

Algorithmen, KI und Data Science 1 (AKIDS 1): Introduction to Python

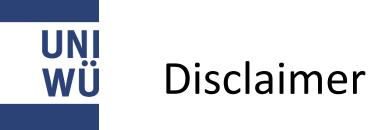
Herbst- / Wintersemester 2023/24

Prof. Dr. Goran Glavaš

Benedikt Ebing

Fabian David Schmidt

adapted from https://web.stanford.edu/class/cs224n/readings/cs224n-python-review.pdf





- Expectations: Learning & Master Python in 1.5 hours
- Reality
 - Overview on setup & tooling
 - Quick rundown of language basics to cover barebone essentials as quickly as possible



- 1. Why Python?
- 2. Setup
- 3. Language Basics
- 4. Practical Python Tips
- 5. Other Great References



1. Why Python?

- 2. Setup
- 3. Language Basics
- 4. Practical Python Tips
- 5. Other Great References



- **Rich standard library:** Python is a widely used, general purpose programming language with batteries included
- Simple Semantics & Syntax: Easy to start working with.
- Ecosystem: packages for anything available as Python is amongst most popular languages
- Machine Learning:

Scientific computation functionality similar to Matlab and Octave. Used by major deep learning frameworks such as PyTorch and TensorFlow.



- 1. Why Python?
- 2. Setup
- 3. Language Basics
- 4. Practical Python Tips
- 5. Other Great References



Jupyter Notebook / Google Colab

- Jupyter Notebook: https://jupyter.org/install
 - A Jupyter notebook lets you write and execute Python code locally in your web browser
 - Interactive, code re-execution, result storage, can interleave text, equations, and images
 - Can add conda environments to Jupyter notebook
- Google Colab: <u>https://colab.research.google.com/</u>
 - Google's hosted Jupyter notebook service, runs in the cloud, requires no setup to use, provides free access to computing resources including GPUs
 - Comes with many Python libraries pre-installed



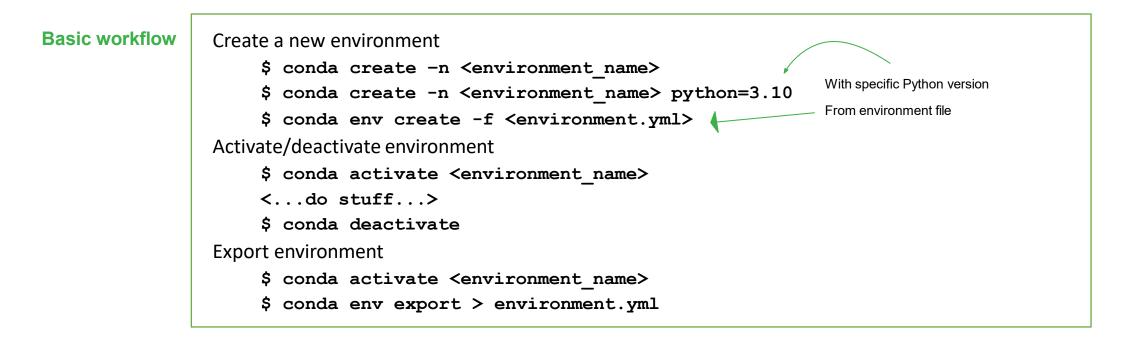
Environment Management

• Problem

- Different versions of Python
- Countless Python packages and their dependencies
- Different projects require different packages
 - Even worse, different versions of the same package!
- Solution
 - Keep multiple Python environments that are isolated from each other
 - Each environment
 - Can use different Python version
 - Keeps its own set of packages (can specify package versions)
 - Can be easily replicated

Anaconda

- Anaconda is a popular Python environment/package manager
 - o Install from <u>https://www.anaconda.com/download/</u>
 - Or better yet, mamba, a much faster drop-in replacement for conda, <u>https://github.com/conda-forge/miniforge</u>
 - Supports Windows, Linux, MacOS
 - Can create and manage different isolated environments



UNI WÜ IDEs / text editors for Python

- PyCharm: tailored to Python, comparably heavy but fully loaded
- VS Code: lighter, language agnostic alternative though not as Python-specific
- Jupyterlab: interactive out-of-the-box and great for presenting code
- Sublime Text: very lightweight and fast editor
- Emacs / Vim: extremely extensible editors that are "lifetime tools"
- Write a Python program in your IDE or text editor of choice.
- In terminal, activate conda environment and run program with command:
- \$ python <filename.py>



Recommendations

- Jupyter Notebooks are very limited: Pick up an IDE and learn the basics quickly!
- Get as much practice as possible with Python: Do the class exercises by yourself; implement something useful to you personally; contribute to open-source
- Learn commandline basics (eg <u>https://missing.csail.mit.edu/</u>) Programming continues to be centered around CLI; Extremely valuable in the long-run to acquire familiarity with git, grep (ripgrep), bash (process management, shell scripting); Any compute infrastructure will run on a Linux distribution



- 1. Why Python?
- 2. Setup
- 3. Language Basics
- 4. Practical Python Tips
- 5. Other Great References



Common Operations

$\mathbf{x} = 10$		<pre># Declaring two integer variables</pre>	
y = 3		# Comments start with hash	
х + у	>> 13	# Arithmetic operations	
x ** y	>> 1000	# Exponentiation	
х / у	>> 3	<pre># Dividing two integers</pre>	
x / float(y)	>> 3.333	# Type casting for float division	
<pre>str(x) + ``+" + str(y)</pre>	>> ``10 + 3″	<pre># Casting integer as string and string concatenation</pre>	



Built-in Values (1/2)

True, False # Usual true/false values None # Represents the absence of something x = None # Variables can be assigned None array = [1, 2, None] # Lists can contain None # Lists can contain None # Functions can return None



Built-in Values (2/2)

and # Boolean operators in Python written
or
or
&&,||, ! in C++

not

- if [] != [None]:
 print("Not equal")
- # Comparison operators == and != check
 for equality/inequality, return
 true/false values



Brackets \rightarrow Indents

- Code blocks are created using indents, instead of brackets like in C++
- Indents can be 2 or 4 spaces, but should be consistent throughout file

```
def sign(num):
    # Indent level 1: function body
    if num == 0:
        # Indent level 2: if statement body
        print("Zero")
    elif num > 0:
        # Indent level 2: else if statement body
        print("Positive")
    else:
        # Indent level 2: else statement body
        print("Negative")
```

UNI WÜ

Language Basics (1/3)

Python is a strongly-typed and dynamically-typed language.

- **Strongly-typed**: Interpreter always "respects" the types of each variable. [1]
- **Dynamically-typed**: "A variable is simply a value bound to a name." [1]
- **Execution**: Python is first interpreted into bytecode (.pyc) and then compiled by a VM implementation into machine instructions. (Most commonly using C.)

UNI WÜ

Language Basics (2/3)

Python is a strongly-typed and dynamically-typed language.

- **Strongly-typed:** Types will not be coerced silently like in JavaScript.
- **Dynamically-typed:** Variables are names for values or object references. Variables can be reassigned to values of a different type.
- Execution: Python is "slower", but it can run highly optimized C/C++ subroutines which make scientific computing (e.g. matrix multiplication) really fast.



Language Basics (3/3)

Python is a strongly-typed and dynamically-typed language.

- Strongly-typed:
 - $1 + 1' \rightarrow \text{Error!}$
- Dynamically-typed: foo = [1,2,3] # initial declaration foo = 'hello!' # later redeclaration
- Execution:

np.dot(x, W) + b \rightarrow Fast!



Lists are **mutable arrays**

```
names = ['Zach', 'Jay'] names[0] == 'Zach'
names.append('Richard') print(len(names) ==
3) >> True
print(names) >> ['Zach', 'Jay', 'Richard'] names +=
['Abi', 'Kevin']
print(names) >> ['Zach', 'Jay', 'Richard', 'Abi', 'Kevin']
names = [] # Creates an empty list
names = list() # Also creates an empty list
stuff = [1, ['hi', 'bye'], -0.12, None] # Can mix types
```



Tuples are **immutable arrays**.

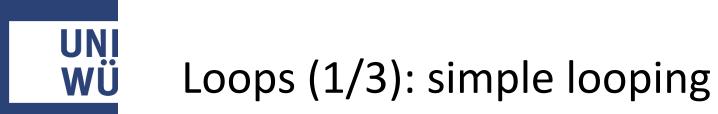
```
names = ('Zach', 'Jay') # Note the parentheses
names[0] == 'Zach'
print(len(names) == 2) >> True
print(names) >> ('Zach', 'Jay')
names[0] = 'Richard' >> TypeError: 'tuple' object does not
support item assignment
empty = tuple() # Empty tuple
single = (10,) # Single-element tuple. Comma matters!
```



Collections (3/3): Dictionary

Dictionaries are hash maps.

```
phonebook = {} # Empty dictionary
phonebook = dict() # Also creates an empty dictionary
phonebook = {'Zach': '12-37'} # Dictionary with one item
phonebook['Jay'] = '34-23' # Add another item
print('Zach' in phonebook) >> True
print('Kevin' in phonebook) >> False
print(phonebook['Jay']) >> '34-23'
del phonebook['Zach'] # Delete an item
print(phonebook) >> {'Jay' : '34-23'}
```



Instead of for (i=0; i<10; i++) syntax in languages like C++, use range ()

```
for i in range(10):
    print(i)
>> 1
    2...
    9
    10
```



Loops (2/3): Iterating Over Lists

```
To iterate over a list
names = ['Zach', `Jay`, `Richard`]
for name in names:
    print(`Hi` + name + `!`)
```

```
To iterate over indices and values
# One way
for i in range(len(names)):
    print(i, names[i])
```

```
# A different way
for i, name in enumerate(names):
    print(i, name)
```

>> Hi Zach!
Hi Jay!
Hi Richard!

```
>> 1 Zach
2 Jay
3 Richard
```



Loops (3/3): Iterating Over Dictionaries

To iterate over a dictionary phonebook = { `Zach': `12-37', `Jay': `34-23' }				
for name in phonebook:	>>	Jay		
<pre>print(name)</pre>		Zach		
<pre>for number in phonebook.values():</pre>	>>	12-37		
<pre>print(number)</pre>		34-23		
<pre>for name, number in phonebook.items():</pre>	>>	Zach 12-37		
<pre>print(name, number)</pre>		Jay 34-23		



```
class Animal:
    def _init_(self, species, age):
        self.species = species
        self.age = age
```

```
def is_person(self):
    return self.species
```

```
def age_one_year(self):
    self.age += 1
```

```
class Dog(Animal):
    def age_one_year(self):
        self.age += 7
```

Constructor `a =
Animal(`human', 10)`
Refer to instance with `self`
Instance variables are public

```
# Invoked with `a.is_person()`
```

```
# Inherits Animal's methods
# Override for dog years
```



- 1. Why Python?
- 2. Setup
- 3. Language Basics
- 4. Practical Python Tips
- 5. Other Great References



List Comprehensions

- Similar to **map**() from functional programming languages.
- Can improve readability & make the code succinct.
- Format: [func(x) for x in some_list]

```
squares = []
for i in range(10):
    squares.append(i**2)
```

```
squares = [i**2 for i in range(10)]
```

```
Can be conditional:
odds = [i**2 for i in range(10) if i%2 == 1]
power = [i**2 if i%2 == 1 else i ** 3 for i in range(10)]
```



Convenient Syntax

Multiple assignment / unpacking iterables:

```
age, name, pets = 20, `Joy', [`cat']
x, y, z = (`Tensorflow', `PyTorch', `Chainer')
```

Returning multiple items from a function

```
def some func():
    return 10, 1
ten, one = some func()
```

Joining list of strings with a delimiter

", ".join([1, 2, 3]) == `1, 2, 3'

String literals with both single and double quotes

```
message = `I like ``single" quotes.' reply = ``I
prefer `double' quotes."
```



pip installs Python packages, conda installs packages which may contain software written in any language

Issues may arise when using pip and conda together. It is best practice to first use conda to install as many packages as possible and use pip to install remaining packages after. [1]

```
conda install -n myenv [package_name][=optional version number]
```

Install packages using pip in a conda environment (necessary when package not available through conda)

```
conda install -n myenv pip
```

conda activate myenv

pip install

```
[package_name] [==optional version number]
```

- # Install pip in environment
- # Activate environment
- # Install package

pip install -r <requirements.txt> # Install packages from file



Importing Package Modules

Import `os' and `time' modules
import os, time

```
# Import under an alias
import numpy as np
np.dot(x, y)  # Access components with pkg.fn
```

Import specific submodules/functions
from numpy import linalg as la, dot as matrix_multiply
Can result in namespace collisions...



- 1. Why Python?
- 2. Setup
- 3. Language Basics
- 4. Practical Python Tips
- 5. Other Great References



- Official Python 3 documentation: <u>https://docs.python.org/3/</u>
- Official Anaconda user guide: <u>https://docs.conda.io/projects/conda/en/latest/user-guide/index.html</u>
- Practice, practice, practice: https://automatetheboringstuff.com/
- Program a lot of useful (personal projects) or required (exercises) stuff: there are no shortcuts!



Orga

- Try to upload the exercise sheets by Thursday EoD
- Exercise sheets are solved in groups of 3
- Two parts:
 - Pen & paper
 - Implementation
- Submit solutions on WueCampus until Friday 10:00 am before the exercise
 - Upload a .zip file containing the following three files:
 - .pdf with solutions for the pen & paper exercises
 - .ipynb with solutions for the implementation exercises
 - .txt with the names and matriculation number of your team members