Kelly Rivers
PhD Candidate
HCII, School of Computer Science

TEACHING & LEARNING
SUMMIT
2016

Ken Koedinger
Professor
HCII & Psychology

Carnegie Mellon University

ITAP: The Intelligent Teaching Assistant for Programming

Our goal: provide novice programmers with automatic, data-driven, personalized feedback and hints for code-writing problems.

We hope to supplement TA and instructor feedback with always-available assistance, in order to reduce frustration and increase learning in coding practice.

Abstract

Students who are novice programmers frequently need feedback and assistance while they're learning to code, but it is difficult to provide individualized help at scale, especially with growing interest in computer science.

To address this need, we've built ITAP, the Intelligent Teaching Assistant for Programming, which can automatically generate next-step hints for students based on data collected from the work of previous students. We're currently testing ITAP on practice problems in 15-110 and 15-112, the main introductory programming courses at CMU, with the goal of determining whether always-available hints can help improve student learning in the classroom.

We hope to eventually make ITAP generally available, to support teachers in designing personalized instruction and give students more opportunities for individualized learning.

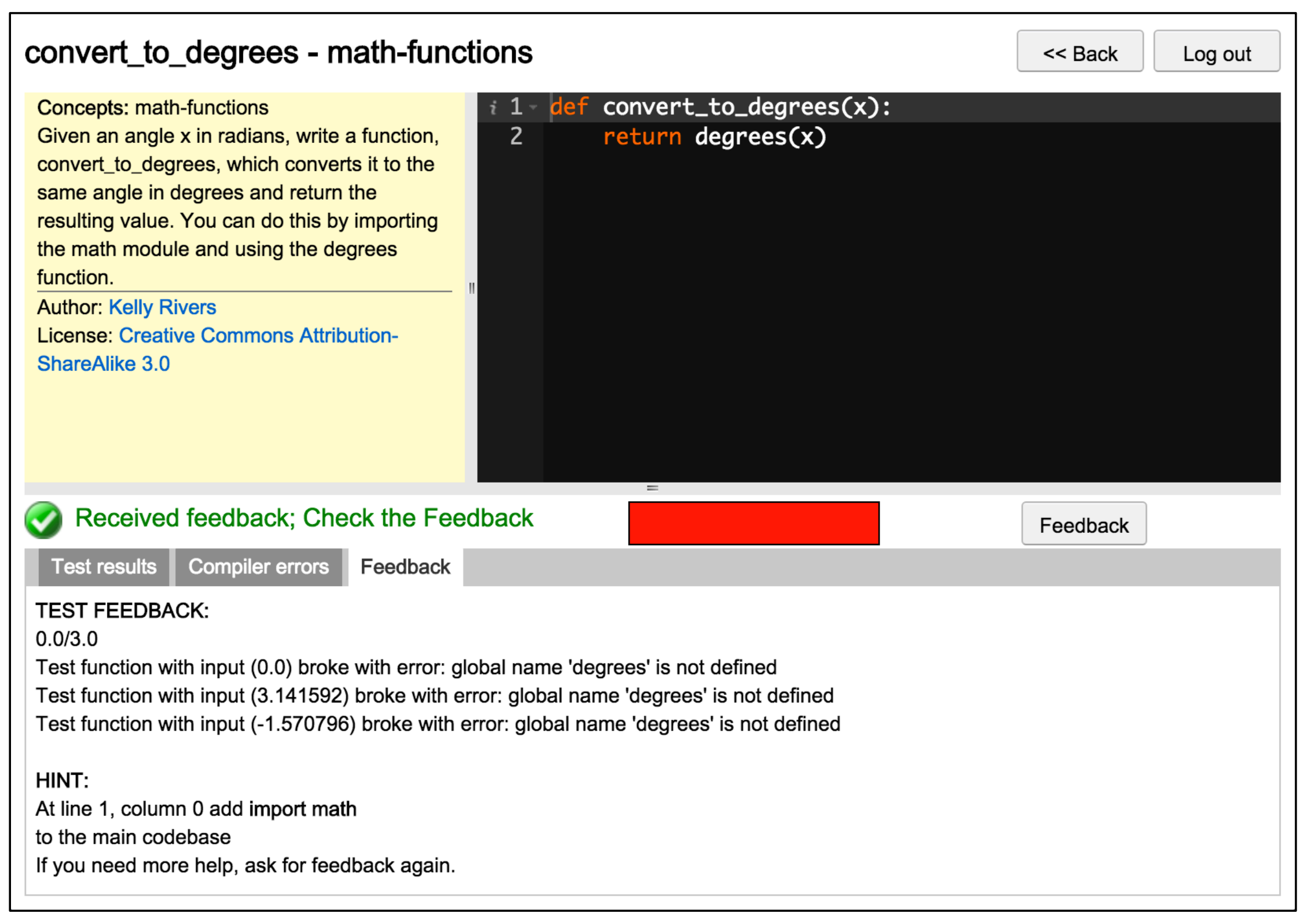
Lessons Learned

1: Instructors may want to give students the minimal amount of help required, but students often want more support, and become frustrated when more detailed help is not provided. Students treat practice problems more like worked examples than independent work.

2: Different students may have different preconceptions about asking for help. In the more advanced classes that we studied, students seemed to dislike the idea of asking for hints, while in the more general classes, students were much more willing to ask for assistance while working. This could be due to different classroom cultures. Also, students like the idea of practicing, but have difficulty finding time to do extra (un-required) practice. Making practice a built-in part of the course may help alleviate this problem.

3: Even when hints provide explicit information on how to correct errors, novices may have difficulty understanding where the change in their program needs to occur. Providing more structured hints may help this.

Project Details



convert_to_degrees - math-functions

Concepts: math-functions
Given an angle x in radians, write a function, convert_to_degrees, which converts it to the same angle in degrees and return the resulting value. You can do this by importing the math module and using the degrees function.

Author: Kelly Rivers
License: Creative Commons Attribution-ShareAlike 3.0

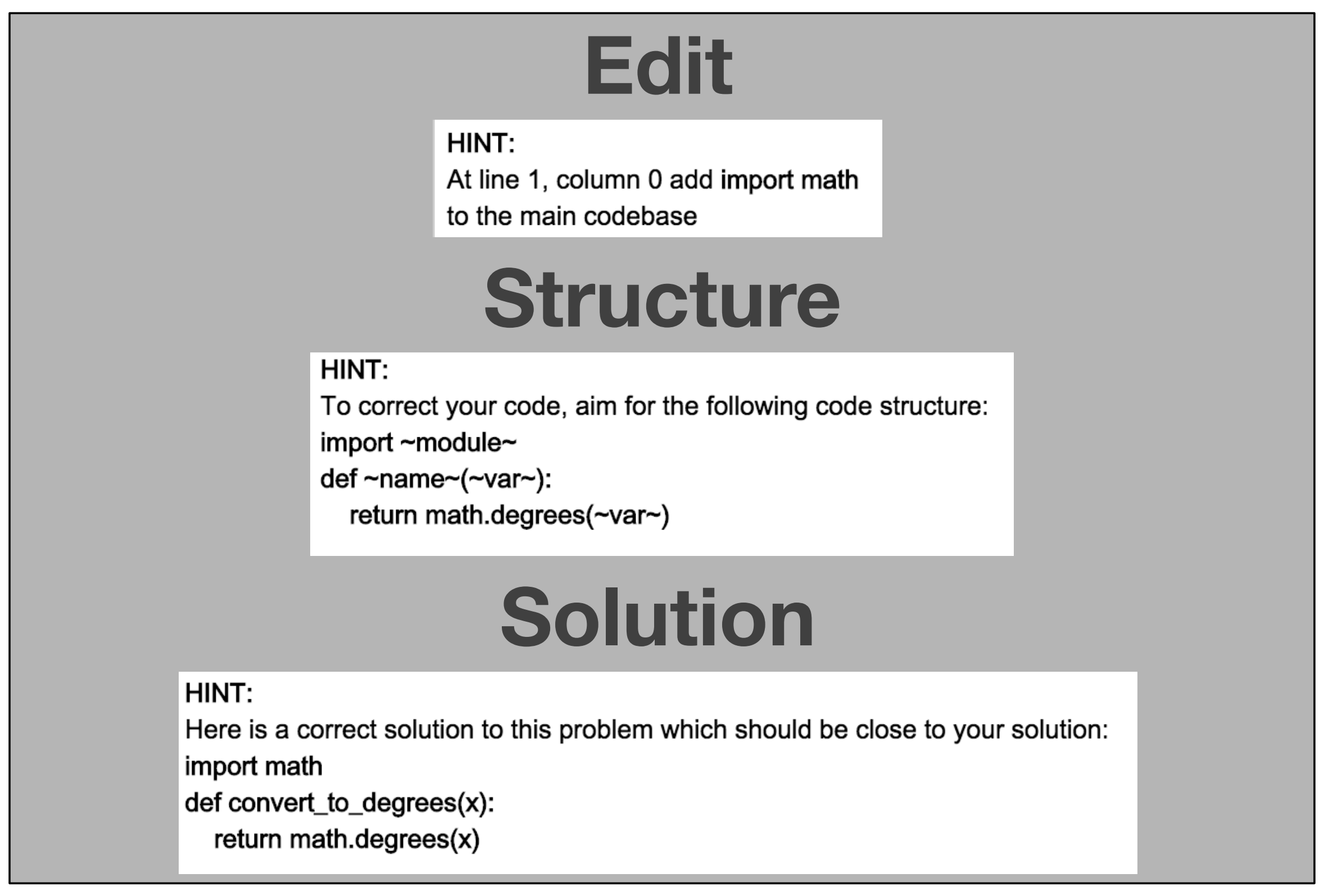
```
def convert_to_degrees(x):
    return degrees(x)
```

Received feedback; Check the Feedback

TEST FEEDBACK:
0.0/3.0
Test function with input (0.0) broke with error: global name 'degrees' is not defined
Test function with input (3.141592) broke with error: global name 'degrees' is not defined
Test function with input (-1.570796) broke with error: global name 'degrees' is not defined

HINT:
At line 1, column 0 add **import math**
to the main codebase
If you need more help, ask for feedback again.

Student View. This is the online system that students interact with when solving programming problems. We use Cloudcoder, an open-source educational IDE, to provide the general interface. ITAP is activated by pressing the **Feedback** button, which sends a request to our system. ITAP processes the student's data, then returns text containing the feedback and personalized hint for the student.



Edit

HINT:
At line 1, column 0 add **import math**
to the main codebase

Structure

HINT:
To correct your code, aim for the following code structure:

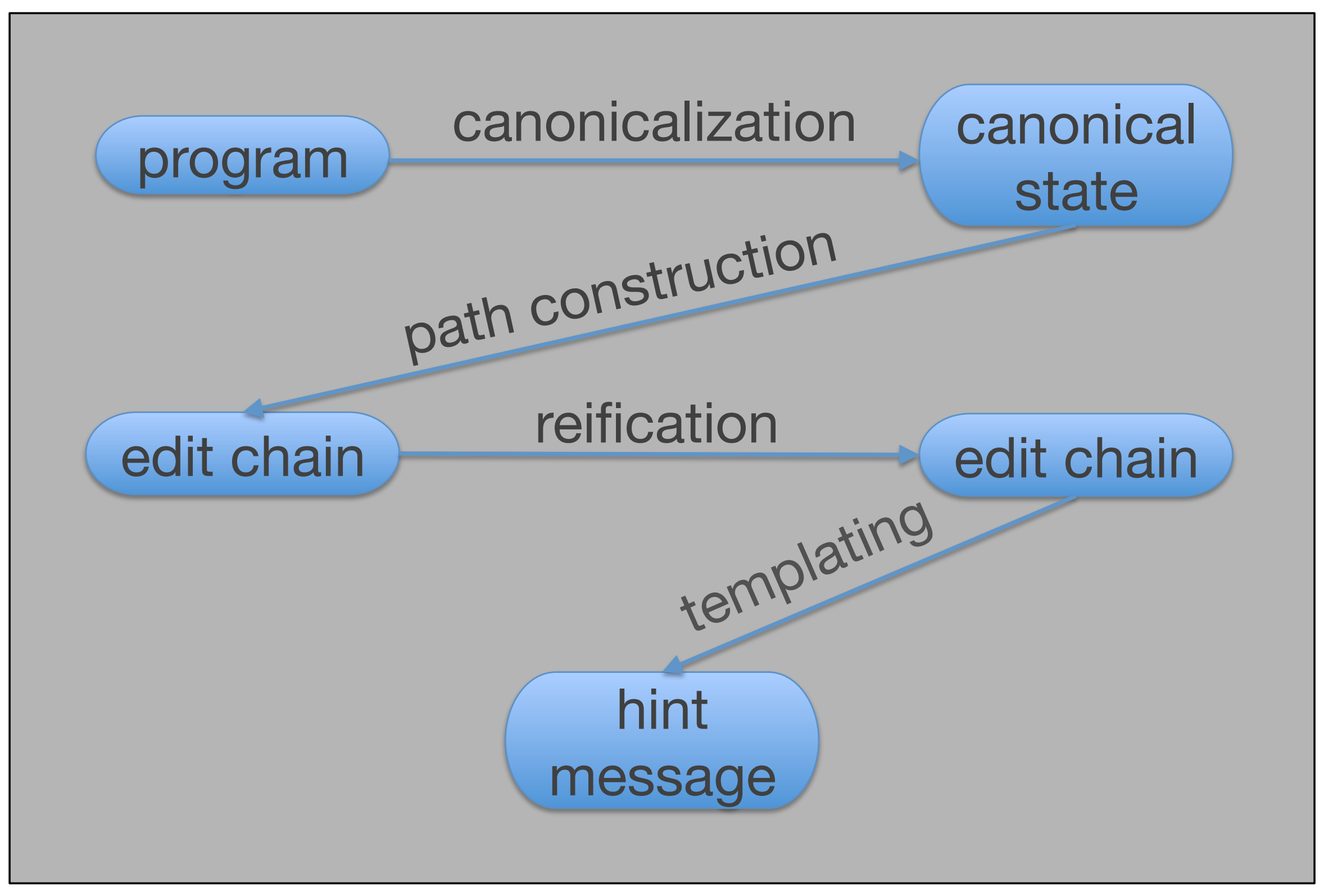
```
import ~module~
def ~name~(~var~):
   return math.degrees(~var~)
```

Solution

HINT:
Here is a correct solution to this problem which should be close to your solution:

```
import math
def convert_to_degrees(x):
   return math.degrees(x)
```

Hint Types. Originally, we only generated *edit* hints (shown at top), which told students how to edit their code to get closer to a working solution. However, usability testing showed us that students did not always interpret these hints as we had wished. Therefore, we used the ITAP process to create two new kinds of hints- *structure* hints (which show structural information while obfuscating low-level details), and *solution* hints (which display the personalized goal state for the student).



ITAP Implementation. ITAP uses a data-driven approach to find the optimal goal state for a student out of a set of solutions that have been seen before. It then determines which edits the student needs to make to turn their code into the correct solution, and translates those edits into hint messages for the student to read.

HCII
Carnegie Mellon

PIER @ Carnegie Mellon
PROGRAM IN INTERDISCIPLINARY EDUCATION RESEARCH

ies