Algorithms and Data Structures

A2. A Very Brief Introduction to Python

Gabriele Röger and Patrick Schnider

University of Basel

February 18, 2026

Python

Python

- interpreted high-level programming language
- supports imperative, object-oriented and functional programming
- easily readable code
- high productivity: for the same functionality, we need significantly less code than e.g. with Java
- extensive libraries
- execution often slower than with compiled languages
- named after Monty Python
  (English comedy troupe from the 1970s)

Python Interpreter

- we use Python 3.x
- program `python3` can execute programs or be used as an interactive interpreter:

Python Interpreter

```
Python 3.13.3 (main, Jan  8 2026, 12:03:54)
[GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>> 5 * 4
20
>>> exit()  (Linux: Ctrl+d)
```

Resources

- Python: `https://www.python.org/downloads/` or from a package repository (Ubuntu: apt install python3)
  - alternatively: scientific computing distribution Anaconda (`https://www.anaconda.com/`), contains much more than you need for this course
- reference and tutorial: `https://docs.python.org/3/`
- IDE: e.g.. PyCharm (`https://www.jetbrains.com/pycharm/`)
- or editor: e.g. emacs or vim (if you already know them), otherwise e.g. Visual Studio Code (`https://code.visualstudio.com/docs/python/python-tutorial`)
- style checker: e.g. Flake 8 (`http://flake8.pycqa.org/`) (Ubuntu: apt install python3-flake8)

Brief Language Overview

Jupyter Notebook

Jupyter notebook: `jupyter_intro.ipynb`

Dynamic Typing

- variables are type-less, only the objects they are referring to have a type.
- type checking only during runtime

```
>>> a = 3
>>> a/2
1.5
>>> a = "now the variable references a string"
>>> a/2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Indentation instead of Braces

indentation defines statement blocks
(such as functions, loop bodies, ...)

```
def count(to):
    for val in range(to):  # val = 0, ..., to-1
        print(val + 1)
    print("done")
```

Java: braces

tab $\neq$ space
recommendation: 4 spaces per level

range

- `range(stop)` generates integers from 0 to stop - 1
  → `range(3) yields 0, 1, 2`
- `range(start, stop[, step])`:
  generates integers from `start` to (excluding) `stop` with steps size `step`
  → `range(3, 11, 2) yields 3, 5, 7, 9`
  → `range(2, -3, -1) yields 2, 1, 0, -1, -2`
  → `range(2, 5) yields 2, 3, 4`

Lists and Tuples

- lists and tuples contain sequences of objects
- lists are written with brackets:
  `[3, "egg", "bacon"]`
- tuples are written with parentheses:
  `("sausage", 31, ["spam", "baked beans"])`
- difference
  - lists are mutable, we can add and remove elements.
  - tuples are immutable, they always contain the same objects in the same order (but the objects can be mutable).

Indexing and Manipulation

- We can index sequences from the front (non-negative integers) or the back (negative integers).
- The first element has index 0.
  `(4, 5, 2, 9)[1]` references 5.
- The last element has index -1.
  `(4, 5, 2, 9)[-2]` references 2.
- In mutable sequences, new assignments are possible.
  `a[2] = 4` for list `a`
- With `append`, we can extend a list by one element.
  `a.append(8)` appends 8 at the end of the list.

Example for Indexing and Manipulation

```
>>> fibonacci = (1, 1, 2, 3, 5, 8)
>>> print(fibonacci[0], fibonacci[2], fibonacci[-1])
1 2 8
>>> fibonacci_list = list(fibonacci)
>>> print(fibonacci_list)
[1, 1, 2, 3, 5, 8]
>>> fibonacci_list.append(14)
>>> print(fibonacci_list)
[1, 1, 2, 3, 5, 8, 14]
>>> fibonacci_list[-1] = 13
>>> print(fibonacci_list)
[1, 1, 2, 3, 5, 8, 13]
```

Immutability of Tuples

```
>>> l = (3, "egg", ["bacon"])
>>> l[2].append("spam")
>>> l
(3, 'egg', ['bacon', 'spam'])
>>> l[1] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: 'tuple' object does not support item assignment
```

More on Tuples

- Tuple Unpacking "unpacks" values on the right-hand side to assign them to the variables on the left-hand side.
  ```
  (number, name) = (3, "Johann Gambolputty")
  ```
- In general, we can omit parentheses around tuples if there is no ambiguity.

More on Tuples

- Tuple Unpacking "unpacks" values on the right-hand side to assign them to the variables on the left-hand side.
  `(number, name) = (3, "Johann Gambolputty")`
- In general, we can omit parentheses around tuples if there is no ambiguity.
- tuple unpacking thus possible without parentheses:
  `number, name = 3, "Johann Gambolputty"`
- often used to swap the values of two variables:
  `var1, var2 = var2, var1`

More on Tuples

- Tuple Unpacking "unpacks" values on the right-hand side to assign them to the variables on the left-hand side.
  `(number, name) = (3, "Johann Gambolputty")`
- In general, we can omit parentheses around tuples if there is no ambiguity.
- tuple unpacking thus possible without parentheses:
  `number, name = 3, "Johann Gambolputty"`
- often used to swap the values of two variables:
  `var1, var2 = var2, var1`
- note: tuples with only one element written with a comma:
  `(2,)`

Control Structures: `if`, `elif`, `else`

```
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

conditions: logical connectives with `and`, `or`, `not`
e.g. `x > 5 and y < 3`

Control Structures: `while`, `for`

Count down from 9 to 1 (two variants):

```
x = 9
while x > 0:
    print(x)
    x -= 1

for x in range(9, 0, -1):
    print(x)
```

- exit a loop with `break`
- skip the current iteration with `continue`

Functions and Main Function

```python
import sys

def power(base, exponent):
    return base ** exponent

def main():
    base, exp = int(sys.argv[1]), int(sys.argv[2])
    print(power(base, exp))

if __name__ == "__main__":
    # called if file is executed but not at import
    main()
```

Selection Sort in Python

Example: Selection Sort

```python
def selection_sort(a):
    """Selection sort sorting algorithm

    >>> selection_sort([3, 1, 6, 3, 2])
    [1, 2, 3, 3, 6]
    >>> selection_sort([])
    []
    """
    for i in range(len(a) - 1):
        min_index = i
        for j in range(i + 1, len(a)):
            if a[j] < a[min_index]:
                min_index = j
        a[i], a[min_index] = a[min_index], a[i]
    return a
```

Example: Selection Sort

selection_sort.py

```
import random

def selection_sort(a):
    cf. previous slide

if __name__ == "__main__":
    a = [n for n in range(40)]  # [0, 1, ... 39]
    random.shuffle(a)  # randomly shuffle the array
    print(a)
    selection_sort(a)
    print(a)
```

Example: Selection Sort

- unit test with `python3 -m doctest selection_sort.py`
- style check with `python3 -m flake8 selection_sort.py`
- execute with `python3 selection_sort.py`