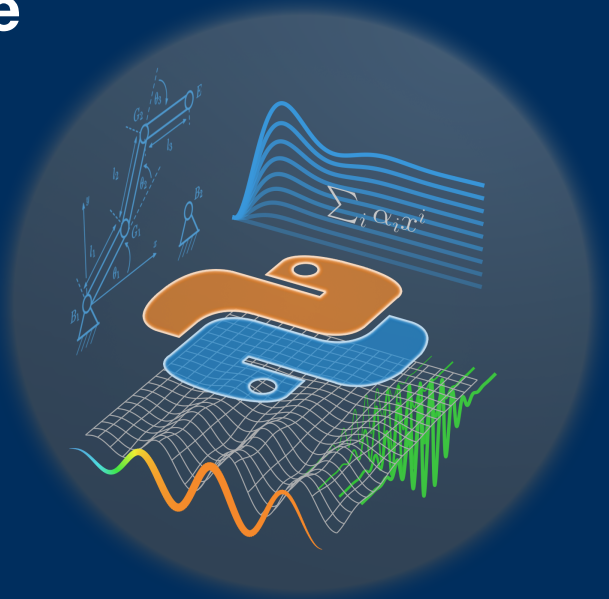
TECHNISCHE UNIVERSITÄT DRESDEN

**Faculty of Electrical and Computer Engineering** Institute of Control Theory

# PYTHON FOR ENGINEERS
# PYTHONKURS FÜR INGENIEUR:INNEN

## Communication with External Hardware
## Kommunikation mit externer Hardware



**Carsten Knoll**

https://tu-dresden.de/pythonkurs
https://python-fuer-ingenieure.de

Dresden, 2023-01-27

# Preliminary Remarks

**Goal:**

- which interfaces $\leftrightarrow$ which Python modules
- general recommendations

**Structure:**

- introduction
- interfaces
  - serial interface
  - parallel interface
  - GPIB
  - Ethernet
- using a DLL driver
- application examples
  - USB Missile Launcher
  - Mobile Robot (Arduino platform)
- general recommendations

# Outline

# Possible Communication Targets?

- measuring devices
- function generators
- controls for so called positioning units
- optical components (lamps, lasers, filters, ...)
- temperature controllers
- microcontroller (’µC’)
- other computers
- ...

# Possible Communication Targets?

- measuring devices
- function generators
- controls for so called positioning units
- optical components (lamps, lasers, filters, ...)
- temperature controllers
- microcontroller (’µC’)
- other computers
- ...

- background: own experience ( Fraunhofer IIS , iapp , RST )

# Python in the Laboratory? (I)

- top dog: LabVIEW (National Instruments)
- extensive (driver) library
- data flow oriented $\rightarrow$ graphical programming
- intuitive parallelization, very easy creation of GUIs

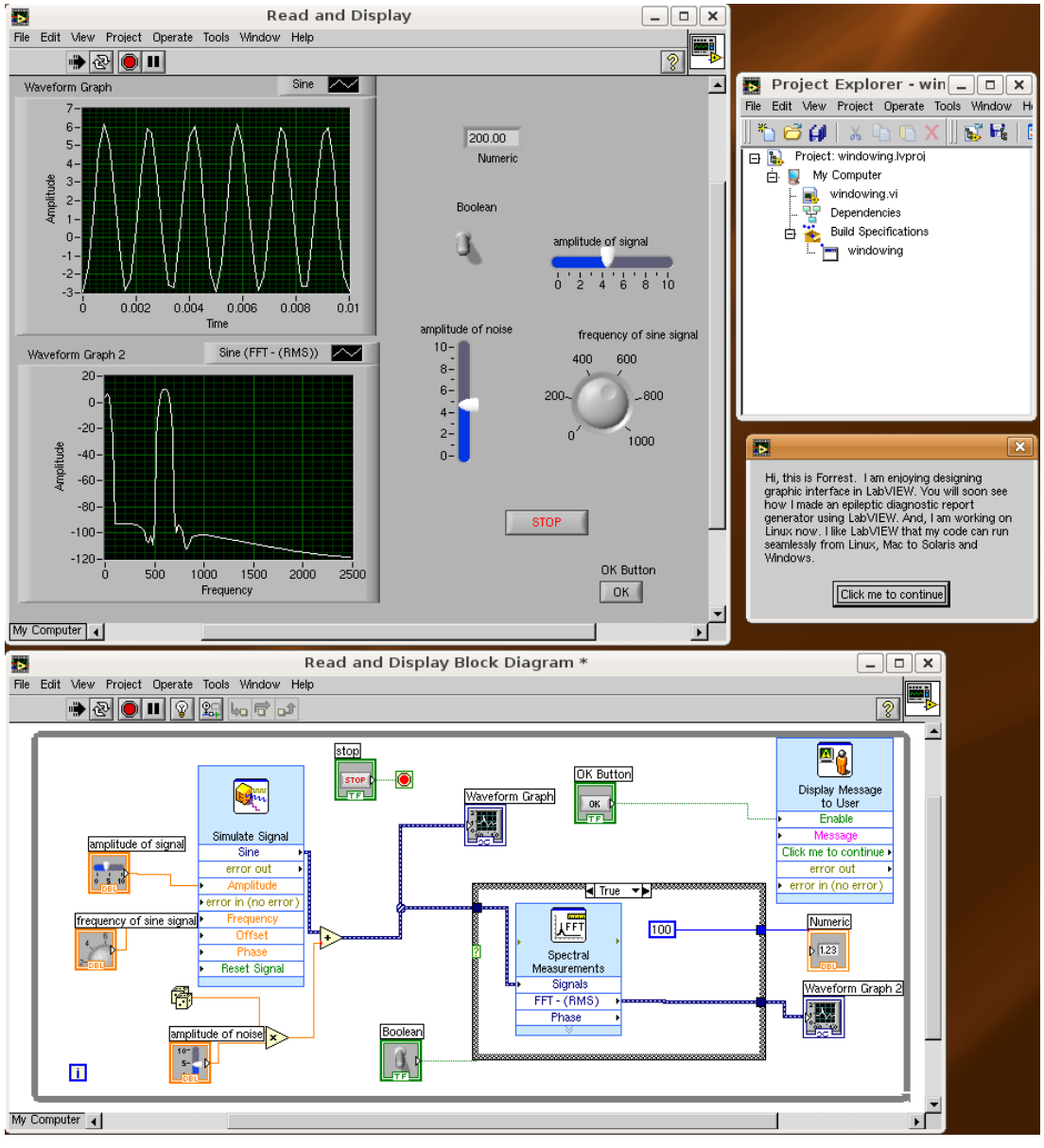
# Python in the Laboratory? (I)

- top dog: LabVIEW (National Instruments)
- extensive (driver) library
- data flow oriented $\rightarrow$ graphical programming
- intuitive parallelization, very easy creation of GUIs

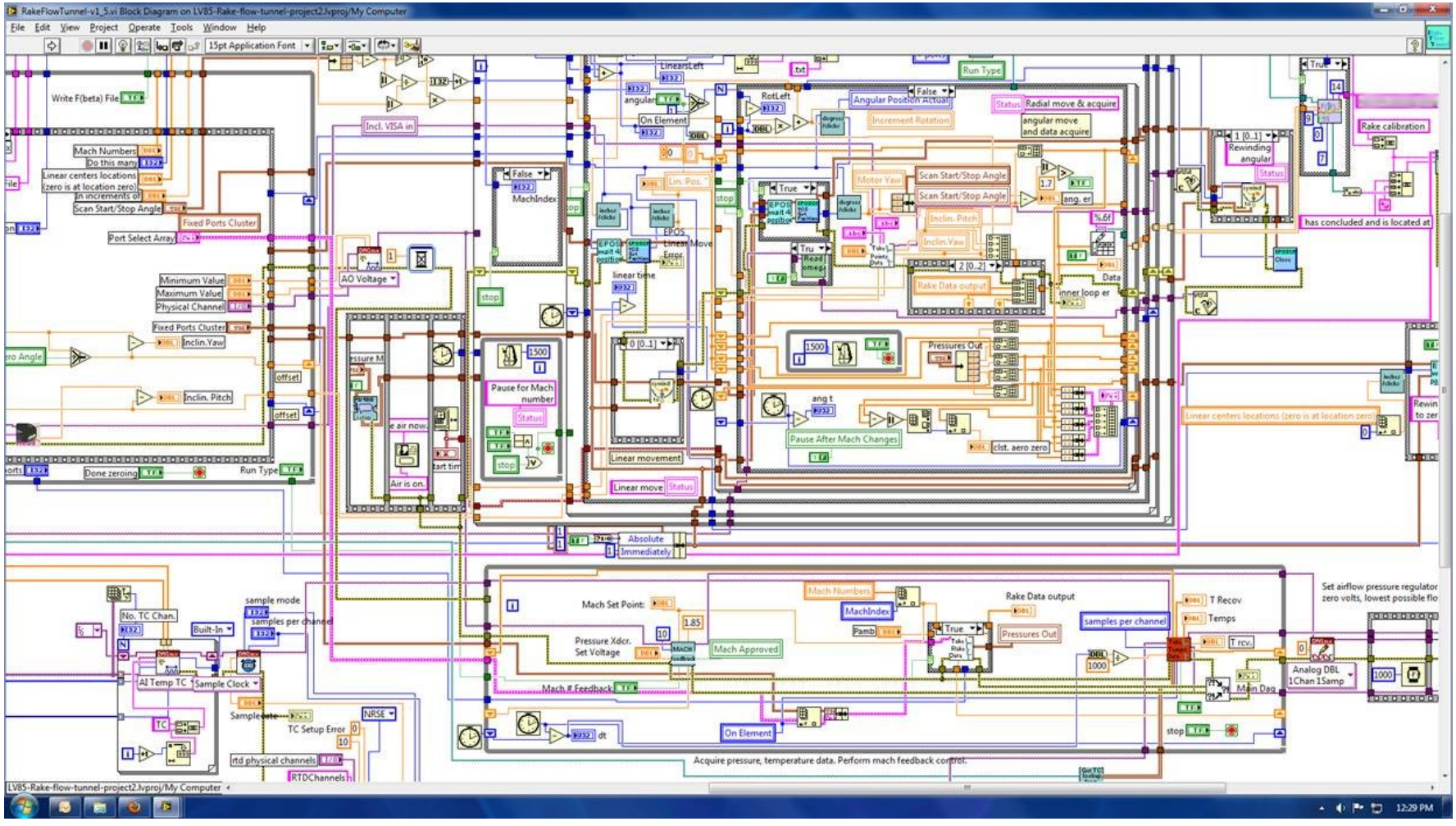
**Disadvantages** (personal opinion):

- main input via mouse (= bottleneck compared to keyboard)
- modularization/encapsulation more complex
- tends to clutter quickly
- permanent scarcity of screen space $\rightarrow$ tendency to to omit comments
- maintainability $\searrow$, extensibility $\searrow$
- ...

# Desired Look



# Reality: “LabVIEW Horror”



# Python in the Laboratory? (II)

When is Python a possible alternative?

- ... no (almost) ready-to-use LabVIEW solution exists
- ... no special LabVIEW features are needed (FPGA, hard realtime)
- ... actual algorithms have to be implemented (not only moving data)
- ... license costs play a role

# Real-Time Capability

- two concepts: “hard” and “soft” real-time conditions
  - hard: violation unacceptable (control of aircraft hydraulics)
  - soft: violation unattractive but not tragic (DVD player)
- approx. 95% of applications: soft
- main criterion: deterministic execution time
- Python programs have non-deterministic execution time (autom. garbage collector)
  $\rightarrow$ only soft real-time possible
- sufficiently fast? $\rightarrow$ depending on task (oftentimes: yes)

# Outline

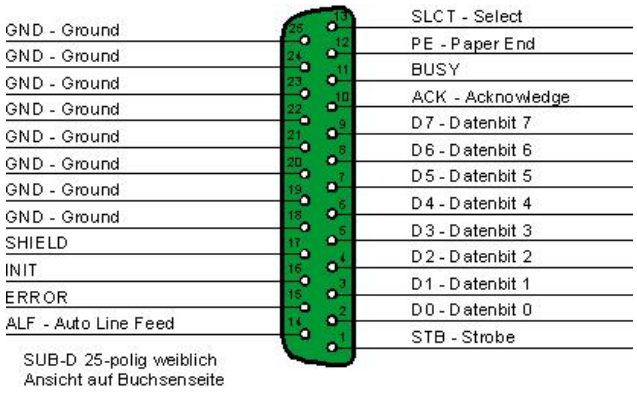
# Serial Interface

- “serial”: data (bit sequence) is transmitted one after the other
- mostly meant: `RS-232`
- nowadays often emulated via USB (Universal **Serial** Bus)
- Windows: `COM1,...` Unix: `/dev/ttyS1...`
- many (measuring) devices equipped with it
- package `serial` (project pySerial)

```
import serial
ser = serial.Serial(0, 19200, timeout=2.5) # open first serial port
print(ser.portstr) # check which port was really used
ser.write("hello") # write a string
response = ser.readline() # wait for the response
ser.close()
```

# Parallel Interface

- “parallel”: multiple bits transmitted simultaneously over parallel lines transmitted
- also known as “printer output”
- simplest option for digital I/O
- hardly available today
- sensitive to overvoltage and short circuit
- package `parallel` (also from project pySerial)




```
import parallel
p = parallel.Parallel() # open LPT1
p.setData(0x57) # write 0101 0111
responseBit = p.getInPaperOut()
```

# GPIB

- general purpose interface bus
  (also: “HP-IB”, “IEC-625 bus”,... )
- up to 15 devices in parallel

- package `visa` (project pyvisa)

```
import visa
keithley = visa.instrument("GPIB::12")
ident = keithley.ask("*IDN?")
assert ident.startswith("KEITHLEY INSTRUMENTS INC.")  # check consistency
keithley.write(":CONF:VOLT:DC")
v = keithley.ask(":READ?")
print(float(v))
```

# Ethernet (Local Network)

- modern devices: often with network connection
- internal web server (program) for communication
- socket:
  = endpoint of a network connection
- → client-server architecture
- package `socket` (python standard library)

```
# client program
import socket

HOST = "141.30.61.152"     # ip address of remote host
PORT = 50007               # same port as used by the server
s = socket.socket()
s.connect((HOST, PORT))    # connect to server
byte_arr = bytes("Hello, world", "utf8")
s.send(byte_arr)           # send something
data = s.recv(1024)        # wait for answer
s.close()
print("Received:", repr(data))
```

background information: on Python datatypes `bytes` vs `str` and encoding and unicode.

# Outline

# DLL Driver

- device drivers sometimes supplied by the manufacturer as compiled code
  Windows: `.dll` (“**d**ynamically **l**inked **l**ibray”); Linux `.so`; “**s**hared **o**bject”
- access it with package `ctypes` (from Python standard library)
  - required: knowledge about names and signature of functions
  - data types need special attention

```
# Example: read optical line sensor
# (Communication via USB (with unknown protocol))

import ctypes
camdll = ctypes.windll.LoadLibrary("D:/devices/cam123.dll")
camdll.setIntegrationTime(5) # 5ms  (value from docs)

# create arry: length = 512, data type = unsigned sort
valuearray_type = ctypes.c_ushort * 512
valuearray = valuearray_type()

# pass reference to array:
pixels = camdll.ReadData( ctypes.byref(valuearray) )
assert pixels == 512 # check consistency
values = list(valuearray) # convert ctypes array -> python list
```

# Outline

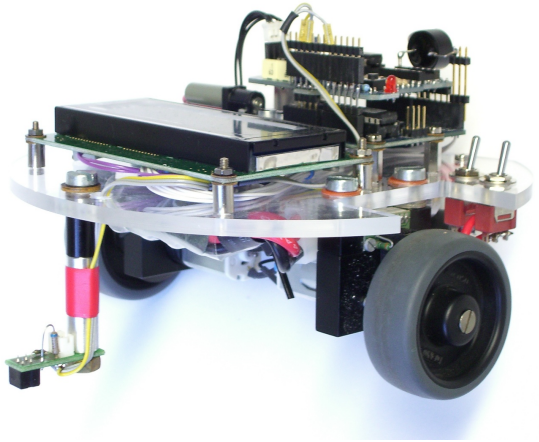
# Low-Cost Beam-Shutter

- application in optics laboratory
- task: interrupt laser beam (without time requirements)
- professional equipment: $>$500€ (project budget already depleted)
- idea: misuse of ... toys
- $\rightarrow$ two-axis adjustable launcher
- control from PC by Windows GUI program $\rightarrow$ doesn't matter

- $\exists$ DLL $\rightarrow$ Python wrapper (own development) $\rightarrow$ task solved

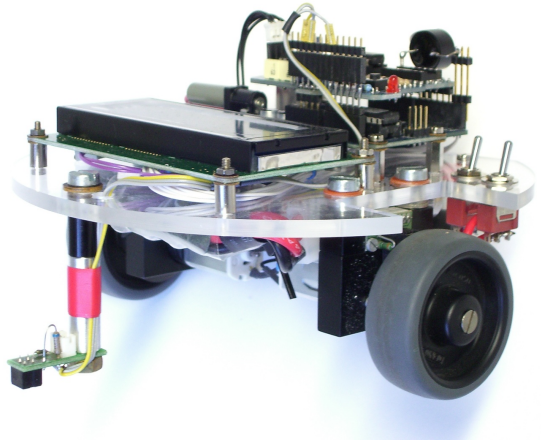
# Mobile Robot with Arduino Microcontroller (µC)

- project for student internship (7th/8th grade).
- abilities: driving, flashing, make sound, detecting underground (brightness sensors), communicate (display, RS-232)



# Mobile Robot with Arduino Microcontroller (µC)

- project for student internship (7th/8th grade).
- abilities: driving, flashing, make sound, detecting underground (brightness sensors), communicate (display, RS-232)



**Arduino:**

- open-source µC development platform simple and powerful
- Atmel AVR microcontroller + board
- USB, power supply, LEDs, reset button
- plug'n play
- digital + analog input and output (AO via pulse width modulation)
- software: IDE + libraries + examples
- $\rightarrow$ strongly simplified C++ programming

# Mobile Robot with Arduino Microcontroller (µC)

- project for student internship (7th/8th grade).
- abilities: driving, flashing, make sound, detecting underground (brightness sensors), communicate (display, RS-232)

**Arduino:**

- open-source µC development platform simple and powerful
- Atmel AVR microcontroller + board
- USB, power supply, LEDs, reset button
- plug'n play
- digital + analog input and output (AO via pulse width modulation)
- software: IDE + libraries + examples
- $\rightarrow$ strongly simplified C++ programming

More on this:

# Python Application: remote control

**Goal:**

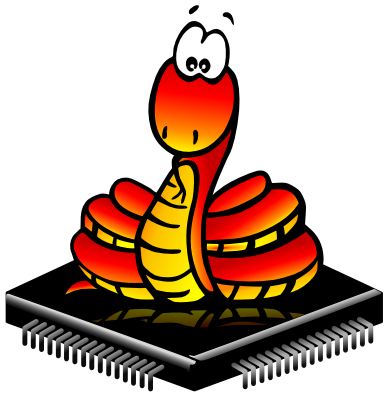
- Control the robot from the PC

**Implementation:** (see `exercise/external_code/robot/robot.ino`)

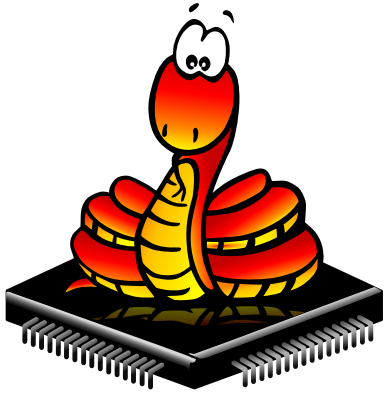
- `C++`-program on µC waits for commands
  and then executes certain actions
- Python program to send the commands
- each command consists of two bytes:
  - command ('F'=Forward, 'B'=Backward, ...)
  - argument (numeric value of the byte, [0, 255])
- inclusion of `ipython` shell ⇒ interactive text interface

# MicroPython

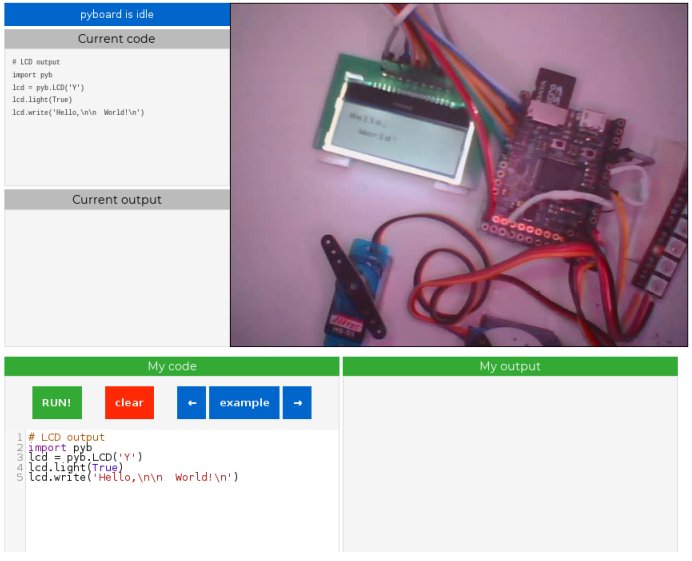
- Python interpreter that runs directly on microcontrollers
- “as compatible as possible”
- including interactive prompt (“REPL”)
- no own experience yet



# MicroPython



- Python interpreter that runs directly on microcontrollers
- “as compatible as possible”
- including interactive prompt (“REPL”)
- no own experience yet




- ∃ online live demo:
  `http://www.micropython.org/live`

# Outline

# General Recommendations

- thoroughly think through the (automation) task → make written notes (use cases, devices, relevant quantities, variables, ...)
- use as much existing as possible

- object orientation: one device ↔ one class (↔ one file)
- modularization: own package (e.g. `devices`)
- for every device: initialization phase and proper shutdown if necessary
- provide simulation mode (allows program testing without devices connected)
- consistency tests (check actually known information)
- check of permissible value ranges
- logging functionality (log to screen and to file, see https://docs.python.org/3/howto/logging.html)

# Links

- https://github.com/pyserial/pyserial
- https://pyvisa.readthedocs.io/en/stable/
- http://docs.python.org/library/socket.html
- http://docs.python.org/library/ctypes.html
- https://docs.python.org/3/howto/logging.html

- http://www.arduino.cc
- http://www.arduino.cc/playground/Interfacing/Python

- http://www.micropython.org

- http://wiki.python.org/moin/BitwiseOperators
- Python data types `bytes` vs `str` (short)
- general information on encodings and Unicode (longer)