(Big) Data Curation, Pipelines, and Management

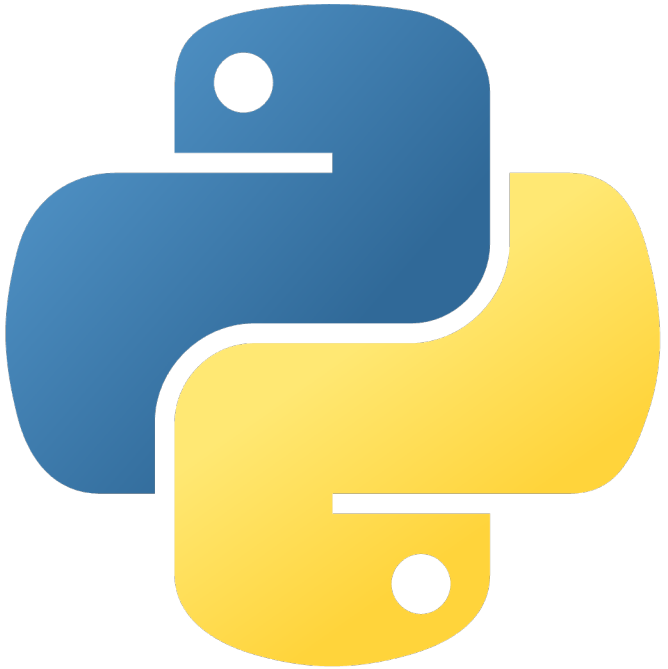
Lecutre 1: Introduction & Python Part 1

Steven Hicks
steven@simula.no

August 6, 2023

Agenda

1. Introduction
   - Course info and structure
   - Resources
2. Python (part 1)
   - Installation and getting started
   - Variables and types
   - Control structures
   - Error handling
   - Functions and classes
3. Exercises



Course info and structure

- Lectures will be held weekly on Friday from 12:00 - 14:00 between August 25th to November 24th.
- I have office hours at BI every Friday - 09:00 – 16:00.
  - Outside of these hours you can contact me at steven@simula.no.
- Lectures and updates will be on GitHub (https://github.com/BI-DS/GRA4157)

Course info and structure

- Grading
  - 40% mid-term
  - 60% final
- Student presentations
  - Every week one of you will present an exercise of their choice.
  - Email me for which week you would like to present I’ll slot you in.
- Group projects
  - Two group projects will be held during the second half of the course.
  - Will not be graded but will be useful for final exam.

Course info and structure

- Part 1 – Python skills
  - Basic Python lists, dictionaries and operations.
  - Reading from and writing to files, flexible solutions.
  - Numerical python with numpy, arrays, array slicing for vectorized computations.
- Part 2 – Working with datasets
  - Working with the pandas library.
  - Reading data from websites.
  - Data visualization.
- Part 3 – Analyzing data
  - Cleaning data, combining data sets.
  - Machine learning workflows with scikit-learn.
  - Assess machine learning models based on various assumptions on data (outliers etc.)

Resources

- Books
  - Introduction to Scientific Programming with Python
  - Python for data analysis: data wrangling with pandas, NumPy, and Jupyter
  - Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems
- Written material can be found here: https://rl.talis.com/3/binorway/lists/B42AA3E8-0EBF-28A8-FD85-D29EA695AF92.html

Python

- Python is an interpreted programming language that emphasizes readability.
- It has long history and its current stable release is 3.11
  - Please make sure to use Python 3 and not Python 2.
- Has become very popular in the data science community and is the most used language.
- Supports several toolboxes and libraries that make it easy to get started.

Installation

- There are several ways of installing Python, and it may already be installed on your computer, but we recommend using Anaconda.
- Anaconda is a distribution of Python that simplifies package management and deployment.
- Anaconda also supports environments that can use specific Python versions with contained list of installed packages.

Create an environment

```
$ conda create -n env python=3.8
```

Activate environment

```
$ conda activate env
```

Deactivate environment

```
$ conda deactivate env
```

Package management

- Python can be extended by install packages.
- The standard package manager for Python is the Python Package Index (PyPI).
- Anaconda comes with a package manager called Conda that can install any package (not only Python) on any system.

PyPI

```
$ pip install scipy
$ pip list
```

Conda

```
$ conda search scipy
$ conda install scipy
$ conda list
```

Documentation and resources

- The official documentation for Python can be found at https://docs.python.org/3.
- There is also a build-in documentation tool that can be accessed using pydoc <module>.
  - Example: `pydoc3 print`

Writing Python and editors

- You can write Python in any editor but some make it easier than others.
- Unless you have any strong opinions, I recommend that you write Python in either an interactive notebook (like Jupyter Notebook) or Visual Studio Code.

- I will be using Visual Studio Code for running code and notebooks.

Exercise (10 minutes)

1. Download and install Python on your machine, I recommend doing this using Anaconda.
2. Download and install an editor to write Python, I recommend Visual Studio Code.
3. Write the following Python program and run it using python <file> 5.

```
1 #!/usr/bin/env python3
2 from math import sin
3 import sys
4
5 x = float(sys.argv[1])
6 print(f"Hello world, sin({x}) = {sin(x)}"
```

Your first Python program

```
#!/usr/bin/env python3
"""Import the sin function from the math library"""
from math import sin

"""Import the sys library to access command line arguments"""
import sys

"""Cast command line argument to float and assign tovariable x"""
x = float(sys.argv[1])

"""Format a string with the variable x and sin(x)"""
print(f"Hello world, sin({x}) = {sin(x)}"
```

Python as a calculator

Addition

```
>> 1 + 2
>> 3
>> 1 + 4 + 7
>> 12
```

Subtraction

```
>> 2 - 1
>> 1
>> 2 - 3
>> -1
```

Multiplication

```
>> 1 * 2
>> 2
>> 10 * 2
>> 20
```

Division

```
>> 4 / 2
>> 2.0
>> 1 / 2
>> 0.5
```

Power

```
>> 2**2
>> 4
>> 2**8
>> 256
```

Complex Numbers

```
>> a = 1 + 2j
>> b = 3 - 5j
>> a * b
>> (13+1j)
```

Modulo

```
>> 4 % 2
>> 0
>> 7 % 5
>> 2
```

Advanced

```
>> from math
import log
>> log10(5)
```

Python variables and types

- Basic types
  - strings: `"strings for storing text"`
  - numbers: `1, 1.5`
  - tuples: `(1, 2, 3)` for storing static collections
  - lists: `["a", "b", "c"]` for mutable, ordered sequences
  - dicts: `{"key": "value"}` for storing key-value pairs
  - sets: `{"do", "re", "mi"}` for storing unique, unordered collections
- You can use `type` to determine what type a variable is.
  - `type(variable)`

Python lists

- Python lists allow you to group together a sequence of values.
- Generally lists are used when you have an ordered collection of the same *kind* of thing:
  - Filenames
  - URLs
  - Objects
  - Numbers
- Lists do not **require** items to have the same type although in practice they usually do.
- Lists are **mutable** meaning that they can be changed.
- Lists can be accessed using slicing [start:end:step]

List examples

Add to list

```
>> a = [1, 2, 3]
>> a.append(4)
>> a
>> [1, 2, 3, 4]
```

Remove from list

```
>> a = [1, 2, 3,
4]
>> a.remove(4)
>> a
>> [1, 2, 3]
```

Access list

```
>> a = [1, 2, 3]
>> a[0]
>> 1
```

Slicing

```
>> a = [1, 2, 3]
>> a[0:2]
>> [1, 2]
```

Concatenation

```
>> a = [1, 2, 3]
>> a + [4]
>> [1, 2, 3, 4]
```

Change

```
>> a = [1, 2, 3]
>> a[2] = 4
>> a
>> [1, 2, 4]
```

Length

```
>> a = [1, 2, 3]
>> len(a)
>> 3
```

Sum

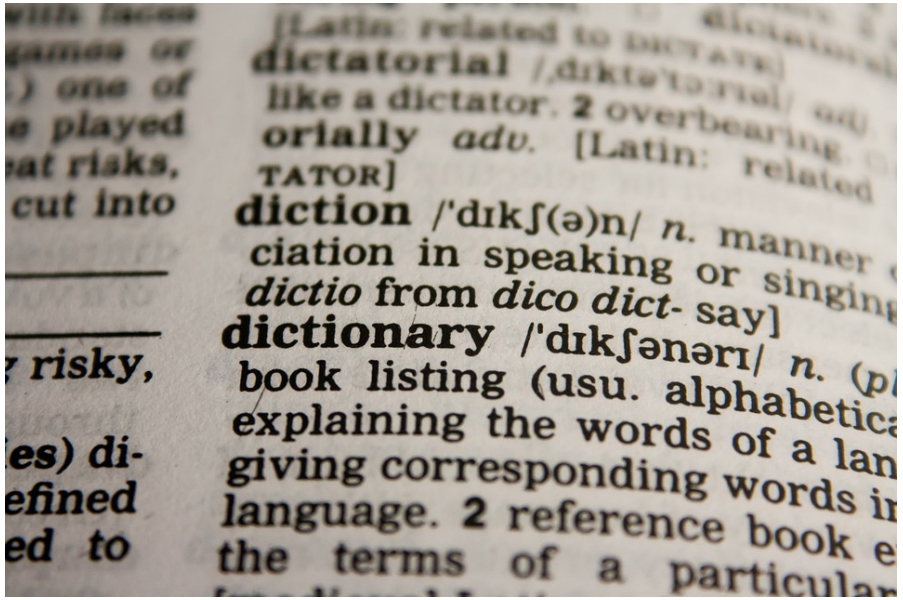
```
>> a = [1, 2, 3]
>> sum(a)
>> 6
```

Python strings

- Strings can be created using single-quotes (‘string’) or double-quotes (“string”) or triple-quotes (”””string”””).
- Triple-quoted strings can be multiline.
- Certain special characters need to be escaped using the backslash character (\).
- Strings are **immutable**, meaning once they are created they cannot be changed.
- Strings can be concatenated or combined using + or *.
  - + can combine strings
  - * can multiply strings with numbers
- Strings can be thought about as a list of characters.

Python dictionaries

- We create dictionaries with the `{}` syntax.
- For each dictionary entry, we need to provide one (immutable) key and its value.
  - `dictionary[key] = value`
- Keys need to be unique and dictionaries are unorederd.
- Dictionaries are **mutable**, meaning once they are created they cannot be changed.
- Some common methods include `keys()`, `values()`, `items()`, `get(key, default)`.



Dict examples

Add value

```
>> a = {“name”: “bill”}
>> a[“age”] = 5
>> a
>> {”name”: ”bill”,
“age”: 5}
```

Remove value

```
>> a = {”name”: ”bill”,
“age”: 5}
>> del a[“age”]
>> a
>> {”name”: ”bill”}
```

Access value

```
>> a = {”name”: ”bill”,
“age”: 5}
>> a[“age”]
>> 5
```

Python control structures - Conditions

- `if` Statement: Checks a condition and executes code if `True`.

```
if condition:
    code_to_execute
```

- `elif` Statement: Used after an `if` to check another condition if the previous one is `False`.

```
if first_condition:
    code_for_first_condition
elif second_condition:

    code_for_second_condition
```

- `else` Statement: Executes code if no preceding conditions are `True`.

```
if first_condition:
    code_for_first_condition
elif second_condition:
    code_for_second_condition
else:
    no_condition_met
```

Python control structures - Conditions Quiz

```
a = 15
b = 5
c = 10
result = ""

if a - b > c:
    result = "Apple"
elif a + b == 2 * c:
    result = "Banana"
else:
    result = "Cherry"

print(result)
```

Python control structures - Loops

- `for` loops through items in an iterable object like lists, tuples, or strings.

```
1 shoppinglist = ["tea", "butter", "milk"]
2
3 for item in shoppinglist:
4     print(f"Remember to buy {item}.")
```

- `while` loops keep executing until some condition is met.

```
1 count = 0
2
3 while count < 5:
4     print("Count is:", count)
5     count += 1
```

Python control structures - Loops Quiz

```
1 x = 1
2 result = ""
3
4 while x < 10:
5     if x % 2 == 0:
6         result += str(x)
7     x += 2
8
9 print(result)
```

Python control structures - Loops Quiz

```
1 x = 1
2 result = ""
3
4 while x < 10:
5     if x % 2 == 0:
6         result += str(x)
7     x += 2
8
9 print(result)
```

Functions

- Python functions allow you to encapsulate a task - they combine many instructions into a single line of code.

```
1 def split(string, char):
2     """ Split the string at the given character """
3
4     position = string.find(char)
5
6     if  position > 0:
7         return string[:position+1], string[position+1:]
8     else:
9         return string, ''
```

Error handling

- `try` and `except` can help you handle unexpected errors in your code.

```
try:
    # Code that might raise an exception
except ExceptionType:
    # Code to handle the exception
```

- Use the `raise` keyword to throw exceptions on specific conditions.

```
if condition:
    raise Exception("Description")
```

Classes

- In Python, everything is an **object**, every object has a **class**.
- A class collects attributes and functions together.
- This class contains three methods
  1. The `__init__` method which is called when a new class object is instantiated.
  2. The `start` method is a user-defined function.
  3. The `accelerate` method is also a user-defined function.

```python
class Car:
    """ A class representing a car """

    def __init__(self, year, fuel="electric"):
        self.year = year
        self.fuel = fuel
        self.speed = 0

    def start(self):
        if self.fuel == "electric":
            print("sssss")
        else:
            print("Wohmmm")

    def accelerate(self, new_speed):
        self.speed = new_speed
```

Classes

- Python classes can implement some magic methods that are called on certain operations.

```
str(a)              # calls a.__str__(), called also with print(a)!
len(a)              # calls a.__len__()
c = a*b             # calls c = a.__mul__(b)
a = a+b             # calls a = a.__add__(b)
a += c              # calls a.__iadd__(c)
d = a[3]            # calls d = a.__getitem__(3)
a[3] = 0            # calls a.__setitem__(3, 0)
f = a(1.2, True)    # calls f = a.__call__(1.2, True)
if a:               # calls if a.__len__()> 0: or if a.__nonzero__():
a == b              # calls a.__eq__(self, b)
```

Inheritance

- Remember: A class collects attributes and functions that belong together.
- Subclasses can be used to specialize and extend existing classes.
- Python supports single and multiple inheritance
  - No private/protected variables (the effect can be "simulated")

```python
class Car:
    def __init__(self, color):
        self.color = color
        self.sound = "Brooom"

    def start(self):
        print(self.sound)
```

```python
class ElectricCar(Car):
    def __init__(self, color):
        super().__init__(color)

        self.sound = "Sssss"

    def stop(self):
        print("Engine stopped")
```

Exercises

- You can find this weeks exercised on GitHub: https://github.com/BI-DS/GRA4157
- I expect that you solve the problem sets after each lecture.
- Starting next week, we will have weekly student presentations where one of you will present an exercise from last weeks lecture.
- Contact me after class to book your slots.

- **This weeks exercises:** Sundnes: 2.7, 2.8, 2.9, 2.15, 2.18, 3.3, 3.6, 3.17