Dev I and Dev II

Dr. Giuseppe Maggiore, Tony Busker

July 6, 2015

1 Introduction

General description of course.

2 Learning goals

As a general goal, after the courses of `DEV I` and `DEV II` the student can read and write basic imperative code. Moreover, the student is aware of techniques to refactor or design code in order to improve encapsulation and single responsibility of classes and functions.

More specifically after the courses, the student can:

skill	**goal**	**course**	**ID**
understand and describe	the logical model of computation	`DEV I`	`LMC`
understand and describe	the concrete model of computation	`DEV I`	`CMC`
use	variables and basic data types	`DEV I`	`VAR`
use	arithmetic and boolean expressions	`DEV I`	`EXPR`
use	conditional control-flow statements	`DEV I`	`COND`
use	looping control-flow statements	`DEV I`	`LOOP`
understand	the concept of abstraction through function definition	`DEV II`	`FUNABS`
use and design	functions	`DEV II`	`FUNDEF`
use and design	recursive functions	`DEV II`	`FUNREC`
understand	the concept of abstraction through class definition	`DEV II`	`CLSABS`
use and design	classes without inheritance or interfaces	`DEV II`	`CLSDEF`
use	recursively defined data structures	`DEV II`	`RECDATA`
use	arrays	`DEV II`	`ARR`

3 Examination

The course is tested with two exams: a series of practical assignments and a single theoretical exam. The final grade is determined as follows:

```
if theoryGrade >= 75% 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: *i*) correct; *ii*) sufficiently fast.

There are other criteria related to usability, flexibility, readability of code, etc. Because `DEV I` and `DEV II` refer to a basic level of skill, we will only focus on correspondingly basic criteria.

In order to produce correct and sufficiently fast code, a programmer needs 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 theoretical exam tests that the required foundation of knowledge is also present to avoid a purely intuitive way of programming.

3.1 Theoretical examination DEV I

The general shape of a theoretical exam for `DEV I` is made up of a series of highly structured open questions.

Question I: formal rules

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

Concrete example of question: *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?*

Concrete example of answer: *The trajectory is:*

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

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`.

Question II: stack and heap

General shape of the question: *Fill-in the stack and the heap with the objects that are allocated while running the sample below.*

Concrete example of question: *Fill-in the stack and the heap with the objects that are allocated while running the sample below. Add an asterisk (*) next to stack or heap values that are deallocated.*

```
incr x = x + 1
double x = x * 2
f x = if x >= 0 then incr(x) else double(x)

for i = 1 to 2 do
  print(f(i - 2))
```

Concrete example of answer: *The stack allocations are:*

`f(-1)` (t_0)	`-2` (t_3)	`print(-2)` (t_4)	`f(0)` (t_5)	`1` (t_8)	`print(1)` (t_9)
`double(-1)` (t_1)	`-2` (t_2)		`incr(0)` (t_6)	`1` (t_7)	

There are no heap allocations in the given code.

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`.

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?*

Concrete example of question: *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
```

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

Variable	Type	Value
`v`	`int`	`0`
`i`	`string`	`"Hello + world"`
`j`	`string`	`"Helloworld"`
`k`	`float`	`3.3`

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`.

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 = 1 to 10 do
  if i % 2 <> 0 && i % 3 <> 0 then
    v = v + i
```

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

Variable	Value
`i`	21
`v`	23

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`.

3.2 Practical examination DEV I

After setting up the environment, every two lectures a practicum is due, for a total of **three practicums**. Each practicum must be handed in at the beginning of the second lecture, and that lecture is used to grade and discuss what was handed in. The overall procedure is made out of the following steps:

Practicum	**Activity**	**Hand-in**	Points
1	Write a simple Python script that declares and assign variables with various int, string, and float expressions.	Nothing	2
2	Discuss and get grades.	Variables and expressions.	
3	Write a simple *turtle* script that reads some input values and makes the turtle move according to the input values by using `if` statements.	Nothing	4
4	Discuss and get grades.	`if` statements.	
5	Write a simple *turtle* script that makes the turtle move in a loop, square, or even spiral by using `for` or `while` statements.	Nothing	4
6	Discuss and get grades.	`for` and `while` statements.	

Grades are simply determined based on correctness. Each practicum script must not produce a compile-time error, a run-time error, and perform the desired computation. Each practicum is either fully correct (full points) or fully incorrect (no points). No intermediate scores are allowed.

3.3 Theoretical examination DEV II

The general shape of a theoretical exam for `DEV II` is made up of a series of highly structured open questions.

Question I: abstracting patterns with functions

General shape of the question: *Given the following block of code, define functions in order to reduce repetition, and rewrite a shorter but equivalent version of the original code by using your functions.*

Concrete example of question: *Given the following block of code, define a function in order to reduce repetition, and rewrite a shorter but equivalent version of the original code by using your functions.*

```
x = 10
y = x + 1
z = x + y * 2
print(x,y,z)

a = 5
b = a + 2
c = a + b * 4
print(a,b,c)
```

Concrete example of answer: *The resulting code is:*

```
f(s, d, f) =
  v0 = s
  v1 = s + d
  v2 = v0 + v1 * f
  print(v0,v1,v2)

f(10,1,2)
f(5,2,4)
```

Points: *25%.*

Grading: *All points for correct answer, correct function but wrong use or correct use but function with minor mistakes half points, wrong function and wrong use zero points.*

Associated learning goals: `FUNABS`, `FUNDEF`.

Question II: abstracting patterns and structures with classes

General shape of the question: *Given the following block of code, define a series of classes in order to reduce repetition of variables and code, and rewrite a shorter but equivalent version of the original code by using your classes.*

Concrete example of question: *Given the following block of code, define a class in order to reduce repetition of variables and code, and rewrite a shorter but equivalent version of the original code by using your class.*

```
cnt1 = 0
incr1() = cnt1 <- cnt1 + 1

cnt2 = 1
incr2() = cnt2 <- cnt2 * 2

for i = 1 to 100 do
  incr1()
  incr2()
```

Concrete example of answer: *The resulting code is:*

```
class Counter(z,step) =
  cnt = z
  incr() = cnt <- step(cnt)


cnt1 = new Counter(0, + 1)
cnt2 = new Counter(1, * 1)

for i = 1 to 100 do
  cnt1.incr()
  cnt2.incr()
```

Points: *25%.*

Grading: *All points for correct answer, correct function but wrong use or correct use but function with minor mistakes half points, wrong function and wrong use zero points.*

Associated learning goals: `CLSABS`, `CLSDEF`.

Question III: recursion on simple data structures

General shape of the question: *Define a recursive function that performs a simple operation on the nodes of a list or a tree. Show an instance of the tree, call its* `Add` *method, and indicate what the result would be.*

Concrete example of question: *Fill-in the body of the (recursive)* `Add` *method in order to add all elements of the binary tree.*

```
struct Node
  Left
  Value
  Right

  ... constructor ...

  Add() = ?
```

Concrete example of answer: *The resulting code is:*

```
struct Node
  Left
  Value
  Right

  ... constructor ...

  Add() =
    res = Value
    if Left =/= null then
      res += Left.Add()
    if Right =/= null then
      res += Right.Add()
    return res
```

An instance of the tree would be:

```
sample = new Node(new Node(10), 5, new Node(-10))
```

Calling

```
sample.Add()
```

returns value 5.

Points: *25%.*

Grading: *All points for correct answer, correct function but wrong use or correct use but function with minor mistakes half points, wrong function and wrong use zero points.*

Associated learning goals: `FUNREC`, `RECDATA`.

Question IV: array processing

General shape of the question: *Define a loop that performs some simple operation on an array.*

Concrete example of question: *Define a loop that sums all positive elements of the array* `numbers`. `numbers` *contains only integers.*

Concrete example of answer: *The resulting code is:*

```
sum = 0
for i = 0 to numbers.length - 1
  if numbers[i] > 0 then
    sum += numbers[i]
return sum
```

Points: *25%.*

Grading: *All points for correct answer, otherwise zero points.*

Associated learning goals: `ARR`.

3.4 Practical examination DEV II

Every two lectures a practicum is due, for a total of **three practicums**. Each practicum must be handed in at the beginning of the second lecture, and that lecture is used to grade and discuss what was handed in. The overall procedure is made out of the following steps:

Practicum	**Activity**	**Hand-in**
1	Write a series of functions that: *i*) add the parameters together; *ii*) return the smallest of the parameters; *iii*) make the *turtle* perform some set of movements; combine the movements with more functions; *iv*) recursively add numbers from `0` to `n`. .	Nothing
2	Discuss and get grades.	Variables and expressions.
3	Write a series of classes that: *i*) represent a fraction; *ii*) encapsulate a *turtle* and make it perform some sets of movements; *iii*) represent a recursive list; *iv*) represent a recursive binary tree. .	Nothing
4	Discuss and get grades.	`if` statements.
5	Write a series of array-processing functions to: *i*) add all elements of an array; *ii*) add all even elements of an array; *iii*) find the smallest element of an array; *iv*) find a specific element of an array; *v*) revert the order of the array elements. .	Nothing
6	Discuss and get grades.	`for` and `while` statements.