The many shapes of the Python programming language

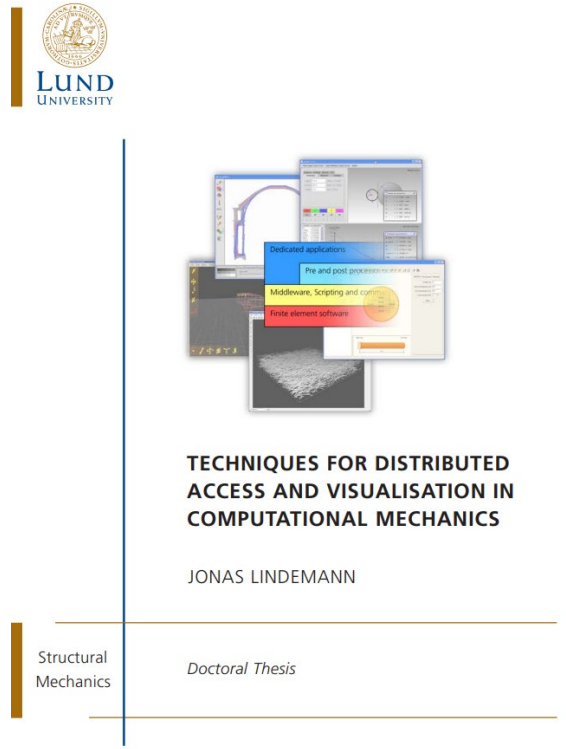
Jonas Lindemann
LUNARC, Lund university

LUND UNIVERSITY


WHO
AM I?

Who am I?

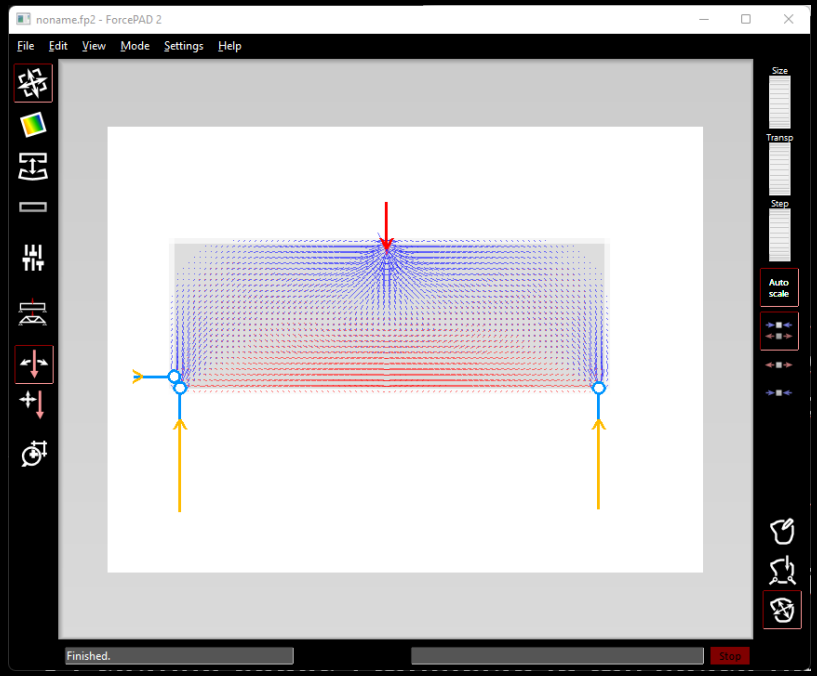
- PhD in Structural Mechanics
  - User interfaces concepts for finite element codes for architects and designers
  - Component based finite element applications (CORBA)
  - Visualisation techniques for fibre networks
  - Python / C++ / Fortran (yes you heard right)
- Director of LUNARC
  - Centre for Scientific and Technical Computing at Lund university

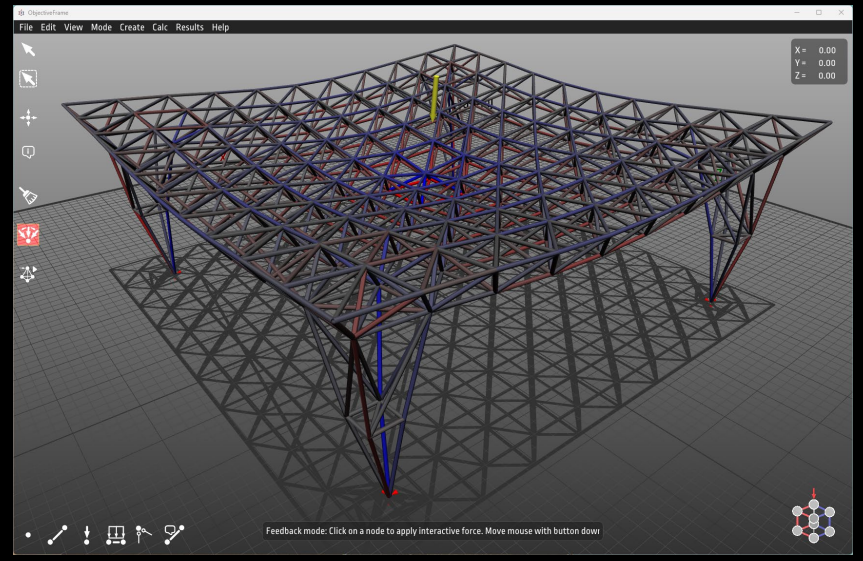


LUNARC

Who am I?

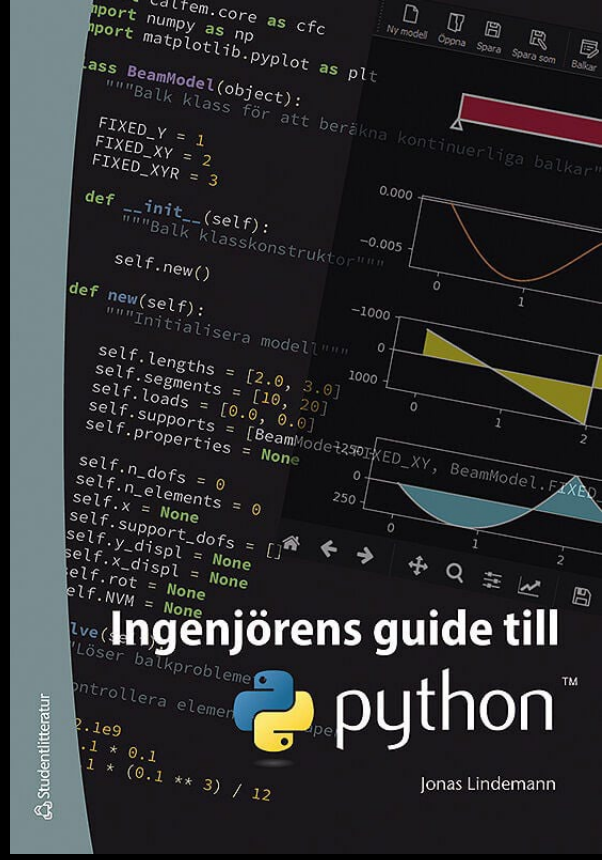
- Developer
  - ForcePAD – Educational software for 2D finite element
  - ObjectiveFrame – Educational 3D beam application
  - Hacon – FEA tool for simulating hardening concrete
  - Interactive Visualisation Framework – Ivf++
  - Co-author of CALFEM for MATLAB and Python (FE-toolbox)
  - QtCreator Fortran extensions
  - GfxLauncher – Software for launching graphical applications on compute clusters (Python)
- Creative coder / musician in my spare time
  - Processing / py5 / Renoise

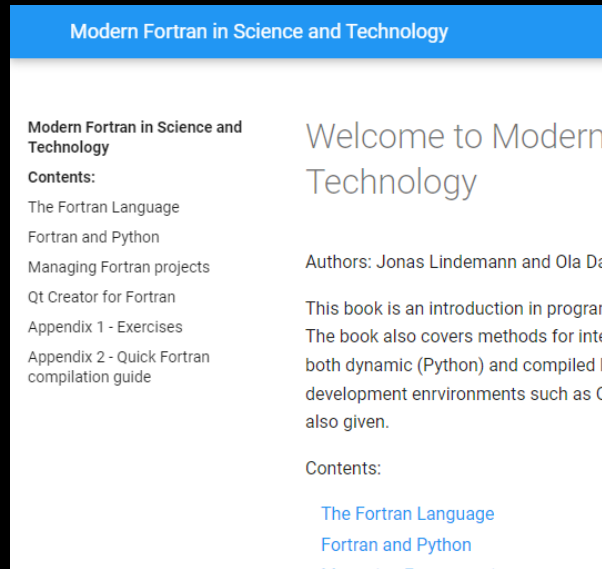




Who am I?

- Author
  - Ingenjörens Guide till Python (The Engineers Guide to Python)
  - Modern Fortran in Science and Technology (Online)
- Teaching
  - Software Development for Technical Applications (Python)
  - Programming in Science and Technology (Python/Fortran)
  - Introduction in to Programming in Science and Technology (Python/Fortran)
  - Efficient programming of modern HPC (Python/Fortran)
  - Advanced Programming in Science and Technology (C++)
  - Scientific Programming in Python and Fortran





Background

How I stumbled into Python programming?

…after my PhD defence – 2003

- Most of my research was in distributed computing and visualisation
- A lot of C++ and Fortran code
- My opponent Hans-Petter Langtangen asked me:

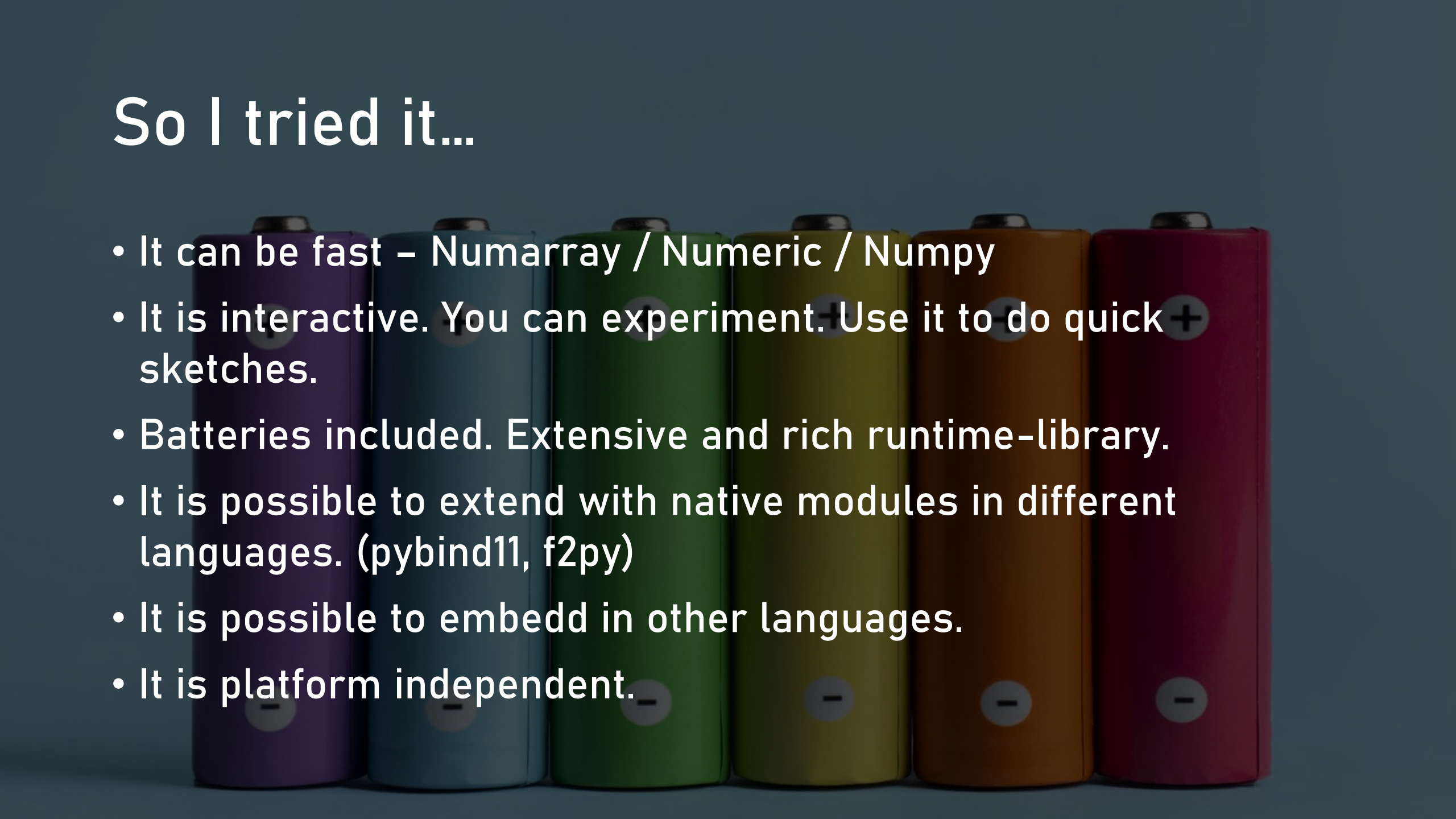
"Have you tried Python?"

Python 2.2

Python and a C++/Fortran programmer?

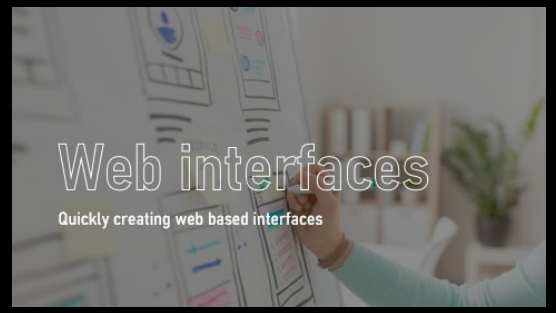
- Python is a scripting language?
- Python can’t be fast
- Why should I switch from a compiled language?
- What use-cases?

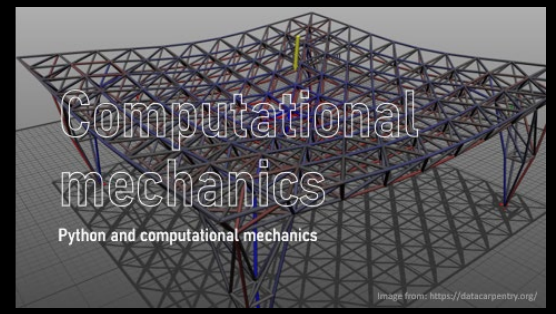
So I tried it...



- It can be fast – Numarray / Numeric / Numpy
- It is interactive. You can experiment. Use it to do quick sketches.
- Batteries included. Extensive and rich runtime-library.
- It is possible to extend with native modules in different languages. (pybind11, f2py)
- It is possible to embedd in other languages.
- It is platform independent.

The shapes of Python





Teaching

Using Python in Teaching

Image from: https://datacarpentry.org/

Active learning

Getting students to practice and retrieve knowledge learnt during the lecture

Image from: https://datacarpentry.org/

Engaging students during a lecture

- Historically, all lectures were static PowerPoint slides
  - Difficult to get students involved in the courses
  - I could be as committed as possible, and some fell asleep anyway…
- How can you create more interaction and engagement during the lecture?

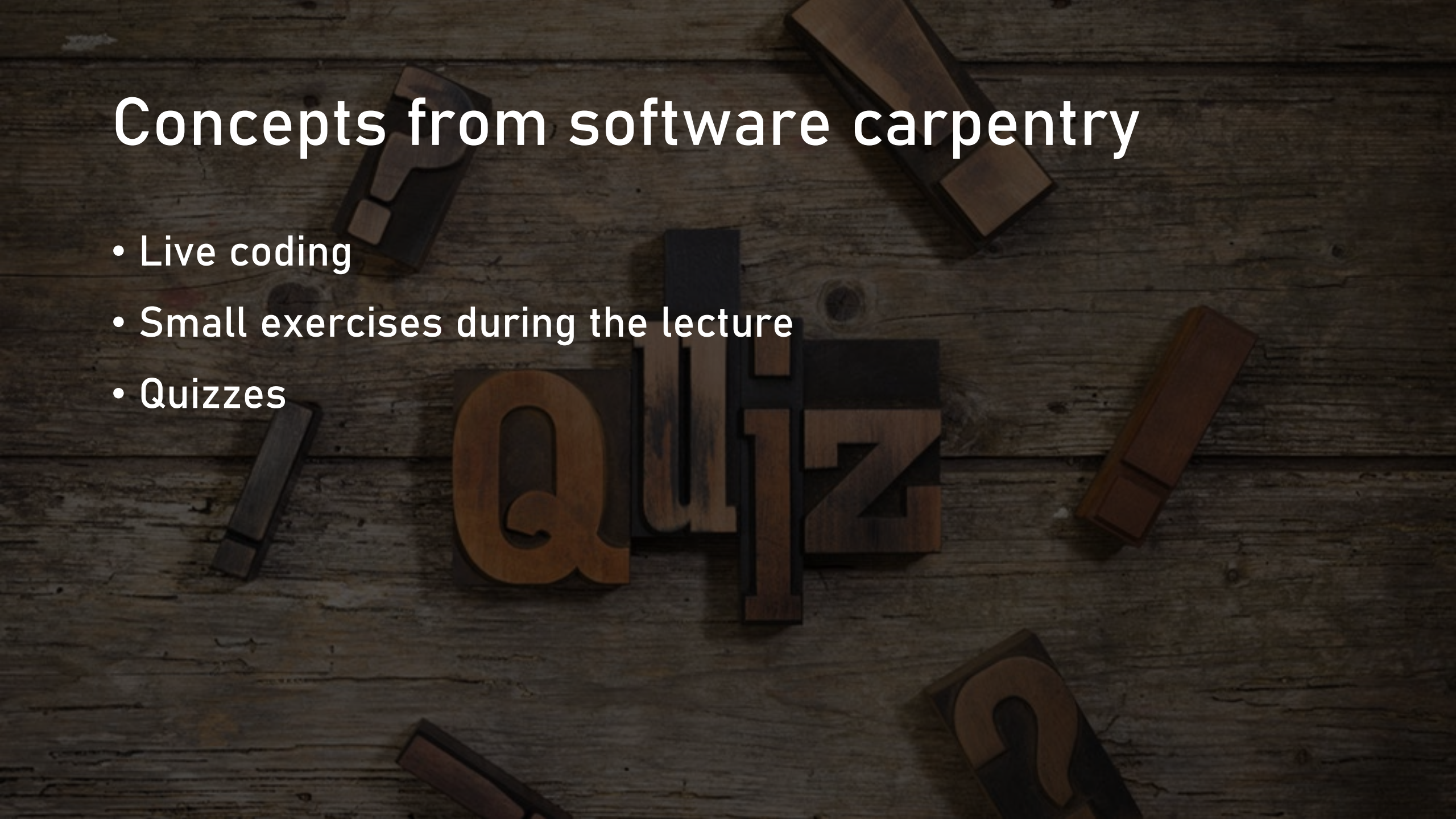
How Does Active Learning Support Student Success? - Teach Online (asu.edu)

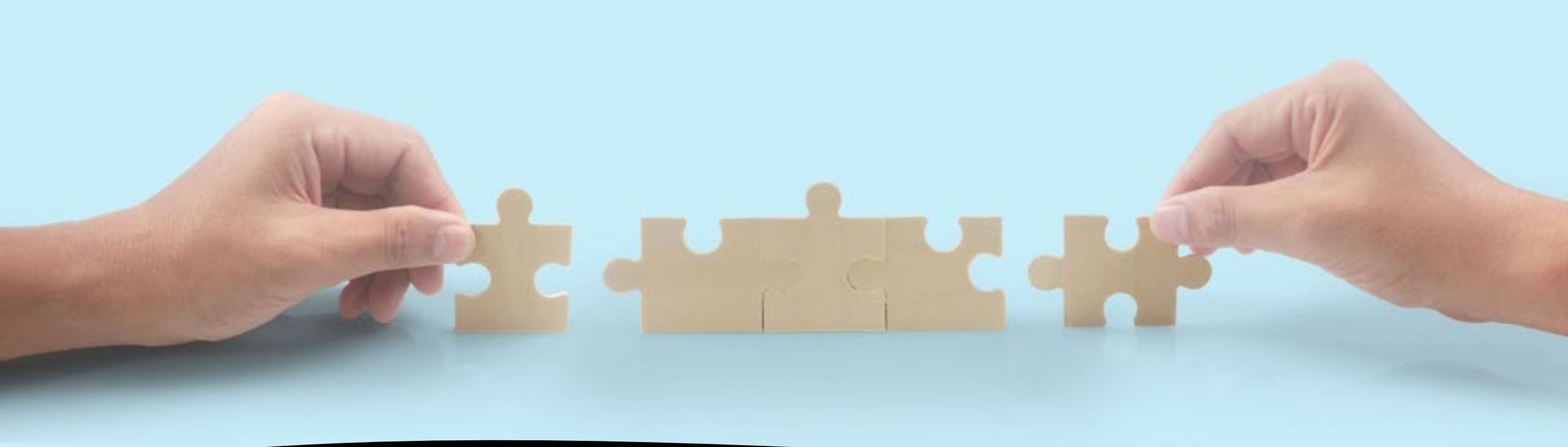
Software carpentry

- Students are encouraged to follow the lectures that the teacher conducts live.
- Post-It notes to inform the teacher if you are done with your assignment
- The concept is exciting but challenging to implement at LTH / LU
  - Requires extra teaching assistants who can help the students during live lectures.
- We can use some of the concepts

Concepts from software carpentry

- Live coding
- Small exercises during the lecture
- Quizzes





Solutions

- Provide environments where students can follow along during lectures.
- Break lecture with opportunities to practice acquired knowledge
- How do we implement this?

Replacing PowerPoint slides

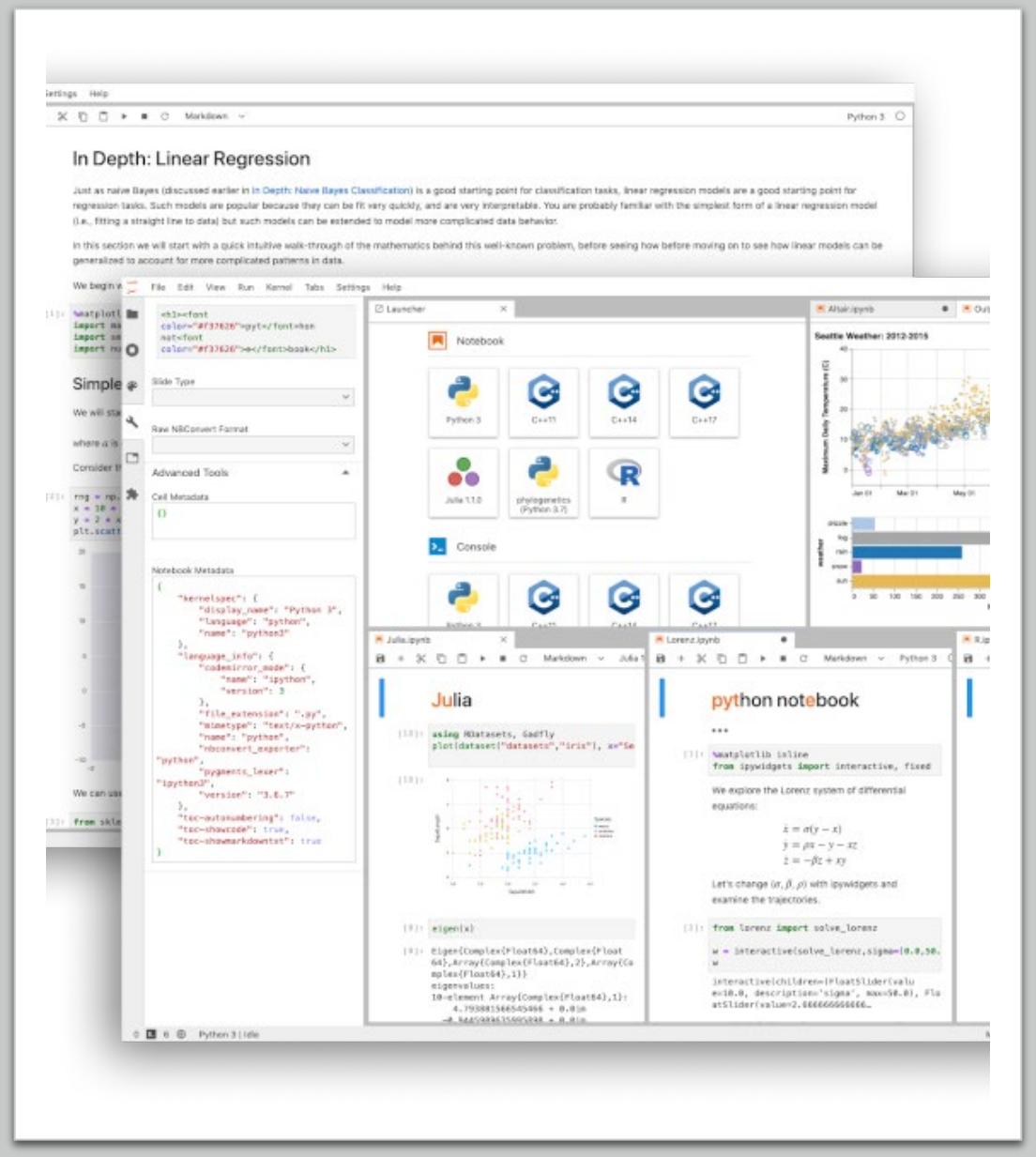
Finding an alternative interactive PowerPoint alternative

Finding a PowerPoint alternative

- PowerPoint slides are not interactive for the students
- Code examples are static
- Most of my existing course material where slides with code examples…
- Jupyter Notebooks can show both text and code?
- This could work!

Jupyter Notebooks

- Document format based on JSON
  - Record of user session, containing code, text, equations and rich output
- Interactive Computing protocol
  - Communicates with computational kernels
- Kernel
  - Runs interactive code in a specific language and returns output



Running notebooks

- Notebooks can be run locally using Anaconda or similar environments
- Many providers have notebook services that enable users to run the notebooks in the cloud

Google Colab

- Provides Notebooks in the cloud
- Free tier works well for teaching purposes
- A URL can be provided to the students before the lecture
- Requires a google-account to run.
- Notebook can be downloaded and run locally

Example lecture Colab

https://bit.ly/pycon-2022-python-intro-colab

SCAN ME

Introduction to Python

Arkiv Redigera Vy Infoga Körning Verktyg Hjälp Senast redigerad 1 september

Kommentar Dela

Anslut Redigerar

Section outline

Starts a kernel for running code

Files generated by running code

Extra comments/explanations for the lecture

Language history

- Created by Guido van Rossum in 1991

ty, to be able to write la

Language features

- Dynamically typed (described in detailed later in this notebook)
- Supports multiple programming paradigms
  - Structured/procedural
  - object-oriented
  - functional

Language versions

- Python version 2.0 released in 2000
- Python version 3.0 released in 2008 (Not entirely backwards compatible)
- Latest 2.x version is 2.7.18 have reached end of life and new developments on this version should be avoided.
- Latest 3.x version is 3.10.3. We will be using 3.9 in this course.

Main Python page: https://www.python.org

Anaconda Python distribution: https://www.anaconda.com/

WinPython: https://winpython.github.io/

Language philosophy

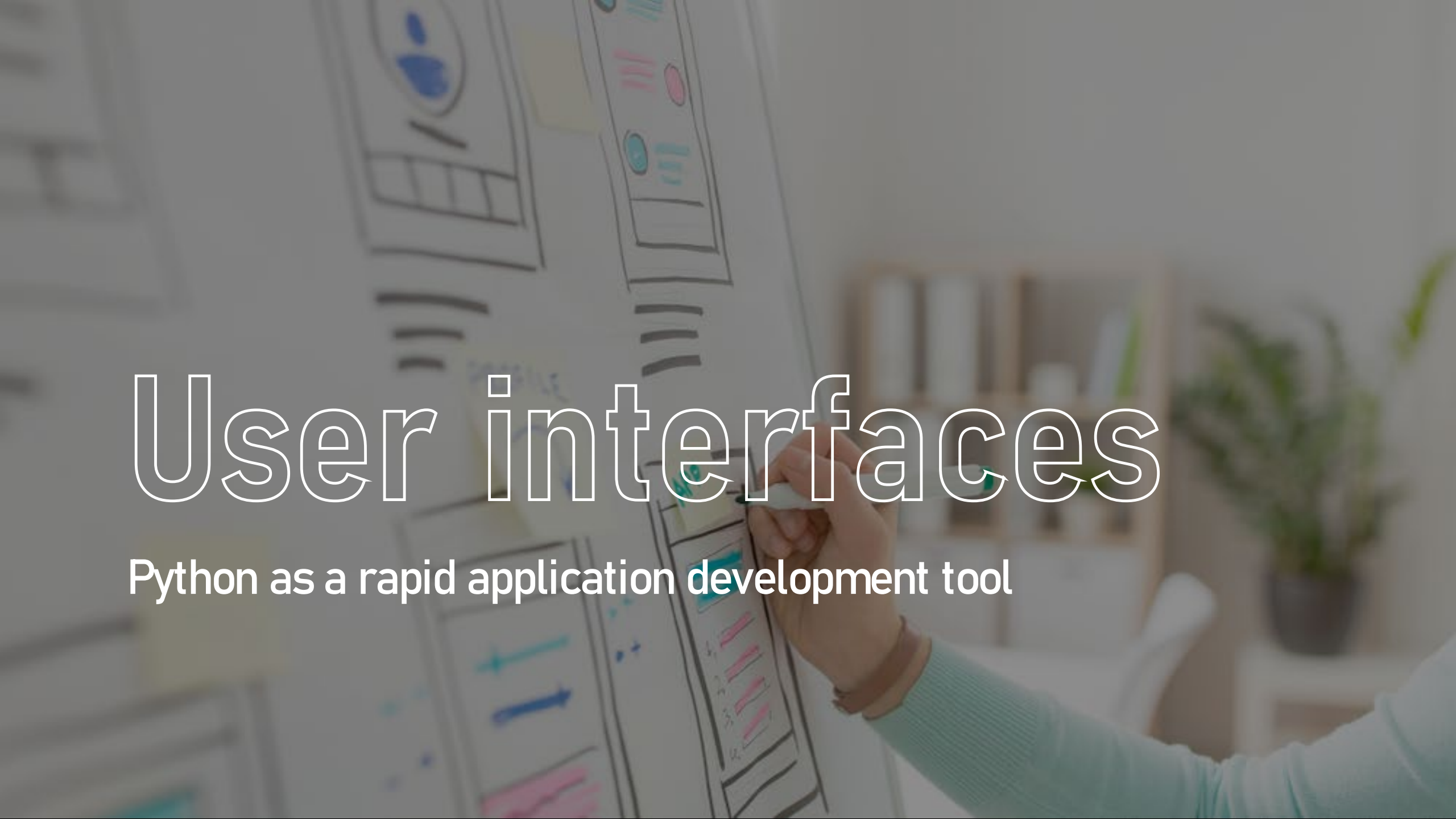
Jonas Lindemann
6 sep. 2021

The main Python page for downloading the base version of Python is:

https://www.python.org/

Jonas Lindemann
6 sep. 2021

For Scientific Computing you usually install a Python distribution that conatins all relevant packages, such Anaconda or PythonXY.



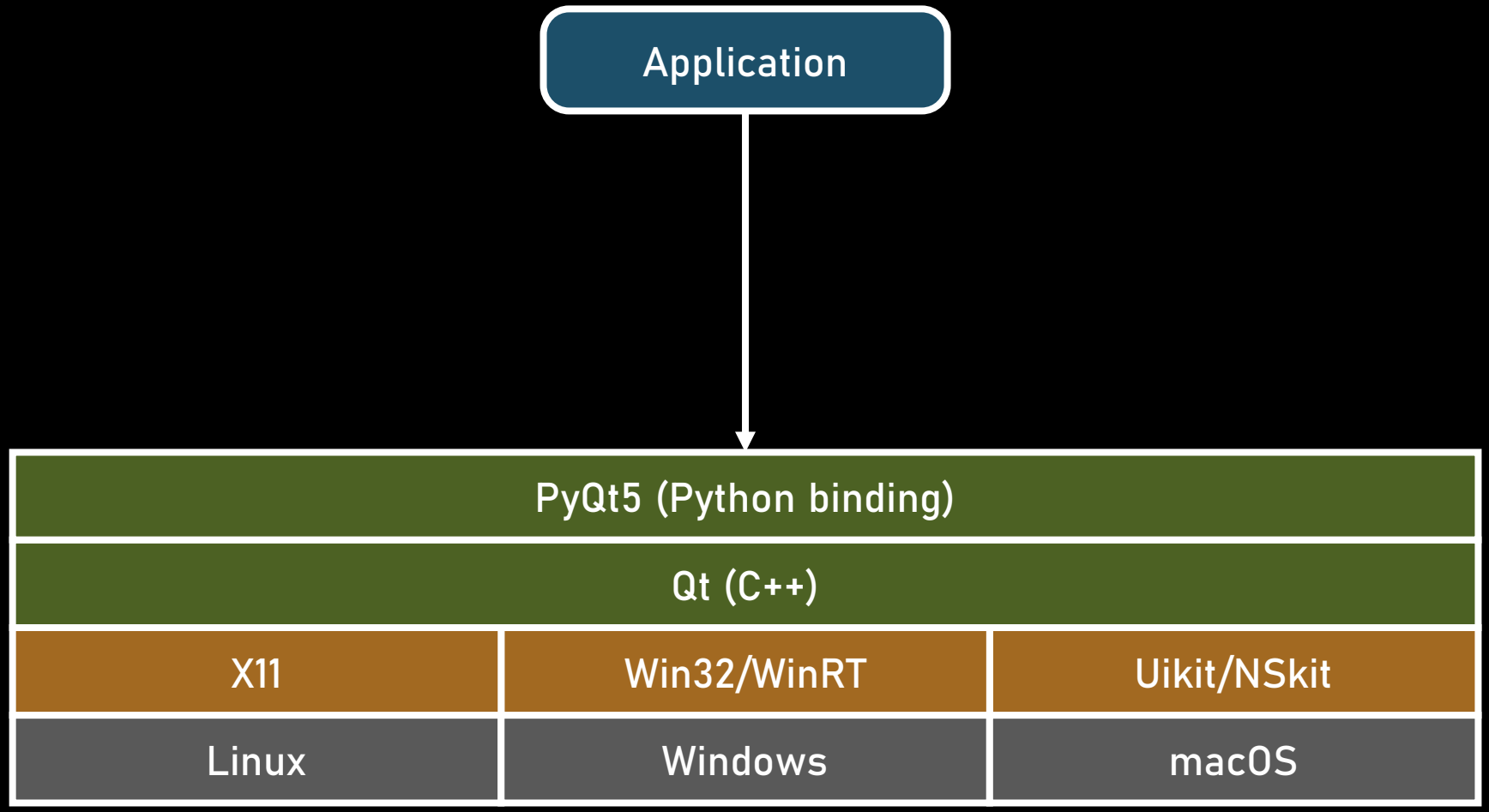
User interfaces

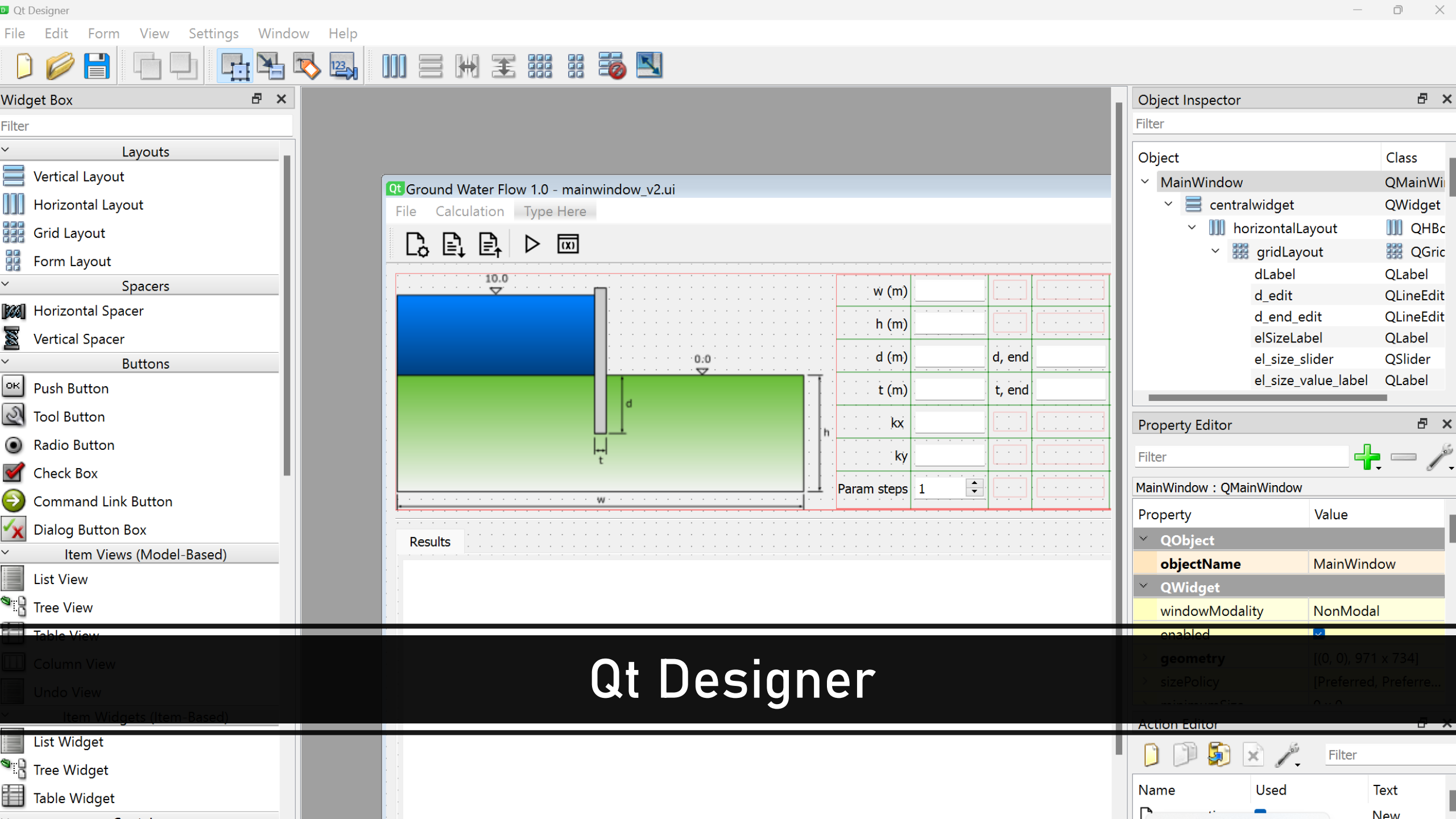
Python as a rapid application development tool

User interfaces in Python

- The dynamic nature of Python is very suitable to quickly implement user interfaces
- There exists several user interface toolkits for Python
- Tkinter
  - Comes with Python – Easy to use. Looks a bit dated.
- wxPython
  - Python binding for the wxWidget C++ library. Well-proven.
- PyQt/PySide/Qt for Python
  - Python bindings for the Qt C++ library. Open source / Commercial license. Well proven. Comes with a RAD tool Qt-designer

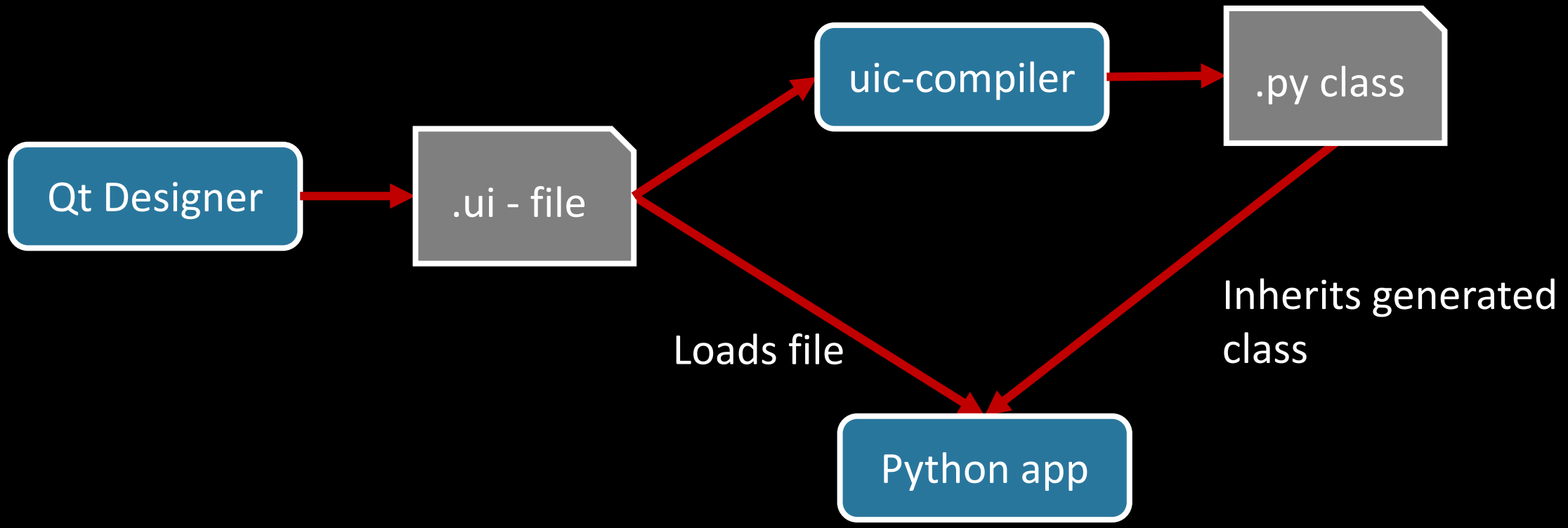
Qt/PyQt library architecture




Qt Designer
File Edit Form View Settings Window Help
Widget Box
Filter
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Dialog Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Undo View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Ground Water Flow 1.0 - mainwindow_v2.ui
File Calculation Type Here
10.0
0.0
d
t
h
w
w (m)
h (m)
d (m)
d, end
t (m)
t, end
kx
ky
Param steps
1
Results
Object Inspector
Filter
Object
Class
MainWindow
QMainWi
centralwidget
QWidget
horizontalLayout
QHBc
gridLayout
QGric
dLabel
QLabel
d_edit
QLineEdit
d_end_edit
QLineEdit
elSizeLabel
QLabel
el_size_slider
QSlider
el_size_value_label
QLabel
Property Editor
Filter
MainWindow : QMainWindow
Property
Value
QObject
objectName
MainWindow
QWidget
windowModality
NonModal
enabled
geometry
[(0, 0), 971 x 734]
sizePolicy
[Preferred, Preferre...
Action Editor
Filter
Name
Used
Text
New

Qt Designer

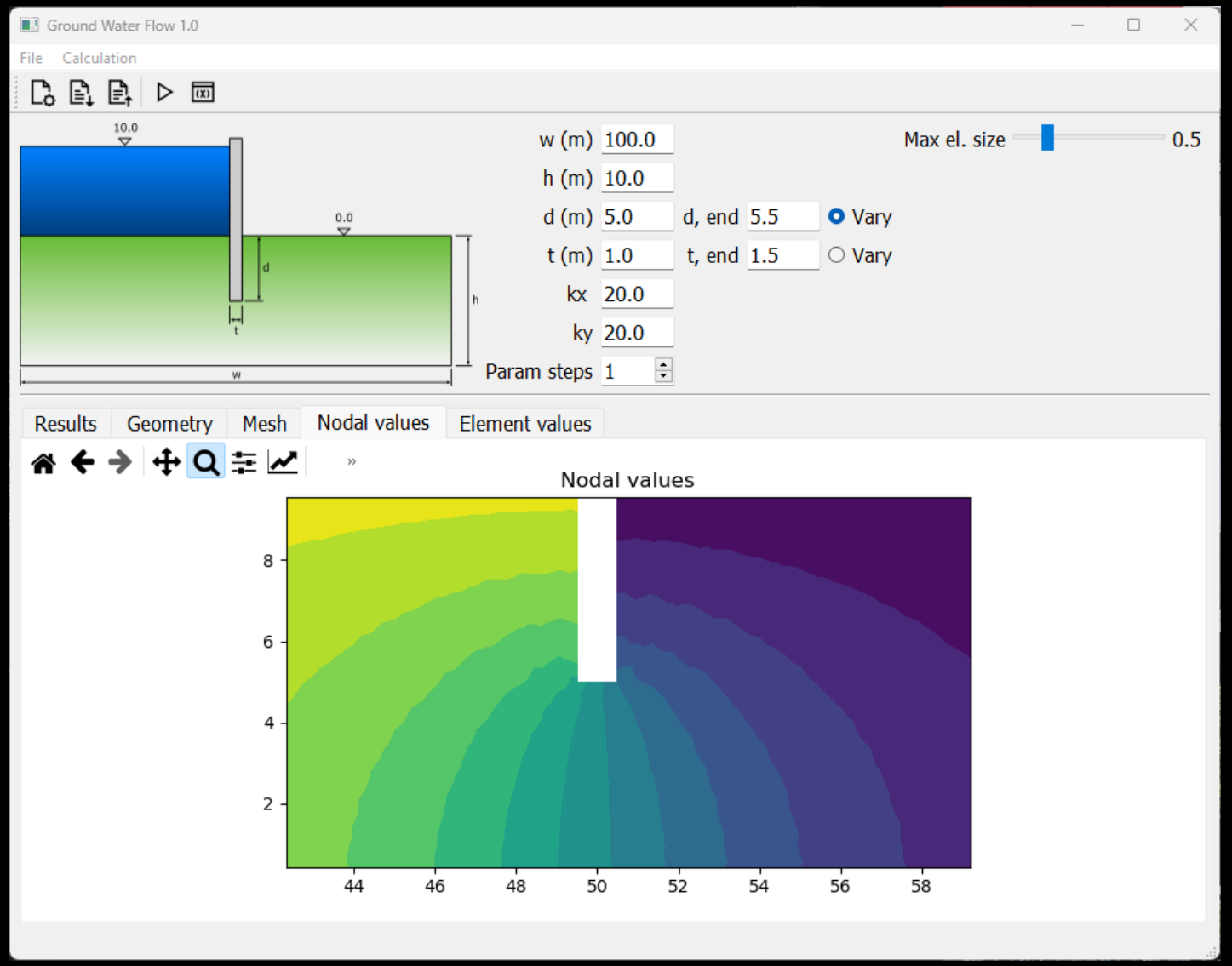
Creating user interfaces in PyQt

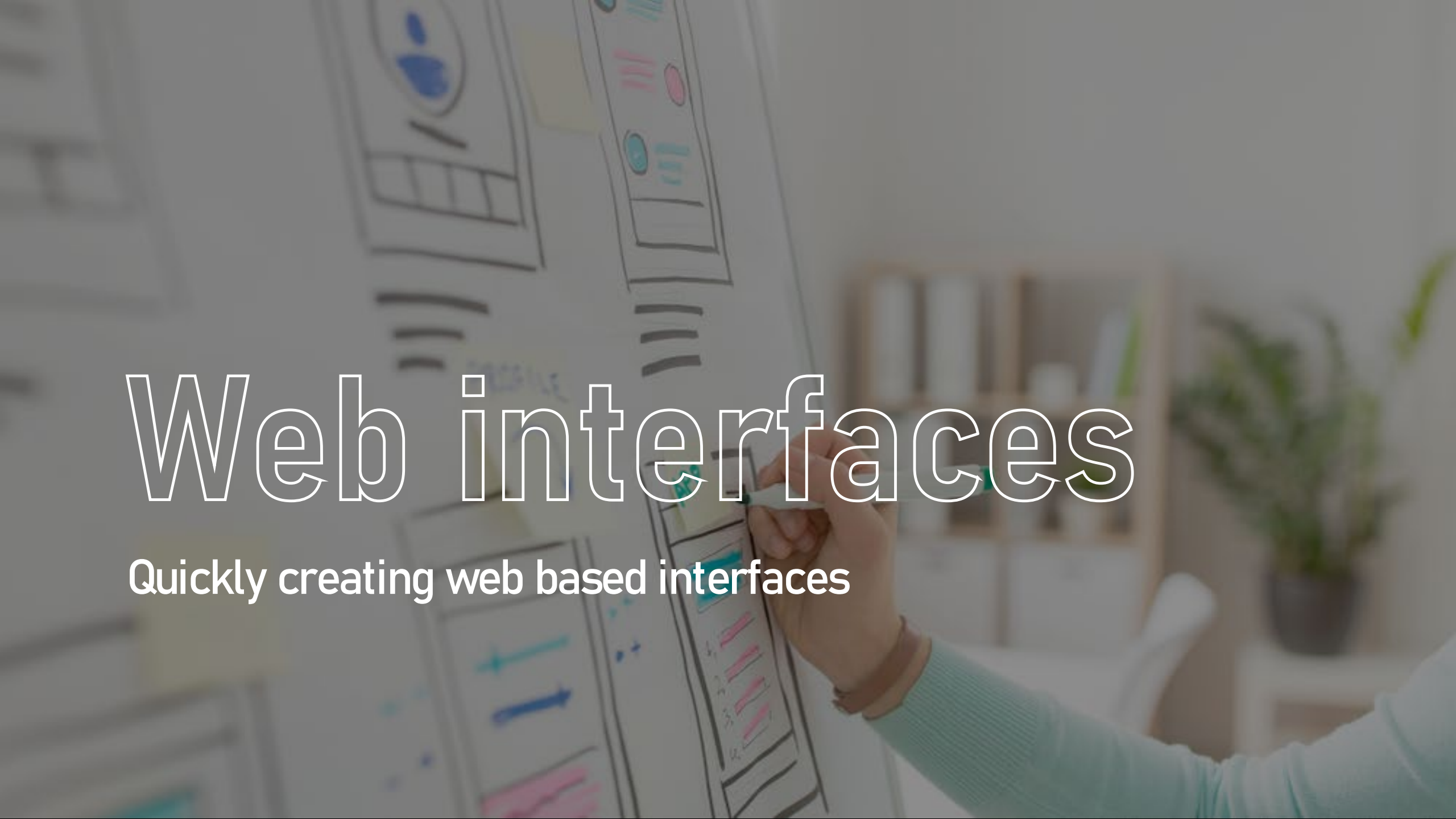


Example of PyQt code

```
class MainWindow(QMainWindow):
    """MainWindow-klass som hanterar vårt huvudfönster"""

    def __init__(self, app):
        """Class constructor"""

        super().__init__()

        # --- Lagra en referens till applikationsinstansen i klassen

        self.app = app

        # --- Läs in gränssnitt från fil

        uic.loadUi("mainwindow_v2.ui", self)

        # --- Koppla kontroller till händelsemetoder

        self.new_action.triggered.connect(self.on_new_action)
        self.open_action.triggered.connect(self.on_open_action)
```

Example user interface developed in the VSMN20 course (Mechanics)





Web interfaces

Quickly creating web based interfaces

Easy web interfaces in Python

- There is a multitude of web frameworks for Python
- Often very complex to use
- Flask is an easy to use framework for quick development of web based applications

A very short example

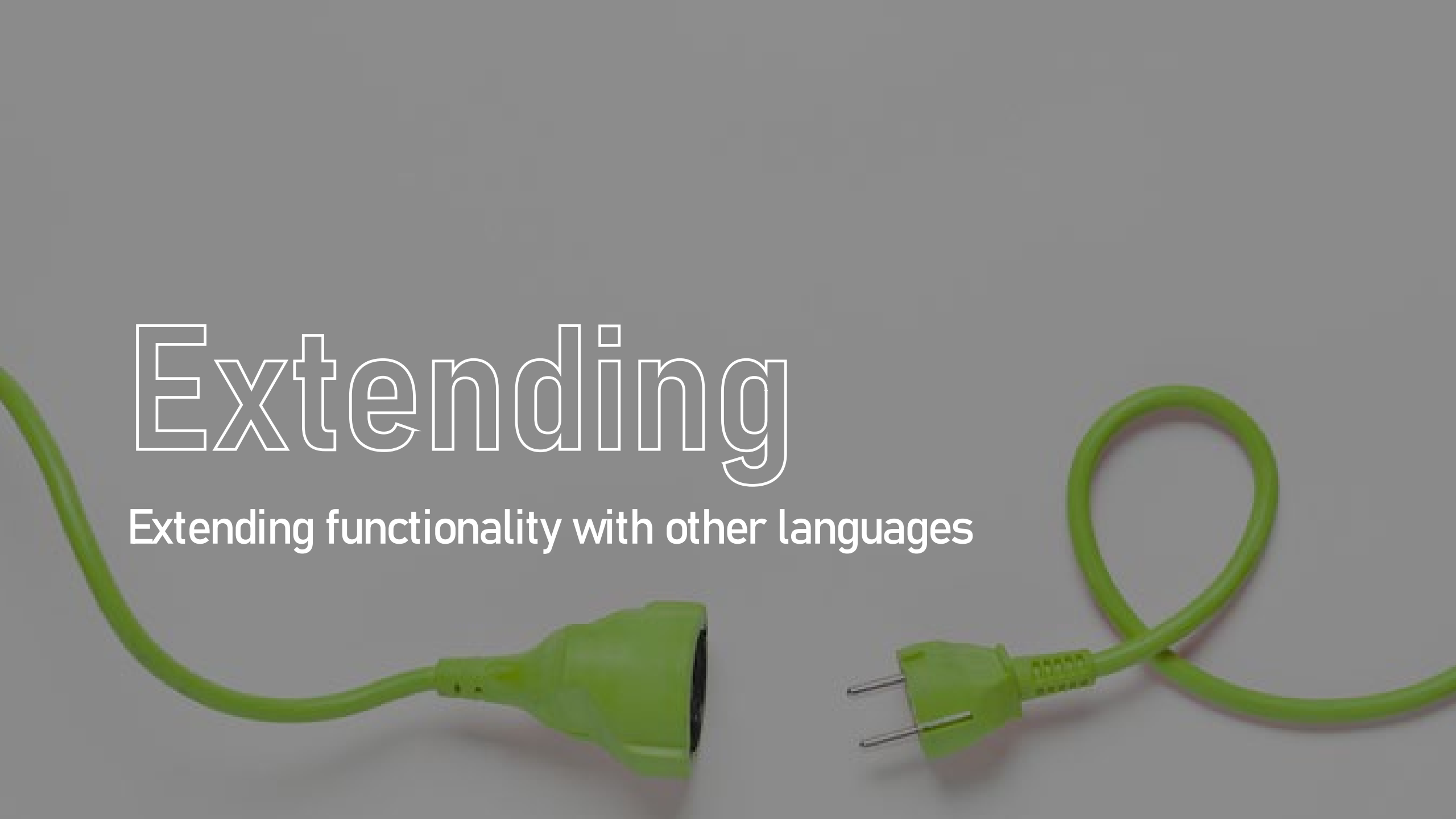
```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

```
$ flask --app hello run
 * Serving Flask app 'hello'
 * Running on http://127.0.0.1:5000 (Press
CTRL+C to quit)
```

Electronic sign implemented in Python with Flask and Raspberry Pi

Extending

Extending functionality with other languages



Extending Python

- External code can be linked into Python using extension modules
- Extension modules in Python are implemented using a C Python API
- Works just like normal Python modules
- Implementing a Python extension module is hard...
- Manually coding this is HARD, especially for arrays

Tools for implementing Python modules

- Simplified Wrapper and Interface Generator – SWIG
  - Can generate wrappers for C/C++ code for many script languages
  - Somewhat cumbersome to use
- PyBind11
  - Operability between C++ and Python
  - No need for interface files. Interface declared in C++ source
  - Existing code have to be extended
- f2py
  - Operability between Fortran and Python
  - Very easy to use.
  - Only Fortran ;)

Fortran as a Python accelerator?

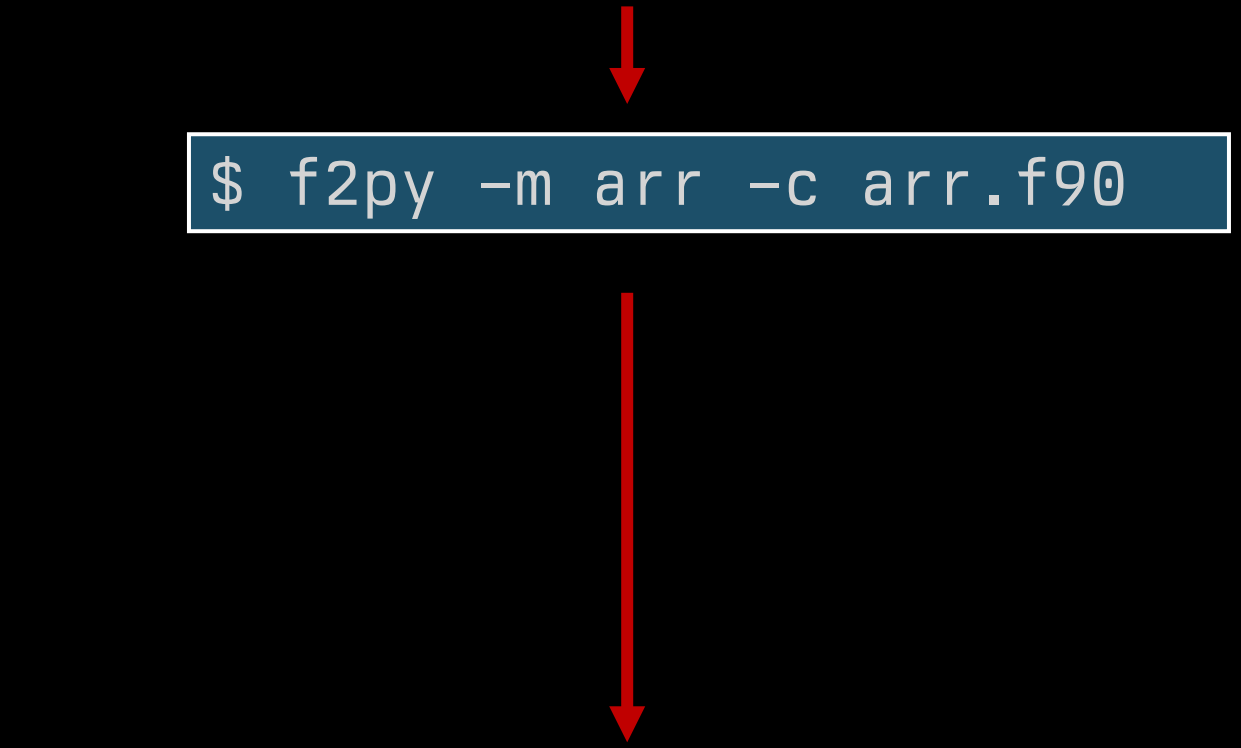
- High performance language
- Compiles to optimised machine code
- Supports parallel computing
  - OpenMP
  - MPI
- Built-in array syntax
- Using f2py make it very easy to create Fortran-based extension modules

Using f2py

```
! A[r,s] * B[s,t] = C[r,t]
subroutine matrix_multiply(A,r,s,B,t,C)
    integer :: r, s, t
    real, intent(in) :: A(r,s)
    real, intent(in) :: B(s,t)
    real, intent(inout) :: C(r,t)

    C = matmul(A,B)
end subroutine matrix_multiply
```

Fortran source code – arr.f90

Python extension module name



Extension module (shared object)

arr2.cpython-37m-x86_64-linux-gnu.so

Using f2py

```
import arr
print(arr.__doc__)

This module 'arr' is auto-generated with f2py (version:1.21.6).
Functions:
  matrix_multiply(a,b,c,r=shape(a,0),s=shape(a,1),t=shape(b,1))
```

Using f2py

```
print(arr.matrix_multiply2.__doc__)
```

```
matrix_multiply(a,b,c,[r,s,t])

Wrapper for ``matrix_multiply``.

Parameters
----------
a : input rank-2 array('f') with bounds (r,s)
b : input rank-2 array('f') with bounds (s,t)
c : in/output rank-2 array('f') with bounds (r,t)

Other Parameters
----------------
r : input int, optional
    Default: shape(a,0)
s : input int, optional
    Default: shape(a,1)
t : input int, optional
    Default: shape(b,1)
```

Using f2py

```
A = np.ones((6,6), 'f', order='F') * 10.0
B = np.ones((6,6), 'f', order='F') * 20.0
C = np.zeros((6,6), 'f', order='F')

print("id of C before multiply =",id(C))

arr.matrix_multiply(A, B, C)

print("id of C after multiply =",id(C))

print(C)
```

order=’F’ ensure array is created with column ordering. Avoids copying

```
id of C before multiply = 139866421235408
id of C after multiply = 139866421235408
[[1200. 1200. 1200. 1200. 1200. 1200.]
 [1200. 1200. 1200. 1200. 1200. 1200.]
 [1200. 1200. 1200. 1200. 1200. 1200.]
 [1200. 1200. 1200. 1200. 1200. 1200.]
 [1200. 1200. 1200. 1200. 1200. 1200.]
 [1200. 1200. 1200. 1200. 1200. 1200.]]
```

C++ and Python a perfect fit?

- C++ is a very powerful language for implementing scientific codes
- Hard to implement user extensible applications in C++
- Wrapping a C++ code as Python extension can provide a flexible layer for users not familiar with C++
- The application can be used in new ways and combined with other Python-modules.
- Enables interactive use of the application!

pybind11

- Header-only C++ library for implementing Python extension modules
- Works on macOS, Windows and Linux
- Add directives to your code to expose it as Python modules and functions
- Supports all Python features including NumPy
- Can be used without modifying existing code

pybind 11 - Function

```cpp
int add(int i, int j) {
    return i + j;
}
```

```cpp
#include <pybind11/pybind11.h>

namespace py = pybind11;

PYBIND11_MODULE(example, m) {
    m.doc() = "pybind11 example plugin";
    m.def("add", &add, "Add numbers");
}
```

```
$ cmake ..
$ make
```

Extension module (shared object)

```
example.cpython-37m-x86_64-linux-gnu.so
```

pybind11 - Classes

```
class ModelParams {
private:
    double m_width;
    double m_height;
    double m_thickness;
public:
    ModelParams(double width, double height, double thickness)
    :m_width{width}, m_height{height}, m_thickness{thickness} {}

    void setWidth(double width) { m_width = width; }
    double width() { return m_width; }

    void setHeight(double height) { m_height = height; }
    double height() { return m_height; }

    void setThickness(double thickness) { m_thickness = thickness; }
    double thickness() { return m_thickness; }

    void print() { std::cout << m_width << ", " << m_height << ", "
<< m_thickness << "\n"; }
};
```

pybind11 – Defining the class

```cpp
#include <pybind11/pybind11.h>

namespace py = pybind11;

PYBIND11_MODULE(example, m) {
    m.doc() = "pybind11 example plugin"; // optional module
docstring

    m.def("add", &add, "A function that adds two numbers");

    py::class_<ModelParams>(m, "ModelParams")
        .def(py::init<double, double, double>())
        .def("setWidth", &ModelParams::setWidth)
        .def("width", &ModelParams::width)
        .def("setHeight", &ModelParams::setHeight)
        .def("height", &ModelParams::height)
        .def("setThickness", &ModelParams::setThickness)
        .def("thickness", &ModelParams::thickness)
        .def("print", &ModelParams::print);
}
```

pybind11 – using the new module

```
>>> import example
>>> model_params = example.ModelParams(0.1, 0.2, 0.3)
>>> model_params.print()
0.1, 0.2, 0.3
>>> model_params.setWidth(0.5)
>>> print(model_params.width())
0.5
>>> model_params.print()
0.5, 0.2, 0.3
```

Embedding

Using Python as an application language

Emedding Python

- Many large graphical applications us Python as a language for extending functionality without recompiling
- Enables easy creation of plugins without recompiling the application
- Enable users to expand functionality
- Enable a user interface application to be scripted
- Pybind11 can be used for embedding as well.

Embedding with pybind11 - CMake

```
cmake_minimum_required(VERSION 3.4)
project(example)

find_package(pybind11 REQUIRED)  # or
`add_subdirectory(pybind11)`

add_executable(example main.cpp)
target_link_libraries(example PRIVATE pybind11::embed)
```

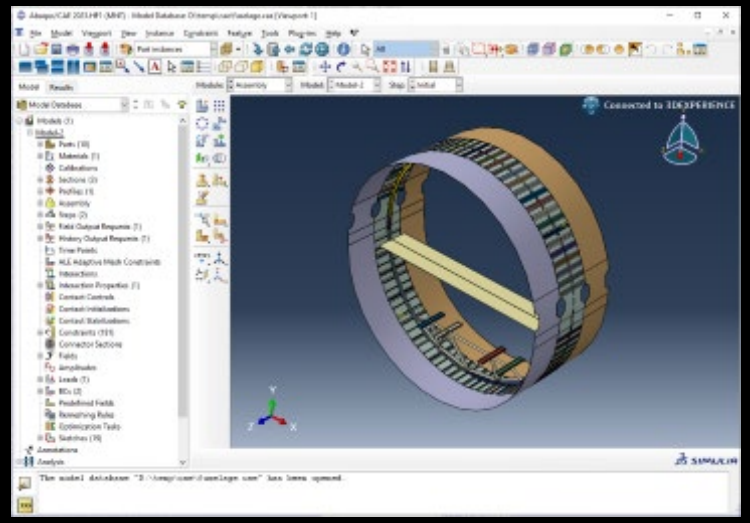
Embedding with pybind11 - Initialise

```cpp
#include <pybind11/embed.h> // everything needed for embedding
namespace py = pybind11;

int main() {
    py::scoped_interpreter guard{}; // start the interpreter

    py::print("Hello, World!"); // use the Python API
}
```

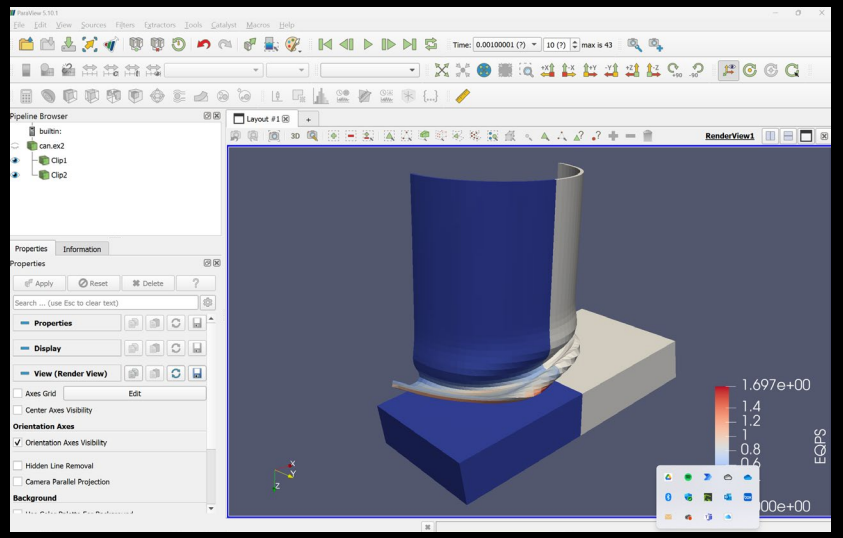
Embedding – expose functionality

```cpp
#include <pybind11/embed.h>
namespace py = pybind11;

PYBIND11_EMBEDDED_MODULE(fast_calc, m) {
    m.def("add", [](int i, int j) {
        return i + j;
    });
}

int main() {
    py::scoped_interpreter guard{};

    auto fast_calc = py::module_::import("fast_calc");
    auto result = fast_calc.attr("add")(1, 2).cast<int>();
    assert(result == 3);
}
```

Examples of embedded Python



ABAQUS/CAE



ParaView

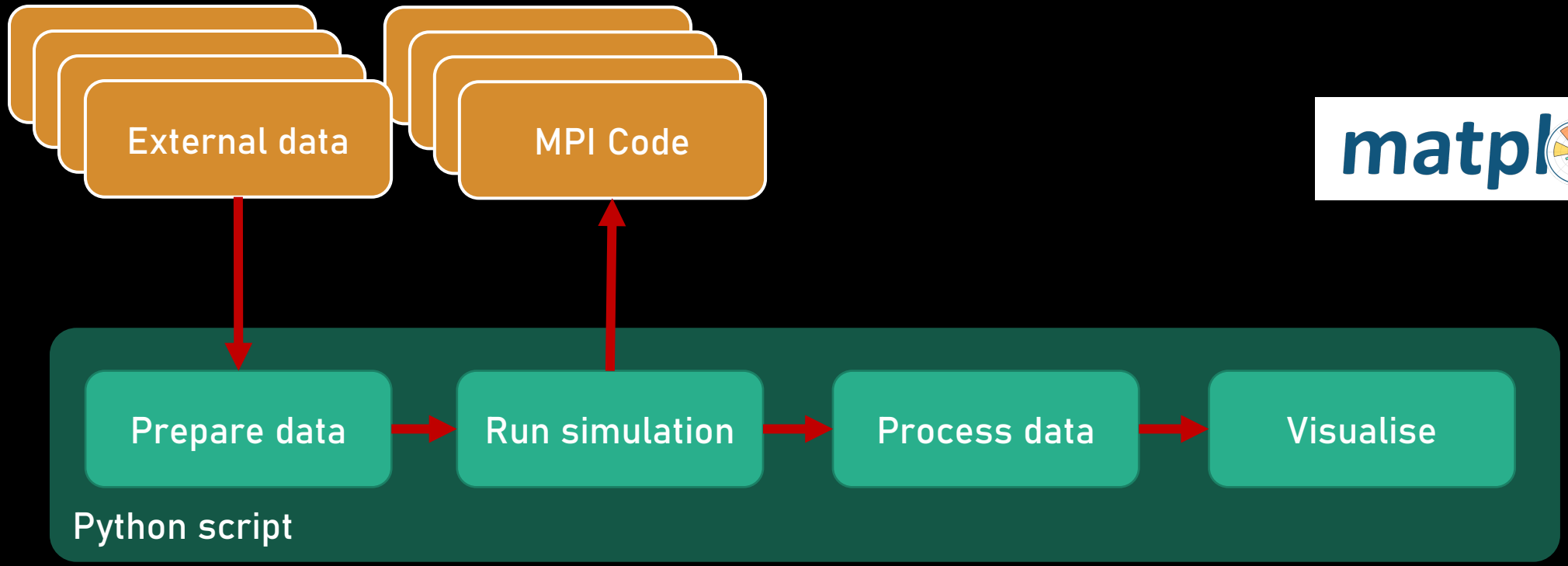
FreeCAD


Workflows
Using Python for reproducible scientific workflows

Python for scientific workflows

- Multiple software tools are required in most scientific work
- Important to document how to reproduce results from data analysis and simulations.
- Scripts can be used, but have limited functionality
- Python can be used to automate the entire workflow
- Glue language
- Important to use virtual environments to document which versions of Python and modules used in the workflow

Example workflow



matplotlib

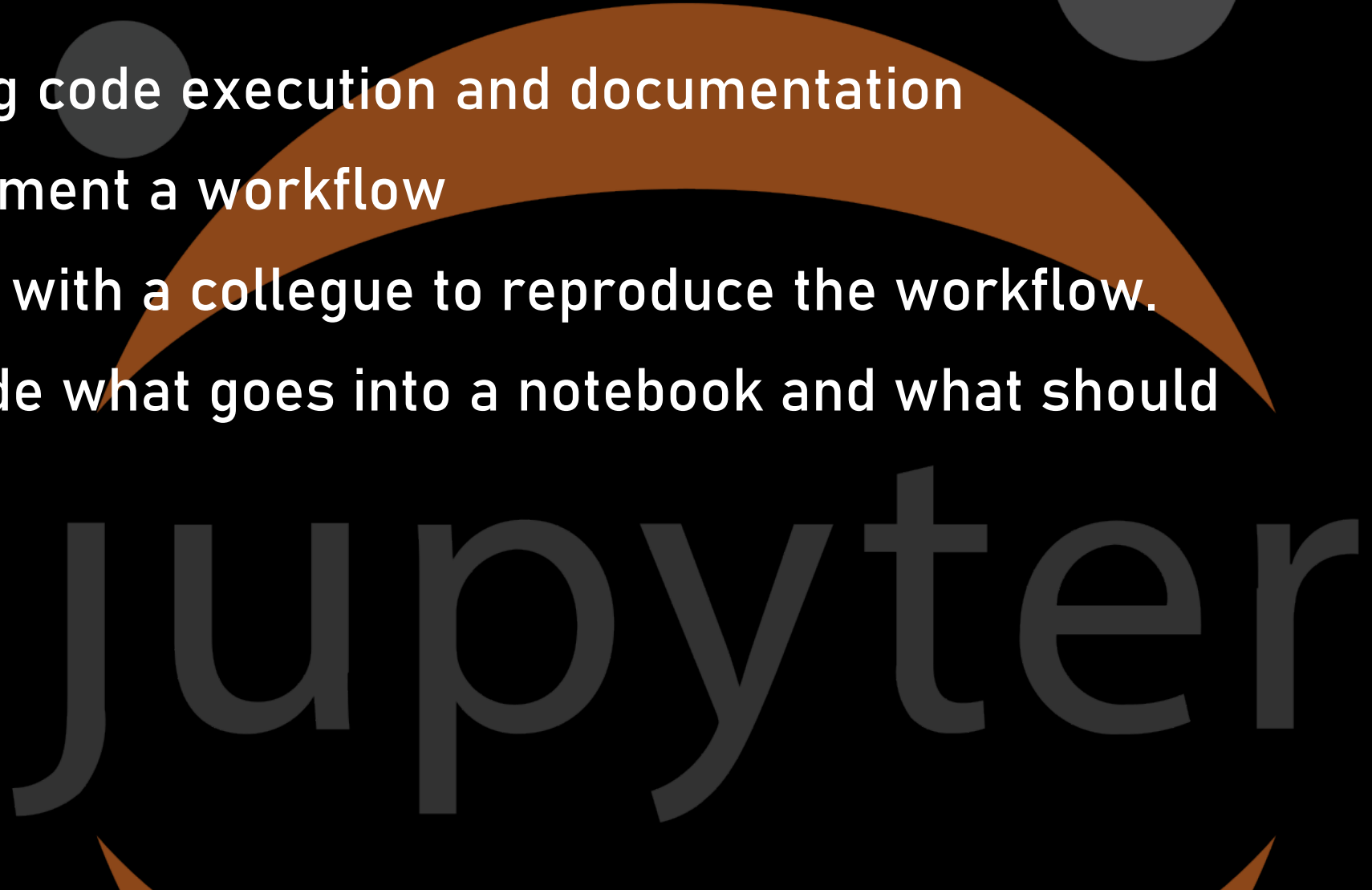
pandas

HDF

ParaView

Jupyter Notebooks

- Tool for combining code execution and documentation
- Good way to document a workflow
- Share a notebook with a collegue to reproduce the workflow.
- Important to decide what goes into a notebook and what should go into modules.



Computational mechanics

Python and computational mechanics

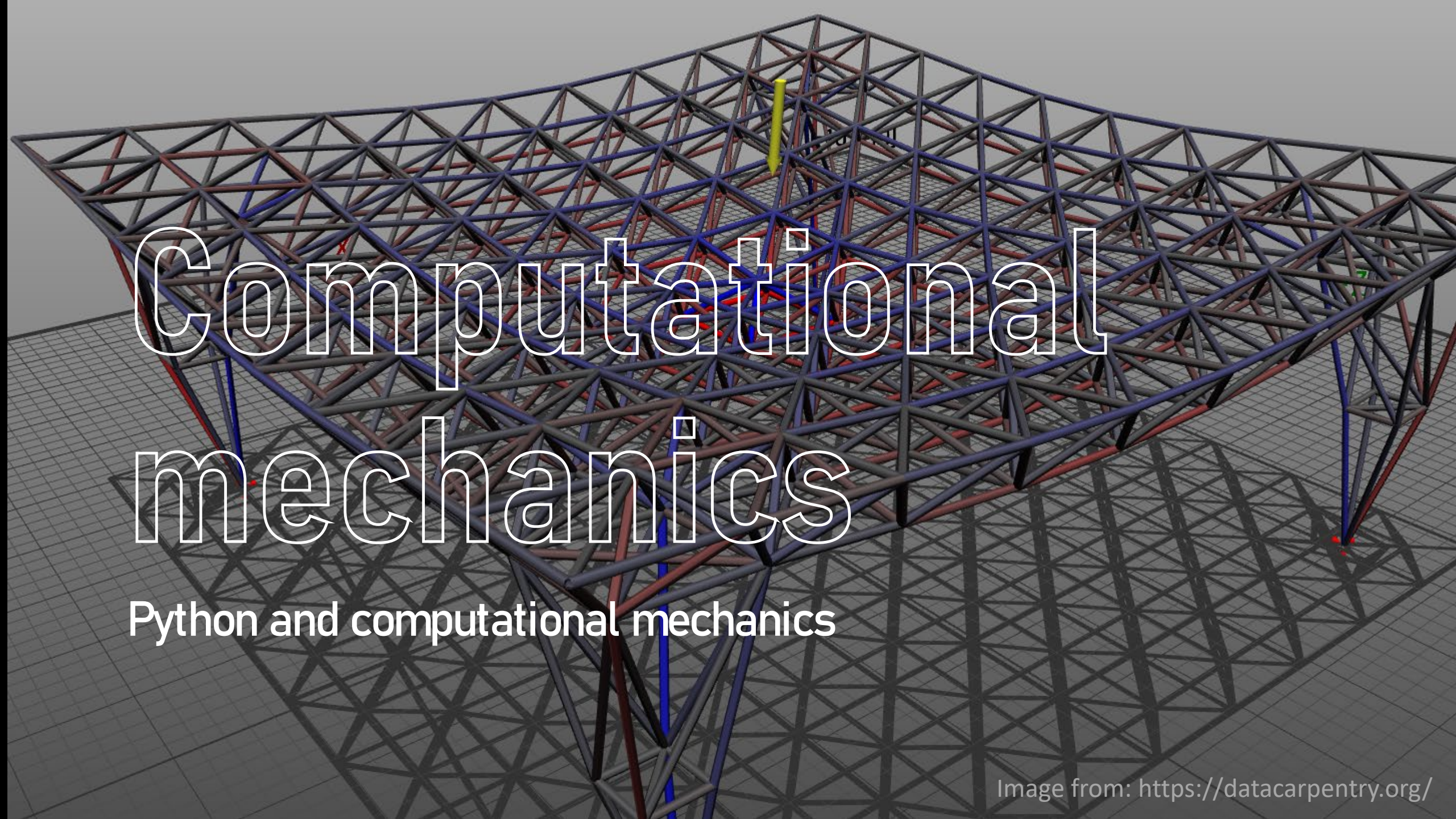
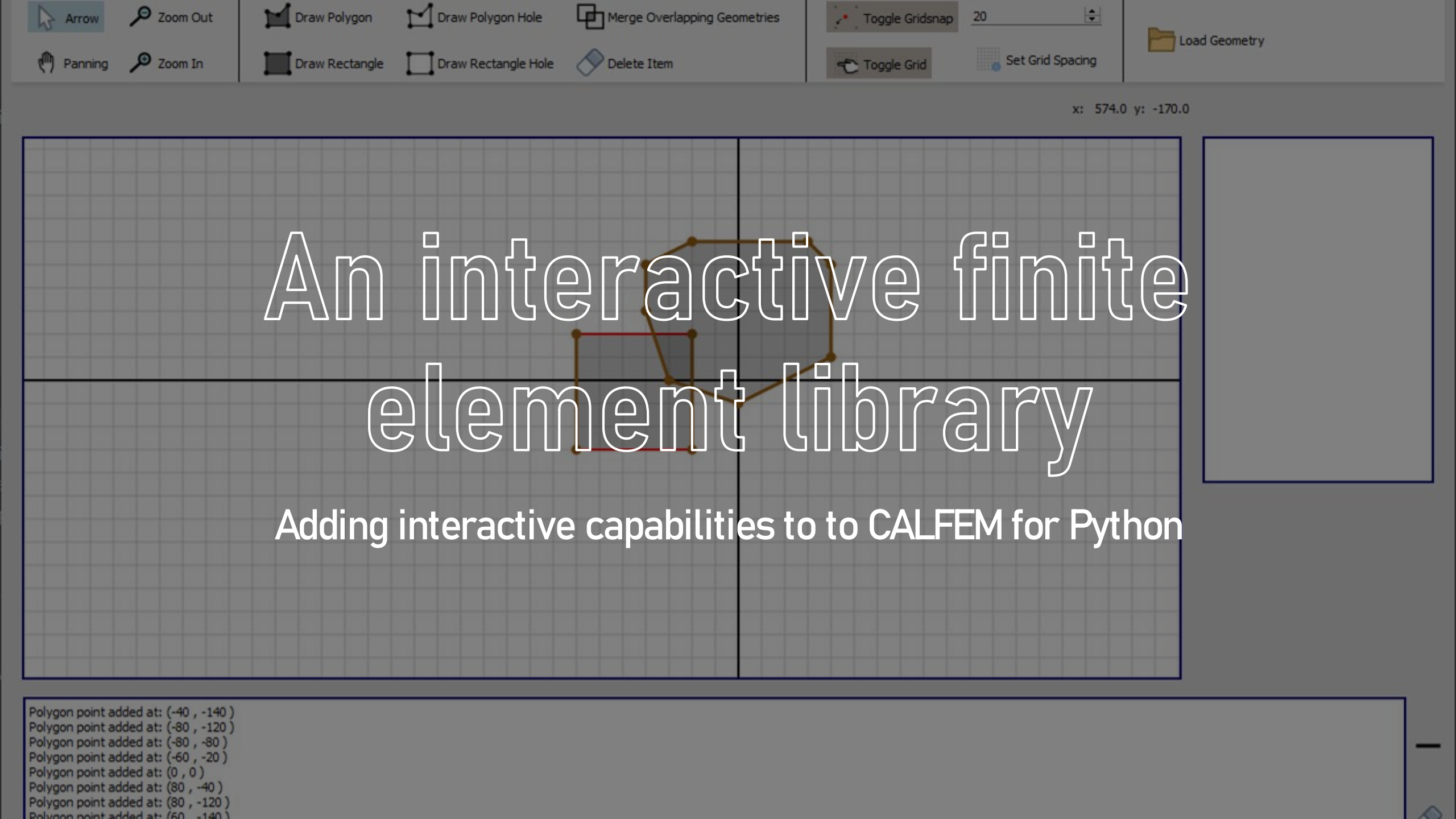


Image from: https://datacarpentry.org/


Arrow
Zoom Out
Draw Polygon
Draw Polygon Hole
Merge Overlapping Geometries
Toggle Gridsnap
20
Load Geometry
Panning
Zoom In
Draw Rectangle
Draw Rectangle Hole
Delete Item
Toggle Grid
Set Grid Spacing
x: 574.0 y: -170.0
An interactive finite element library
Adding interactive capabilities to to CALFEM for Python
Polygon point added at: (-40 , -140)
Polygon point added at: (-80 , -120)
Polygon point added at: (-80 , -80)
Polygon point added at: (-60 , -20)
Polygon point added at: (0 , 0)
Polygon point added at: (80 , -40)
Polygon point added at: (80 , -120)

Adding interaction in Python libraries

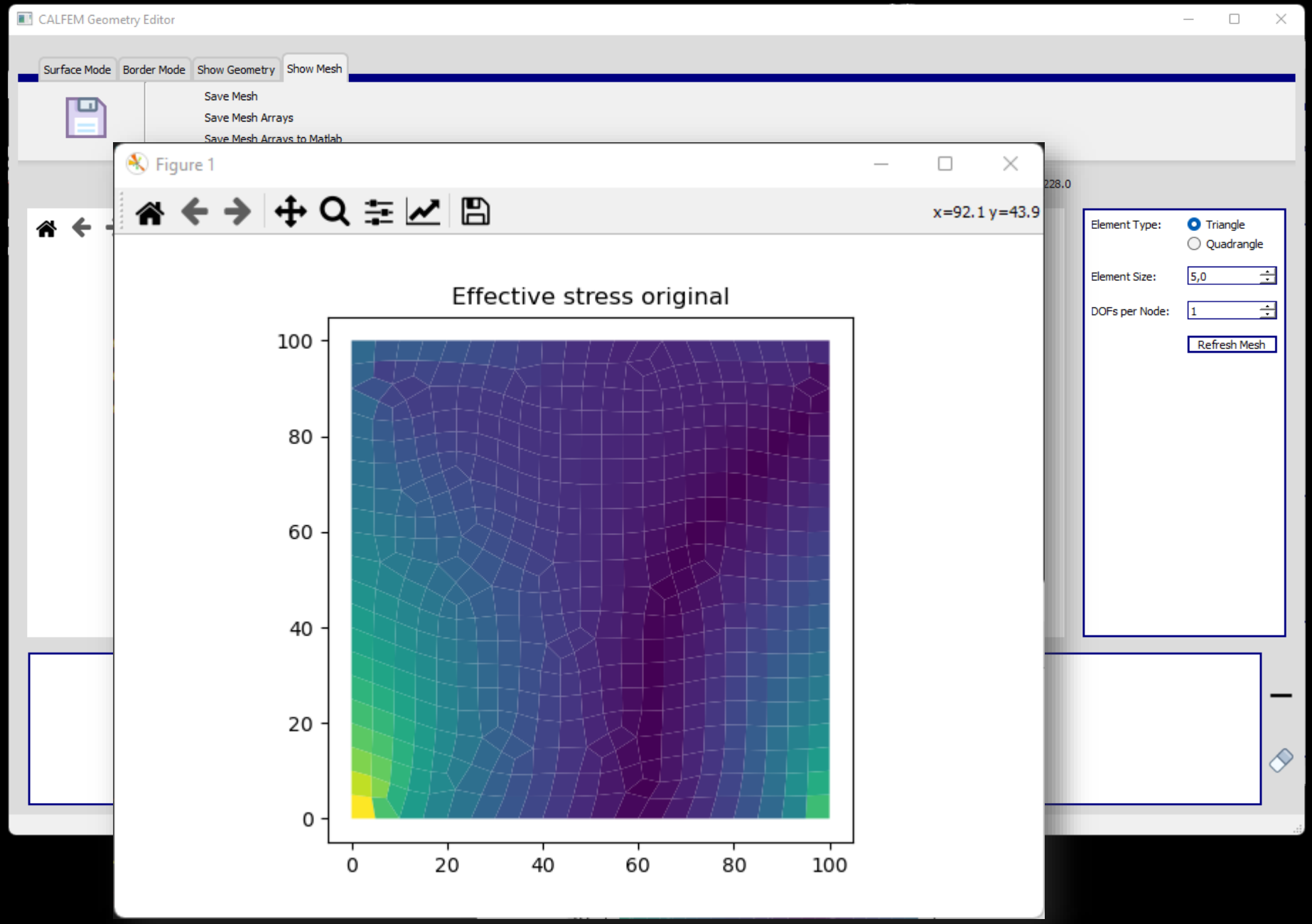
- When learning finite element programming concepts, it can be difficult for students to experiment with the code.
- CALFEM for Python is a Python package used in teaching the finite element method.
- To enable the experimentation with changes in geometry functions for interactively editing graphics was added.
- This provides a way of quickly changing models without many code changes.

An example

```
# --- Creating a square geometry with two markers

g = cfg.Geometry()

g.point([0.0, 0.0])    # point 0
g.point([100.0, 0.0])  # point 1
g.point([100, 100])    # point 2
g.point([0, 100])      # point 3

g.spline([0, 1])       # line 0
g.spline([1, 2])       # line 1
g.spline([2, 3])       # line 2
g.spline([3, 0])       # line 3

g.surface([0, 1, 2, 3])  # Connect lines to form
surface
g.setCurveMarker(0, 10)
g.setCurveMarker(2, 20)

# --- Open the geometry to allow changes in the
CALFEM Geometry Editor

new_geometry, marker_dict = cfe.edit_geometry(g)
```

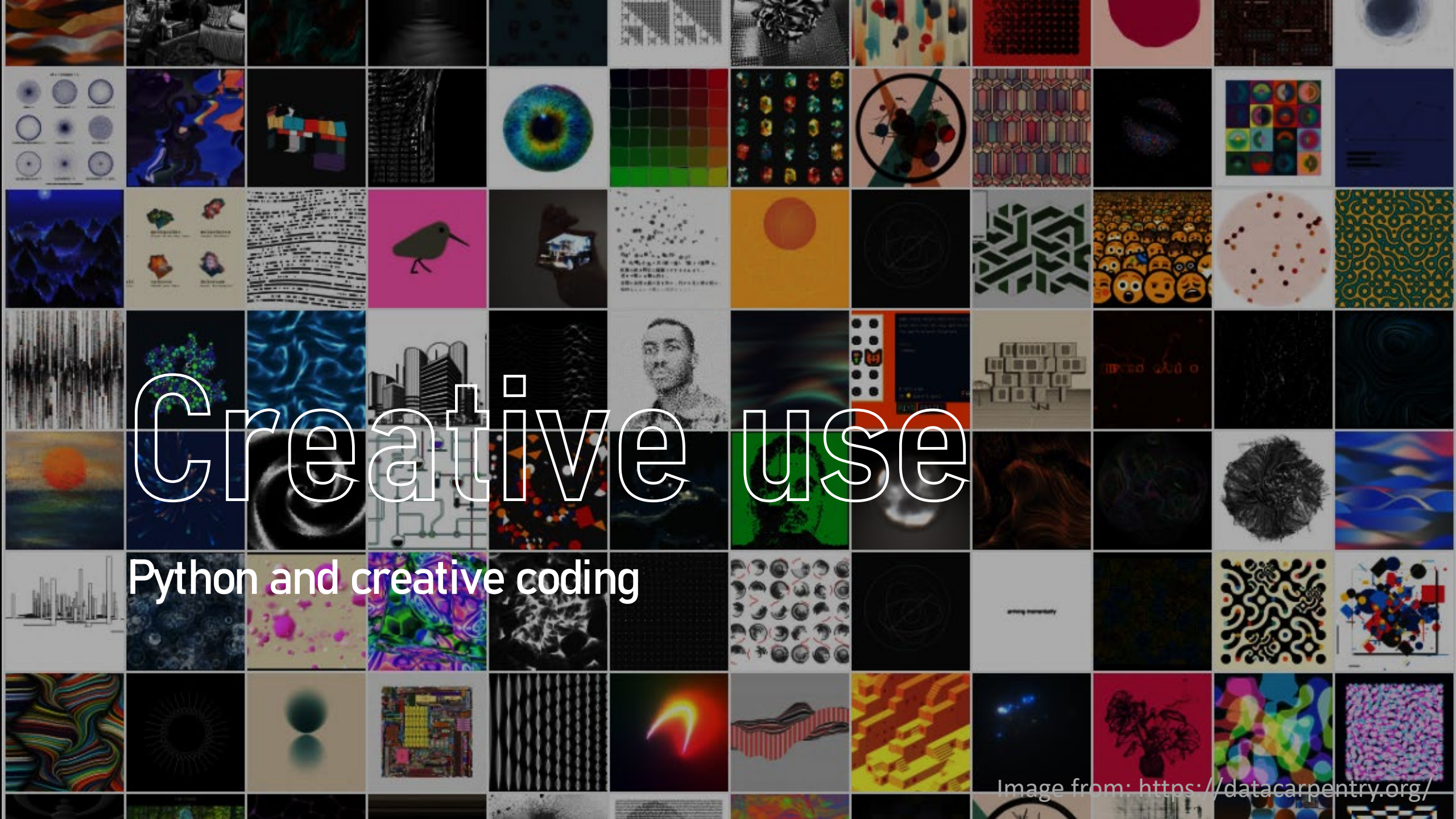
This command brings up a user interface for modifying the geometry.

CALFEM Geometry Editor



Creative use

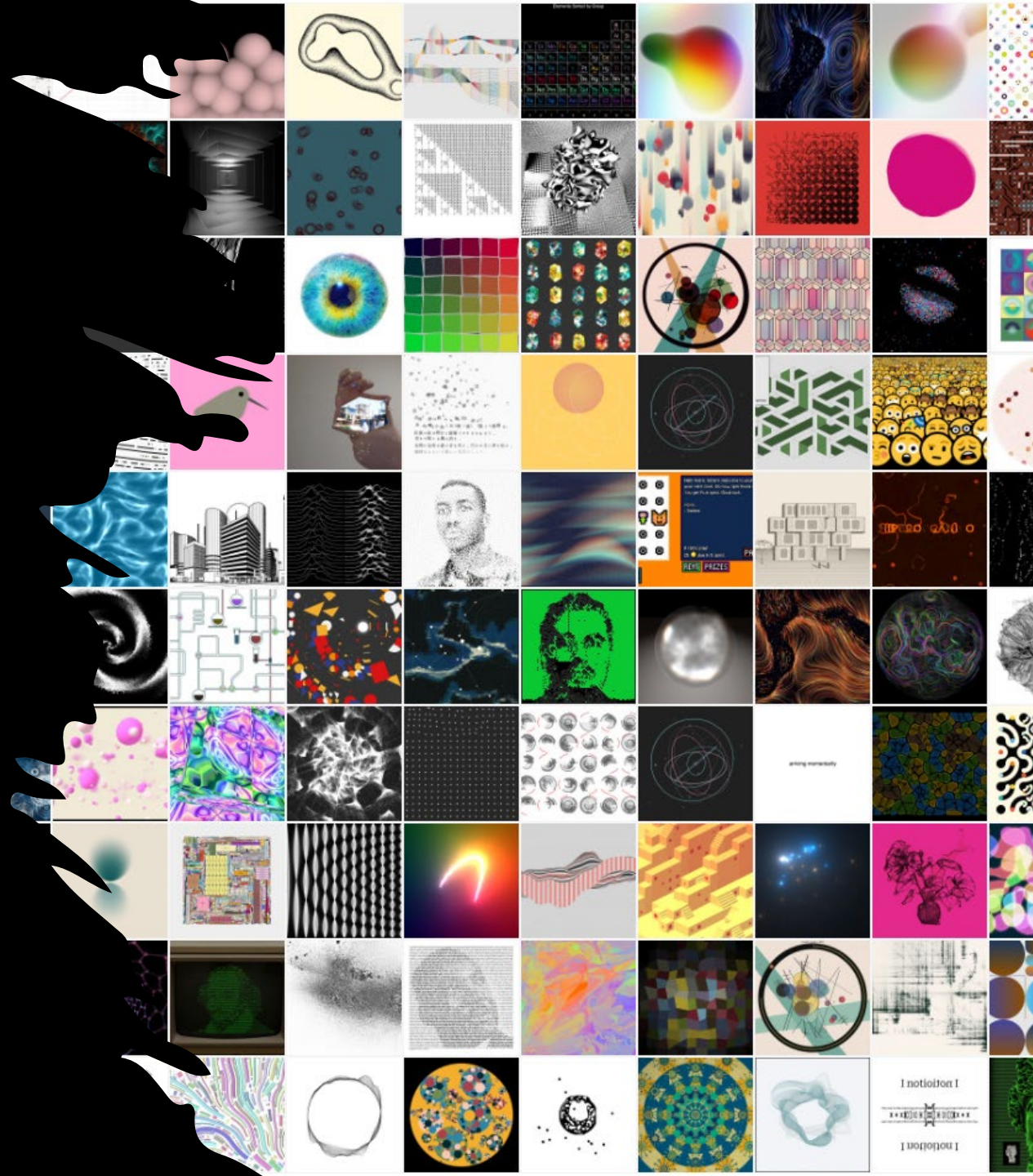
Python and creative coding

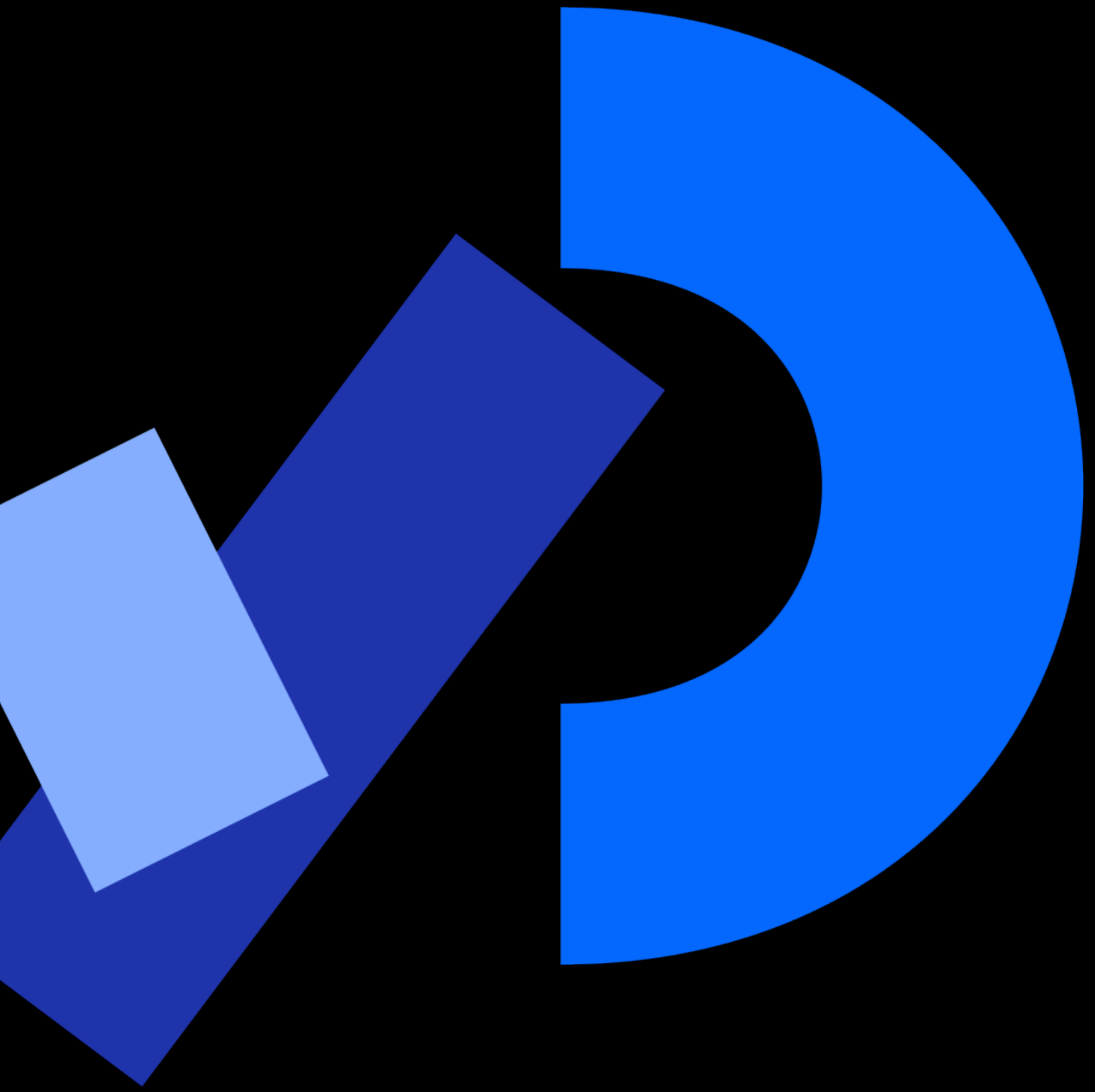


Creative coding

“Creative coding is a type of computer programming in which the goal is to create something expressive instead of something functional. It is used to create live visuals and for VJing, as well as creating visual art and design, entertainment (e.g. video games), art installations, projections and projection mapping, sound art, advertising, product prototypes, and much more.”

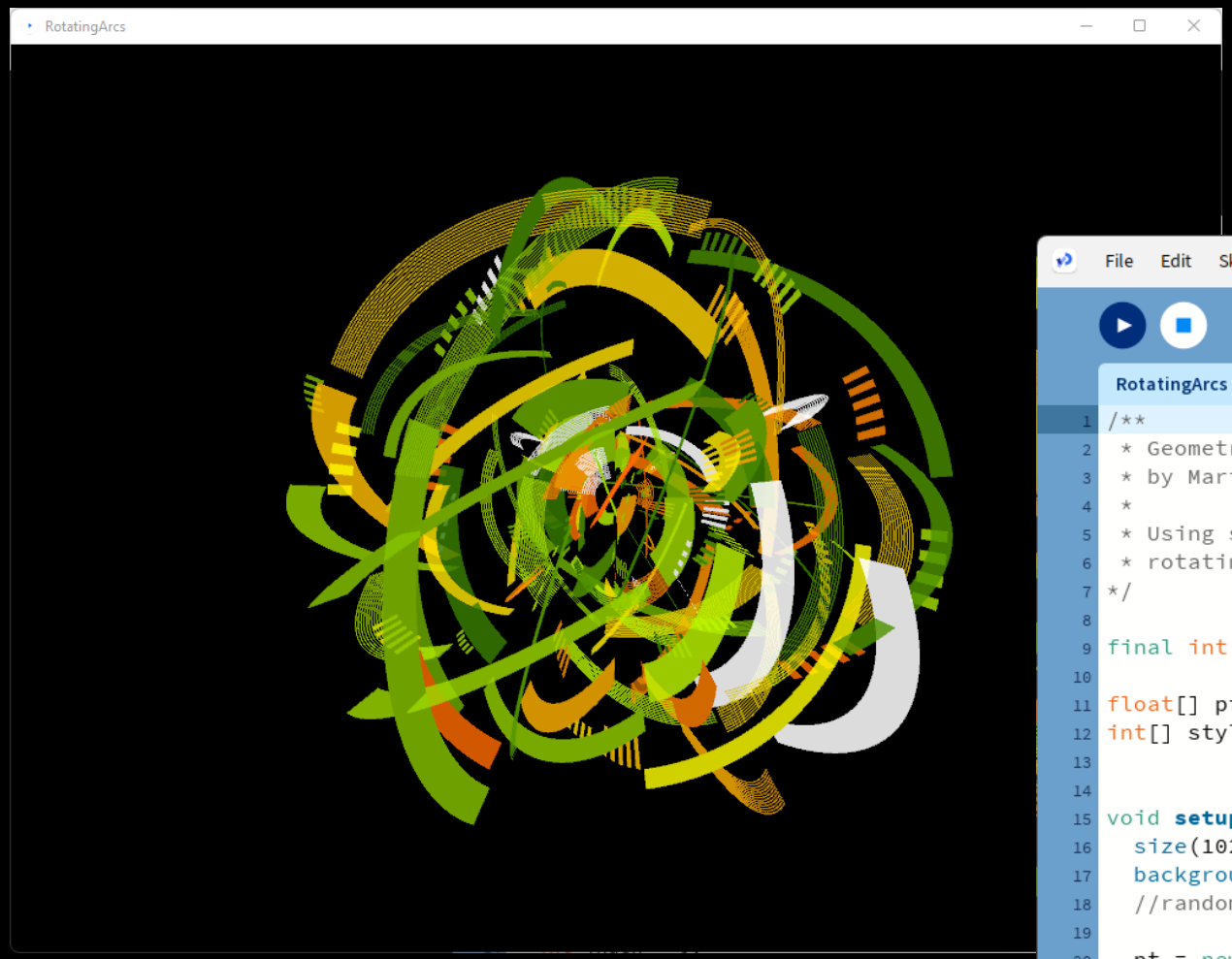
Wikipedia





Processing

- Processing is an interactive environment for Creative coding.
- Simplified Java-based language
- Large API for creating graphical applications
  - 2D/3D/Audio support
- Currently only supports Python 2 … No NumPy



File Edit Sketch Debug Tools Help

RotatingArcs | Processing 4.0.1

Java

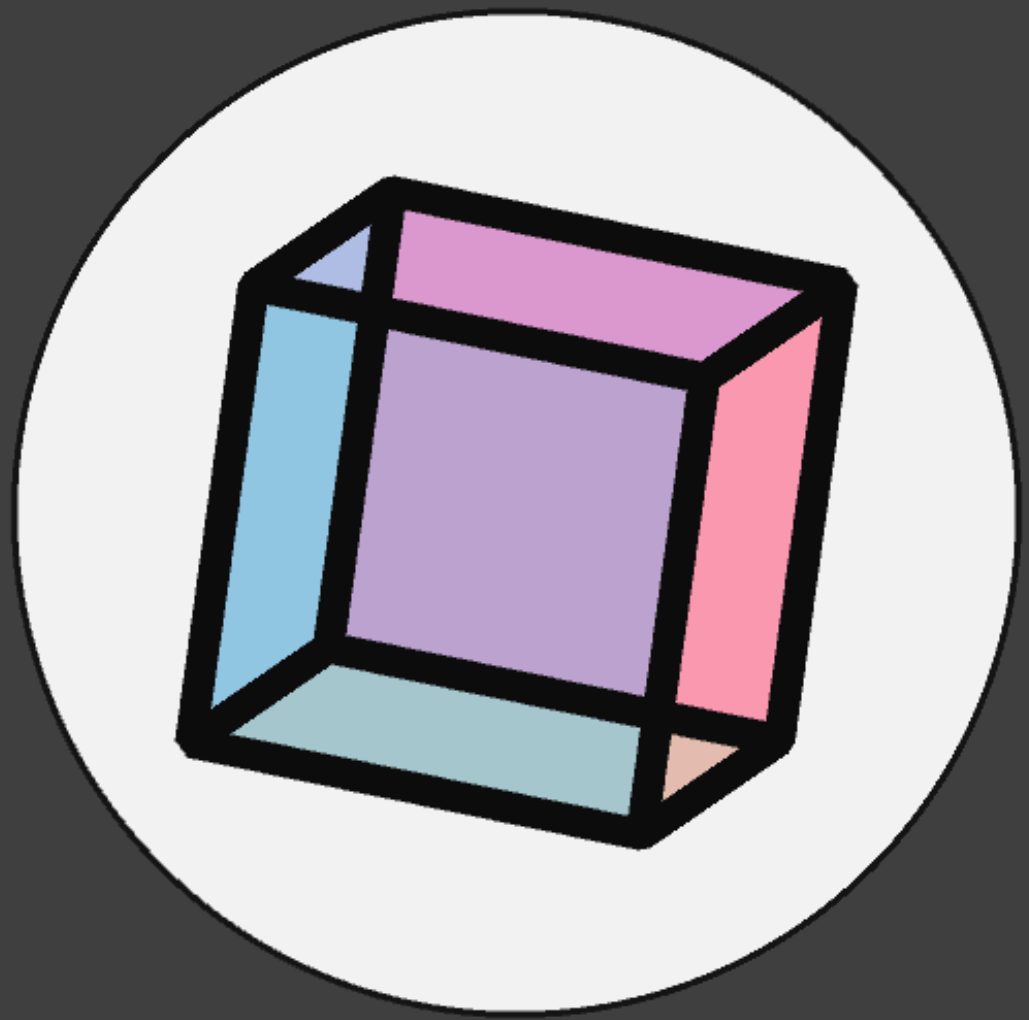
RotatingArcs

```
/**
 * Geometry
 * by Marius Watz.
 *
 * Using sin/cos, blends colors, and draws a series of
 * rotating arcs on the screen.
*/

final int COUNT = 150;

float[] pt;
int[] style;


void setup() {
  size(1024, 768, P3D);
  background(255);
  //randomSeed(100);  // use this to get the same result each time

  pt = new float[6 * COUNT]; // rotx, roty, deg, rad, w, speed
  style = new int[2 * COUNT]; // color, render style

  // Set up arc shapes
  int index = 0;
  for (int i = 0; i < COUNT; i++) {
```

Console Errors Updates 1

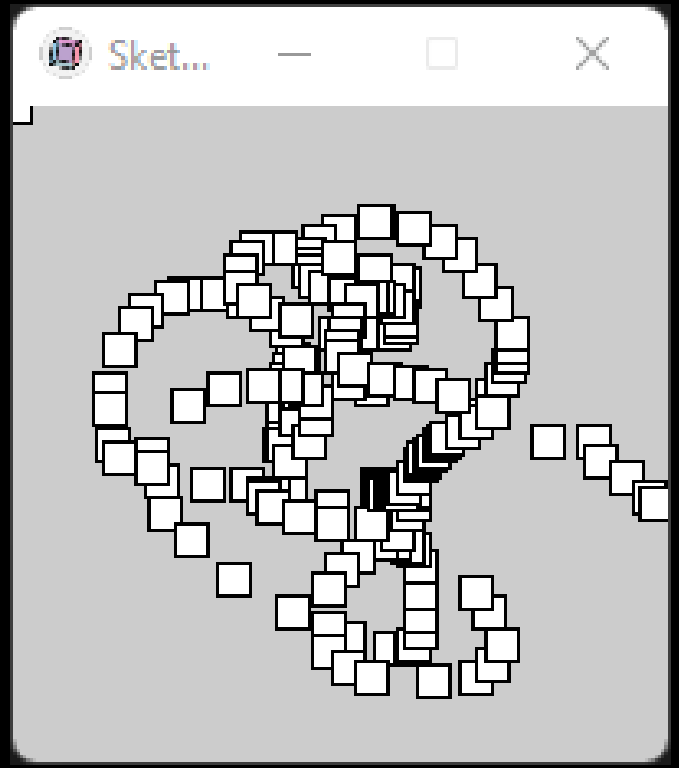
Creative coding in Modern Python using py5

- py5 is a project making the Processing API available for in a modern Python 3.8 environment.
- Uses JPype to provide the functionality to the CPython interpreter



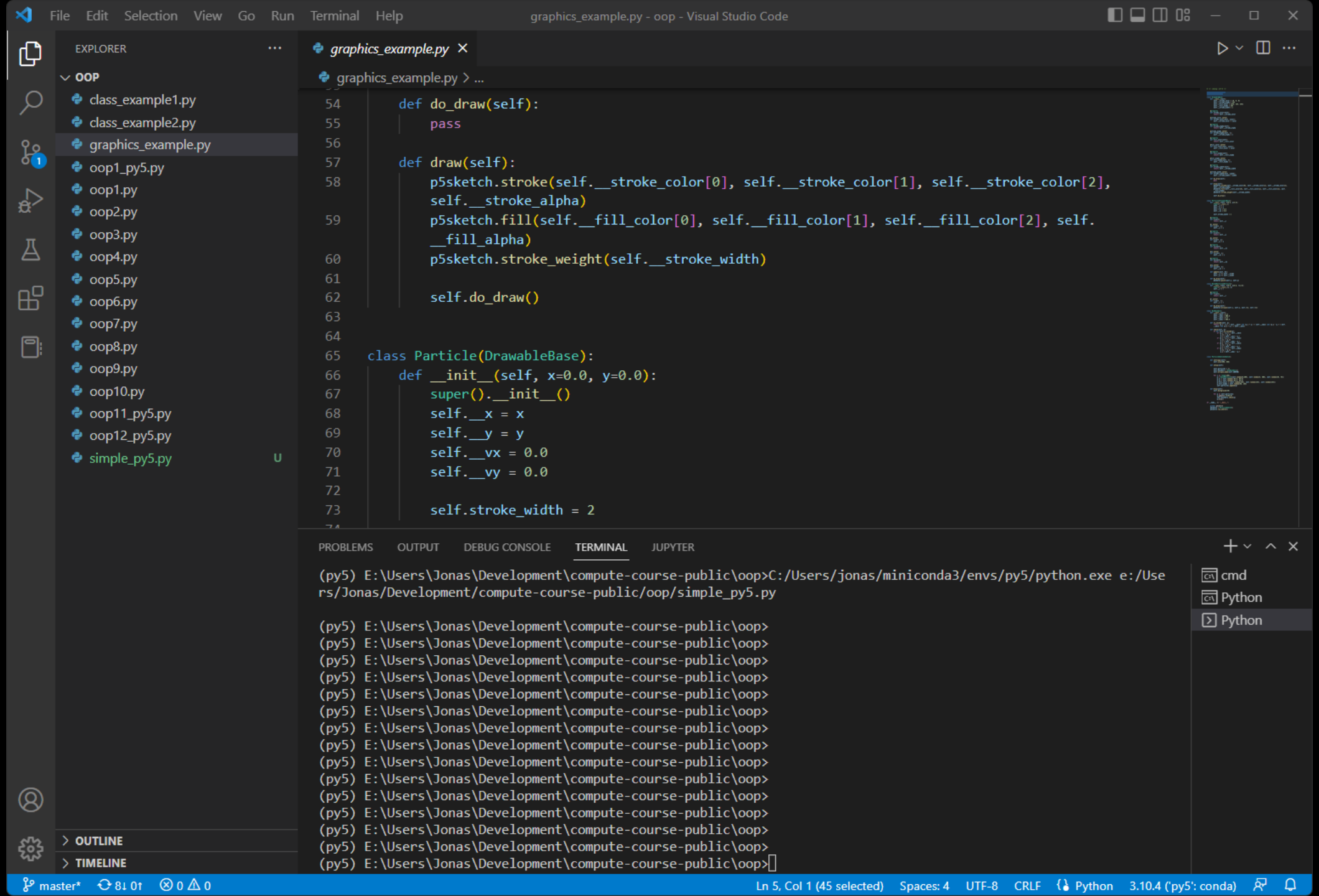
py5 for Python

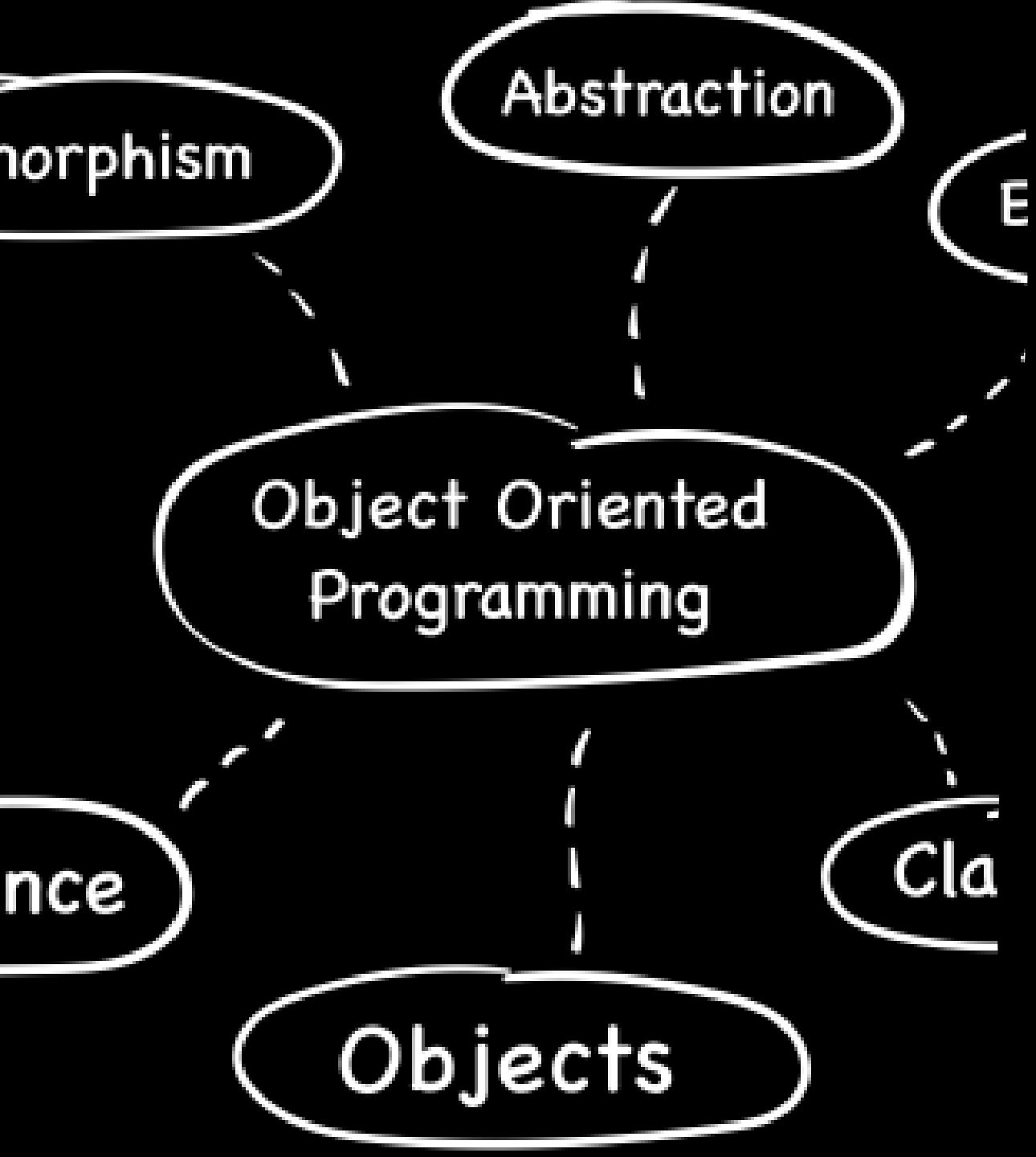
```python
import py5

def setup():
    py5.size(200, 200)
    py5.rect_mode(py5.CENTER)

def draw():
    py5.rect(py5.mouse_x, py5.mouse_y, 10, 10)

py5.run_sketch()
```



py5 live coding environment

Visual Studio code and a custom py5 environment


File Edit Selection View Go Run Terminal Help
graphics_example.py - oop - Visual Studio Code
EXPLORER
OOP
class_example1.py
class_example2.py
graphics_example.py
oop1_py5.py
oop1.py
oop2.py
oop3.py
oop4.py
oop5.py
oop6.py
oop7.py
oop8.py
oop9.py
oop10.py
oop11_py5.py
oop12_py5.py
simple_py5.py U
graphics_example.py
graphics_example.py > ...
54 def do_draw(self):
55 pass
56
57 def draw(self):
58 p5sketch.stroke(self.__stroke_color[0], self.__stroke_color[1], self.__stroke_color[2], self.__stroke_alpha)
59 p5sketch.fill(self.__fill_color[0], self.__fill_color[1], self.__fill_color[2], self.__fill_alpha)
60 p5sketch.stroke_weight(self.__stroke_width)
61
62 self.do_draw()
63
64
65 class Particle(DrawableBase):
66 def __init__(self, x=0.0, y=0.0):
67 super().__init__()
68 self.__x = x
69 self.__y = y
70 self.__vx = 0.0
71 self.__vy = 0.0
72
73 self.stroke_width = 2
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
(py5) E:\Users\Jonas\Development\compute-course-public\oop>C:/Users/jonas/miniconda3/envs/py5/python.exe e:/Users/Jonas/Development/compute-course-public/oop/simple_py5.py
(py5) E:\Users\Jonas\Development\compute-course-public\oop>
cmd
Python
Python
OUTLINE
TIMELINE
master* 8↓ 0↑ ⊗ 0 ⚠ 0
Ln 5, Col 1 (45 selected) Spaces: 4 UTF-8 CRLF Python 3.10.4 ('py5': conda)



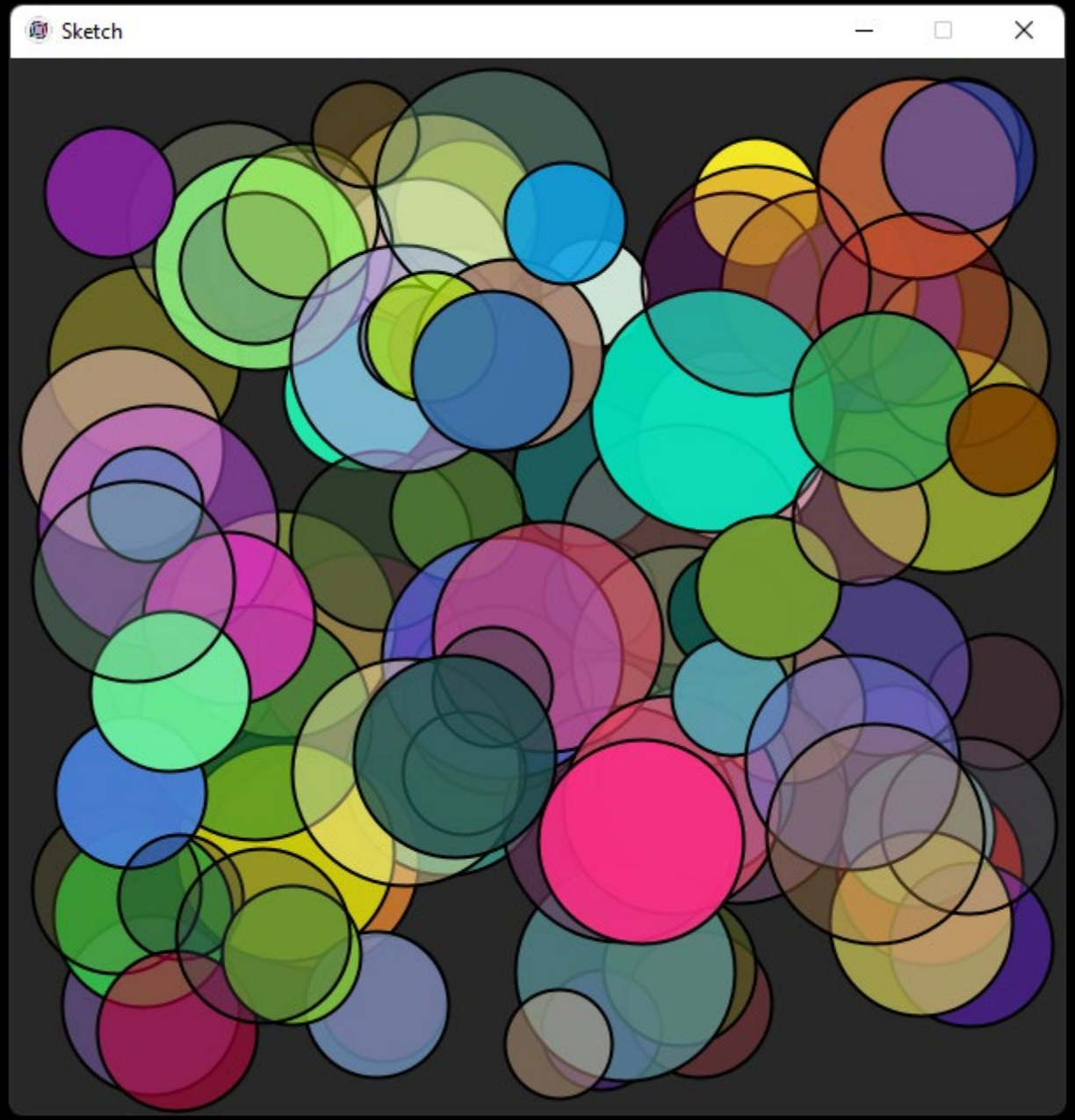
Updated OOP course material

- Instead of using abstract Point, Circle, Box, and Line examples, we create classes that can be drawn in py5.
- Example code a bit more complicated, but 10x more engaging and fun ☺
- A more prominent example with simple particles is implemented.
- Live coding with Visual Studio Code

```
class Point:
    def __init__(self, x=0.0,
        self.__x = x
        self.__y = y

    def set(self, x, y):
        self.__x = x
        self.__y = y

    @property
    def x(self):
        return self.__x

    @x.setter
    def x(self, x):
        self.__x = x
```

Explaning OOP graphically

- Object-oriented programming is hard to teach
- Examples often are boring and text-based.
- Is there a way of making this more engaging and understandable?


Sketch

Conclusions

- Python is a very versatile language
- Python can compliment and strengthen other languages
- Easy to integrate and extend
- By being interactive by design, it is well suited for use in teaching
- Can be used as a glue in scientific workflows
- It is FUN!

Contact info

Email

jonas.lindemann@lunarc.lu.se

Twitter

@jonas_lindemann

@LUNARC_LU

Github

https://github.com/jonaslindemann

YouTube

https://www.youtube.com/c/JonasLindemann


Thank you!