DEVELOPING A DISTANCE EDUCATION PROGRAMMING SKILLS COURSE FOR GEO-INFORMATION SCIENCES AND EARTH OBSERVATION STUDENTS

Ivánová, I., Huisman, O., de By, R.A., Rutzinger, M., Bakker, W., Feringa, W.

ITC Faculty of Geo-Information Science and Earth Observation of the University of Twente, The Netherlands
[ivanova, huisman, deby, bakker, feringa, rutzinger]@itc.nl

Commission VI

KEY WORDS: Programming, courseware, Python, e-learning, open-source

ABSTRACT:

This paper describes the development of a distance education course on programming skills. The purpose of this course is twofold: first, it supports ITC's Joint- Education Projects (JEPs) currently underway in several developing countries, and secondly, it broadens the target market for our educational services. One of the key aims in teaching programming skills using any programming language is to teach students how to approach computational problem-solving in a structured and logical way. The challenge in doing so is how to develop adequate learning resources on the programming fundamentals, which are both intuitive and functional. Drawing upon several years of experience of teaching programming skills at the Faculty of Geo-Information Science and Earth Observation of the University of Twente (ITC, The Netherlands), we designed a course based on an interactive learning environment, developed for a previous distance education course (Ivánová et al. 2008), but significantly extended to support new content. The programming language taught is Python, which is a general-purpose, open-source computer language well-suited for use in the hybrid context of databases, Geographic Information Systems, image processing, and web applications. It is well-known for its lean syntax, which allows the novice student to devote more time to algorithmic and programming fundamentals. The course is designed around an interactive learning infrastructure. It is built in an environment that seamlessly integrates the Python environment (= program scripting and command line) and the teaching environment (= lecture material, exercise environment, and a textbook). The course material is developed for plug-and-play deployment on a student's PC, reducing the initial requirements on installation and optimization of the work environment. Although the nature of studying a programming language is intrinsically self-study, this plug-and-play solution also allows components of the course to be deployed in a face-to-face set-up, as well as in distance education mode. This type of integrated learning environment currently does not exist anywhere; existing solutions are mostly in the form of isolated tutorials or books. From a didactic perspective, the course presented here is a system of self-instructional learning objects, allowing students to proceed in their own location, in their own time, and at their own pace, with an extensive direct feedback embedded in the course materials. Most of the course content is based upon open-source materials – the textbook and some of the exercises. However, we are extending these materials to tailor the course better to geoinformation science students by adding material (in the form of lessons, as well as adding a chapter to the book itself). By doing so, we release the new chapter into the open-source realm through creative commons licensing.

1. INTRODUCTION

1.1 Programming Skills at ITC

A substantial fraction of geoscientific problems can be viewed as problems in which geoinformation in one format needs to be transcribed into another one. Solutions to such problems require high-level analytic thinking, with the application of algorithmic and computational principles, and then a realization of the solution through appropriate application of programming tools. The standard application fields where programming skills are used are databases and Geographical Information Systems (GIS) analysis, image processing, and web services.
Programming Skills is a geoinformatics course component split into two parts. The first part is integrated into the current core courses, which provide the fundamentals in geoinformatics and geosciences (four three-week, full-time educational modules) of ITC's MSc and Master degree programme. Using a variety of theoretical and applied examples, these courses aim to teach students how to approach problem-solving in a structured, logical way, and to develop computational solutions for given problems.

Python is a general-purpose, open-source computer programming language used by thousands of developers around the world in areas as diverse as spatial modelling, internet scripting, user interfaces, product customization, and more. The language is generally regarded as easy to learn, due to its no-frills syntax and insightful command structure. It is also recognized as an extremely powerful language, with more and more industrial-quality extension modules becoming available. Finally, it is widely used in geo-software projects and libraries such as in hybrid contexts of databases, GIS, image processing and web applications.

1.2 Rationale for a distance education course on programming skills

The distance education course on programming skills (DEPS) covers the same topics as the core module Programming Skills 1 of ITC's GFM degree programmes, with extended and integrated material to support e-learning. The key objective was that no face-to-face contact is required. The DEPS course offers a 'stand-alone' introduction to Python of eight weeks duration (plus one 'warm-up' week), with materials developed

specifically for plug-and-play deployment on student PCs via a USB stick.
Drawing upon several years of experience of the GFM Programming Skills course, DEPS is deployed as an *integrated* environment for student learning. Effectively, the Python 'environment' (program scripting and command-line) is accessible directly from within the learning environment, providing seamless integration of lecture material, exercises, and other support documentation developed specifically for the course.
This type of environment currently does not exist anywhere, and is a valuable learning resource that contributes to reduced requirements for tutor input during the course run. Together with a focus on student-centred content, it also provides greater flexibility for students (for study in own time, at own pace, in own location, with own learning style preference) within an interactive learning environment.
It is the intention that the course material is also usable for e-learning in a campus setting (to enhance classical learning by individualizing learning) and for joint/decentralized programmes. Upon course completion, students should be able to decompose and structure a problem, formulate algorithms that solve a given simple problem, and be able to implement these using Python. Successful completion of the course also implies exemption from having to follow the respective module at ITC in context of a Master or MSc course.

2. TEACHING DISTANCE-EDUCATION PROGRAMMING SKILLS: SPECIFIC ISSUES

2.1 Programming and Geo-information Science

The developed course is the starting point for different types of student. Some of them have never had any contact with computer science or a related field yet. Others already know one or two high-level programming languages. Some members of those groups may have a background in mathematics. The course should therefore offer content that is relevant for both groups. This requires a mixed approach: Initially, it requires the basic programming methods and techniques to be taught, programming examples from other disciplines, such as mathematics and geometry, which make it easy to translate the concepts with which the student is already familiar into the new programming concepts. Finally, after successful course participation this rather diverse group of students will achieve the same base level of computational concepts and programming skills to prepare them for more advanced problems in spatial data processing.
The understanding of computational concepts and the ability of implementing algorithms are a fundamental pre-requisite for working in geoinformatics. Programming skills are needed to handle different types of geo-data, convert between formats, and to automate workflows to reduce manual user interaction, time, and cost. Current data volumes in the geosciences and remote sensing are such that they can no longer be handled manually.
The learned programming skills will help students to solve geo-related problems such as information capture, extraction and fusion, data storage and management, spatial data analysis, and data presentation, for instance, by plotting for publishing or providing the data as web services to a larger audience.
Effectively, the challenge of distance education is to compensate the missing face-to-face contact hours, which normally give room for more detailed explanation, discussion and direct questions. The extensive implementation of animated content and interactive learning tools in this course will close this gap, and the modular structure of the course itself allows an individual learning approach in terms of speed and time.

2.2 How to teach problem solving skills?

A substantial portion of problems in the geosciences can be viewed as problems in which geospatial information in one format needs to be transcribed into another format. That transcription process may intend to highlight parts of the contained information, aggregate information, or bring together disparate sources. Some of the problems are common and well-known and have found solutions that are now nicely embedded in GIS or related tools. Examples are topological predicates for vector-based data, overlay operators, and spatial reference transformations of data sets. Other problems are less common, are often of a higher complexity, and do not have standardized solutions built into off-the-shelve systems. We want to equip our students for tackling such problems as well.
Solutions to them require high-level analytic thinking, with the application of computational principles, and then a realization of the solution through appropriate application of one or more tools. The standard tools and techniques include database and GIS functions, image processing, web services, and as a catch-all, general purpose programming. In realizing the solution, the student should develop an appropriate strategy: parts to an overall solution may be well-known sub-problems with posted solutions, while other parts fundamentally are in need for solution still. In both situations, programming can help to realize the solution. In the first, it can be applied to combine partial solutions into an orchestrated complete solution, while in the second, it can be used to script a solution entirely.

As an example, a problem of geocoding a free natural language description of a locality, like the following, is given:
***Paciencia**: A village or farm in São Paulo state, visited by Natterer, 24-25 Mar. 1824: Natterer's itinerary indicates locality is in ne São Paulo, north of Casa Branca [2146/4705 (USBGN)]*.
The, admittedly under-specified, problem is to obtain as precise as possible the lat/long of this locality, with an accompanying indication of spatial accuracy. A strategic solution might involve: (1) natural language parsing, (2) use of a gazetteer service for 'Casa Branca', (3) use of a web feature service for Brazilian states, (4a) use of a gazetteer service for Paciencia, or (4b) use of a freeform internet search engine for it, and (5) application of an advanced spatial predicate to combine the various obtained inputs to list Paciencia candidate localities. This proposed solution is rather a linear pipeline of steps, but in the details a more functional structure would be apparent.

Students should thus learn how to exploit existing functionality, as well how to develop new functionality. Python is as easy a tool for both as one can possibly find. But what is needed first is the ability to think computationally. We cannot emphasize enough that 'knowing how to program' is about more than just the specific syntax of a computer language. It involves computational thinking, an ability to conceptualise solutions, by applying abstractions, in which the notions of functional decomposition, transformation, mathematical semantics, data modelling, modularity, iteration and recursion, generalization, but also decidability, complexity and performance are key metaphors. Computers, as well as programming languages, are merely the tools to turn these abstractions into reality.
We feel the best way to introduce these rich ideas to our students is by presenting them real life examples of their application, and with exercises (at first easier ones) that invite them to apply the principal ideas. We have found that it is

important for students to grow their confidence to a level where they actually exercise abstraction, in the sense that they will feel to have developed a solution even when not a single line of code has been created. This should come with a level of confidence in their own fundamental abilities and cleverness that abstractly the problem has been solved, and what rests is coding that solution. And then, they should realize that this is the only ideal procedure and that while coding their understanding of the problem and its solution will only deepen.
Finally, our institute's educational system, for various reasons, is organized in three-week modules. We have found that for students with no previous exposure to the world of computational problem solving, three weeks constitutes too short an incubation time to digest the materials. This is why the regular course spreads out over several months, and why we advice the distance takers a somewhat similar digestion.

2.3 Didactical principles applied in distance course

DEPS is developed as a self-instructive course. Our choice of a self-instructive distance education course is motivated by the aim to offer students a learning environment that can be accessed in their *own time*, can be processed at *own pace* in *own location*. In DEPS, we apply the same didactical principles used in the existing distance education course - Distance education: Principles of Databases developed by ITC (Ivánová, 2010). ICARE, one of the instructional design models, is the didactical system in use. ICARE stands for Introduce, Connect, Apply, Reflect and Extend. As described in detail in (Ivánová et al., 2008), ICARE is a didactical system that contains various taxonomies, including Gagne's *Nine events of instruction*, Merill's *Component display theory*, and Bloom's *Taxonomy of higher order learning*. ICARE system in DEPS courseware is depicted in Table 1.

ICARE System	DEPS
Introduce	Introduction to the lesson
Connect	Lecture, Demonstration
Apply	Demonstration, Exercise
Reflect	Self-test
Extend	e-Book, Additional links

Table 1 DEPS and its compliance to didactical principles of ICARE system

Figure 1 shows the didactical system used in DEPS. The course consists of three units, every unit contains four lessons. A lesson is the atomic didactical element of DEPS.
All elements used to build a lesson are seen as course components (Ivánová et al., 2008). Although we allow the students to choose their own path through the course components, we suggest an example path through units and within a lesson. We use the study guide for this. However, the course is designed in such a way, that every lesson can be consumed as an independent learning object. The course is embedded in a learning management system (LMS) used at ITC. Every lesson is introduced by a short document – Introduction to the lesson, in which the main objectives of the lesson and components of the lesson are listed. An overview of topics covered by lessons in DEPS is displayed in Table 2.

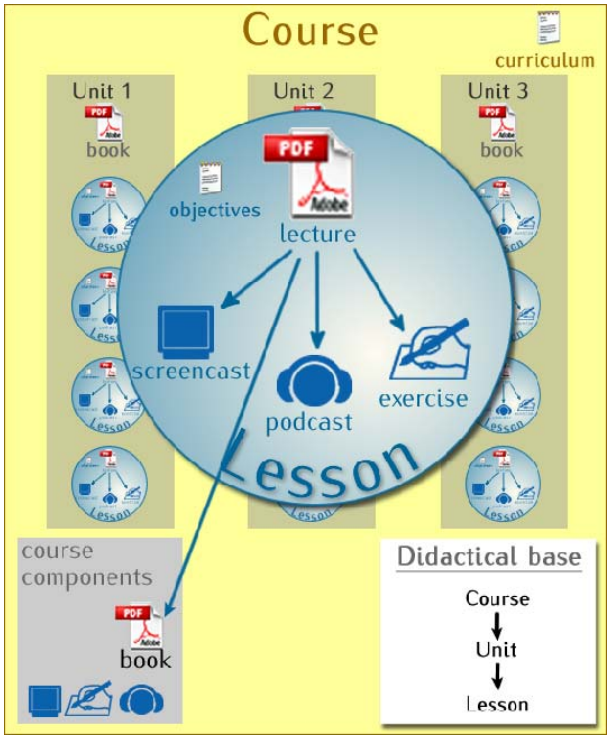

Figure 1 Didactical system of DEPS

.

Unit	Lesson
1 Getting started	1.1 Introduction to programming
	1.2 Variables, expressions and statements
	1.3 Functions
	1.4 Conditionals and recursions
2 Python fundamentals	2.1 Fruitful functions
	2.2 Iterations
	2.3 Strings
	2.4 Lists and tuples
3 Intermediate Python programming	3.1 Dictionaries
	3.2 Files (I/O)
	3.3 Classes and objects
	3.4 Case study – spatial processing

Table 2 Overview of DEPS course content

3. THE COURSEWARE

DEPS courseware is depicted in Figure 2. A solid blue line represents containment – of the course on the USB stick and on ITC's server. A dashed red line represents the communication amongst tutor and students through the LMS.

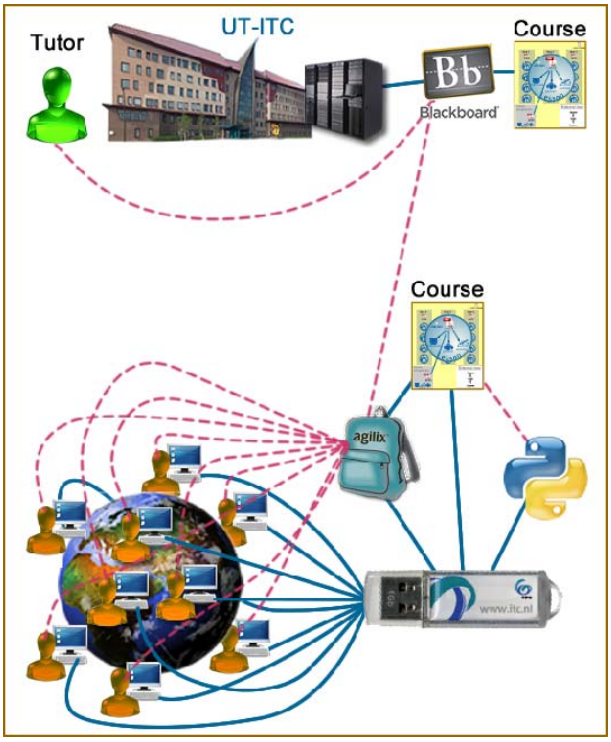

Figure 2 Structure of the course

3.1 Plug-and-Play learning environment

The course material is being developed for plug-and-play deployment on a student's PC, reducing the initial technicalities related to installation and optimization of the work environment. The course will run entirely off a USB stick, requiring no installation and configuration on the student side. Students plug in the USB stick, start up the LMS client. This is a small browser application that acts as user interface to the course, which allows to browse the course content. This LMS client interface also allows students to interact with tutors, by following the discussion board. Since the LMS client contains an automatic update mechanism, students will always access the most up-to-date version of the course (observe the 'communication line' between Blackboard and Agilix in Figure 2). Part of the USB content is a portable Python application used in exercises (see Chapter 3.3). In this way, there is no need for any type of installation. Python installation tips are included as bonus material at the end of the course. Although the nature of studying a programming language is self-study, this plug-and-play solution also allows components of the course to be deployed in a face-to-face set-up, as well as in the distance education mode.

3.2 Lectures with Python flavour

In DEPS lessons, the lecture acts as 'knowledge glue'. The lecture relates ideas presented in other elements of the courseware (e-Book, Demonstration, Exercise, Self-test). We typeset most of our courseware using LaTeX, a high-quality typesetting system based on TEX, developed by Donald Knuth in 1978 (Knuth, 1984). The Python language has a special flavour, by which we mean native Python code typesetting and use of colours in most Python interpreters is catered for by using a special environment in LaTeX, the 'pythonlisting' environment. Authors of the lecture can simply copy code snippets from the interpreter and paste them into the *.tex* source file of the lecture. All it requires is inserting a piece of code (observe the indentation) into the *.tex* file of the lecture such as the following:

```
\begin{pythonlisting}
def squaredPositive(x):
   if x<0:
      print "Number is negative"
      return
   return x**2
\end{pythonlisting}
```

The above example will be typeset as shown on Figure 3.

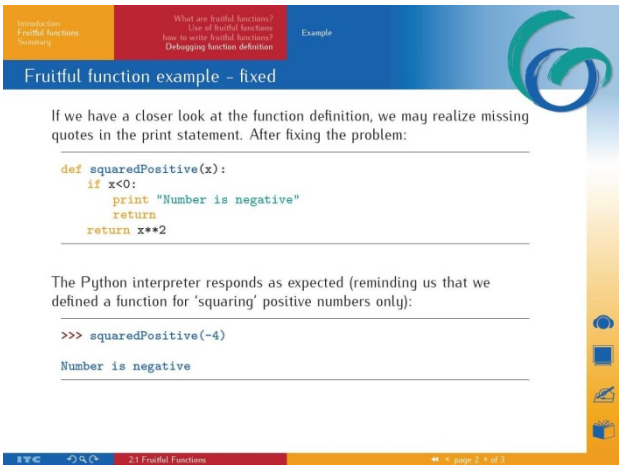

Figure 3 DEPS lecture - the 'knowledge glue'

As mentioned earlier, the DEPS lecture didactical system understands the lecture as 'knowledge glue'. This is exemplified by a multimedia strip on the right margin of the lecture page. Every page of the lecture can point to a podcast (audio file), a screencast (video demonstration), an exercise, and/or a specific location in the book.

3.3 Self-instructive learning of practical programming skills

Teaching practical programming skills is always an area in which self-study and supervised study are demanding priority. The demand for supervision is two-fold. On the one hand asking for assurance while starting to write programs is needed, and on the other hand asking for help to solve a problem when programs do not run is required. The traditional approach to distance courses often used practical assignments to check student progress in applying the knowledge explained by the lecture. Staff processing of assignments and providing feedback requires huge time investments. In a self-instructive distance education course, the main challenge is to *embed* the supervisor into the courseware. In other words, to enhance the application tools (i.e., exercises) with a capability to provide feedback that is timely and to-the-point. Embedding extensive feedback directly to the courseware allows tutors of the distance course to provide only additional clarification and explanation.
In DEPS, to teach practical programming skills in Python we use a Python application called *Crunchy*. Crunchy is a program, developed by André Roberge in 2007. It takes a Python tutorial and turns it into an interactive session within a normal browser. Conventional text is mixed with snippets of Python code and command lines or complete text editors that can be used to execute a single line or even multiple lines of Python code (see Figure 4).

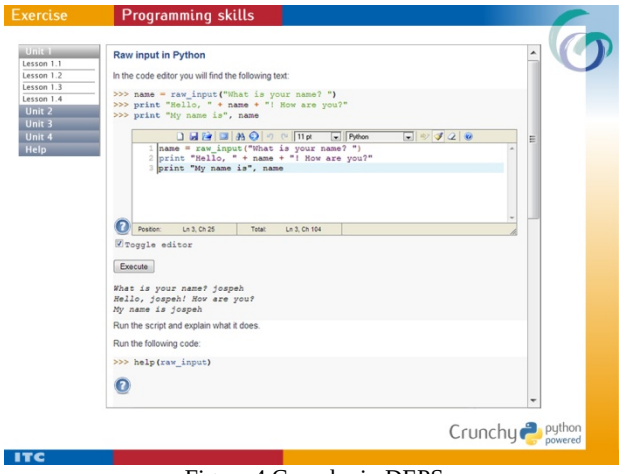

Figure 4 Crunchy in DEPS

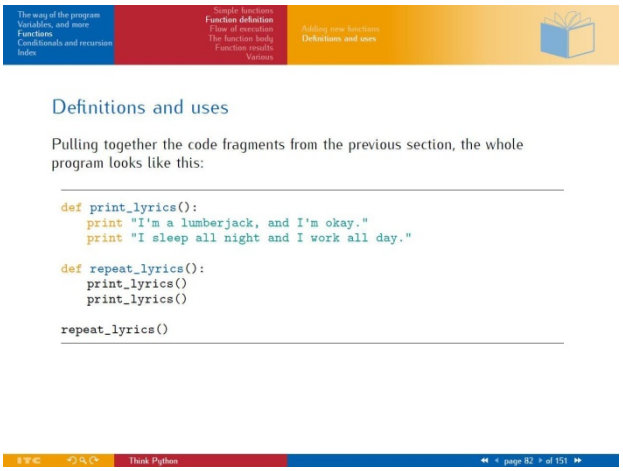

Figure 5 DEPS e-Book

Crunchy installs itself as a wrapper around the browser, translates normal html pages into attractive and interactive pages using a bit of JavaScript. Any Python code is executed by a regular Python interpreter and Crunchy catches its output and puts it inside the browser page. In this way, the student experiences no difference between interacting with a normal Python interpreter and interacting with Python through Crunchy. Crunchy offers an attractive editor for Python code with syntax high-lighting. Python's help system and *docstrings* can be accessed in various ways. The code that is entered by the student can be checked with a method that is used in Python known as *doctest*. The *doctest* can be used to check the correctness of a piece of Python code. For instance, if the task is to build a function for squaring numbers then the *doctest* can test the student's function for a number of predefined, and strategically chosen values to see if the function performs correctly. The current version of Crunchy has an *exam* mode, in which the *doctest* can only be passed if it falls within a certain amount of time. This is useful for building online exams. Other capabilities of Crunchy include dynamic insertion of images, building GUI's in separate windows, and a simple turtle system for programming vector graphics. All this makes Crunchy the ideal tool for building online Python tutorials, exercises and tests.

A minimum of effort is needed to turn a piece of standard text into a controlled interactive session with the Python interpreter. The student, while reading the text, can interactively try out pieces of code or do exercises that can be tested on the fly. Unit tests can be designed to test the knowledge of the student.

3.4 Book

The book used in DEPS presents background reading material to the student of our course on Programming Skills. It was written by Allen Downey, with the help of other authors, especially Jeff Elkner. Allen's original book used Java as the principal language, not Python. That's where Jeff entered. Allen had decided to put the book in the open domain, allowing, amongst others, us to use it for the purpose here. The original book by (Downey, 2002) has copyright information, while text of the book's Preface can be found here: http://www.greenteapress.com/thinkpython/.
Figure 5 illustrates how the original book shows up in DEPS.

3.5 Demonstrations

The main purpose of the demonstration is to expand the theoretical knowledge by showing how to do certain things. Demonstration in DEPS is developed as a narrated screencast. For more technical details, we refer the reader to (Ivánová, 2008). For a visual impression of the DEPS demonstration, see Figure 6.

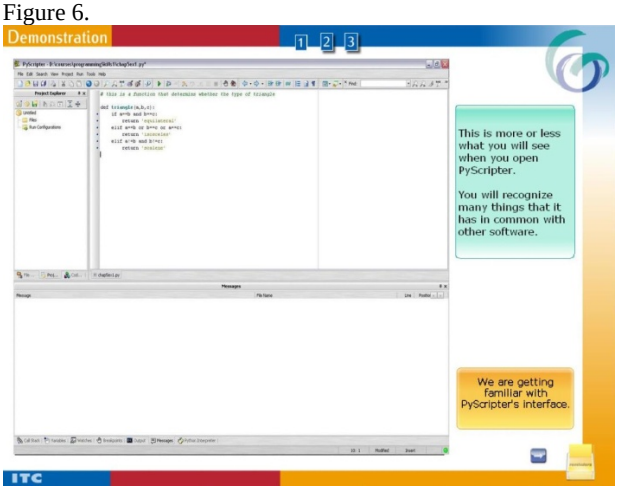

Figure 6 Demonstration in DEPS lesson

3.6 Assessment tools

There are various ways and tools for assessing the knowledge used in DEPS. Every lesson includes one self-test. Self-tests are small Flash applications that allow students to test their knowledge by answering questions in various forms, such as short answer, multiple choice, fill-in-the-blanks and others (see Figure 7).

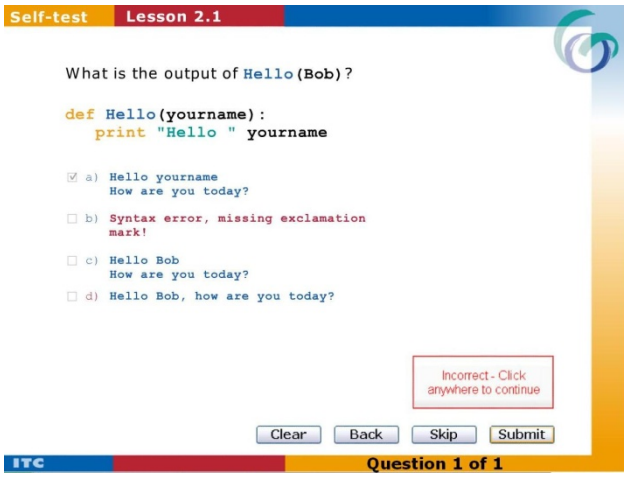

Figure 7 Self-test in DEPS

Some exercises allow a discussion which is also a means to verify student understanding. A discussion is moderated by the tutor. At the end of every unit, there is a unit test. Unit tests are developed in the LMS and results contribute to the student's final score. Tests are available during the whole course and must be completed before the exam, at the end of the course. Table 3 provides an overview of monitoring the assessment tools.

Assessment tool	Monitoring	Contribution to final score
Self-test	NO	-
Unit test	YES	45%
Exercise – programming	PARTLY – consultation provided through discussion board	-
Exercise as discussions	YES	5%
Exam	YES	50%

Table 3 Assessment tools in DEPS

4. USE AND CONTRIBUTION TO OPEN-SOURCE COMMUNITY RESOURCES

Most of the course content — the textbook and some of the exercises — is based on open-source material. This material has been extended to tailor the course to geoinformation science students by adding material in the form of lessons, as well as adding a chapter to the book itself (Case study: spatial processing). We will release the new chapter into the open-source realm through a Creative Commons license. This will be a return of value-added components based upon open content to the community.

The content of the Case study chapter is very much ITC-specific, focusing on the use of Python to process spatial data. The main points discussed in this new chapter address the following questions:

- What is spatial vector data?
- Which Python modules allow working with spatial data?
- How to input spatial data to Python and what type of output spatial data can be produced?
- What type of spatial processing functions can be used?
- How can one develop custom spatial functions?

5. SUMMARY

This paper discusses key aspects of the development of a distance education course on programming skills. There are unique requirements for teaching programming – most noteworthy is to instil in the students a logical approach to problem solving, coupled with improved algorithmic skills. The Python programming language, with its simple syntax and command structure is an ideal environment to facilitate this.

We also note that specific courseware is required to successfully teach a distance education course on programming skills. Most importantly, a didactical system is needed that compensates for the lack of face-to-face contact by providing as much flexibility as possible that maintains simplicity and navigability. Other key elements of the courseware include assessment tools such as exercises, self-tests and exams.

The incorporation of, and contribution to, open-source learning material is an important feature of the course, and fits well with the emerging view of knowledge as a shared resource.

REFERENCES

Downey, A. 2002. How to think like a computer scientist : learning with Python / Allen Downey, Jeffrey Elkner, Chris Meyers. – 1st ed., Green Tea Press, Wellesley, ISBN 0-9716775-0-6

Harel, D.., Feldman, Y., 2004. *Algorithmics—The Spirit of Computing*, third edition. Addison Wesley/Pearson, Harlow, England, pp. 536.

Ivánová, I., Feringa, W.F., de By, R.A., Retsios, V., 2008. Self - instructive course on principles of databases. In: *Sharing good practices : e-learning in surveying, geo - information sciences and land administration, proceedings FIG International Workshop*, Enschede The Netherlands, pp. 35-50.

Ivánová, I., 2010. Principles of databases – distance education course, ITC, Enschede, The Netherlands.

Knuth, D., 1984. the TeXbook, Adison-Wesley, Reading, Massachusetts, 1984, ISBN 0-201-13448-9, 483pp

Roberge, A., 2007. Introduction to Crunchy, Series of screencasts at showmedo.com, http://showmedo.com/videos/video?name=1430000&fromSeriesID=143 (accessed 13. April 2010)

Roberge, A., Woolard, J., 2010. Python Crunchy: Interactive Python tutorials served through a web browser. http://crunchy.sourceforge.net (accessed 13. April 2010)

Snyder, L., 2006. *Fluency with Information Technology: Skills, Concepts and Capabilities*, second edition. Addison Wesley, Boston, pp. 784.

Wing, J., M., 2006. *Computational Thinking*, Communications of the ACM, Vol. 49, 3, pp. 33–35.