MAD76 Academy: Python

Frank Tränkle*
Hochschule Heilbronn, Germany

August 25, 2025

*frank.traenkle@hs-heilbronn.de

Contents

1 Agenda

- What is Python? Why Python? (see Section 2)
- Math with Python (see Section 3)
- Procedural programming with Python (see Section 4)
- Object-oriented programming with Python (see Section 5)

Teaching Objectives

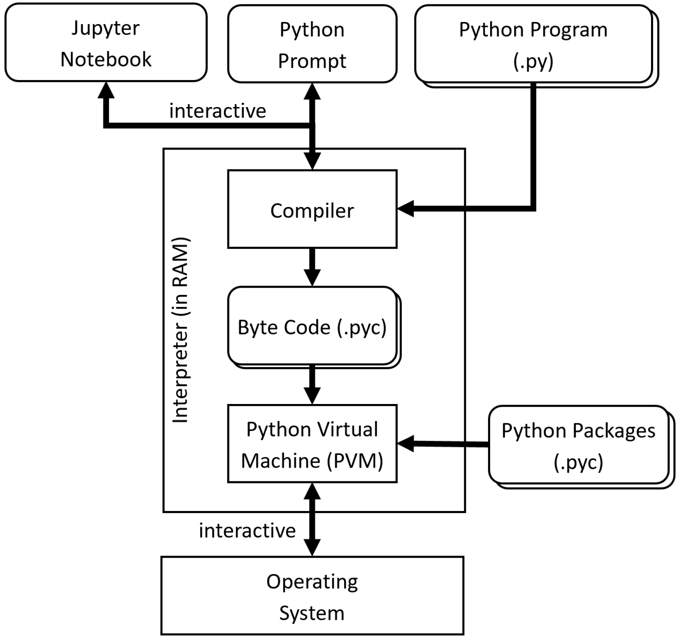
- Understand Python as an interpreted programming language
- Learn Python syntax and semantics
- Learn how to use Python for mathematics at school
- Learn how to use Jupyter Notebooks for Python
- Learn procedural programming with Python
- Learn object-oriented programming with Python
- Learn how to use VS Code as an IDE for Python

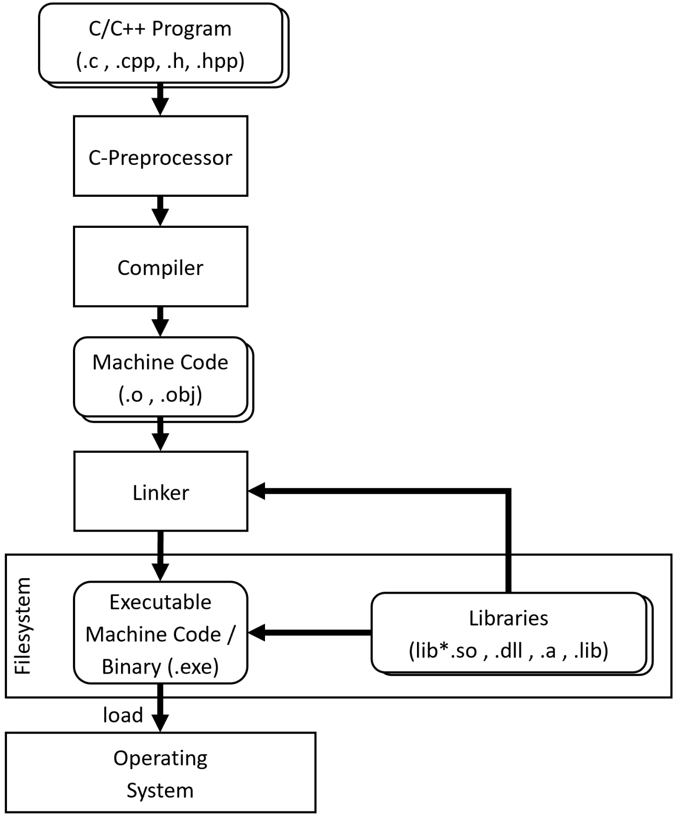
2 What is Python

- Python is a high-level programming language [1]
  - for procedural and object-oriented programming
- Python is an *interpreter*
  - Python prompts for input
  - User enters Python command
  - Python replies to command
- Python is the most popular programming language in the world (`https://www.tiobe.com/tiobe-index/`)
- Python is THE programming language in Artificial Intelligence (AI)
- Python is strong for scripting as a high-level alternative to Bash
- Python is one of the programming languages of ROS2 and MAD76
- Python is straight-forward to use for beginners
  - which is not the case for C++ or Rust
- Python is strong in numerics (mathematics on computers)

  - not as strong as MATLAB, though, but free to use
- But:
  - Python is comparably slow and resource-intensive [2]
  - This leads to high energy consumption $\rightsquigarrow$ high CO_2 emissions of AI Foundation Models training
  - Python is not reliable and non-realtime $\rightsquigarrow$ not suitable for safety-critical applications (in cars, aeroplanes)

Interpreter versus Compiler





3 Math with Python

Agenda

- Use Jupyter with Python as a calculator (see Section 3.1)
- Solve quadratic equations (see Section 3.2)
- Solve linear equation systems (see Section 3.3)
- Function plotting (see Section 3.4)
- Python data types (see Section 3.5)

3.1 Python as Calculator

- Jupyter is an interactive user interface
- Jupyter Notebooks are *living documents*
- Jupyter Notebooks contain both
  - Python code
  - and documentation (in Markdown)
  - including math formulas (in LaTeX)
- Jupyter mainly supports Python, but also other programming languages
- Jupyter is available both locally in VS Code and online (see `https://jupyter.org/`)

1. Start VS Code and create a new Jupyter notebook `math.ipynb`

```
cd
mkdir -p src/pythonmath
cd src/pythonmath
touch math.ipynb
code math.ipynb
```

2. Create new cell by hitting `Alt+Enter` button

3. Enter the following code in the cell:

```
1 + 2
```

   - Note that `1 + 2` is a valid Python expression

4. Hit `Ctrl+Enter` to execute the cell
5. Select `Python Environments ...` and then `Python 3.x.x /usr/bin/python` (or similar)
6. Jupyter then displays the result of the calculation
7. Try some more calculations by hitting `Alt+Enter` and `Ctrl+Enter`

```
3 * 4 / 2 * 3
3 * 4 / (2 * 3)
2 ** -1
2 ** -3 * 8
```

8. Import Python package `numpy` for numerical calculations

```python
import numpy as np
```

9. Use `numpy` to calculate the square root of a number

```
np.sqrt(2)
```

10. and for trigonometry

```
np.pi
np.pi * 0.5
np.pi * 5e-1
np.sin(np.pi * .5)
np.cos(np.pi)
np.cos(np.deg2rad(180))
np.sin(101)**2 + np.cos(101)**2
np.tan(np.pi * .5)
```

11. Use variables

```
x = 2
y = 3
z = x + y
```

3.2 Solve Quadratic Equations

- Solve quadratic equations of the form $ax^2 + bx + c = 0$
- Use the quadratic formula $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

1. Create a new Markdown cell for documentation by hitting `+ Markdown`

```
# Quadratic Equations
* equation: $a x^2 + b x + c = 0$
* solutions: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 a c}}{2 a}$
```

2. Create a new Code cell

```
a = 4
b = -20
c = -200
d = b**2 - 4*a*c
x1 = (-b + np.sqrt(d)) / (2*a)
x2 = (-b - np.sqrt(d)) / (2*a)
x1 , x2
```

3. Easier: Use `roots` function of `numpy` that solves polynomial equations $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = 0$ of any degree $n \geq 1$

```
np.roots([a, b, c])
```

Exercises

1. Solve the quadratic equation $x^2 + 2x + 1 = 0$
2. Solve the cubic equation $x^3 + 3x^2 + 3x + 1 = 0$

3.3 Solve Linear Equation Systems

- Solve linear equation systems of order $n \geq 1$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n &= b_m \end{aligned}$$

- This equation system can be formulated as a matrix equation

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

- with

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & ... & a_{mn} \end{pmatrix} \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

- Example with $m = 2$ and $n = 2$:

$$\begin{aligned} x_1 + 2x_2 &= 5 \\ 3x_1 + 4x_2 &= 6 \end{aligned}$$

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mathbf{b} = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$$

- Use the `numpy.linalg.solve` function

1. Create a new Markdown cell

```
# Solve Linear Equation System
```

2. Create a new Code cell

```
A = np.array([[1, 2], [3, 4]])
b = np.array([5, 6])
x = np.linalg.solve(A, b)
x
```

Exercises

1. Solve the linear equation system of order $n = 2$

$$\begin{aligned} 2x_1 + 4x_2 &= 2 \\ x_1 + 2x_2 &= 1 \end{aligned}$$

2. Solve the linear equation system of order $n = 3$

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 10 \\ 4x_1 + 5x_2 + 6x_3 &= 11 \\ 7x_1 + 8x_2 + 9x_3 &= 12 \end{aligned}$$

3.4 Function Plotting

1. Create a new Markdown cell

```
# Function Plotting
```

2. Create a new Code cell to import Python package `matplotlib.pyplot` for plotting

```
import matplotlib.pyplot as plt
```

3. Define 1000 sampling points in the range $[-100, 100]$

```
x = np.linspace(-100, 100, 1000)
```

4. Compute the values at the sampling points of the Sinc function $y = f(x) = \frac{\sin(x)}{x}$

```
y = np.sin(x) / x
```

5. Plot the function using `matplotlib`

```
plt.plot(x, y)
plt.title('Sinc Function')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.show()
```

Exercises

1. Plot a cosine function with frequency 50 Hz and amplitude 230 V.

3.5 Python Data Types

- Python variables are dynamically typed
  - The data type is determined at runtime
  - The data type is determined by assigning values to a variable
  - A variable can hold values of different types at different times
- The data type can be determined using the `type` function

```
type(-1.0)
type(2)
type(np.pi)
type(A)
type(x)
```

Numeric Types

`int`	integer with no limits	`1, -1`
`float`	64bit floating point on most CPUs	`3.14, -0.001, .5, 10e-1`
`np.float32`	32bit floating point	`np.float32(3.14)`
`np.float64`	64bit floating point	`np.float64(3.14)`
`complex`	complex numbers	`1+2j, -3-4j`
`bool`	boolean values	`True, False`

Text Type

`string`	string with UTF-8 characters	`"Max-Planck-Str. 39", 'Heilbronn'`

Aggregate Types

`list`	mutable list with different data types	`[ 1, -1, 3.14, "text" ]`
`list`	empty list	`[]`
`np.array`	mutable array with one data type	`np.array([ 1, -1, 3.14 ])`
`tuple`	immutable list with different data types	`( 1, -1, 3.14, "text" )`
`tuple`	empty tuple	`()`
`range`	immutable sequence of numbers (9 integers from 1, 2, ..., 9)	`range(1, 10)`
`set`	unordered collection of unique elements	`set( [ 1, 2, 3, 4, 5 ] )`
`dict`	dictionary of key-value pairs	`{"lastname": "Mouse", "firstname": "Mickey"}`

Element Access

- Define list and arrays

```
l = [ 1, -1, 3.14, "text" ]
A = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
x = np.array([1, 2, 3, 4, 5])
```

- Fist element (Python is zero-indexed)

```
l[0]
A[0, 0]
x[0]
```

- Last element

```
l[-1]
A[-1, -1]
x[-1]
```

- Sizes

```
len(l)
A.shape
x.size
```

- Slicing

```
l[1:3]
A[1, 1:3]
A[0, 1:]
x[1:]
```

- Slicing with step

```
l[0:-1:2]
A[0, ::2]
x[::2]
```

4 Procedural Programming

- Python is imperative: programs are sequences of statements
  - which are executed in order
- Python is procedural
  - Sequences of statements can be grouped into functions $\rightsquigarrow$ *reusability*
  - Functions can be called with parameters
  - Functions can return values
  - Functions can be called from different places in the program
- Python is object-oriented (see Section 5)

Agenda

- Simple hello world program (see Section 4.1)
- Hello world program with functions (see Section 4.2)
- Function for solving quadratic equations (see Section 3.2)

4.1 Hello World

1. Create a Python program `helloworld.py` in VS Code
   (a) by hitting `File - New File...`
   (b) selecting `Python File` in the dropdown menu
2. Enter the following code in the text editor

```
#!/usr/bin/env python3

print("Hello, World!")
```

   - `#!/usr/bin/env python3` is a *shebang* line that tells the system to use the Python interpreter
   - `print` is a built-in function in Python to generate output in the terminal
   - `"Hello, World!"` is a string of UTF-8 characters
   - `()` are parentheses used to call functions and to enclose function arguments
3. Save the program to a file `helloworld.py` by hitting `Ctrl+S`
4. Run the program by hitting `F5`
5. Select `Python File` from the dropdown menu

6. In the `terminal` window, the output should be displayed as `Hello, World!`

Run program from terminal

1. open a terminal window
2. navigate to the directory where the file is saved
3. run the Python interpreter and start the program

```
python helloworld.py
```

Run program from Python prompt

1. open a terminal window
2. navigate to the directory where the file is saved
3. run the Python interpreter `python`

   ```
   python
   ```

   which gives you a Python prompt
4. load and run the program at the Python prompt

```python
import helloworld
```

Run program from Jupyter notebook

1. create and execute a new cell

```python
import helloworld
```

4.2 Hello World with Functions

1. Create a new Python program `helloworld_function.py` in VS Code
2. Enter the following code in the text editor

```
#!/usr/bin/env python3

def hello_world():
    print("Hello world")

def hello(name):
    print("Hi", name, "!")
    print("How are you today?")

if __name__ == "__main__":
    hello_world()
    hello("Asterix")
    hello("Obelix")
```

- `def` is a keyword in Python to define a function
- `def <functionname> (<argument1>, <argument2>, ...):` defines a function where
  - `<functionname>` is the name of the function

  - `<argument1>, <argument2>, ...` is a list of arguments passed to the function
- The function body is indented by the `TAB` keyword
- The function body contains the statements to be executed when the function is called
- `hello_world` is a function that prints a greeting message
- `hello` is a function that takes a name as a parameter and prints a personalized greeting
- The `if __name__ == "__main__":` block ensures that the functions are only called when the script is run directly
  - The functions are only called if the program is run by hitting `F5` in VS Code
  - or by running the program from the terminal with `python helloworld_function.py`
  - but not when the program is imported with the `import` statement

Exercises

1. Extend the function `hello` by adding the additional message `"And how is Idefix?"` if the function argument `name` is equal to `"Obelix"`.

4.3 Function for Solving Quadratic Equations

1. Create a new Python program `quadratic.py` in VS Code
2. Enter the following code in the text editor

```
#!/usr/bin/env python3

import numpy as np

def solve_quadeqn(a, b, c):
    """
    Solves the quadratic equation ax^2 + bx + c = 0 for real roots.

    Parameters:
        a (float): Coefficient of x^2
        b (float): Coefficient of x
        c (float): Constant term

    Returns:
        tuple: A tuple containing the two real roots (x1, x2)
    """
    d = b ** 2 - 4.0*a*c
    x1 = (-b + np.sqrt(d)) / (2.0 * a)
    x2 = (-b - np.sqrt(d)) / (2.0 * a)
    return (x1, x2)
```

```
if __name__ == "__main__":
    a = 1.0
    b = -3.0
    c = 2.0
    roots = solve_quadeqn(a, b, c)
    print("The roots are:", roots)
```

3. Run the code by hitting `F5`
4. Change `b` to `1.0` and rerun the code

Exercises

1. Modify the code to check for a negative discriminant and return an empty tuple in that case.

5 Object-Oriented Programming

- Python is object-oriented
  - Data and functions can be grouped into *classes*
  - Classes can model real-world entities
  - Classes represent the *properties* as well as *behaviors* of entity types
    - *attributes* = properties = data
    - *methods* = behavior = functions
  - A class can be instantiated multiple times to create *objects* (*class instances*)
    - Each object represents an individual entity and has its own set of attribute values
  - Classes are encapsulated
    - Implementation details are hidden from the user
    - Users interact with objects through well-defined interfaces (subsets of the methods and attributes)
  - Classes can inherit from other classes and thus model sub-typing relationships between entities
    - A *quadratic equation* is a sub-type of a *polynomial equation*, which in turn is a sub-type of an *equation*

5.1 Class for Quadratic Equations

- The following code defines a class `QuadraticEquation` for solving quadratic equations of the form $ax^2 + bx + c = 0$
- Class `QuadraticEquation` represents a quadratic equation and stores the equation coefficients as attributes
- Its behavior is defined by the following methods

Method	Description
`__init__(self, a, b, c)`	Constructor to initialize the equation coefficients a, b, and c. This constructor is called when instantiating the class with `QuadraticEquation(a, b, c)`
`solve(self)`	Solves the quadratic equation and returns the real roots as a tuple
`__str__(self)`	Returns a string representation of the quadratic equation. `__str__(self)` is internally called by `print(eq)`.

- Within the class definition, `self` refers to the current instance of the class
- Upon instantiation, the variable `eq` holds a reference to the newly created `QuadraticEquation` object
- With `eq.solve()`, the `solve` method is called for object `eq`

```python
#!/usr/bin/env python3

import numpy as np

class QuadraticEquation:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def solve(self):
        """
        Solves the quadratic equation ax^2 + bx + c = 0 for real roots.

        Returns:
            tuple: A tuple containing the two real roots (x1, x2)
        """
        d = self.b ** 2 - 4.0 * self.a * self.c
        if d < 0.0:
            return None # No real roots
        x1 = (-self.b + np.sqrt(d)) / (2.0 * self.a)
        x2 = (-self.b - np.sqrt(d)) / (2.0 * self.a)
        return (x1, x2)

    def __str__(self):
```

```
        return f"QuadraticEquation({self.a}*x^2 + {self.b}*x + {self.c} = 0)"

if __name__ == "__main__":
    eq = QuadraticEquation(1, -3, 2)
    print(eq)
    roots = eq.solve()
    print("The roots are:", roots)
```

References

[1] E. Matthes. *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*. 3rd. No Starch Press, 2021. ISBN: 978-1718502703.

[2] Rui Pereira et al. "Ranking programming languages by energy efficiency". In: *Science of Computer Programming* 205 (2021), p. 102609.