Data structures

TEAM INFDEV

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Introduction

Lecture topics

- Walkthrough retake exam
- Mechanism of abstraction
- The need for data structures
- Classes as data structures in Python

What is abstraction?

What is abstraction?

Introduction

- The big issue of the whole course is **abstraction** in programming
- Abstraction is a fundamental concept in programming to reduce repetition
- We sit atop a mountain of abstraction, which we make taller at every iteration

What is abstraction?

Grab the student next to you

- Describe what you just did so that someone else can perform the same action

What is abstraction?

Grab the student next to you

- Describe what you just did so that someone else can perform the same action
- Now add specific details about the movements of your arm and phalanges (pieces of fingers)

What is abstraction?

Grab the student next to you

- Describe what you just did so that someone else can perform the same action
- Now add specific details about the movements of your arm and phalanges (pieces of fingers)
- Now realize that there are even more subcomponents: individual muscles, tendons, etc.

What is abstraction?

Grab the student next to you

- Describe what you just did so that someone else can perform the same action
- Now add specific details about the movements of your arm and phalanges (pieces of fingers)
- Now realize that there are even more subcomponents: individual muscles, tendons, etc.
- But then we have also cells that make these up
- ...

What is abstraction?

Human love for abstraction

- Our brain cannot handle so many details
- To cope with this, we are structured in layers
- Our consciousness manipulates only the upper layers with simple instructions
- *Raise arm above head*

What is abstraction?

Human love for abstraction

- The same happens with regular language
- "*Go buy a liter of milk*" is quite a short description
- The underlying operation is very complex

Complexity of simple instructions

```
1  Go buy a liter of milk =
2    Turn game off
3    Get up from the couch
4    Curse the instruction giver
5    Get dressed
6    Put money in pocket
7    Leave house
8    Reach nearest shop
9    Enter shop
10   Find milk
11   Take one liter bottle
12   Pay milk
13   Go home
14   Give milk to instruction giver
```

What is abstraction?

Human love for abstraction

- And clearly something like "*reach nearest shop*" is not a trivial instruction by itself
- Think about all the things you give for granted
  - Crossing roads
  - Traffic lights
  - Pathfinding
  - Road work and obstructions
  - Use of transportation methods
  - ...

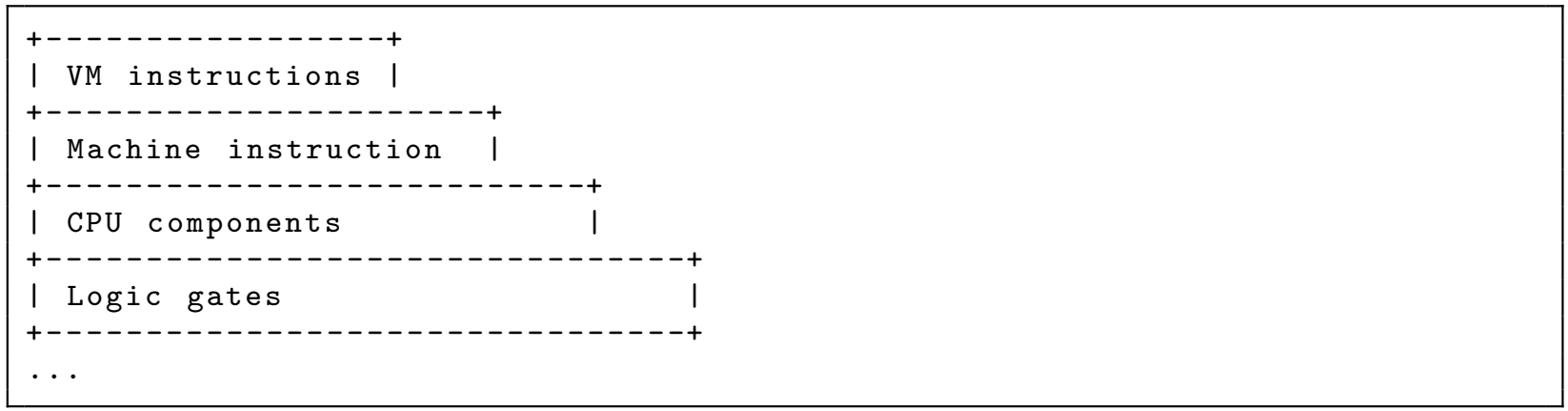
Data structures

Data structures

Flying back to Earth

- How is this relevant for programmers?
- We have a similar issue with a modern computer

A single Python instruction runs



Data structures

Flying back to Earth

- Moreover, sometimes we have repetition of constructs in our own code
- This means that we would like to extend the pyramid with our own stuff

A single Python program runs


```
+------------+
| Own stuff  |
+-----------------+
| VM instructions |
+----------------------+
| Machine instruction  |
+---------------------------+
| CPU components            |
+-----------------------------------+
| Logic gates                       |
+-----------------------------------+
...
```

Data structures

What kind of "*own stuff*"?

- Any recurring structure, code, etc.
- We do not want to repeat it every time
- We just give it a name, instead of specifying it every time
- The actual goal is to make things simpler
  - Code reuse, maintainability, etc. do not exist
  - It is all just **properly built abstractions that make reasoning about code easier**

Repeated code


```
1  playerOneName = "P1"
2  playerOnePositionX = 0.0
3  playerOnePositionY = 0.0
4
5  playerTwoName = "P2"
6  playerTwoPositionX = 5.0
7  playerTwoPositionY = 0.0
8
9  playerThreeName = "P3"
10 playerThreePositionX = 10.0
11 playerThreePositionY = 0.0
```

Repeated code


```
1  playerOneName = "P1"
2  playerOnePositionX = 0.0
3  playerOnePositionY = 0.0
4
5  playerTwoName = "P2"
6  playerTwoPositionX = 5.0
7  playerTwoPositionY = 0.0
8
9  playerThreeName = "P3"
10 playerThreePositionX = 10.0
11 playerThreePositionY = 0.0
```

Now let's add a score, an exp level, etc.

Repeated code


```
1  playerOneName = "P1"
2  playerOnePositionX = 0.0
3  playerOnePositionY = 0.0
4
5  playerTwoName = "P2"
6  playerTwoPositionX = 5.0
7  playerTwoPositionY = 0.0
8
9  playerThreeName = "P3"
10 playerThreePositionX = 10.0
11 playerThreePositionY = 0.0
```

Now let's add a score, an exp level, etc.

Does it scale well?

Data structures

Make some examples

- **Everyone make an example of repeated structures of data.**
- **Some of you will present theirs**

General idea

General idea

Introduction

- A possible solution to this problem is capturing the repetition of data structures
- With a name, and a specification of what is common about them

General idea

Fundamental ingredients of the solution

- Brains of the programmer, always active
- Abstraction requires awareness and experience
- It is as much technique as it is art

Repeated code


```
1  playerOneName = "P1"
2  playerOnePositionX = 0.0
3  playerOnePositionY = 0.0
4
5  playerTwoName = "P2"
6  playerTwoPositionX = 5.0
7  playerTwoPositionY = 0.0
8
9  playerThreeName = "P3"
10 playerThreePositionX = 10.0
11 playerThreePositionY = 0.0
```

General idea

Fundamental ingredients of the solution

- We observe that there is an underlying pattern, which we will call **abstraction**
- The pattern, or abstraction, comes repeated in several **concrete instances** in our program

General idea

Fundamental ingredients of the solution

- We observe that there is an underlying pattern, which we will call **abstraction**
- The pattern, or abstraction, comes repeated in several **concrete instances** in our program
- In the program above this is fairly obvious, in real life not always really :)

General idea

Fundamental ingredients of the solution

- A proper name for the abstraction
- **For example?**

General idea

Fundamental ingredients of the solution

- A proper name for the abstraction
- **For example?** `Player`

General idea

Fundamental ingredients of the solution

- A set of common attributes
- All characterizing aspects of the abstraction that are common to all its instances
- **For example?**

General idea

Fundamental ingredients of the solution

- A set of common attributes
- All characterizing aspects of the abstraction that are common to all its instances
- **For example?** `Name`, `PositionX`, `PositionY`

The blueprint (**THIS IS NOT CODE!**)


```
1  Abstraction Player =
2    Name, which is a sequence of characters
3    PositionX, which is a number
4    PositionY, which is a number
```

The abstraction above is called a **data structure**.

It is not valid Python code, but it is a blueprint specifying a recurrent set of attributes that often go together to identify a player.

Assignment

Assignment

Assignment

- Think of the project,
- identify (at least) 3 data structures and
- define the blueprints.

Assignment

Examples

- We are now ready to implement our player data type
- We will use a Python class to do so
- We will then create concrete instances of it, and use them

The blueprint to implement


```
1 Abstraction Player =
2   Name, which is a string
3   PositionX, which is a number
4   PositionY, which is a number
```

The implemented class


```
1  class Player:
2    def __init__(self, name, posX, posY):
3      self.Name = name
4      self.PositionX = posX
5      self.PositionY = posY
```

Creating concrete instances


```
1  playerOneName = "P1"
2  playerOnePositionX = 0.0
3  playerOnePositionY = 0.0
4
5  playerTwoName = "P2"
6  playerTwoPositionX = 5.0
7  playerTwoPositionY = 0.0
8
9  playerThreeName = "P3"
10 playerThreePositionX = 10.0
11 playerThreePositionY = 0.0
```

Becomes:

```
1  playerOne   = Player("P1", 0.0, 0.0)
2  playerTwo   = Player("P2", 5.0, 0.0)
3  playerThree = Player("P3", 10.0, 0.0)
```

The implemented class


```
1  class Player:
2    def __init__(self, name, posX, posY):
3      self.Name = name
4      self.PositionX = posX
5      self.PositionY = posY
```

General template:

```
1  class <<Name>>:
2    def __init__(self, <<v1>>, <<v2>>, ..., <<vN>):
3      self.<<A1>> = <<v1>>
4      self.<<A2>> = <<v2>>
5      ...
6      self.<<AN>> = <<vN>>
```

The class has thus: `name`, `initial values` v_1 through v_N, and `attributes` A_1 through A_N initialized with `__init__`. `self` is a reference to the `concrete instance` that is being set up.

Usage of Python classes

Using a class:

```
1 playerOne   = Player("P1", 0.0, 0.0)
```

General template:

```
1 x = <<Name>>(<<v1>>, <<v2>>, ..., <<vN>)
```

Sets up a `concrete instance` of <<Name>> with some `initial values`.

Usage of Python classes

Reading:

```
1 print(x.<<A2>>)
```

Prints the value of the second `attribute` of the `concrete instance` called `x` of class `<<Name>>`.

// Writing:

```
1 x.<<A3>> = y
```

Assigns `y` as the new value of the third `attribute` of the `concrete instance` called `x` of class `<<Name>>`.

Assignment

Assignment

Write your classes in Python

- Write the data structures you defined before in Python classes
- Test them out.

Designing data structures

Designing data structures

Are we there yet?

- We can keep extending our knowledge about the problem
- For example, we might notice that `PositionX` and `PositionY` might happen in other places of the program
- **What could we do?**

Designing data structures

Are we there yet?

- We can keep extending our knowledge about the problem
- For example, we might notice that `PositionX` and `PositionY` might happen in other places of the program
- **What could we do?**
- We could define a `Point2D` (or `Vector2D`) data structure!

Refined data structures


```
1  class position:
2    def __init__(self, x, y):
3      self.X = x
4      self.Y = y
5
6  class PlayerRefined:
7    def __init__(self, name, posX, posY):
8      self.Name = name
9      self.position = position(posX,posY)
```

Designing data structures

Refined data structures

- Creation is precisely identical to the previous sample
- The `__init__` of the `PlayerRefined` has the same inputs
- Where we had `playerOne = Player("P1", 0.0, 0.0)`
- Now we have `playerOne = PlayerRefined("P1", 0.0, 0.0)`

Designing data structures

Refined data structures

- Usage of the new player definition is almost identical to the previous
- Only changes are lookups like: `playerOne.PositionY`
- **What do they become now?**
- `playerOne.Position.Y`

Assignment

Assignment

Build, in class, a series of data structures

- Tyre
- Wheel
- Engine
- Seat
- Light
- Person (driver and passenger)
- Car

Assignment

What characterizes a good design of data structures?

- **Reuse** of code in places where otherwise repetition would happen
- **Encapsulation** of the semantics of the data structure
- **Loose coupling** between the data structure and the rest of the program

Assignment

Reuse of code

- Repetition is dangerous
- A small change in one place but not in the others can lead to unexpected consequences
- More code to read means more mental overhead
- Actual work of the program is hidden under lots of noise and thus less visible

Assignment

Encapsulation

- A data structure has a single, clear, well-defined goal
- Its name clearly explains what it contains and does
- There is no multiple functionality mix

Assignment

Encapsulation

- A data structure has a single, clear, well-defined goal
- Its name clearly explains what it contains and does
- There is no multiple functionality mix
- It's a cold beer, not a cocktail

Assignment

Loose coupling

- A data structure is a closed and complete unit
- To use it, you just need to declare it and initialize it
- The rest of the program integrates a well-designed data structure with minimal modification

Assignment

How do we verify all this?!?

- Takes experience and good taste
- It is an old story
- Remember: you have the power to make your own life a living Hell...

Assignment

How do we verify all this?!?

- Takes experience and good taste
- It is an old story
- Remember: you have the power to make your own life a living Hell...
- ...unless you reason first and write code after

Conclusion

Conclusion

Lecture topics

- Abstraction is the fundamental mechanism that allows us to group concepts together and refer to them as if they were a single concept
- For example, a name and two numbers became a `player`
- We then use the new concept (the `player`) without having to explicitly mention all of its components every time
- This makes it leaner for us to manipulate complex programs, as less concepts ("actors") make an appearance

This is it!

The best of luck, and thanks for the attention!