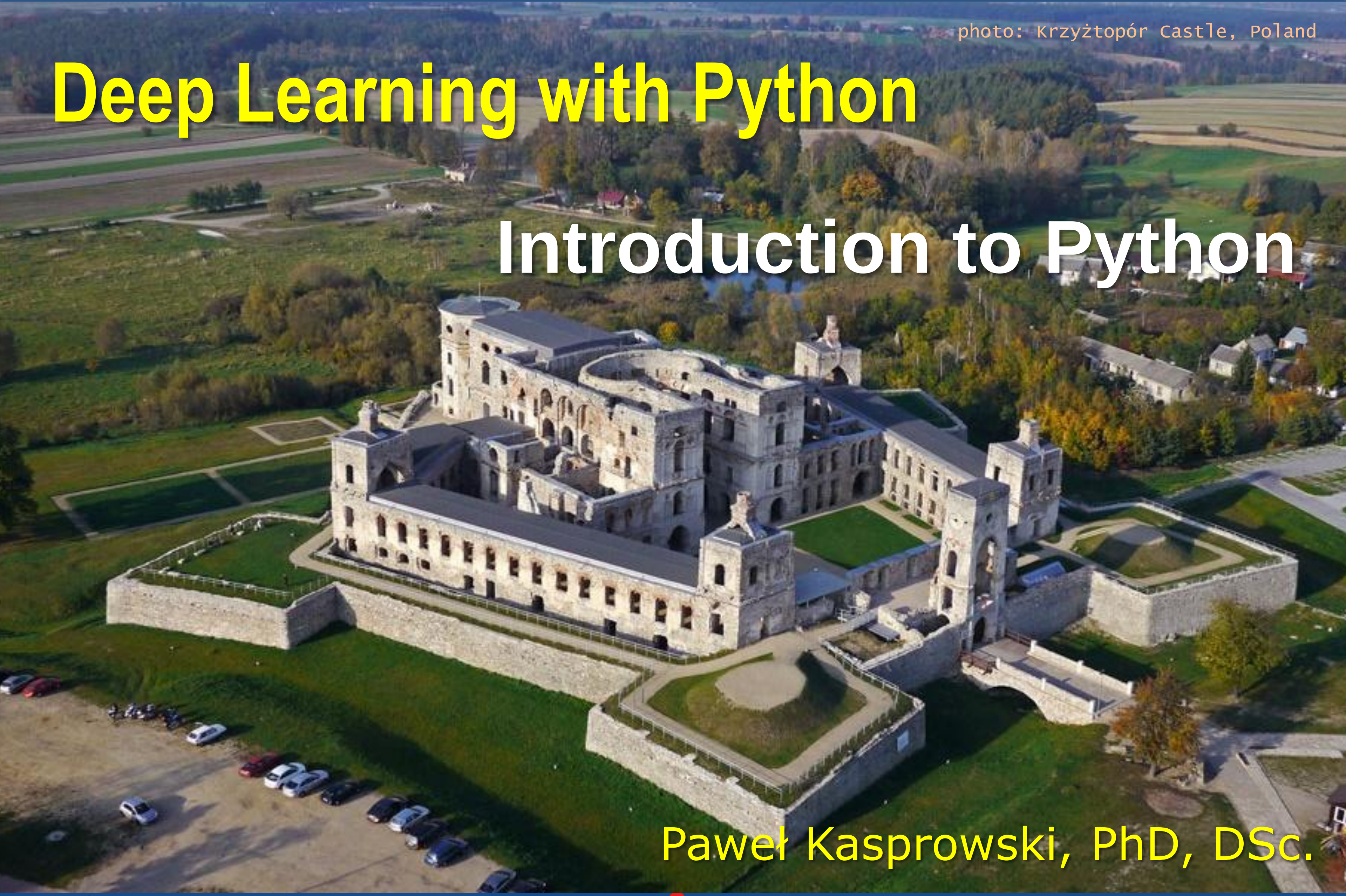


photo: Krzyżtopór Castle, Poland

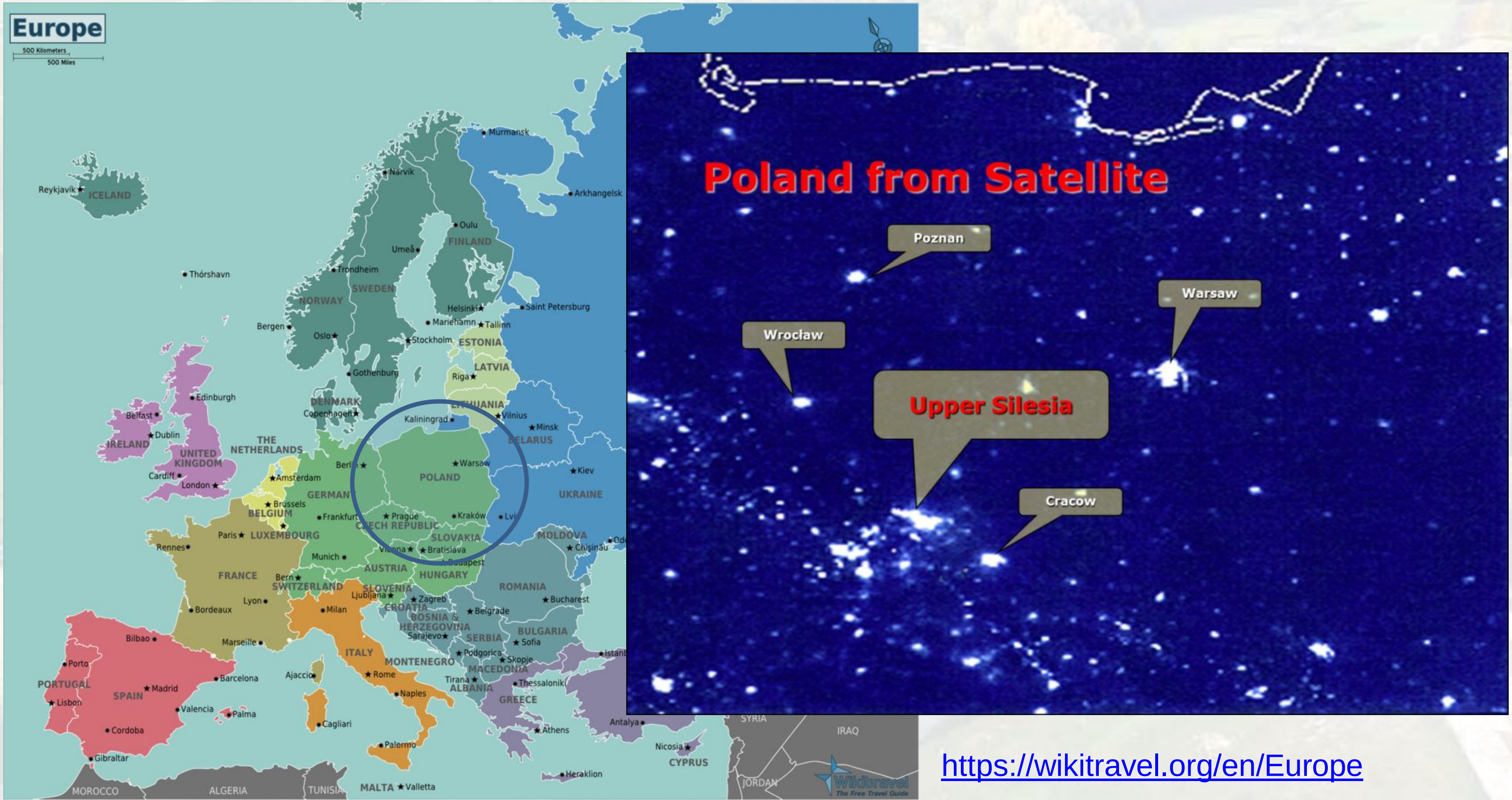
Deep Learning with Python

Introduction to Python

Paweł Kasprowski, PhD, DSc.

About my home

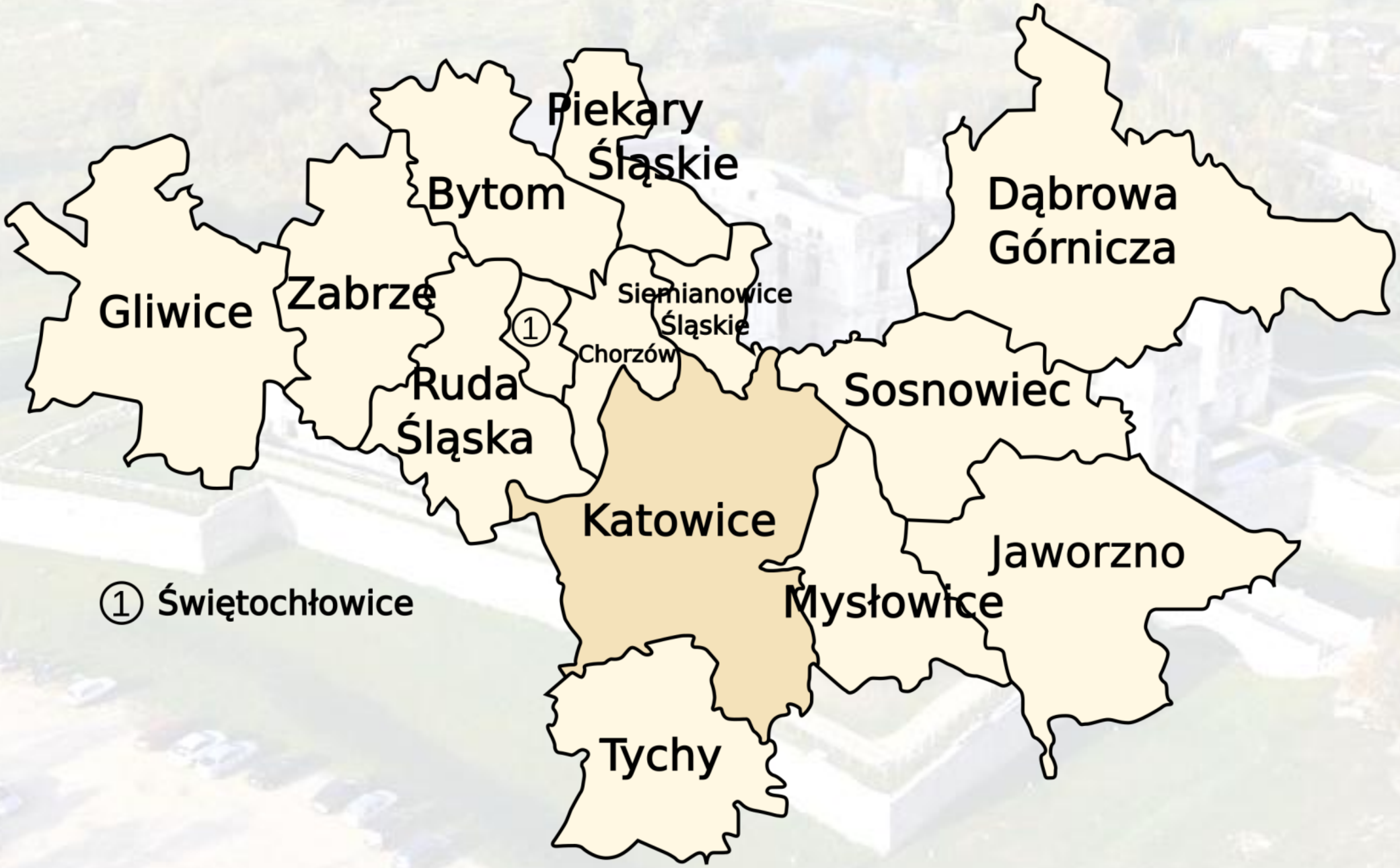
Silesian University of Technology, Gliwice, Poland...?



https://wikitravel.org/en/Europe

Upper Silesian Industrial Region

14 cities, population about 5 mln, 13% of Polish industry



Silesian University of Technology (Politechnika Śląska)

Politechnika Śląska

- Rankings among all universities in Poland
  - 2nd place in educating top managers
  - 4th place whose graduates are most desired by employers
  - 6th place among all higher education technology schools in Poland
- **13** Faculties
- More than:
  - **15,000** students
  - **600** PhD students
  - **3,150** employees
  - **150** full-professors

My Faculty

Faculty of Automatic Control, Electronics and Computer Science

(AEI - "vowel faculty")

Majors:
Automatic Control
Electronics and Teleinformatics
Computer Science
Computer Science (English)
Macrofaculty (English)
Bioinformatics

Staff:
40 professors
160 associate professors and lecturers
>2000 students

General plan

- Introduction to Python
  - working environment, Python basics, pandas, numpy, matplotlib
- Introduction to Data Mining
  - classification, regression, scikit-learn
- Neural Networks in Keras/Tensorflow
- Convolutional Neural Networks
  - image classification, parametrization, training
- Recurrent Neural Networks
  - networks with memory, texts classification
- GradientTape, Hypernetworks, Adversarial Attacks
- Autoencoders, Generative Adversarial Networks (GANs)
- Object detection with Fast and Faster RCNN

Rules

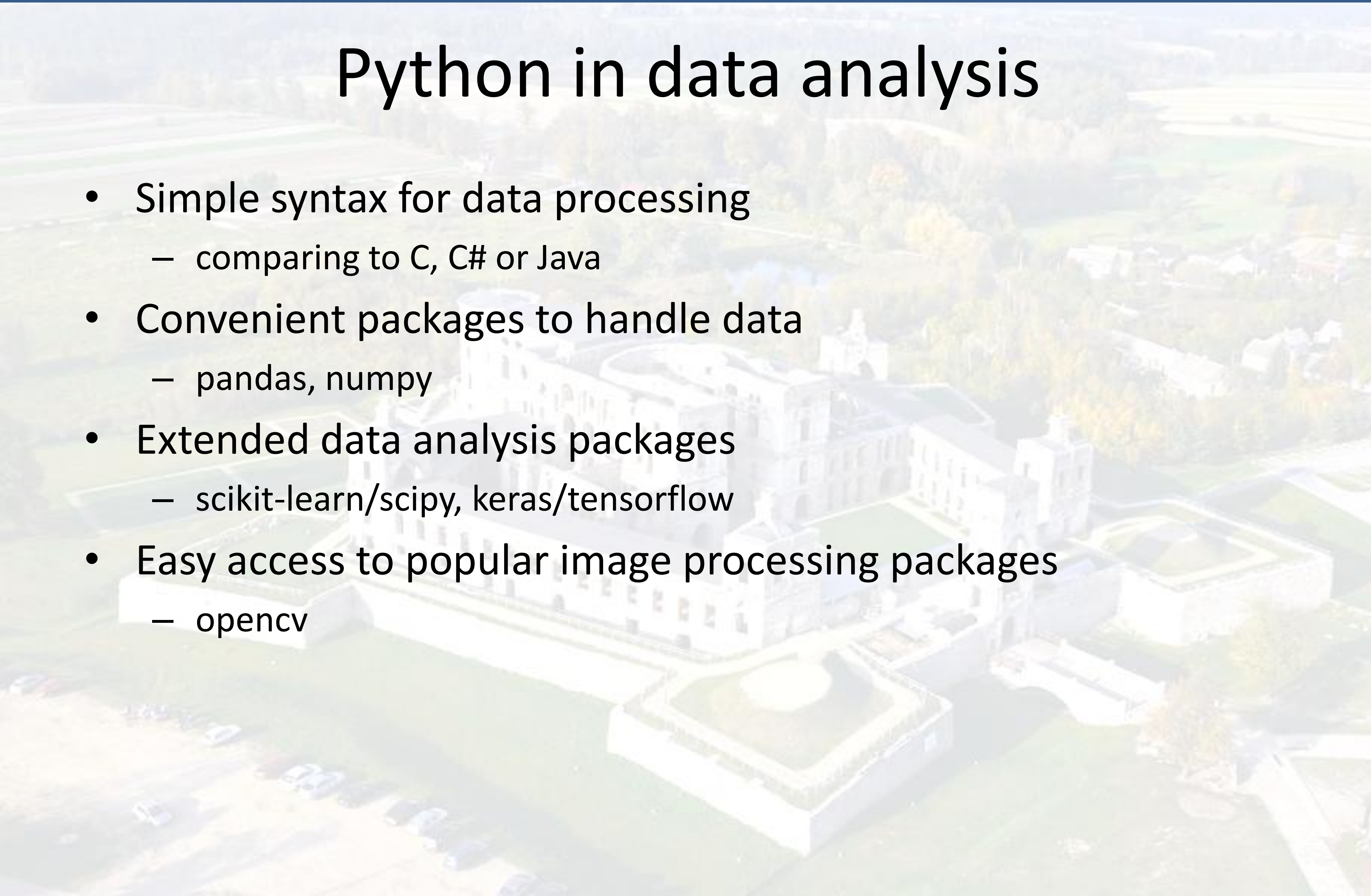
- All materials accesible on GITHUB:
  - https://github.com/kasprowski/yanshan
- Tasks for students after every lecture
  - you may work single or in pairs
- One week to solve the task and send me by email
  - pawel.kasprowski@polsl.pl
- Theoretical test at the end of the course

Python

- Created in 1991 by Guido Van Rossum
- Current version: 3.9
- Interpreted language
- Many libraries/packages (typically written in C or Cython)
- Dynamically typed
- Object oriented
- Open source

Python installation

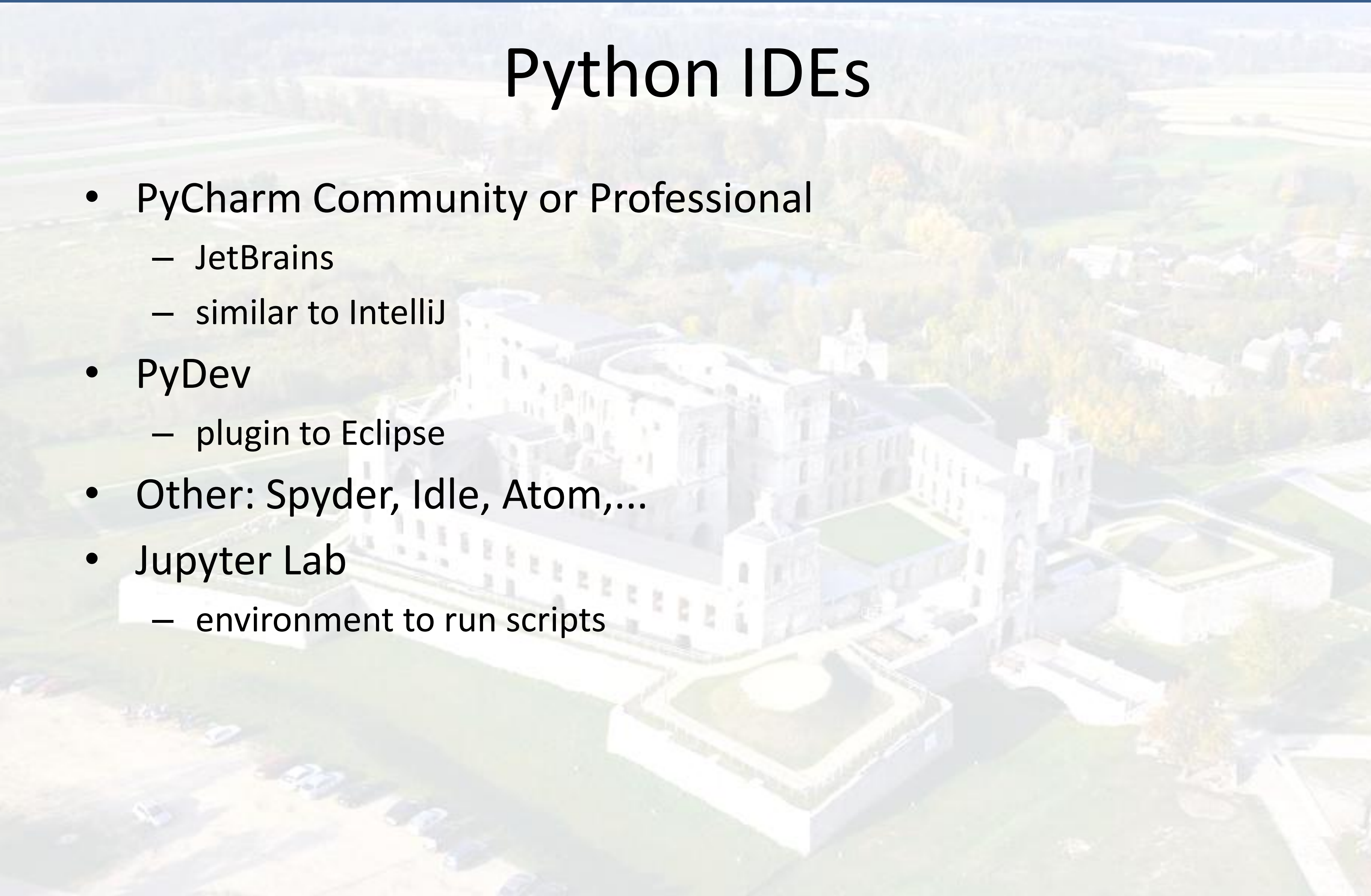
- Downloadable from www.python.org
- After installation shell client: python.exe
- The simplest program:
  - print("Hello world")
- It is possible to create "virtual environments" with different library sets



Python in data analysis

- Simple syntax for data processing
  - comparing to C, C# or Java
- Convenient packages to handle data
  - pandas, numpy
- Extended data analysis packages
  - scikit-learn/scipy, keras/tensorflow
- Easy access to popular image processing packages
  - opencv

Python IDEs

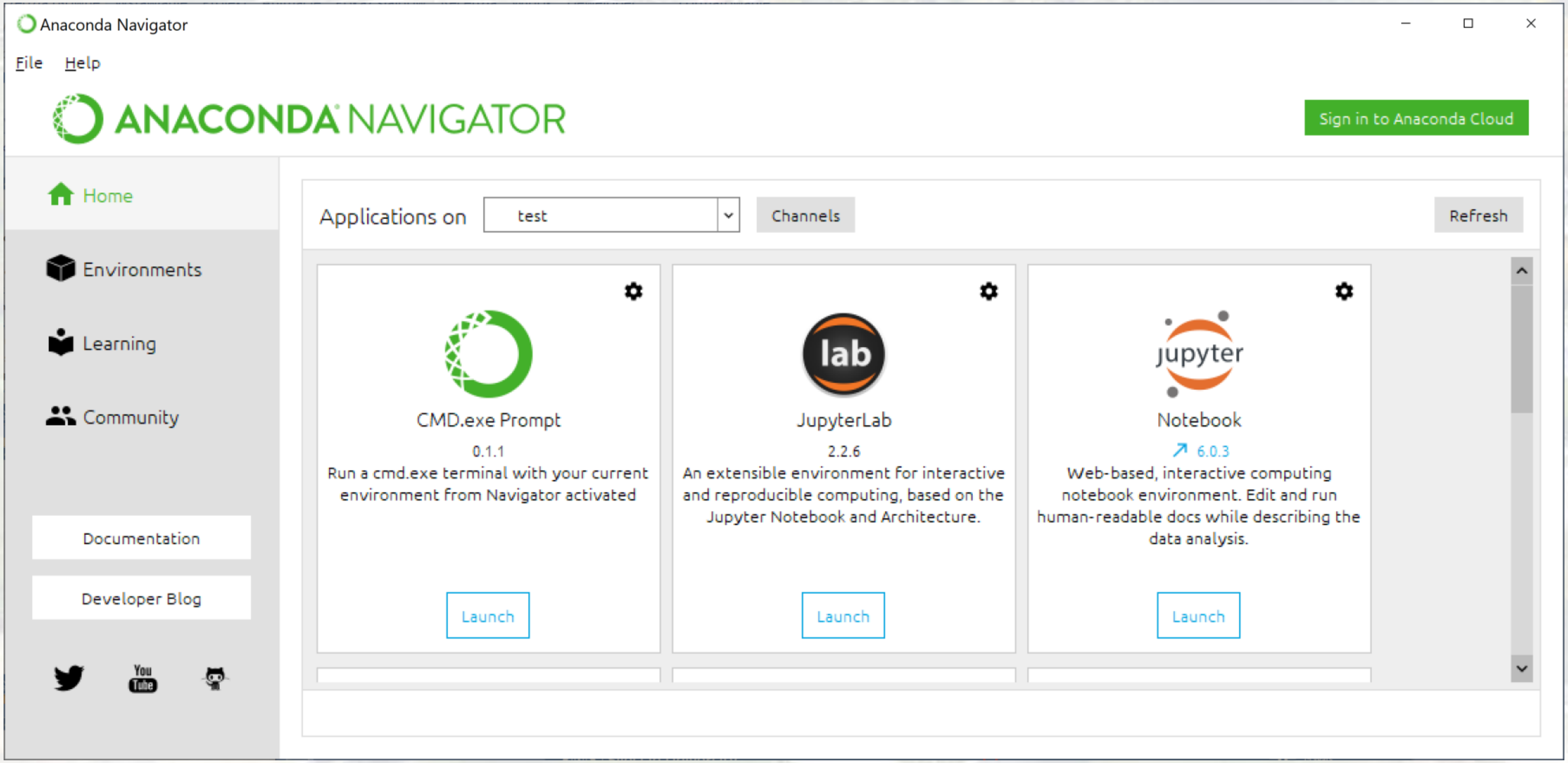


- PyCharm Community or Professional
  - JetBrains
  - similar to IntelliJ
- PyDev
  - plugin to Eclipse
- Other: Spyder, Idle, Atom,...
- Jupyter Lab
  - environment to run scripts

Anaconda/Miniconda

- Working environment for Python and R
- Convenient tool to handle packages
  - many "environments" with different setups
- Special application to control environments:
  - Anaconda Navigator
- Command line tool to install packages:
  - conda (similar to PIP but better in handling dependencies)

Anaconda Navigator



Anaconda Navigator

File Help

ANACONDA NAVIGATOR

Sign in to Anaconda Cloud

Home

Environments

Learning

Community

Documentation

Developer Blog

Applications on test Channels Refresh

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch

JupyterLab
2.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch

Notebook
↗ 6.0.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

Virtual environments

- There is no "project" or "solution"
- Order of operations:
  - create a "virtual environment"
  - activate it
  - install packages (using conda or pip)
  - start IDE
  - write applications using these packages
- It is possible to create many environments with different sets of packages or different versions of the same package

Using conda to manage environments

- Create a new environment:
  - conda create --name deep
- Activate it:
  - activate deep
- Install packages (three possibilities):
  - conda install <package>
  - Anaconda Navigator
  - pip install <package>
- List of currently installed packages:
  - conda list [--explicit]

Using pip

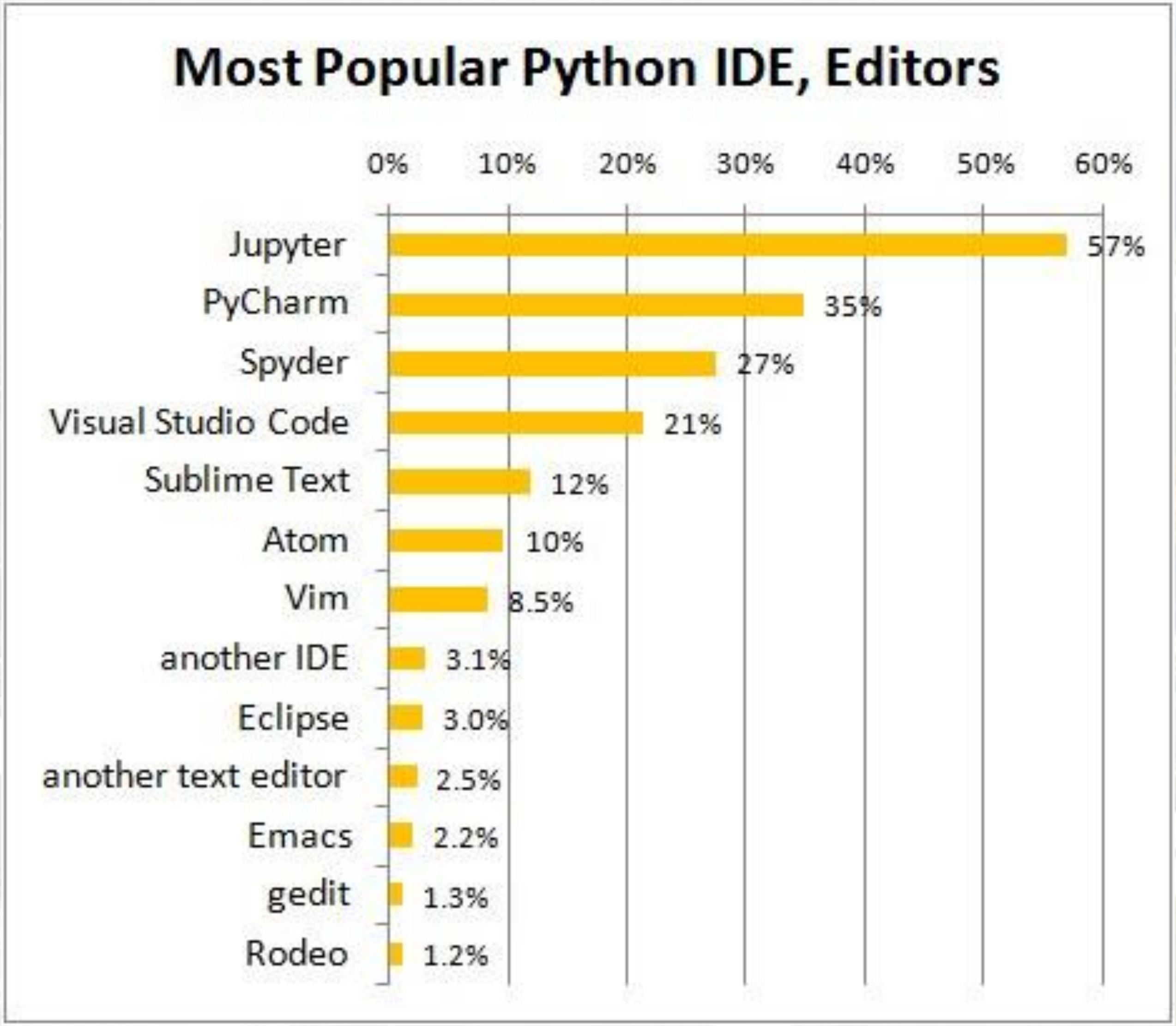
- Some useful packages cannot be found in conda repos
- In such case the standard repository Python Package Index (PyPI) may be used
  - application to use: pip (preferred installer program)
- Installation of the package:
  - pip install [--upgrade] 'package[==version]'
- Installation of multiple packeges from a list:
  - pip install -r requirements.txt

Jupyter Notebook

- Creates files in JSON format that contain "blocks" of code or text (*.ipynb)
- Convenient for scripts
  - REPL (Read-Eval-Print-Loop)
- GUI starts in the web browser
- Uses a kernel that executes code
  - many possible kernels – we will use Python
- The newest version: JupyterLab

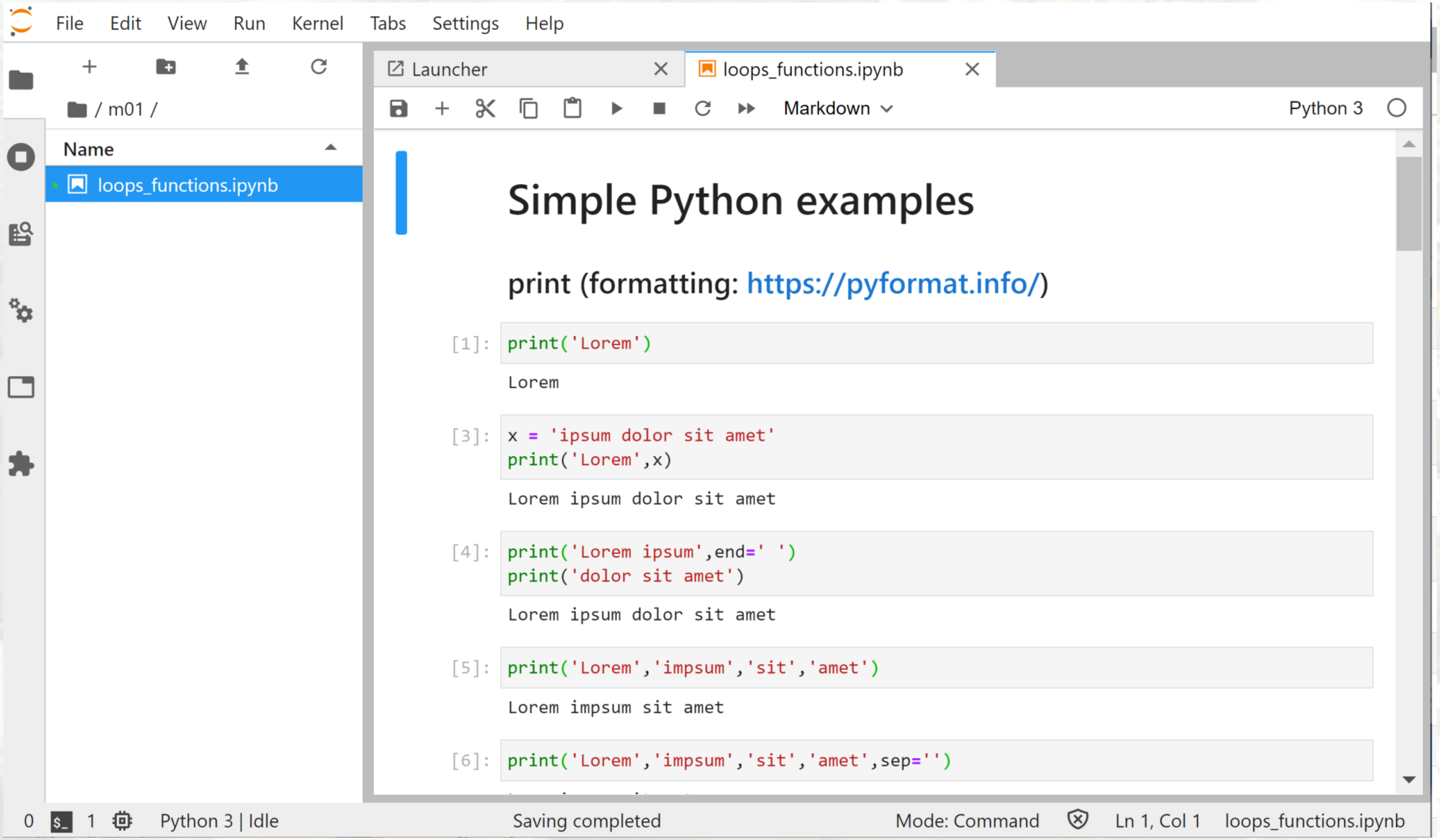
Starting Jupyter Lab

- Choose environment:
  - activate deep
- Install package (and dependencies)
  - conda install jupyterlab
- Start the application (Windows)
  - start jupyter lab [--notebook-dir="c:/directory"]
- Work in the web browser
  - create and modify *.ipynb files
  - execute blocks of code
- Check active applications:
  - jupyter notebook list



https://www.datacamp.com/community/tutorials/top-python-ides-for-2019

JupyterLab



Simple Python examples

print (formatting: https://pyformat.info/)

```
[1]: print('Lorem')
```

```
Lorem
```

```
[3]: x = 'ipsum dolor sit amet'
     print('Lorem',x)
```

```
Lorem ipsum dolor sit amet
```

```
[4]: print('Lorem ipsum',end=' ')
     print('dolor sit amet')
```

```
Lorem ipsum dolor sit amet
```

```
[5]: print('Lorem','impsum','sit','amet')
```

```
Lorem impsum sit amet
```

```
[6]: print('Lorem','impsum','sit','amet',sep='')
```

Using cloud services

- Google Colab
  - https://colab.research.google.com
- Collaborative Calculation and Data Science
  - https://cocalc.com
- Microsoft Azure Notebooks
  - https://notebooks.azure.com/
- and many more...
- Good review of possibilities:
  - https://www.dataschool.io/cloud-services-for-jupyter-notebook/

Python basics

- New line = new command (no semicolons)
- Nesting by identation (no brackets { })
- No strict typing (just a=10 or a="xyz" without a type)
- Comments:
  - # this is a comment

Strings

- Single or double quotes - ' or "
- Three double quotes """ for multiline texts

```
txt = """ Lorem ipsum
dolor sit amet """
```

- Standard methods: find(), index(), len()...
- Comparisons with ==, >, <,...

if and while

- Remember about the indentations!

```
x=10
if x==0:
    print('x is equal to zero')
elif x==10:
    print('x is equal to 10')
else:
    print('x is not zero')

while x>0:
    print("x=",x)
    x=x-1
```

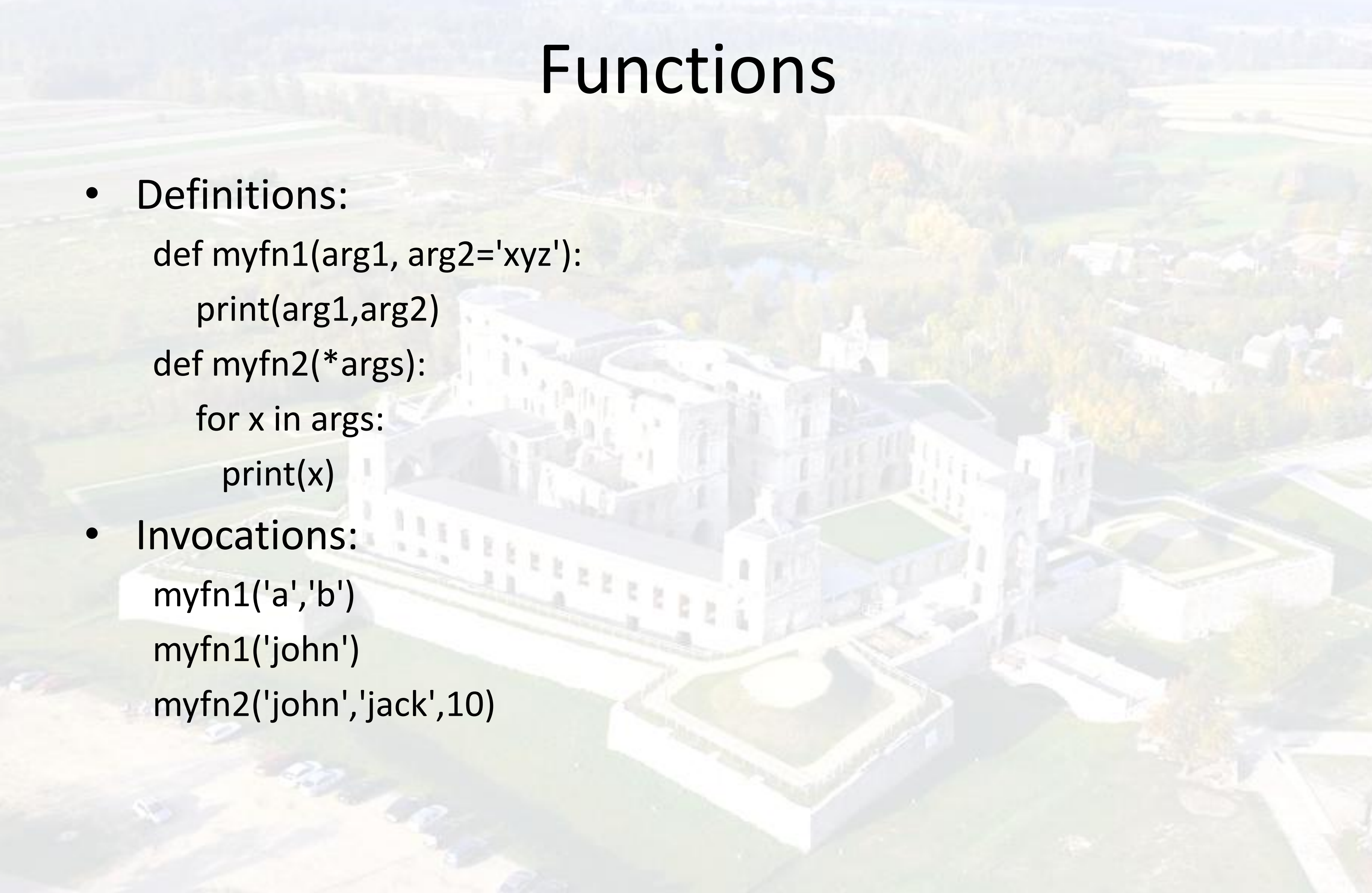
for loop

- for each

```
list = ['john','jack','jane','bob']
for x in list:
    print (x)
```

- for loop – range(start,stop,step) or range(start,stop) or range(stop)

```
for x in range(10):  # 0,1,2,...9
    print(x)
for x in range(10,5,-2): # 10,8,6
    print(x)
```



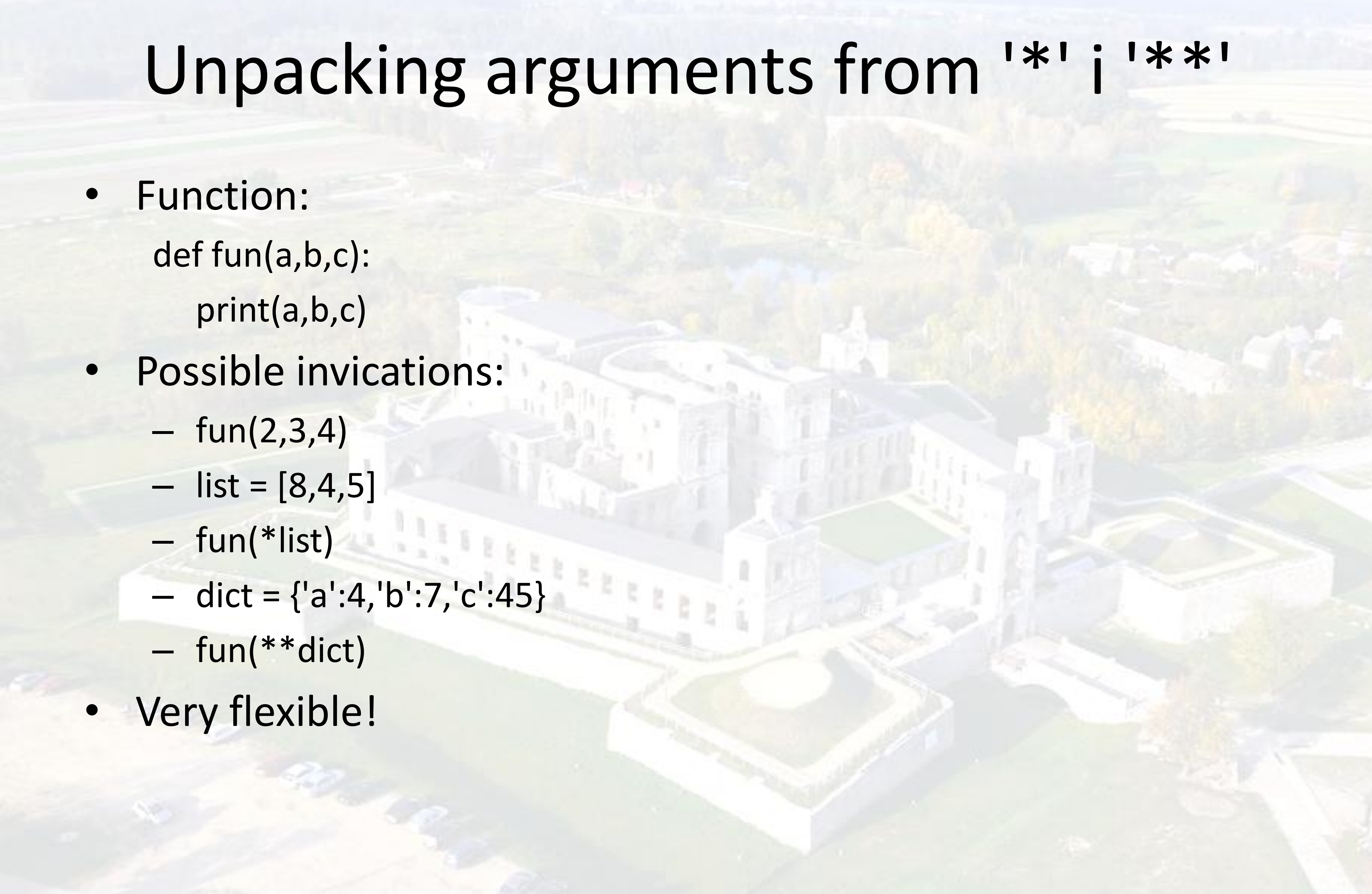
Functions

- Definitions:

```
def myfn1(arg1, arg2='xyz'):
    print(arg1,arg2)
def myfn2(*args):
    for x in args:
        print(x)
```

- Invocations:

```
myfn1('a','b')
myfn1('john')
myfn2('john','jack',10)
```



Unpacking arguments from '*' i '**'

- Function:

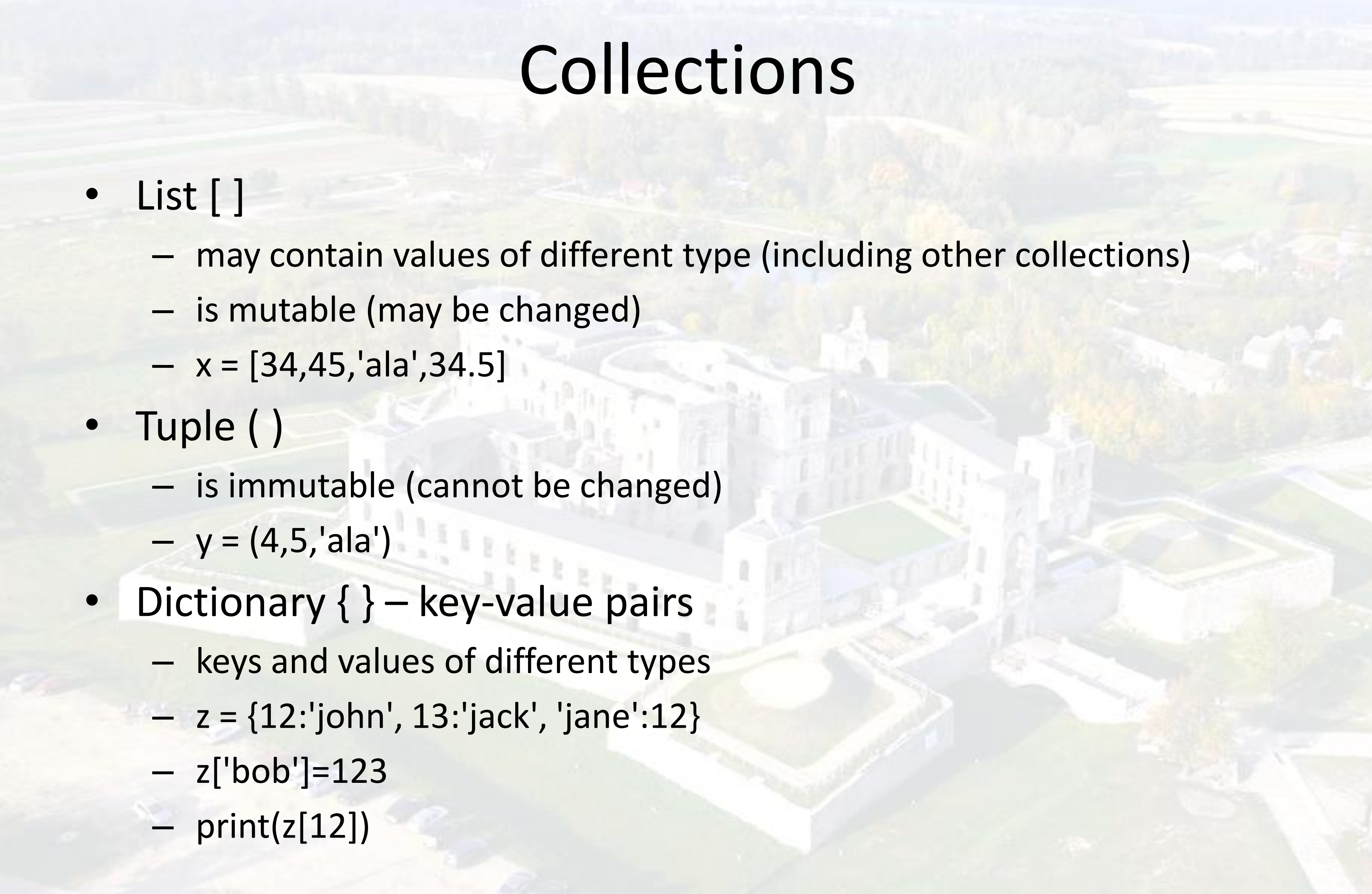
```
def fun(a,b,c):
    print(a,b,c)
```

- Possible invications:
  - fun(2,3,4)
  - list = [8,4,5]
  - fun(*list)
  - dict = {'a':4,'b':7,'c':45}
  - fun(**dict)
- Very flexible!

First Python notebook example

loops_functions.ipynb

- Open: https://colab.research.google.com/
- Choose: GitHub
- Enter: https://www.github.com/kasprowski
- Choose: kasprowski/yanshan
- Choose: m01/loops_functions.ipynb



Collections

- List [ ]
  - may contain values of different type (including other collections)
  - is mutable (may be changed)
  - x = [34,45,'ala',34.5]
- Tuple ( )
  - is immutable (cannot be changed)
  - y = (4,5,'ala')
- Dictionary { } – key-value pairs
  - keys and values of different types
  - z = {12:'john', 13:'jack', 'jane':12}
  - z['bob']=123
  - print(z[12])

Working with collections

collections.ipynb

- list.append(v)
- list2 = list.copy()
- Scopes: list[start:stop:step]
  - list[1:2:2]
  - list[1:] – from the second element
  - list[:2] – first two elements (0 and 1)
  - list[1::2] – from the second every odd element
  - list[:] – the whole list (the same as list.copy())

Some useful functions

- clear()
- len(collection)
- count(x) – how many elements with x value
- extend(list) – add all elements from the other list
- index(x) – indeks of an element with value x
- insert(i,x) – inserts x at index i (the same as list[i]=x)
- pop(i) – deletes i-th element (the same as del list[i])
- remove(x) – deletes the first element with value x
- reverse()
- sort()

Copying lists

- c2 = c1
  - a new reference to **the same** list
- c2 = c1.copy()
  - shallow copy (new list but references the same objects)
- import copy
- c2 = copy.deepcopy(c1)
  - deep copy (all elements are copied)

List comprehension

- Expressions returning lists
- [expr for expr in lista if cond]
- Simple examples:

  [x for x in range(10) if x%2!=0]

  - [1, 3, 5, 7, 9]

  [(x,y) for x in [1,2,3] for y in ['a','b'] if x!=3]

  - [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
- List comprehensions reduce the number of lines in scripts

List may include other lists

- In:
  - x = []
  - y = ['a','b']
  - z = ['c','d']
  - x.append(y)
  - x.append(z)
  - print(x)
- Out
  - [['a', 'b'], ['c', 'd']]

Dictionary

- The dictionary may be built using dict() function
- With fields:

```
dict(alfa = "a", beta = 'b', delta="d")
```

  - {'alfa': 'a', 'beta': 'b', 'delta': 'd'}

- With two-element lists:

```
x = [['a','b'],['c','d']]
dict(x)
```

  - {'a': 'b', 'c': 'd'}

- Using zip() function:

```
x = ['first','second','third']
y = [11,22,33]
dict(zip(x,y))
```

  - {'first': 12, 'second': 22, 'third': 33}

Set

- Collection without duplicates
  - c = [1,2,4,1,2,3,4,0]
  - my_set = set(c)
    - {0, 1, 2, 3, 4}
- No random access (using index)
  - my_set[2] – KeyError!
- Adding and deleting elements
  - s.add(x), s.remove(x), s.discard(x)
- Conversion to list:
  - my_list = list(my_set)

Packages

- Package is a code that may be used in our code
  - it contains useful functions and classes
- It is possible to import the package in another module/file:
  - import pack1 [as alias]
  - functions and variables from pack1 are available as: pack1.function() or pack1.var
- It is also possible to import single functions/variables:
  - from pack1 import func1
  - in that case we don't need a package name for the invocation: func1()
- It is also possible to write: from pack1 import *
  - but typically it is not a good idea – results in a mess in the code

Standard packages

- os, os.path
- sys
- math
- time
- …
- Simple example

```
import os
indir = "."
for file in os.listdir(indir):
    print(file)
```

Useful packages

- Pandas
  - loading data from text files (2D tables)
- NumPy
  - processing multidimensional structures
- Scikit-learn
  - data mining – classification, regression, many algorithms
- Matplotlib
  - charts, visualization
- OpenCV
  - image processing
- Keras/Tensorflow
  - deep learning, neural networks

Package installation

- conda install pandas
  - (installs also the numpy package)
- conda install jupyterlab
- conda install scikit-learn
- conda install matplotlib
- conda install opencv
- conda install tensorflow

Pandas package

pandas.ipynb

- Package for 2D tables (DataFrame)
  - import pandas as pd
- Load data
  - df = pd.read_csv(plik,sep='',...)
- Show columns and types
- Add new columns
- Drop columns and rows
- Search for rows
- Save data

NumPy package

- Basic structure: NumPy array
  - multidimensional array (tensor) of objects of one type

numpy.ipynb

- Creating:
  - array = np.array(<elements>)
  - np.zeros(<dimension>)
  - np.ones(< dimension >)
  - np.full(< dimension >,value_to_fill)
  - np.random.random(< dimension >)
- Check array dimension:
  - np.shape(array) or array.shape

Multidimensionality

- array3d = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
- array3d.shape
  - (2,2,3)

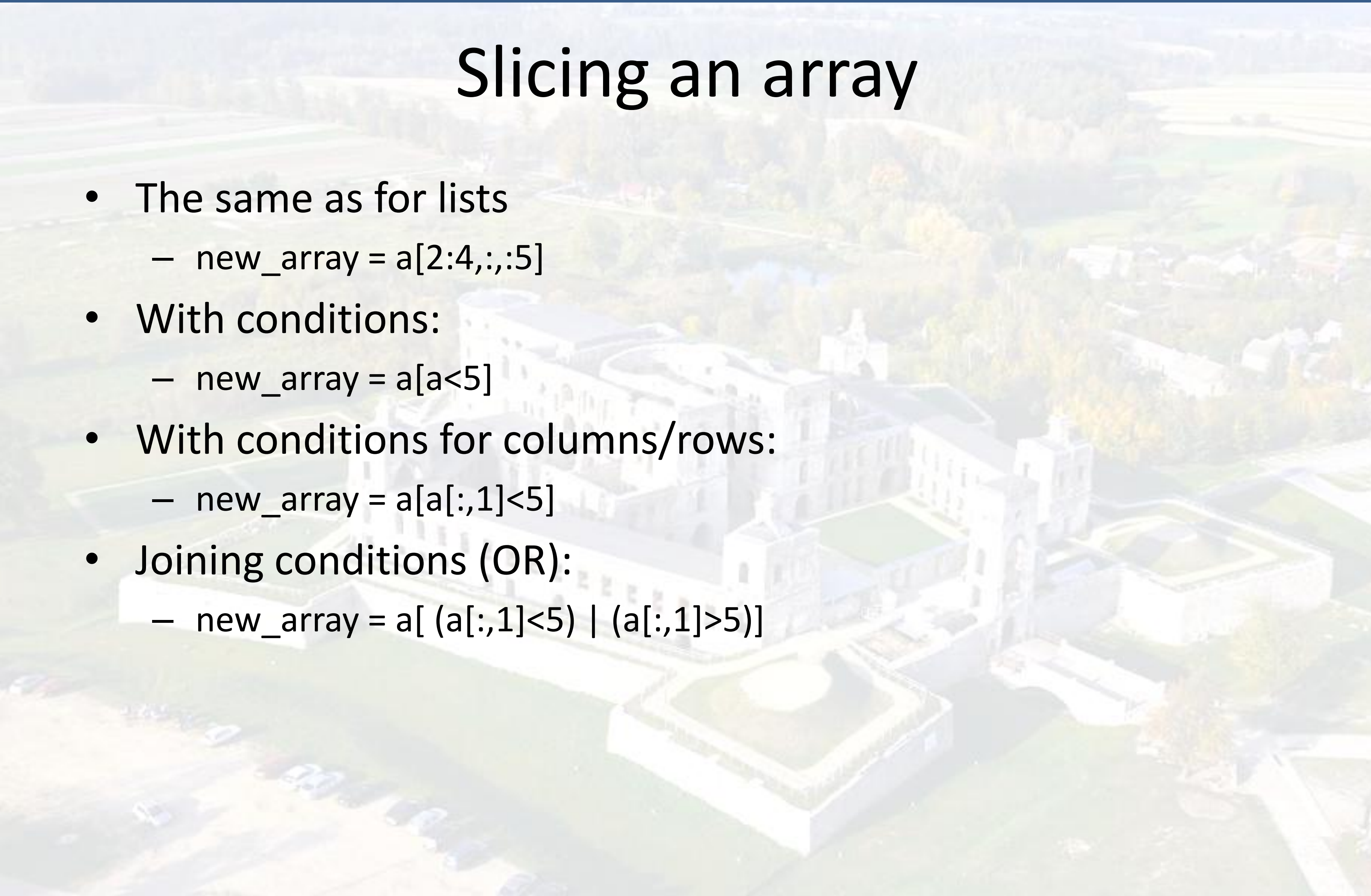
```
[
    0:[
        0:[ 1 2 3]
        1:[ 4 5 6]
    ]
              0 1 2
    1:[
        0:[ 7 8 9]
        1:[10 11 12]
    ]
] >> array3d[1,1,1] >> 11
```

(X Y Z)

Changing shape (dimensions)

- Transposition:
  - a.T
- reshape():
  - a.reshape(2,3,4)
  - a.reshape(1,-1) # -1 means "choose a correct value"
- Funkcja ravel() – flattening to 1D
  - a.ravel() # tthe same as a.reshape(-1)
- expand_dims() – adding a new dimension (of size=1)
  - x = np.expand_dims(a, axis=0)
  - y = np.expand_dims(a, axis=1)

Slicing an array



- The same as for lists
  - new_array = a[2:4,:,:5]
- With conditions:
  - new_array = a[a<5]
- With conditions for columns/rows:
  - new_array = a[a[:,1]<5]
- Joining conditions (OR):
  - new_array = a[(a[:,1]<5) | (a[:,1]>5)]

Load/save

- Binary files:
  - array = np.load('file.npy')
  - np.save('file',array)
- Compressed (or many arrays) :
  - np.savez_compressed('file',array)
  - array_dict = np.load('file.npz')
- Text (only 1D or 2D):
  - new_array = np.loadtxt('array')
  - np.savetxt('array',array)
  - np.genfromtxt(...) – more params than loadtxt!

Operations on arrays

- Arrays must have compatible dimesions
- Adding, substracting, multiplying, dividing:
  - x = a+b
  - y = a*b
- Aggregations:
  - np.sum(a)
  - np.mean(a)
  - np.std(a)
  - and much more...
- Also for single columns/rows:
  - np.sum(a, axis=1)
  - a.sum(axis=0)

Broadcasting

- It is possible to do operations on tables with different dimensions – but one must have a dimension with size=1
- Example:

```
a = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]]) # shape (4,3)
b = np.array([1,3,6]) # shape (1,3)
print(a+b)
        [[ 2, 5, 9], [ 5, 8, 12], [ 8, 11, 15], [11, 14, 18]]
```

- Array b is added to every row of array a

Matplotlib

- Extended library to create charts and visualizations
  - import matplotlib.pyplot as plt

plot.ipynb

- Some types of charts:
  - plt.plot(x,y)
  - plt.scatter(x,y)
  - plt.bar(x,y)
  - plt.hist(x) – the default number of bins=10

Deep Learning with Python

photo: Krzyżtopór Castle, Poland

Next lecture: Introduction to Data Mining

Introduction to Python

Paweł Kasprowski, PhD, DSc.