



Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CAS Practical Machine Learning Introduction

First Steps with Python

Prof. Dr. Jürgen Vogel (juergen.vogel@bfh.ch)

Python

Language Overview

Python

Python (<https://www.python.org/>)

- current version 3.x
 - verify installation: `python --version`
- main features
 - interpreted language (no compiler)
 - interactive mode via command line
 - start: `python`
 - hello world: `print("hello world")`
 - end: `quit()`
 - object-oriented and/or functional programming
 - types
 - dynamic but strong typing (no variable declaration)
 - all variables are actually object references: `x = 2`
 - predefined complex types: `tuple`, `list`, `set` and `dict`
 - statements
 - control flow: `if...elif...else`, `for` and `while`
 - grouping via whitespace indentation (no braces, no delimiter)
 - many libraries
- see <https://docs.python.org/>

Python Hello World

```
x = 34 - 23 # A comment
y = "Hello"
z = 3.45
if z == 3.45 or y == "Hello":
    x += 1
    y = y + " World"
print("Result",x,y)
```

- assignment with `=`, comparison with `==`
- first assignment to a variable creates it
 - no explicit type declaration (but strong typing)
 - in fact `x` is a reference to an object of the corresponding type
 - `type(x)` shows type of `x`
 - `dir()` shows all variables in namespace
- for numbers `+` `-` `*` `/` `%` are as expected
 - ordering as expected
 - use `()` for custom order
- special use of `+` for string concatenation
- logical operators are words: `and`, `or`, `not`
- printing via `print` function
- definition statements (`if`) end with `:`
- blocks via indentation
- newline ends a line of code
 - use `\` to continue line of code on next line

Strings

- string literals are defined with `"bla"` or with `'bla'`
- strings are immutable
- special characters need to be escaped, e.g., `'\\'`, `'\n'`
- search for substring
 - `str.find("this")` # returns index or -1
 - `"this" in str` # returns True or False
 - `str.startswith("this")` # returns True or False
- split: `str.split(". ")`
- remove whitespaces: `str.strip()`
- replace substring: `str.replace(old, new)`
- from any type to string: `"Hello " + str(x)`
- from string to list: `li = str.split(',')`
- from list to string: `str = " ".join(['a', 'b', 'c'])`

Sequences: Strings, Tuples and Lists

- **string**
 - immutable sequence of characters: `myString = 'string'`
- **tuple**
 - ordered, immutable sequence of objects: `myTuple = (1,2)`
 - mixed types: `myTriple = (1,2,"3")`
- **list**
 - ordered, mutable sequence of objects: `myList = [1,2]`
 - adding elements: `myList.append(3)` and `myList.insert(0, 3)`
 - removing elements: `myList.remove(0, 3)`
- **manipulation**
 - length: `len(myList)`
 - indexes: `myString[0] ... myString[len(myString)-1]`
 - slices create a copy of the original list: `subList = myList[start:end]`
 - `subList = myList[:end]`
 - `subList = myList[start:]`
 - `listCopy = myList[:]` # different from `listCopy = myList`
 - concatenation via `+`: `l3 = l1 + l2`
 - minimum, maximum: `min(myList), max(myList)`
 - contains element `x`? `if x in myList:`

Control Flow

- `if condition:`
 `do something`
`elif condition2:`
 `do something`
`else:`
 `do something`
- `while condition:`
 `do something`
- `for i in list: # or: for i in range(len(list))`
 `do something with i`
- loops may be ended with `break` and skipped with `continue`

Conditions

Boolean literals: `True`, `False`

Boolean operators: `and`, `or`, `not`

comparison operators (content):
`<`, `<=`, `==`, `!=`

object comparison: `is`, `is not`

Efficient List Processing

List Comprehension

- create a new list by processing all elements of an existing one
 - e.g. applying some function `func`

```
newList = [func(elem) for elem in oldList]
```

- a filter may be applied

```
newList = [func(elem) for elem in oldList if elem > 3]
```


Dictionaries

- dictionary stores key-value pairs (= hashtable)
 - `hs = {'user1': 'password1', 'user2': 'password2'}`
 - keys are unique, can be any immutable type
 - values can be any type
- lookup: `print(hs['user1'])`
- adding: `hs['user3'] = 'pw3'`
- deleting: `hs.pop('user1')`
- list of all keys: `hs.keys()`
 - `for k in hs.keys(): # or: for k in hs:`
`print(k)`
- list of all values: `hs.values()`
 - `for v in hs.values():`
`print(v)`
- list of all key-value pairs: `hs.items()`
 - `for k,v in hs.items():`
`print(k,v)`

Functions

```
def functionName(arg1, arg2, arg3=default):  
    do something  
    do something  
    return result
```

```
result = functionName(a1, a2, a3)
```

- function names cannot be overloaded
 - as in other languages via argument variations
- functions without explicit return implicitly return `None`
- functions can be used as any other data type
 - assigned to variables
 - managed in tuples, lists, ...
 - used as arguments and returns for function calls
- lambda: function without name

Classes

- everything is an object: `"hello".upper()`
- definition via `class`

```
class myClass:
    def __init__(self, arg1, arg2):
        self.var1 = arg1
        self.var2 = arg2
    def getVar1(self):
        return self.var1
```
- every class method needs the instance reference `self` as first argument
- class attributes are defined implicitly (as with all types)
- all constructors have the name `__init__`
- create instances: `myInstance = myClass(param1,param2)`
- calling a method: `result = myInstance.getVar1()`
- note that `self` is passed implicitly
- there is no destructor but an automatic garbage collection
- inheritance: `class mySubClass(myClass):`

Modules

- modules structure code (functions, classes) into larger units with separate namespaces
- module is a file with the name `module.py`
- can be used after `import` statement
 - `import module`
 - imports everything and keeps it in the module's namespace
 - `module.func()`
 - `module.className.func()`
 - `from module import *`
 - imports everything under the current namespace
 - `func()`
 - `className.func()`
 - (not recommended)
 - `from module import className`
 - selectively imports under the current namespace
 - `className.func()`
- standard modules: `math`, `os`, `sys`

Files

- **open a file:** `f = open("path/file", "r")` # (r)ead, (w)rite, (a)ppend
- **read the entire file into a string:** `fContent = f.read()`
- **read a single line:** `line = f.readline()`
- **iterate linewise:** `for line in f:`
- **write:** `f.write("bla")`
- **close a file:** `f.close()`
- **manipulating files and directories via module `os`**
 - **list directory content:** `l = os.listdir("/path")`
 - **exists?:** `os.path.exists("/path/file")`
 - **is file?:** `os.path.isfile("/path/file")`
- **reading and writing python objects via module `pickle`**
 - **read:** `object = pickle.load(file)`
 - **write:** `pickle.dump(object, file)`

Python Development Environment

Python Development Environment

1) Installation

- a) custom: Python core plus package managers plus Jupyter plus libraries
 - Python <https://wiki.python.org/moin/BeginnersGuide/Download>
 - Jupyter (Lab or Notebook) <https://jupyter.org/install>
- b) distribution: package with all of the above
 - e.g., Anaconda <https://www.anaconda.com/>

2) Execution Environment

- a) single machine
 - i. local/native: direct install on your machine
 - ii. local (or cloud-based) virtual machine (VM): install on a guest OS
 - e.g., Linux VM (Ubuntu) via VirtualBox
 - iii. local (or cloud-based) Docker container
- b) cluster
 - i. Hadoop
 - ii. Spark

Python Development Environment

1) Installation

- a) custom: Python core plus package managers plus libraries plus tooling
- b) distribution: package of the above
 - e.g., Anaconda <https://www.anaconda.com/>

2) Execution Environment

- a) single machine
 - i. local/native: direct install on your machine
 - ii. local (or cloud-based) virtual machine (VM): install on a guest OS
 - i. e.g., Linux VM (Ubuntu) via VirtualBox
 - iii. local (or cloud-based) Docker container
- b) cluster
 - i. Hadoop
 - ii. Spark

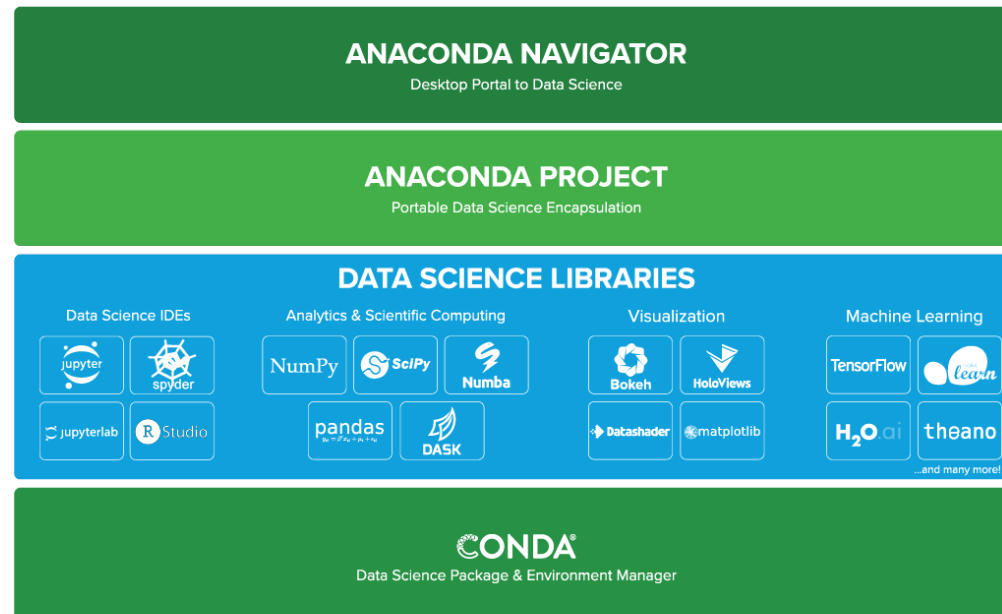
Hardware Requirements for Local Execution

- in most cases, a modern Laptop should be sufficient
 - hardware
 - CPU: 4+ cores recommended
 - RAM: 8GB+ recommended
 - GPU: for some use cases
 - Disk: SSD recommended
 - OS: Linux, Mac OS, or Windows
 - or VM
 - sufficient rights for installation required
- depending on data volume, complexity of feature engineering, and complexity of ML model, a more powerful machine is required (→ CAS Data Engineering)
 - private vs. cloud-based solution

Anaconda Distribution

Anaconda (<https://www.anaconda.com/>)

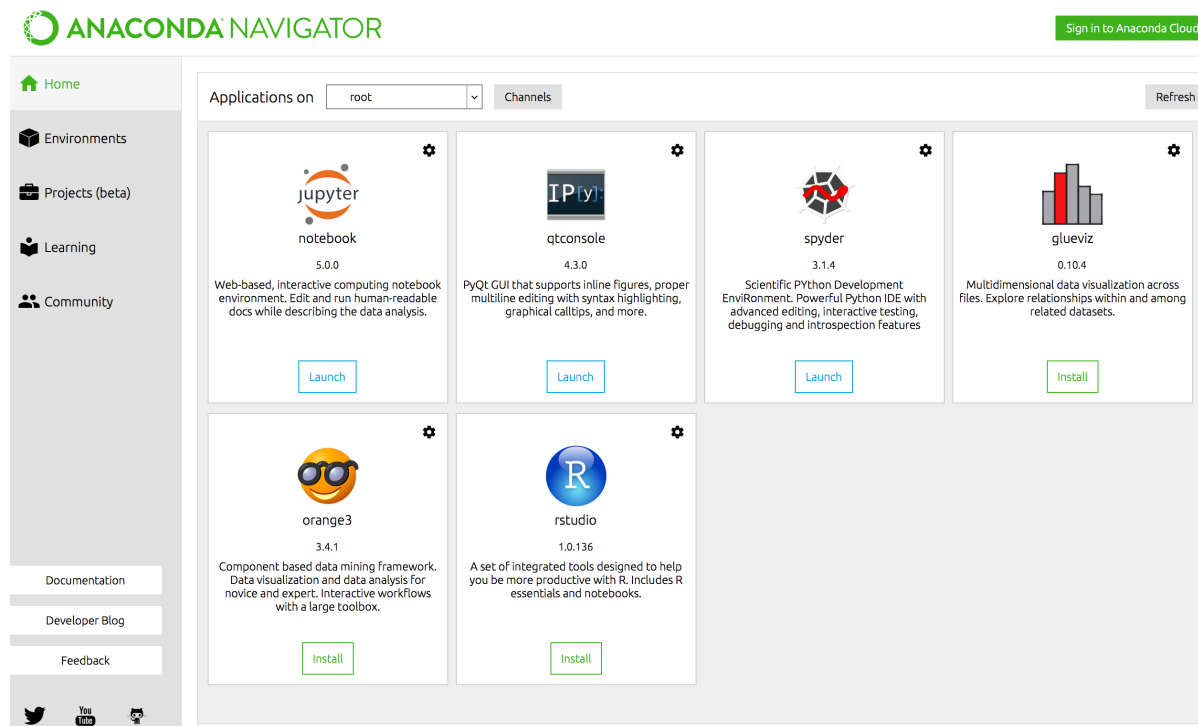
- offers a (free) Python distribution that includes the most important packages and tools for data science tasks
- company with a typical open source-based business model: additional tools, support, consulting, training and cloud hosting
- download free Python distribution via <https://www.anaconda.com/download/>



Anaconda Navigator

Anaconda Navigator (<https://docs.anaconda.com/anaconda/navigator>)

- manages Anaconda installation and included tools



pip

pip (<https://pypi.org/project/pip/>)

- package manager for Python packages (libraries)
- packages are retrieved from a repository
 - Python package index (<https://pypi.org/>)
- verify installation: `pip --version`
- list all installed packages: `pip list`
- install: `pip install [packagename]`
 - specific version: `pip install [packagename] == 2.1`
 - list of dependencies: `pip install -r [requirements.txt]`
- update: `pip install --upgrade [packagename]`
- see https://pip.pypa.io/en/stable/user_guide/

conda

conda (<https://conda.io/docs/>)

- package manager for Python (and other) packages (libraries)
- packages are retrieved from a repository
 - default and configurable repository
- supports multiple environments
 - specific set of packages with specific versions
 - configuration saved in `environment.yaml`
 - can be shared between machines / developers
 - stay in default environment `root` for now

```
conda info --envs
```

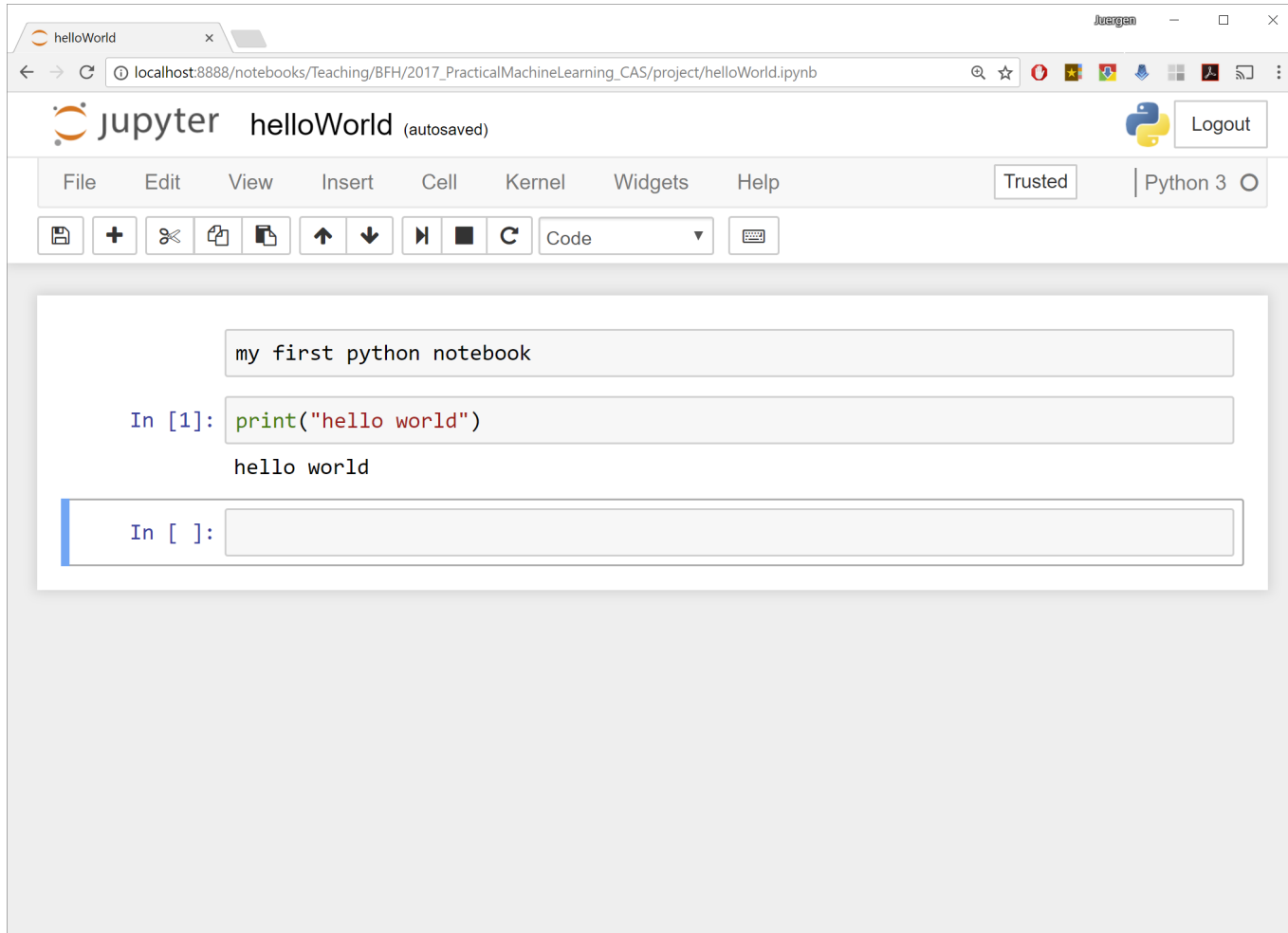
- **verify installation:** `conda --version`
- **list all installed packages:** `conda list`
- **install:** `conda install [packagename]`
- **update:** `conda update [packagename]`
`conda update conda`
- see <https://conda.io/docs/user-guide/getting-started.html>

Jupyter

Jupyter Notebook (<https://jupyter.org/>)

- notebook-style IDE: integrate code with text and interactive data visualizations, ...
- Web application
 - start
 - Anaconda Navigator
 - command line: `jupyter notebook --port [port]`
 - set working directory for notebooks:
`jupyter notebook --notebook-dir='[path]'`
 - runs on `http://localhost:[port]`
 - see running servers: `jupyter notebook list`
 - incl. authentication token
 - create new Notebook via Web UI
 - navigate to desired location and click New -> Python3 notebook
 - file saved as `[name].ipynb`
 - notebook page should open automatically in browser

Jupyter Hello World



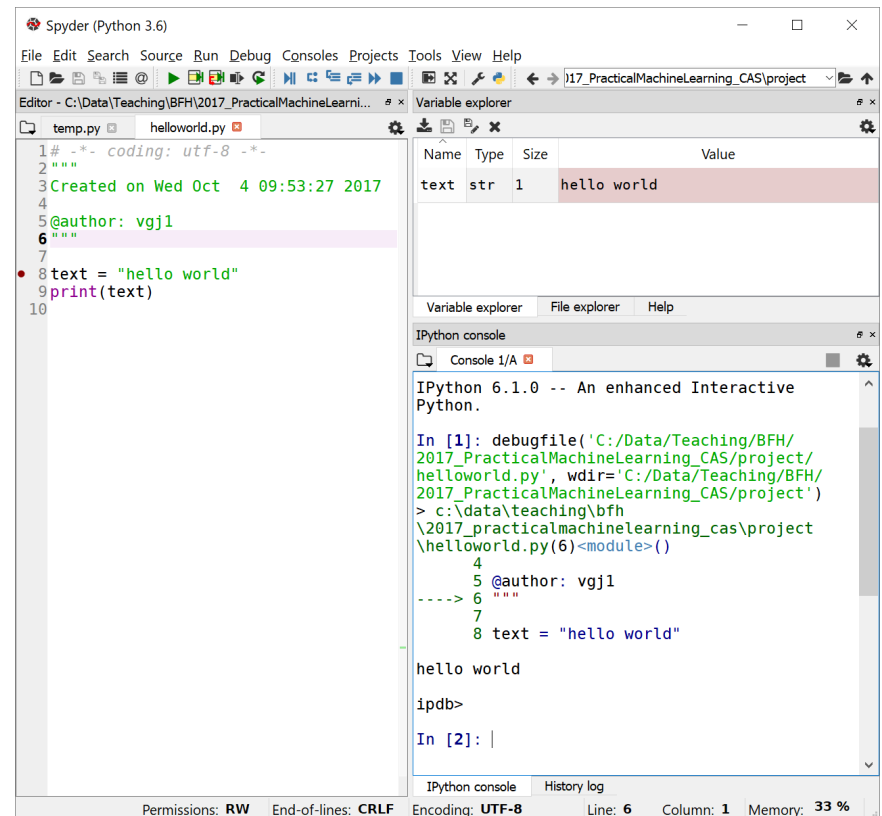
Jupyter

- stop notebook
 - in the notebook page: click File -> Stop and Halt
 - in the jupyter page: click Running **and** Shutdown
- stop server via killing the process
 - Linux
 - `netstat -tulpn` to retrieve pid
 - `kill [pid]`
 - Windows
 - `netstat -ano` to retrieve pid
 - `taskkill /PID [pid] /F`
- see <https://jupyter.org/documentation.html>

Spyder

Spyder (<https://pythonhosted.org/spyder/>)

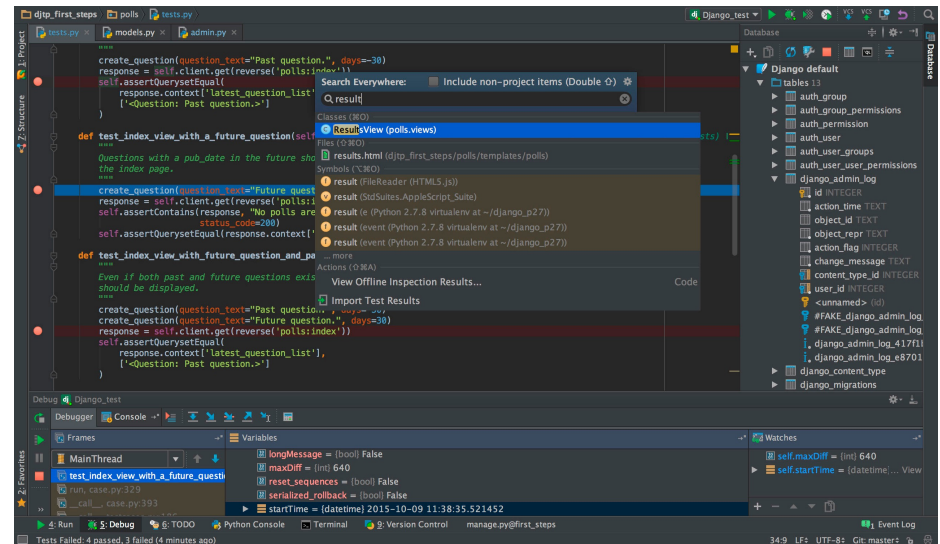
- classical IDE for python
 - editor with syntax highlighting and code completion
 - code analysis
 - Pylint (<https://www.pylint.org/>)
 - code execution and debugging
 - projects with version control
 - git (<https://git-scm.com/>)
- desktop application
 - start via Anaconda Navigator



PyCharm

PyCharm (<https://www.jetbrains.com/pycharm/>)

- traditional IDE for Python
 - code completion, highlighting, refactoring, and analysis
 - debugging
 - unit testing
 - version control
- by JetBrains
 - professional, educational, and open source licensing



Python Libraries for Machine Learning

Popular Python Libraries

- **NumPy**: vectors, matrices and linear algebra, ...
- **SciPy**: linear algebra, integration, interpolation, ...
- **Statsmodels**: statistical data analysis
- **pandas**: structured data and analysis
- **matplotlib**: visualize numerical data
- **scikit-learn**: machine learning
- **PyTorch**: deep learning
- **NLTK**: text analysis
- **Scrapy**: Web scraping
- **BeautifulSoup**: HTML and XML parsing
- ...