HOGESCHOOL ROTTERDAM / CMI

Development 1

INFDEV02-1
2016-2017

Number of study points:	4 ects
Course owners:	Dr. Giuseppe Maggiore, Tony Busker

Modulebeschrijving

<table>
<tr><td>Module name:</td><td>Development 1</td></tr>
<tr><td>Module code:</td><td>INFDEV02-1</td></tr>
<tr><td>Study points and hours of effort:</td><td>This module gives 4 ects, in correspondence with 112 hours:
3 x 6 hours frontal lecture
3 x 6 hours practicum
the rest is self-study</td></tr>
<tr><td>Examination:</td><td>Written examination and practicums (with oral check)</td></tr>
<tr><td>Course structure:</td><td>Lectures, self-study, and practicums</td></tr>
<tr><td>Prerequisite knowledge:</td><td>None.</td></tr>
<tr><td>Learning tools:</td><td>Book: Think Python; author A. B. Downey (<code>http://www.greenteapress.com/thinkpython/</code>)
Presentations (in pdf): found on N@tschool and on the GitHub repository <code>https://github.com/hogeschool/INFDEV02-1</code>
Assignments, to be done at home and during practical lectures (pdf): found on N@tschool and on the GitHub repository <code>https://github.com/hogeschool/INFDEV02-1</code></td></tr>
<tr><td>Connected to competences:</td><td><table><tr><th></th><th>analyse</th><th>advies</th><th>ontwerp</th><th>realisatie</th><th>beheer</th></tr><tr><td>gebruikersinteractie</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>bedrijfsprocessen</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>software</td><td>1</td><td>1</td><td>1</td><td>1</td><td></td></tr><tr><td>infrastructuur</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>hardware interfacing</td><td></td><td></td><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>Learning objectives:</td><td>At the end of the course, the student can:
understand and describe the logical model of computation <code>LMC</code>
understand and describe the concrete model of computation <code>CMC</code>
use variables and basic data types <code>VAR</code>
use arithmetic and boolean expressions <code>EXPR</code>
use conditional control-flow statements <code>COND</code>
use looping control-flow statements <code>LOOP</code>
understand the concept of abstraction through function definition <code>FUNABS</code>
use and design functions <code>FUNDEF</code></td></tr>
</table>

Content:	• basic concepts of computation from a logical standpoint • basic concepts of computation from a concreter perspective in terms of storage and instructions • variables (in Python 2) • primitive datatypes and expressions (in Python 3) • conditional control-flow statements (in Python 3) • looping control-flow statements (in Python 3)
Course owners:	Dr. Giuseppe Maggiore, Tony Busker
Date:	29 augustus 2016

1 General description

Programming is one of the most ubiquitous activities within the field of ICT. Many business needs are centered around the gathering, elaboration, simulation, etc. of data through programs.

This course covers the very basic aspects of programming in the Python programming language (version 2).

1.1 Relationship with other teaching units

All subsequent programming courses build upon the knowledge learned during this course.

Knowledge acquired through the programming courses is also useful for the projects. A word of warning though: projects and development courses are largely independent, so some things that a student learns during the development courses are not used in the projects, some things that a student learns during the development courses are indeed used in the projects, but some things done in the projects are learned within the context of the project and not within the development courses.

2 Course program

The course is structured into lectures. The lectures take place during the six weeks of the course, but are not necessarily in a one-to-one correspondance with the course weeks. For example, lectures one and two are fairly short and can take place during a single week.

2.1 Lecture 1 - logical model of computation

The first lecture covers basic concepts of computation from a logical standpoint.

Topics

- Following a path (example: *take three steps forward, turn left, ...*);
- Following a path with state (example: *read N from the whiteboard, take N steps forward, ...*);
- Following a path with wrongly typed state (example: *take Monday steps forward, ...*);
- Following a path with state and conditionals (example: *take N steps forward if the lecturer is smiling, ...*).

Activities

- Let students follow instructions;
- Introduce elements of state and let students follow instructions with state (*take N/4 steps forward; N is your age*);
- Introduce elements of writable state and let students follow instructions with writable state (*take N/4 steps forward; N is written under the yellow sticker*);
- Introduce elements of decision-making and let students follow instructions with state and decision making (*if the sun is shining, then take N/4 steps forward; otherwise, go sit down*);
- Introduce elements of iteration and let students follow instructions with state, decision making, and iteration (*divide the students in teams, and let run some script battling for the toy farm*).

2.2 Lecture 2 - concrete model of computation

The second lecture covers basic concepts of computation from a more concrete perspective in terms of storage and instructions.

Topics

- CPU and memory;
- a basic overview of the various things that an imperative language can do, independent of syntax;
- introduction to semantics and post-conditions.

Activities

- Formalize the concept of instructions seen in the previous lecture by rewriting the scripts;
- Formalize the concept of state and mutation seen in the previous lecture by rewriting the scripts;
- Formalize the concept of decision-making seen in the previous lecture by rewriting the scripts;
- Formalize the concept of iteration seen in the previous lecture by rewriting the scripts.

2.3 Lecture 3 - Hello Python! and variables

The third lecture covers an introduction to Python and its concept of variables.

Topics

- brief history of programming languages;
- brief introduction to Python: what it does, what it does not, why we have chosen it;
- variables in python for integers;
- the effect of variable assignment on memory;

2.4 Lecture 4 - datatypes, and expressions

The fourth lecture covers primitive datatypes and their associated expressions in the Python programming language.

Topics

- what are data-types and *why we do need them*?
- different Python data-types;
- arithmetic expressions;
- integer and floating point operators;
- boolean expressions;
- conditional expressions;
- very long expressions with conditionals vs temporary variables: the art of naming to encode knowledge.

Activities Call upon students to solve small riddles related to sample Python scripts on:

- Integers, strings, floats, bools;
- Integer, string, float, and bool variables;
- Semantics and post-conditions on variable-assignments.
- Integers, strings, floats, bool expressions;
- Conditional expressions;
- Semantics and post-conditions on expressions and conditional expressions.

2.5 Lecture 5 - conditional control-flow statements

The fifth lecture covers conditional control-flow statements in the Python programming language.

Topics

- making choices;
- `if-then` statements;
- `if-then-else` statements;
- the importance of an `else` statement;
- (slightly informal) semantics;
- exponential explosion of potential control-paths;
- expressive power of `if-then-else`.

Activities Call upon students to solve small riddles related to sample Python scripts on:

- `if-then` and `if-then-else` statements;
- how many possible final states of a program;
- semantics and post-conditions on conditional statements.

2.6 Lecture 6 - looping control-flow statements

The sixth (and last) lecture covers looping control-flow statements in the Python programming language.

Topics

- repeated behaviors;
- `while` statements;
- (slightly informal) semantics;
- (more than) exponential explosion of potential control-paths;
- expressive power of `while`;
- `for` statements;
- (slightly informal) semantics;
- `for` as a *limited* form of `while`.

Activities Call upon students to solve small riddles related to sample Python scripts on:

- `while` and `for` loops;
- how many possible final states of a program;
- semantics and post-conditions on loops.

2.7 Lecture 7 - functions

The third lecture covers abstraction over (groups of) instructions and statements through functions:

Topics

- Abstraction operations (functions)
- The need for functions;
- Creating and using functions in Python;
- Formal and actual parameters and return;
- Brief introduction to: scope (local and global variables) and visibility;
- Syntax and semantics;
- Introduction to recursion;

3 Assessment

The course is tested with two exams: a series of practical assignments, a brief oral check of the practical assignments, and a theoretical exam. The final grade is determined as follows:

`if theoryGrade ≥ 75% & practicumCheckOK then return practicumGrade else return insufficient`

This means that the theoretical knowledge is a strict requirement in order to get the actual grade from the practicums, but it does not reflect your level of skill and as such does not further influence your grade.

Motivation for grade A professional software developer is required to be able to program code which is, at the very least, *correct*.
In order to produce correct code, we expect students to show: *i*) a foundation of knowledge about how a programming language actually works in connection with a simplified concrete model of a computer; *ii*) fluency when actually writing the code.
The quality of the programmer is ultimately determined by his actual code-writing skills, therefore the final grade comes only from the practicums. The quick oral check ensures that each student is able to show that his work is his own and that he has adequate understanding of its mechanisms. The theoretical exam tests that the required foundation of knowledge is also present to avoid away of programming that is exclusively based on intuition, but which is also grounded in concrete and precise knowledge about what each instruction does.

3.1 Theoretical examination INFDEV02-1

The general shape of a theoretical exam for `INFDEV02-1` is made up of a series of highly structured open questions. In each exam the content of the questions will change, but the structure of the questions will remain the same. For the structure (and an example) of the theoretical exam, see the appendix.

3.2 Practical examination INFDEV02-1

Each week there is a mandatory assignment. The assignments of week 4, 5 and 6 will be graded. Each assignment is due the following week. The sum of the grades will be the *practicumGrade*. If the course is over and *practicumGrade* is lower than $5,5$ then you can retry (herkansing) the practicum with one assignment which will test all learning goals and will replace the whole *practicumGrade*.

Exam structure

What follows is the general structure of a INFDEV02-1exam.

3.2.0.1 Question I: formal rules

General shape of the question: *given a set of rules, and given the starting point, what is the result of excution? Give the intermediate steps as well.*

Points: *25%.*

Grading: *All points for correct answer, partial trajectories count for exactly half the score, a completely orthogonal trajectory counts for no points.*

Associated learning goals: `LMC`.

3.2.0.2 Question II: program state

General shape of the question: *Fill-in the program state with the values that the variables assume while running the sample below.*

Points: *25%.*

Grading: *Full points if all stack frames and return types are correctly listed, with the right time stamps. Half points if at least half of all stack frames is listed. Zero points otherwise.*

Associated learning goals: `CMC`.

3.2.0.3 Question III: variables, expressions, and data types

General shape of the question: *What is the value and the type of all variables after execution of the following code?*

Points: *25%.*

Grading: *All values and types are correct: full-points. At least half the values and at least half the types are correct: half points. Zero points otherwise.*

Associated learning goals: `VAR`, `EXPR`.

3.2.0.4 Question IV: control flow

General shape of the question: *What is the value of all variables after execution of the following code?*

Points: *25%.*

Grading: *All values are correct: full-points. At least half the values are correct: half points. Zero points otherwise.*

Associated learning goals: `COND`, `LOOP`.

Exam sample

What follows is a concrete example of the exam.

3.2.0.5 Question I: formal rules

You start at point (0,0). Take a step in the direction (10,0) until you are above point (45,0). Then take five steps in the direction (0,2). Where do you end up?

Answer: *The trajectory is:*

```
P1 = (50,0)
 +----- P2 = (50,10)
 |
 |
 |
 |
P0 = (0,0)
```

Points: *25%.*

3.2.0.6 Question II: program state

Fill-in the program state with the values that the variables assume while running the sample below.

```
y = 1
for i in range(0, 5):
    y = y * 2
```

Answer: *The variable allocations are:*

y	1	1	2	2	4	4	8	8	16	16	32
i	n.a.	0	0	1	1	2	2	3	3	4	4

Points: *25%.*

Grading: *Full points if all values are correctly listed in the right order. Half points if at least half of values are listed in the right order. Zero points otherwise.*

Associated learning goals: `CMC`.

3.2.0.7 Question III: variables, expressions, and data types

What is the value and the type of all variables after execution of the following code?

```
v = 0
i = "Hello + world"
j = "Hello" + "world"
k = 10 / 3
```

Answer: *The value and type of all variables after execution is:*

Variable	Value	Type
v	0	int
i	'Hello + world'	str
j	'Helloworld'	str
k	3	int
l	3.3333...	float

Points: *25%.*

Grading: *All values and types are correct: full-points. At least half the values and at least half the types are correct: half points. Zero points otherwise.*

Associated learning goals: `VAR`, `EXPR`.

3.2.0.8 Question IV: control flow

General shape of the question: *What is the value of all variables after execution of the following code?*

Concrete example of question: *What is the value of all variables after execution of the following code?*

```
v = 0
for i in range(1,15):
  if (i % 2 == 0) & (i % 3 == 0):
    v = v + i
```

Concrete example of answer: *The value of all variables after execution is:*

Variable	Value
`i`	`14`
`v`	`18`

Points: *25%.*

Grading: *All values are correct: full-points. At least half the values are correct: half points. Zero points otherwise.*

Associated learning goals: `COND`, `LOOP`.

Bijlage 1: Toetsmatrijs

Learning goals	Dublin descriptors
`LMC`	1, 4
`CMC`	1, 4
`VAR`	1, 2, 4
`EXPR`	1, 2, 4
`COND`	1, 2, 4
`LOOP`	1, 2, 4

Dublin-descriptors:

1. Knowledge and understanding
2. Applying knowledge and understanding
3. Making judgments
4. Communication
5. Learning skills