CS 152 Repository Entry
Embedded EthiCS @ Harvard Teaching Lab

Overview

Course:	CS 152: Programming Languages	
Course Level:	Upper-level undergraduate	
Course Description:	"This course is an introduction to the theory, design, and implementation of programming languages. Topics covered in this course include: formal semantics of programming languages (operational, axiomatic, denotational, and translational), type systems, higher-order functions and lambda calculus, laziness, continuations, dynamic types, monads, objects, modules, concurrency, and communication."[1]	
Module Topic:	Designing Usable Programming Languages	
Module Author:	Eliza Wells	
Semesters Taught:	Spring 2021-22	
Tags:	programming languages [CS], justice [phil], egalitarianism [phil]	
Module Overview:	This module focuses on how easy programming languages are to learn and use. In particular, students are introduced to the idea that programming language usability can be a problem of justice. The module introduces an egalitarian framework to help students see that morally irrelevant factors (e.g. vision impairment or not being a native English speaker) impact whether or not some populations are able to access the goods provided by computer science. The module discusses different elements of programming languages, such as documentation and intuitive design, that can impact programming language usability and asks students to think through ways that they can design programming languages so that morally irrelevant factors don't impact usability.	
Connection to Course Material:	Students in this course learn about the formal foundations of programming languages. They are taught about some values that good programming languages realize, such as memory-safety, and ways that they can both test and design for those values. This module considers another value that a good programming language should realize: usability. It encourages students to think about the people who use programming languages, and which people are able to sufficiently take advantage of the technically important features students learn about in the course.	The material in this course is very technical and mathematical. Rather than engage in detail with concepts learned in the course, this module's approach is to zoom out from course material and ask students to think about programming languages in their social context.

Goals

Module Goals:	1. Familiarize students with the philosophical concept of egalitarianism as a way of thinking about justice.

[1] https://groups.seas.harvard.edu/courses/cs152/2022sp/

	2. Explore how programming language usability is a justice issue. 3. Practice thinking through different ways in which students can design programming languages for usability.	
Key Philosophical Questions:	1. Is programming language usability a justice issue? 2. What goods do individuals have access to through computer science? 3. What morally irrelevant traits impact individuals' access to those goods? 4. What responsibility do computer scientists have to make programming languages usable?	This module argues that the answer to question #1 is yes. By thinking through the first three questions, students are prepared to see that the answer to the final question is that computer scientists have a responsibility to ensure where possible that morally irrelevant traits don't impact usability.

Materials

Key Philosophical Concepts:	● Justice ● Egalitarianism ● Morally irrelevant traits	The module focuses on how morally irrelevant traits can impact whether or not some people are able to use different programming languages. Egalitarianism (understood as the claim that, because all people have equal moral standing, different treatment on the basis of morally irrelevant traits is unjust) helps students to see why this is morally problematic.
Assigned Readings:	● Selections from Ryan Long's article on Egalitarianism from the Internet Encyclopedia of Philosophy[2]	This article introduces the basic claims of egalitarianism and discusses different kinds of goods that should be distributed equally under an egalitarian framework – welfare, resources, capabilities, etc. As well as introducing them to the key philosophical topic, this reading prepared students to think about a variety of goods that might be at stake in programming language usability.

Implementation

Class Agenda:	1. Overview. 2. Class discussion of the various goods to which being a computer scientist provides access.

[2] https://iep.utm.edu/egalitarianism/

	3. Egalitarianism as a way of thinking about how those goods should be distributed. 4. Programming language usability as one case where the goods of computer science are distributed unequally. 5. Different ways in which programming languages can be more or less usable. 6. Thinking through usability using case studies of blind and low-vision programmers and non-native English speakers.	
Sample Class Activity:	In small groups, students are asked to discuss: 1. What goods are at stake? 2. What factors limit access to those goods? 3. Are those factors morally relevant? They are then presented with a case study of non-native English speakers who have difficulty using programming languages and asked: 4. What can we do about it?	This activity is an opportunity for students to apply the concepts they've learned to the particular question of programming language usability. When encouraged, students were able to come up with a long list of factors that might limit access to these goods and discuss the nuances of whether those factors (e.g. motivation to learn programming) should be considered morally relevant.
Module Assignment:	Students are asked to respond to the following question on a class thread: Choose a programming language with which you are familiar. Which features of this language make it usable? For whom? Is there a way in which it could be made more usable (e.g. better documentation, more intuitive design, etc.)?	This assignment asks students to take the abstract lessons of the module and apply them concretely to a language they have experience with. In their responses, students were able to bring out how a variety of technical features impact usability. Few of them, however, engaged in much detail with an egalitarian perspective on which populations were less able to use the programming language they discussed. A future version of this question could highlight that dimension more explicitly.
Lessons Learned:	1. Students were engaged with the content of the module and very willing to discuss how big-picture social factors (e.g. access to education, socioeconomic status, values in the community in which one is raised) impacted usability. In the course of the discussion, it became clear that designing programming languages to be more usable is a very small change in a complex system of injustice. Future versions of this module could do more to situate programming language usability within that bigger system. 2. Future versions of this module could do more to highlight the different justice issues brought out by a) not being able to use *any* programming language versus b) not being able to use a *particular* programming language.	

	3. One of the difficult questions of egalitarianism is "What counts as a morally irrelevant trait?" Students were able to bring out interesting edge cases, like motivation to learn a programming language, that may or may not be justice issues. Future modules could lean into this complexity. 4. This module did not have time to discuss the question of how computer scientists can concretely realize their responsibility to make programming languages usable, and what burdens they ought to take on in order to achieve that. Students were interested in this question.	