The joy of teaching Theoretical Computer Science.

Christine Choppy (Université Paris XIII, France)

> "No man can reveal to you aught but that which already lies half asleep in the dawning of your knowledge ... And he who is versed in the science of numbers can tell of the regions of weight and measure, but he cannot conduct you thither. For the vision of one man lends not its wings to another man."
>
> The prophet Khalil Gibran

Reading this may be both comforting and hopeless, and may lead us both to some modesty as regards our impact on what students learn from us, and to the feeling that sometimes our efforts may not be fruitful ...

Teaching is for me a source of various feelings that include joy stemming from various ways. The joy of creativity as regards the way to present and explain the technical content, trying to provide enlightening explanations, and exciting motivations. Interacting with students may also be a source of joy, since quite a lot seem to be happy to see you, smile and say hello, make unexpected and funny remarks, look interested in what is taught to them. After some years, there is also this feeling that we also teach other things, in a more subtle way, that they capture from the way we behave, from the kind of human being we are. This is not computer science per se, but I do believe that this is very important.

Rinus Plasmeijer (University of Nijmegen, The Netherlands)

I can imagine that it is not always a joy to teach Theoretical Computer Science. When I teach, it is a joy, because the theory I teach is always related to some practical problem. That makes things more interesting for most students. That is the reason why I like functional programming so much. But in research as in teaching. Functional programming languages have a very solid theoretical basis. This basis can be used for making implementations of compilers (think of type theory, strictness analysis, program transformations), for making applications with functional languages using the properties of the language (e.g. any expression can be computed in parallel without doing harm), for reasoning about the program (by hand or by using proof systems). It is very nice to see how many theoretical notions can be used in this setting. For me as a researcher it is nice to see how

the theoretical framework we started with in 1984 (term graph rewriting) is used years later inside a huge piece of software used by industry for making serious applications (radar equipment) and not so serious applications (games). So, for me it is always easy to illustrate why theory is important such that teaching theory is a joy both for me as for the students.

Hans Jürgen Schneider (Universität Erlangen-Nürnberg, Germany)

Entering mathematics in 1956, I enjoyed the constructive approach of Bourbakism that proceeds from simple structures to more complicated ones by applying a small set of design principles again and again. Computer science is even more suited to be taught in such a way since it is constructive by its objectives: Its ultimate goal is to build tools of some kind or other, may be in hardware or in software.

A usual programming course, however, is mainly concerned with presenting syntactic sugar and semantic tricks to solve more or less small, standardized problems. In the end, the students know a lot of details how to make a computer run, and at the best, they are familiar with good design practice and some software-engineering tools. But, do they understand what computer science is and what it can bring about? To remedy this, they need not only be taught the theory, but also be supplied with the relationship between fundamentals and practice.

The main task of both programming and theory is making definitions, proving properties of the objects introduced by the definitions, and taking advantage of these properties. We can derive profit from this analogy by teaching theory from a programmer's point of view. A topic which is very closely related to programming, and at the same time is at the heart of theoretical computer science, is computability theory. Therefore, it is well-suited as an entrance to theory. Even freshmen perceive the clarity of Kleene's theory of recursive functions if the definitions are translated one-to-one into higher-order functions of about five lines using some LISP-dialect. The students can then apply these functions to create computable functions one after the other, make them run and can convince themselves that the theory works. It is nice to see that even hackers can be tempted into studying the theory of recursive functions, efficiency problems, denotational semantics, etc., if they can do this sitting in front of their beloved computers. I have never understood why lectures in theoretical computer science must be difficult to follow and restricted to using paper and pencil.