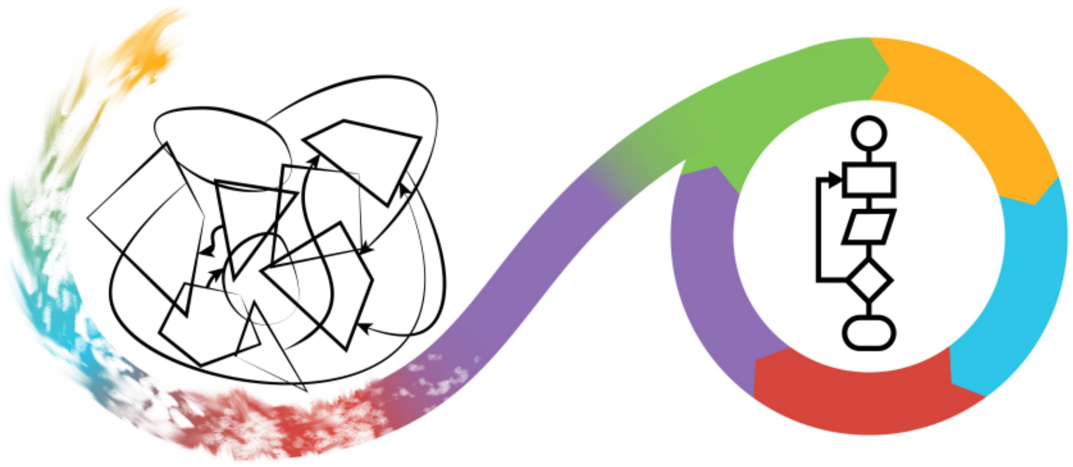
Data structures

Richèl Bilderbeek

1 The Big Picture

https://github.com/UPPMAX/programming_formalisms/blob/main/data_structures/data_structures_lecture/data_structures_lecture.qmd



1.1 Breaks

Please take breaks: these are important for learning.

It can sometimes be painful/annoying when there is a break in the middle of the exercise.

Ideally, do something boring (1)!

1.2 Overview

- Discuss: Reflection
- Do: catch-up: generate and understand merge conflicts
- Do: catch-up: work with branches
- Learn and do: Data structures
- Do: catch-up: Design project
- Learn and do: algorithms

1.3 Schedule

From	To	What
9:00	10:00	Work with git branches
10:00	10:15	Break
10:15	11:00	`struct` design
11:00	11:15	Break
11:15	12:00	class design
12:00	13:00	Lunch

2 Discuss reflection (10 mins)

- Show on Programming Formalisms repository

3 Do: catch-up: generate and understand merge conflicts

Goal: a learner should be able to explain why a merge conflict occurs

Demonstration:

- Ask a student to modify `fairytale.md`
- Show merge conflict, explain why

4 Do: catch-up: work with branches

Goal: a learner should be able to create, switch and merge branches

- Work on `fairytale.md`
- Work on `main`, `develop`, `[name]`

- Show exercise
- Demonstrate, preferably a learner volunteer

5 Data structures

The Zen of Python, by Tim Peters

[...] Readability counts. [...] 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. [...]

5.1 What and why?

Data structure are 'ways to organize your data'.

- Bad way: put all in one `list`

Good ( ) data structures:

- Increase expressiveness
- Bundles data that belongs together
- Ensures correct state of the program

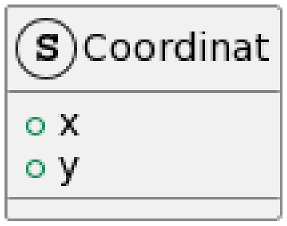
6 `struct` design

Good ( ) data structures:

- **Increase expressiveness**
- Bundles data that belongs together
- Ensures correct state of the program

6.1 Increase expressiveness, in design

"A two-dimensional coordinate **has a** x and a y component"



Class diagram of a two-dimensional coordinate

6.2 Increase expressiveness, in code

```
a = get_a()
print(a)
```

```
[3.14, 2.72]
```

```
print(type(a))
```

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

```
a <- get_a()
a
```

```
[1] 3.14 2.72
```

```
class(a)
```

```
[1] "numeric"
```

What is `a`?

6.3 Increase expressiveness, in code

```
a = get_a()
print(a)
```

```
(3.14, 2.72)
```

```
print(type(a))
```

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

```
a <- get_a()
a
```

```
[1] 3.14 2.72
attr(,"class")
[1] "Coordinat"
```

```
class(a)
```

```
[1] "Coordinat"
```

Ah, it is a **coordinat**!

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

6.4 structure versus class

Parameter	structure	class
Python keyword	`class`	`class`
R object type	S3 class	S4, R5, R6
All states are valid	Yes	No
Maintains a valid state	No	Yes
Example	A number for an x coordinate	A string for a DNA sequence

6.5 Exercise 1: design a struct (15 mins)

Goal:

- to convince design is trickier than one thinks
- to convince design has implications
- to grow appreciation of classes

6.6 Exercise 1: design a struct (15 mins)

What are their elements? Which do you guess are structures? Were they?

- A coordinate in 3 dimensions
- A velocity in two dimensions
- A circle
- A square

6.7 Exercise 1: a coordinate in 3 dimensions

- consists out of an `x`, `y` and `z` coordinate
- `struct`: all combinations valid

. . .

But:

- Spherical coordinate: radial distance, polar angle, azimuthal angle
- distance must be positive, hence `class`

6.8 Exercise 1: a velocity in two dimensions

- consists out of a `delta_x`, `delta_y`: `struct`
- consists out of a `angle`, `speed` ( ): maybe `class`

6.9 Exercise 1: a circle

A circle has:

- a center coordinat and …
  - a diameter ( ): maybe `struct`
  - a radius ( ): maybe `struct`
- three coordinats: class, must avoid colinearity

6.10 Exercise 1: a square

A square has

- a center coordinat and size ( ) and …
  - maybe an angle ( )
- a top-left coordinat and size ( ) and …
  - maybe an angle ( )
- a two opposing coordinats: struct?

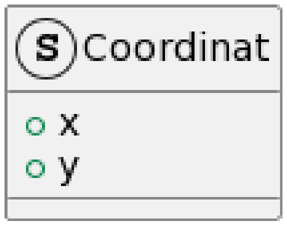
7 `struct` relations

Good ( ) data structures:

- Increase expressiveness
- **Bundles data that belongs together**
- Ensures correct state of the program

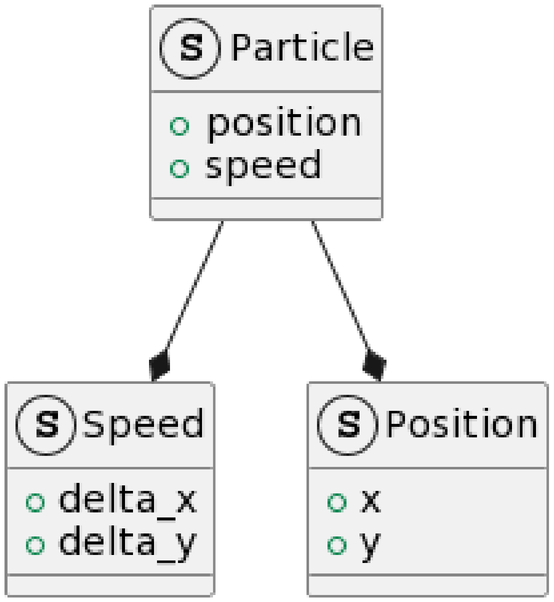
7.1 A has-a relationship

When one data type has a 'has-a' relationship with another, this is called *composition*.



A `Coordinat` has an `x` and `y`

7.2 Composition of structs



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

7.3 Exercise 2: create overview of classes (15 mins)

Goals:

- Experience first step in design
- Allow feeling that design could be improved in hindsight
- Distribute ownership of project

7.4 Exercise 2: overview of classes (20 mins)

- Read project description
- Which classes does the project need?
- Together:
  - make an alphabetic list of structures in the shared document
  - add exactly 1 maintainer to each struct
- Put it on the GitHub repository

7.5 Exercise 3: create structs (30 mins)

- Per struct maintainer, create one group
- Add the struct to the GitHub project repository

7.6 Discussion

- Structures increase expressiveness of code
- The design of data structures has implications
- Structures are useful to bundle data
- It is rare that structures match the real world

Agreed yes/no?

8 `class` design

8.1 An invariant

- 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

9 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

9.1 General class anatomy

- A constructor: all data needed to create it
- Private member variables
- Public member functions

9.2 Class anatomy in R

- R has four class types (S3, S4, R5, R6)
- S3 classes are closest to structures
- R6 classes are real classes

9.3 Class anatomy in Python

```
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"
```

9.4 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.

9.5 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

9.6 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

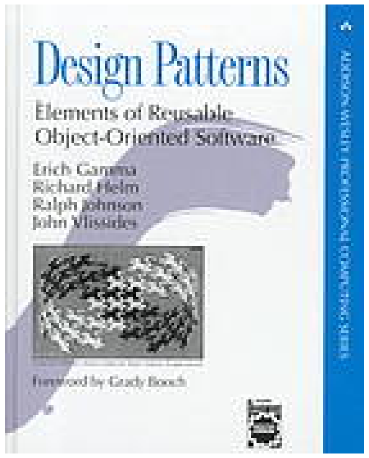

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
Addison-Wesley Professional Computing Series

Figure 1: (2)

9.7 Class design

- Python classes
- C++ Core Guidelines

10 Built-in data structures

10.1 Problem

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

10.2 Python classes

- `list`: heterogeneous container
- `tuple`: immutable `list`
- `set`: sets

- `dict`: dictionary
- Regular expressions: text patterns

From Python 'Data Structures' documentation

10.3 `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

10.4 `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

10.5 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

10.6 Discussion

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;