Teaching an Introductory Programming Course For Non-Majors Using Python

Jeff Rufinus
rufinus@cs.widener.edu

Y. Kortsarts
yanako@cs.widener.edu
Computer Science Department, Widener University
1 University Place, Chester, PA 19013, USA

Abstract

In this paper we present an innovative approach to teaching an introductory programming course for non-majors using the Python programming language. Lecture structure and suggestions of topics (course outline) on developing and designing the course are briefly presented. This teaching approach could be easily adapted to teach introductory programming courses to majors, including Information Systems majors.

Keywords: pedagogy, innovative teaching approach, teaching tips, Python programming language

1. INTRODUCTION AND MOTIVATION

The IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems (Gorgone, 2002) formulates several characteristics of the IS profession that are integrated into the curriculum. One of these characteristics is "IS professionals must have strong analytical and critical thinking skills." To achieve the necessary levels of algorithmic and computational capabilities for strong analytical and critical thinking skills, it is essential to educate students in computation and computational techniques as early as possible, perhaps by the end of their first year. Such practices have been developed and adapted by many Computer Science / Computer Information Systems departments at many universities in the US, where introductory courses have been developed and offered for Information Systems majors as well as for students specializing in other fields (non-majors).

The difficulties of teaching any programming languages (C, C++, etc.) in introductory courses, however, have been widely recognized. By introductory courses we mean courses given for the 'beginners' (majors and non-majors) with no or minimal background in programming. The difficulty lies in the fact that we have to equip these students with the analytical and critical thinking skills that could benefit them while at the same time we have to introduce the whole concept of a programming language. As a result, many innovative teaching methods were developed.

In this paper we present an innovative teaching approach for an introductory programming course using Python programming language that has been developed in our department. This is a pilot

teaching project that has been tested with non-Information Systems/Computer Science majors. The results of this approach (within a reasonable period of time) have proven to be effective and above satisfactory. The evidences come from the overall average grade of the class, evaluation forms and students' feedback, and the increasing number of students taking the course. We believe that the effectiveness of this approach will universally apply for other introductory courses, whether they are for majors or for non-majors. With no and/or slight modifications (e.g. the Python programming language could be substituted with other programming languages), the approach presented here could be easily adapted to teach other introductory courses including those of Information Systems / Computer Science. As a matter of fact, we are now applying this approach into teaching the introductory data structures and algorithms courses (with Java language) for our Information Systems sophomores.

As a brief introduction, our department has an undergraduate program leading to Bachelor of Science degrees in both Computer Information Systems (CIS) and Computer Science (CS). A variety of courses for majors (CIS/CS) and non-majors (non CIS/CS) is currently offered by the department. Several years ago, the department began to offer a course on programming language for non-majors. The curricular need was to educate non-majors with computational techniques and programming capability; this need was fulfilled through the introduction of a programming language and real life applications. The course that we teach is open to all non-majors and the population of the students is always different each semester. For the classes that consist of 85% to 90% science students, the computational approach worked very well, but for the classes with a high percentage of English and other liberal arts majors we found out (through general course evaluation) some topics (e.g. the text and string manipulation, multimedia) were the most interesting and exciting for the students.

2. CHALLENGES

The development of this course was a challenging task for our department for several reasons. First, the students who would be taking this course had never been exposed to computer programming languages or to computer programming techniques. Second, the students who would take this course would come from diverse disciplines (mostly science majors), some with good mathematical background and some without. Third, the programming language to be used in this course had to follow the "current" trend in computer science, including the introduction of object-oriented method. Of course, there are varieties of programming languages available, from C, C++, to Java, but the question was which one would be easy enough for the students to learn while at the same time using it for problem solving. In the first several years after being introduced, this course was taught using C programming language. From our own experience in teaching this course (based on students' evaluations, etc.) we found that the C language, even though it is a very powerful language, is not a suitable language to be introduced to the non-majors with no background and experience in programming.

3. WHY PYTHON?

Currently the course is being taught using the Python programming language (Python website, 2004). The interesting question is why Python? There are several reasons for choosing this language (e.g. "free" open source, portability, object oriented capable, there are many new textbooks on Python available in the market (Deitel, 2004) (Lutz, 2003) (Mertz, 2003) (Zelle, 2004), etc.), but the most important reason is Python is a simple but powerful language to learn. This is a fair statement to write from our own experience in teaching Python to our students in the last several years. The simplicity of the Python programming language allows for concentration on the programming and problem solving techniques rather than on syntax and complex language structures.

4. PEDAGOGICAL APPROACH

Teaching programming techniques for non-major students, even in an introductory course, is a challenging task. The conventional teaching approach (Cohen, 1989) consisting of long lectures plus homework, quizzes, and exams, is not a very good approach for teaching

introductory computer programming techniques for non-major undergraduate students. There are several reasons for this argument. First, the students have diverse backgrounds. Thus, much lecture time is needed to cover basic technical concepts. Second, a majority of students dislike the 50-minute lecture format, meaning they will not be attentive and learn effectively. Third, homework (with exercises, programming assignments, problems, etc.) alone is not enough to build a genuine understanding of the materials. The fact is that students always use the "pattern matching" technique when they try to solve problems in the homework. (Hanson, 2000) (Hanson, 2004)

Realizing all of these obstacles, we have modified our lecture style to use in-class activities as a way to build the students' understanding of the materials. This approach tries to generate the students' capabilities of learning through a research-based process. After the mini lectures we give the students class assignments for individual or team completion. The assignments themselves consist of several different exercises. We have to find very challenging exercises that will help build their understanding and knowledge on the subject. To encourage the students to come to class we always give points for these activities. Overall, this pedagogical approach changes the role of a teacher from "sage on the stage" to "guide on the side" supporting our main teaching objective: to let the students build their own understanding of the materials through challenging problems, exercises and in-class activities.

After doing all these modifications to our in-class teaching method, we noticed some changes in the performance of our students. First, the students seem to like our method of teaching (data come from students' evaluations and/or through private communications with the students). Second, the students spend more time working during class time (some students commented to add more class time). Third, the students have more confidence and better average grades on tests and quizzes. Overall, the students performed much better than those in previous years, as manifested from their average final grades.

5. COURSE OBJECTIVES AND GOALS

Our teaching approach realizes the following needs: (1) to provide students with sufficient knowledge in computer organization (software and hardware); (2) to provide students with independent and applied knowledge of the Python programming language; (3) to provide students with assignments and examples that will help them develop critical thinking and problem solving techniques; and (4) to provide assignments and projects that are related to a variety of specific academic disciplines. The course takes into account the very limited background that students will have in the subject.

Our pedagogical approach meets the following course objectives:

1. To teach students the basic concepts of the Python programming language
2. To teach students the basic concepts of the programming cycle
3. To teach students to develop an understanding of each of the following concepts: abstraction, debugging and program correctness, functions and objects, recursion, efficiency, reusability.
4. To teach students to develop logical and critical reasoning and problem solving techniques
5. To teach students to develop team and collaborative skills
6. To teach students to develop write and read communication skills
7. To teach students to be able to modify existing software to adapt for specific problems
8. To teach students to apply the knowledge of the programming language and programming techniques to their own discipline

6. LECTURE STRUCTURE

To support our pedagogical approach the following lecture structure was developed and tested at our department and received very positive student feedback:

Lecture Features:

- The simplicity of the Python programming language allows for concentration on the programming and problem solving techniques rather than on syntax and complex language structures.

- Logical reasoning, as the most important problem solving strategy, is emphasized through theoretical and practical materials.
- Each weekly lesson is comprised of three lectures with theoretical materials, lecture programming examples, a laboratory part, ideas for the team assignments, and home programming assignments with sample solutions. The homework assignments for each lesson are divided into sections according to the specific topic and to the complexity level. The materials support the typical lecture structure that was used at our department (see Table 1)
- Each lesson includes a "problem solving" section that relates to a specific problem solving strategy that is relevant to the specific week of study. Special attention is devoted to analyzing and translating the word problems that can be solved with a computer and specific examples are presented.
- If time permits, advanced material related to scientific computing using python, and bio-python with bio-informatics examples, can be presented.
- The lecture practice component emphasizes the problem solving techniques in the software development cycle. The programming examples are based on real world problems and motivate students to learn in order to solve practical problems. The lecture practice component emphasizes an active learning approach. Special solution templates are designed to involve students into the active learning process.
- The program testing procedure is explained in detail for each example.
- The lecture practice component includes examples from the different areas to answer the needs of the diversity of the students in the course.
- The laboratory component includes teamwork assignments that can be implemented during additional laboratory sessions and/or during the student practice session at the end of the lecture.

Session	Purpose	% of time (total = 100%)
Warm-up	Multiple-choice quizzes are given (written or verbal) to review material that was covered in the previous lecture.	10%
Theory	Explanation of new material is presented.	30%
Lecture practice	Examples are solved and explained in detail with active student participation.	30%
Student practice	Special sets of assignments are designed and proposed for independent work by the students to practice the theoretical material that was explained at the beginning of the lecture.	30%

Table 1. Lecture structure

7. COURSE OUTLINE

- Introduction: Computer – what is inside?
  - This lecture provides a brief introduction to computer organization, operating systems, and programming languages.
- Python environment:
  - This lecture introduces the Python environment and instructions for Python installation.
- First program in Python:
  - This lecture introduces a few simple examples for the first Python programs.
  - The practice part of this lecture includes a detailed explanation about two different modes with which students can work: immediate and edit modes.
  - The approach that is adopted for the first program will not include the function "main" to reduce the complexity of the examples and to reach a maximum understanding of the programming process.
- Explanation of the software development process:
  - In this lecture, the program development cycle is explained in detail.
  - Examples are given at several different complexity levels.
- Basic Python commands and statements:
  - This lecture will introduce input/output operations and assignment statements.
  - It will also introduce two basic data types in Python: numeric and alphanumeric.
- Decision and loop structures:
  - This lecture explores if, if - else, if – elif – else.
  - It also explores while loop and for loop.
- Functions:
  - Functional programming as a basic programming design technique is introduced.
  - Our teaching experience demonstrates that the functional approach is much easier for non-majors to understand than object-oriented approach.
  - The concept of function, however, can be a difficult concept for the liberal arts students to understand. Many liberal arts students do not have the necessary mathematical background and are not familiar with the concept of function in mathematics.
  - The concept of function is introduced using the idea of transferring messages. The integrated teamwork supports this idea.
- Recursive functions:
  - This topic is the most intellectually challenging topic in the course.
  - The theoretical part of the lecture provides a wide range of solved examples of different complexity to introduce the concept.
- Simple data structures such as lists, arrays, and dictionaries:
  - The concept of simple data structures is presented.
  - Simple examples of lists, arrays and dictionaries are given.
- Algorithms:
  - The basics of the design and analysis of algorithms are presented.
  - The focus is on numerical algorithms to motivate science majors and to show the power of the computer as a computational tool for real world problem solving.
- Advanced topics of the course include introduction to object-oriented design and bio-python with bioinformatics examples.

8. CONCLUSIONS

It is a very challenging task to teach an "Introduction to Programming" course for majors and non-majors. In the case of

non-majors, several reasons for the challenge should be mentioned: First, the students who would be taking this course have possibly never been exposed to computer programming languages or to computer programming techniques. Second, the students who would take this course come from diverse disciplines (mostly science majors), some with good mathematical background and some without. Third, the programming language to be used in this course has to follow the "current" trend in computer science, including the introduction of object-oriented method, yet this language should be simple and powerful. Our own experience demonstrates that the Python programming language is a good choice for these beginners because it is a simple language to learn, is powerful, and has object-oriented capability.

We developed an innovative teaching method that has been tested successfully in our institution. Our experience demonstrates that students could learn more from in-class activities than long lectures. In a 50-minute class we offer a mini-lecture followed by a variety of in-class activities. The in-class activities are always challenging and specially built in the form of research-based guided process. These activities are mostly done in a team even though individual assignments are also given. Overall, this pedagogical approach changes the role of a teacher from "sage on the stage" to "guide on the side."

The approach presented here could be adapted to teach other introductory courses, including those for IS majors.

9. REFERENCES

Cohen, D. K. (1989) "Contributing to Educational Change", Philip W. Jackson, Editor. McCutchan: Berkeley, CA.

Deitel, Harvey M., Paul J. Deitel, Jonathan P. Liperi, Ben Wiedermann (2004) Python: How to Program. Prentice-Hall.

Gorgone, John T., Gordon B. Davis, Joseph S. Valacich, Heikki Topi, David L. Feinstein, Herbert E. Longenecker, Jr. (2002) "IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems." Association for Information Systems.

Hanson, D., and T. Wolfskill (2000) "Process Workshops – A new model for instruction", Journal of Chemical Education 77, 120-129.

Hanson, D., and T. Wolfskill (2004) Personal communications and Discussions during NSF Chautauqua Workshop, SUNY-Manhattan, NY.

Lutz, M., and D. Ascher (2003) Learning Python, 2nd edition. O. Reilly.

Mertz, D. (2003) Text Processing in Python. Addison-Wesley.

Python websites (2005) http://www.python.org and http:// www.biopython.org.

Zelle, John M. (2004) Python Programming: An Introduction to Computer Science. Franklin Beedle & Associates.