# The logical model of computation

The INFDEV Team @ HR

Hogeschool Rotterdam
Rotterdam, Netherlands

# Introduction

# Introduction



## Course topics

- This course is about *basic programming concepts* (DEV I)
- We will discuss computational concepts
- Computational thinking
- Describing computations clearly

# Introduction



## Course topics

- *How does a programming language work?*
- Memory, variables, conditionals, if-statements, and loops
- These are already enough to implement anything (of course not handily!)

# Introduction



**At the end of the course you will be able to...**

- ...describe algorithms clearly
- ...write basic programs in Python
- ...describe the semantics of a basic Python program

# A programming language

# A programming language

## What is programming not about?

- computers
- programming languages
- technology
- programs
- websites
- smartphones
- ...

# A programming language

## What is programming about?

- the encoding of logical thought
- non-ambiguity: there is only one possible mode of execution
- precision: there is no appeal to vagueness or intuition

# A programming language

## What is programming about?

- especially if a machine will eventually run our program
- machines are **dumb as **ck**[a]

[a] rock

# A programming language

## A programming language specifies

- what instructions we have
- what do they perform
- in what order

# Let's start programming

# Let's start programming



## The `stdNt` programming language

- In `stdNt` we let students perform some actions
- It does not require a machine, but only a white-board and alive (and complying students)

## Following instructions[1]

```
1 take 3 steps forward
2 sit on the chair
3 turn left
4 slide 3 steps forward
```

[1] The teacher should ask for a volunteer

# Let's start programming

## The features of `stdNt` so far

- **Instructions**, in English
- **Order of execution** is left-to-right, top-to-bottom
- **State** made up of a living, breathing student

## Following instructions with state *(we need a "volunteer")*

| A | B | C |
|---|---|---|
| your age | 2 | -3 |

```
1 take A/4 steps forward
2 sit on the chair
3 turn left by 90 * B degrees
4 slide C steps forward
```

# Let's start programming

## The features of `stdNt` so far

- **Instructions**, in English
- **Order of execution** is left-to-right, top-to-bottom
- **State** made up of a living, breathing student plus a bunch of cards with data written on them

## What if the state makes no sense? *(we need a "volunteer")*

| A | B | C |
|---|---|---|
| your age | ‘‘nice day today’’ | -3 |

```
1 take A/4 steps forward
2 sit on the chair
3 turn left by 90 * B degrees
4 slide C steps forward
```

# Let's start programming

## State comes with big preconditions

- It only contains information that is:
  - used in a way that makes sense with respect to the instructions
  - logically expressed (numbers, strings, etc. rather than emotions or riddles)
  - actually accessible (there is some connection from the executor to the accessed data)

## The state may change *(we need a "volunteer")*

| B | C |
|---|---|
| -1 | today's weather |

```
1 make a comment on C
2 write on C the index of the current day of the week
3 sit on the chair
4 turn left by 90 * B degrees
5 slide C steps forward
```

# Let's start programming



## The features of `stdNt` so far

- **Instructions**, in English
- **Order of execution** is left-to-right, top-to-bottom
- **Mutable state** made up of a living, breathing student plus a bunch of cards with data written on them

## We can make decisions[2]

| A | B | C | D |
|---|---|---|---|
| shirt colour | -1 | 2 | 3 |

```
1 sit on the chair
2 if A is ‘‘black’’ then
3   turn left by 90 * B degrees
4 otherwise
5   turn left by 90 * C degrees
6 clap D times
```

[2] Teacher should ask the students to perform the action

# Let's start programming

## The features of `stdNt` so far

- **Instructions**, in English
- **Order of execution** is left-to-right, top-to-bottom
- **Mutable state** made up of a living, breathing student plus a bunch of cards with data written on them
- **Decisions** based on elements of the state

# We can repeat behavior[3]

```
while there are green soldiers alive
      AND
      there are brown soldiers alive
  TEAM 1:
    a = pick green soldier
    d = pick brown soldier
    fight(a,d)
  TEAM 2:
    a = pick green soldier
    d = pick brown soldier
    fight(a,d)
```

```
fight(a,d):
  if a = BAZOOKA AND d = GRENADIER then
    both die
  else if a = BAZOOKA then
    d dies
  else if d = GRENADIER then
    a dies
  else if brown team still has leader t
    a dies
  else
    d dies
```

[3] The teacher should ask for two teams of volunteers

# Let's start programming

## The features of `stdNt` so far

- **Instructions**, in English
- **Order of execution** is left-to-right, top-to-bottom
- **Mutable state** made up of a living, breathing student plus a bunch of cards with data written on them
- **Decisions** based on elements of the state
- **Repetition** of code based on elements of the state

# Let's start programming

## Assignment 1 in groups of four

- Reprogram the game
- Make it so that the positioning of defending soldiers makes a difference (positive or negative)
- *One group will be "randomly selected" to present*

# Let's start programming

## Assignment 2 in groups of four

- Think about the actions needed for a game concept (at most 10).
- Write them down and put them in the box.
- Pick a sheet at random (if it is the one you wrote pick again).
- Write the implementation of a game using the actions you have.
- A group will be chosen to play the game.

# This is it!

The best of luck, and thanks for the attention!