

Slicing Lists

The ending index is always excluded (just like in `range`)

```
hp_list = ["harry", "ron", "hermione"]

print(hp_list[0:3]) # ["harry", "ron", "hermione"]
print(hp_list[1:3]) # ["ron", "hermione"]
print(hp_list[2:3]) # ["hermione"]

print(hp_list[:]) # ["harry", "ron", "hermione"]
print(hp_list[:-1]) # ["harry", "ron"]
print(hp_list[:-2]) # ["harry"]
print(hp_list[:-3]) # ???
```

Lists

Slicing Lists

```
Index from rear:    -6  -5  -4  -3  -2  -1
Index from front:    0   1   2   3   4   5
                   +---+---+---+---+---+---+
                   | a | b | c | d | e | f |
                   +---+---+---+---+---+---+
Slice from front:  :   1   2   3   4   5   :
Slice from rear:   :  -5  -4  -3  -2  -1   :
```

```
[a, b, c, d, e, f]
```

Lists

Iterating over Lists

```
hp_list = ["harry", "ron", "hermione"]
print(len(hp_list)) # 3

# Iterating by index
for index in range(len(hp_list)):
    item = hp_list[index]
    # do something with item here
    print(item)

# Iterating by content
for item in hp_list:
    # do something with item here
    print(item)
```

Iterate both by index and content:

```
for index, item in enumerate(hp_list):
    print(index, item)
```

Lists

Evaluating Lists

1. `len(list)`: Returns length of list
2. `max(list)`: Returns maximum item of list
3. `min(list)`: Returns minimum item of list
4. `sum(list)`: Returns sum of items in list
5. `sorted(list)`: Returns sorted copy of list

How could we use these functions to calculate the average?

Lists

Nesting Lists

```
def make_grid(height, width, value=1):
    grid = []
    for y in range(height): # repeat for each row
        row = [] # create empty list
        for x in range(width): # repeat for each cell
            row.append(value) # add value to row
        grid.append(row) # add row to grid
    return grid

my_grid = make_grid(3, 2)
print(my_grid) # [[1, 1], [1, 1], [1, 1]]
```

1	1
1	1
1	1

Tuples

Tuples

1. Same as lists, but immutable (unchangeable)
2. Creating: `my_tuple = (v1, v2, v3)`
3. Unpacking: `x1, x2, x3 = my_tuple`

```
my_list = [1, 2, 3]
my_list[1] = 5
print(my_list) # [1, 5, 3]

my_tuple = (1, 2, 3)
my_tuple[1] = 5
# TypeError
```

Sets

Sets

1. Same as lists, but unique (duplicates automatically removed)
2. Syntax: `s = {v1, v2, v3}`
3. Not ordered (no indexing)

```
my_set = {1, 2, 2, 3}
print(my_set) # {1, 2, 3}
print(my_set[0])
# TypeError
```

Dictionaries

Dictionaries

1. Stores not only values, but also names ("keys")
2. Creating: `my_dict = {key1: value1, key2: value}`
3. Indexing: `my_dict[key1]`
4. Keys must be immutable (e.g. Strings, Tuples)

```
my_dict = {"price_apples": 1.2, "price_banana": 4.8}
print(my_dict["price_apples"]) # 1.2
# Iterating by key value pairs
for key, value in my_dict.items():
        print(key, value)
        value = 7 # this will not change anything
        my_dict[key] = 29 # this will change something
```

Boilerplate Code

```
1 """ Docstring that describes script. """
2
3 def main():
4     print("Hello. I am doing something.")
5
6
7 if __name__ == "__main__":
8     main()
```

This is a typical Python script structure.

Line 7 prevents the code from being accidentally executed when importing this script.

The main function is the first thing being executed when running this script.

Outlook

Next Steps:

1. **Open the Jupyter Notebook (**basic_python_notebook.ipynb)
2. **Accept the homework assignment** (link will be in StudIP announcement)
3. **Complete the homework until Sunday at midnight** (2021-04-26 00:00:00+02:00)

Next Week: Object-oriented programming, comprehensions, exceptions, …

Enjoy the week!