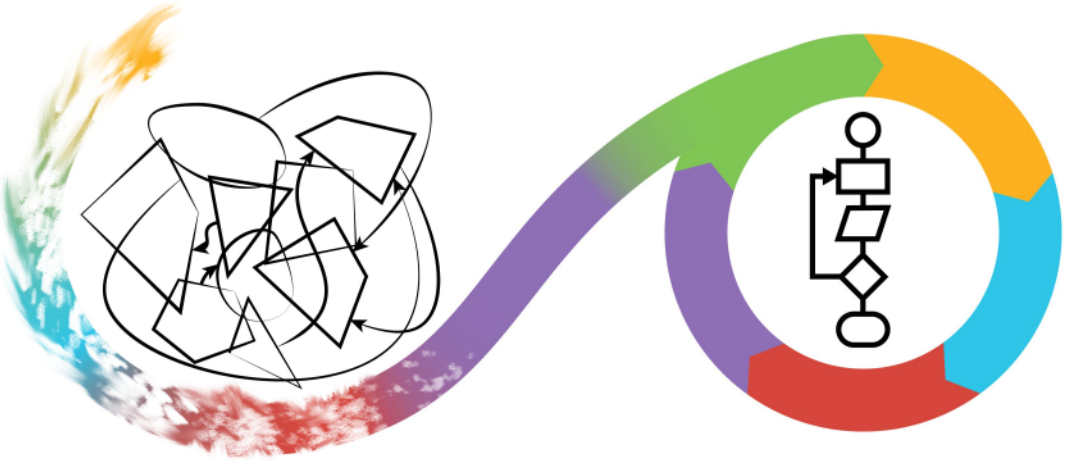
Data structures

Richèl Bilderbeek

1 The Big Picture 

https://github.com/UPPMAX/programming_formalisms/blob/main/tdd/tdd_lecture/tdd_lecture.qmd



1.1 Breaks

Please take breaks: these are important for learning. Ideally, do something boring (1)!

1.2 Schedule

Day	From	To	What
Thu	9:00	10:00	Class diagram for project
Thu	10:00	10:15	Break
Thu	10:15	11:00	Creating files and first tests for project

Day	From	To	What
Thu	11:00	11:15	Break
Thu	11:15	12:00	Lecture: Class design
Thu	12:00	13:00	Lunch

2 Data structures

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

3 Why data structures?

- Increase expressiveness
- To bundle data that belongs together

3.1 Increase expressiveness

```
a = [3.14, 2.72]
print(type(a))
```

```
<class 'list'>
```

A list is created …?

. . .

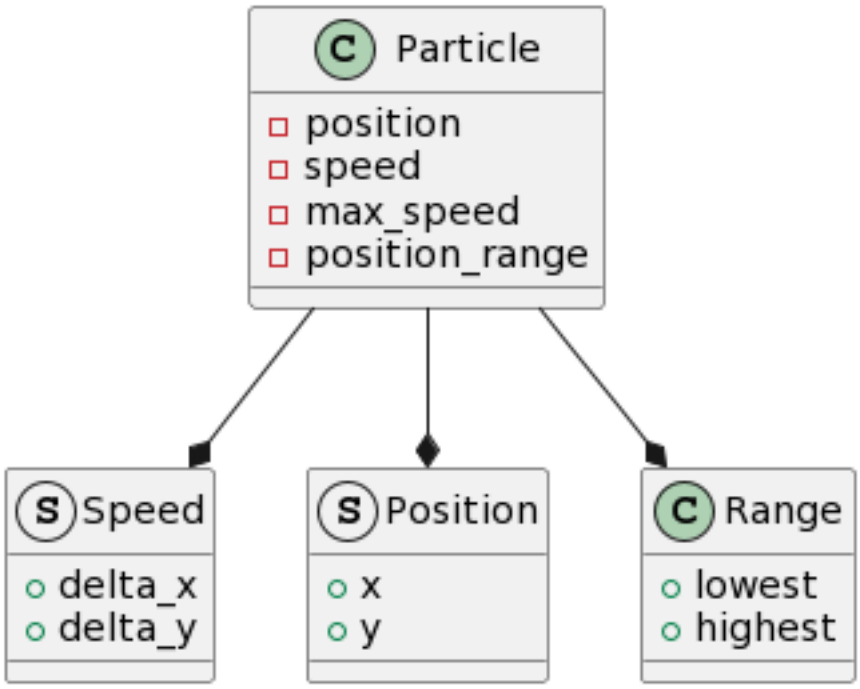
```
b = Coordinat(3.14, 2.72)
print(type(b))
```

```
<class '__main__.Coordinat'>
```

Ah, it is a **coordinat**!

- P.1. Express ideas directly in code
- PEP 20: 'Explicit is better than implicit'

3.2 Bundle data



- C.1. Organize related data into structures (structs or classes)

3.3 Bundle data

- C.2: Use class if the class has an invariant; use struct if the data members can vary independently
- C.8: Use class rather than struct if any member is non-public

4 Built-in data structures

4.1 Problem

Are there classes that can help me solve problems more elegantly?

4.2 Python classes

- `list`: heterogeneous container
- `tuple`: immutable `list`
- `set`: sets
- `dict`: dictionary
- Regular expressions: text patterns

From Python 'Data Structures' documentation

4.3 `set`

Sorted collection of unique elements.

```
data = [3, 1, 4, 1, 5]
s = set(data)
assert 3 in s
assert list(s) == [1, 3, 4, 5]
```

- No need to check for elements existing twice

4.4 `dict`

A dictionary:

```
periodic_table = dict({1: "Hydrogen", 2: "Helium", 3: "Lithium"})
periodic_table[2] = "helium"
assert periodic_table[2] == "helium"
```

- Commonly uses as a look-up table
- A look-up table can store the results of earlier calculations

4.5 Regular expressions

A state-machine for a pattern in text

```
import re
dna_regex = re.compile("^[ACGT]*$")
assert dna_regex.match("")
assert dna_regex.match("A")
```

```
assert dna_regex.match("CA")
assert dna_regex.match("GCA")
assert dna_regex.match("TGCA")
assert dna_regex.match("TGCAAAAAA")
assert not dna_regex.match("nonsense")
```

- https://docs.python.org/3/library/re.html

5 Writing a good class

Q: What is a good class?

. . .

A:

- guarantees its stored data is valid, e.g the class `DnaSequence` is probably a string of one or more A, C, G and T
- the quality requirements for a function, among others a good interface
- writing a design, documentation and tests all help

5.1 Class anatomy

- `__init__`: instantiation operation (a.k.a. 'constructor')
- Private variables: what it secretly is
- Methods: how to work on it

```
class DnaSequence:
    def __init__(self, sequence):
        assert is_dna_string(sequence)
        self._sequence = sequence # convention

    def get_str(self):
        return self._sequence

a = DnaSequence("ACGT")
assert a.get_str() == "ACGT"
```

5.2 Private variables are a social convention

Use of `_` before the name of a private variable is a social convention!

```
self._sequence = sequence # convention
```

Nothing stops you from:

```
a._sequence = "XXX"
assert a.get_str() == "XXX"
```

Some other programming languages offer stronger guarantees.

5.3 Inheritance and polymorphism

> C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it, to the point where it's much much easier to generate total and utter crap with it.
>
> Linus Torvalds, 2007-09-06

5.4 Inheritance and polymorphism

- Can create class hierarchies
  - 'All Animal objects can make a sounds'
- Easy to abuse, hard to use correctly
- Design Patterns are known to work well

5.5 Class design

- Python classes
- C++ Core Guidelines

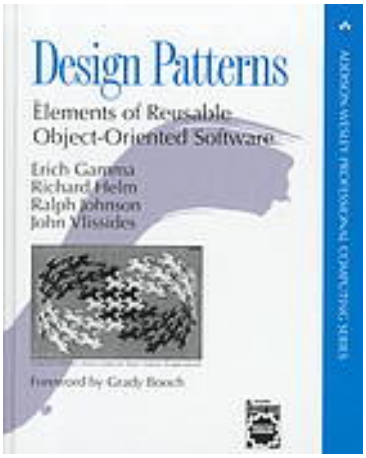


Figure 1: (2)

5.6 Recap

References

1. Newport C. Deep work: Rules for focused success in a distracted world. Hachette UK; 2016.
2. Gamma E, Helm R, Johnson R, Vlissides J, Patterns D. Elements of reusable object-oriented software. Design Patterns. 1995;