# **Teaching debugging: assignments and drills**

The idea is to integrate debugging smoothly with other aspects of “reasoning about code”. For each of the four kinds of reasoning about programs shown below,

* We start with instructor-provided guidance (e.g., multiple-choice for a set of provided invariants) and proceed to students figuring out the invariants.
* We go from simple code constructs to more complex code constructs.

1. *Given: correct code. Identify: invariants that hold in the code.*

EXAMPLES:

x = input()

if x < 0:

x = -x

y = 0

for i in range(x):

y += 1

# A

Question: which of the following are always true at point A:

a) x > 0

b) x >= 0

c) x == -x

d) y == x

2. *Given: correct code. Identify: invariants that don’t hold in correct code*

The progression is similar to before. The point is that now there’s a mismatch between the code they’re seeing and the invariants they are looking at, and they need to be able to think in terms of what is not true.

EXAMPLES:

x = input()

if x % 2 != 0:

y = x+1

else:

y = 2\*x

# A

Question: which of the following are not always true at point A:

a) x > 0

b) y > 0

c) y > x

d) y >= x

e) y is an even number

3. *Given: incorrect code together with invariants that should hold (some of them hold and some don’t). Identify the invariants that are supposed to hold but in fact don’t hold, together with inputs for which they don’t hold.*

EXAMPLES:

# check whether the argument x is divisible by 2 or 3 or 5

def divisible\_by\_2\_or\_3\_or\_5(x):

y = x % 2

### desired invariant A: y is 0 iff x is divisible by 2

z = y % 3

### desired invariant B: z is 0 iff x is divisible by 3

w = x % 5

### desired invariant C: w is 0 iff x is divisible by 5

return (y == 0 and z == 0 and w == 0)

Question: Which of the desired invariants A, B, and C does *not* hold?

4. *Given: incorrect code together with invariants that should hold (some of them hold and some don’t), as well as specific inputs that cause incorrect behavior. Find the code responsible for the problem and fix it so that all the invariants that should hold do, in fact, hold.*

EXAMPLES:

# read a number and check whether it is divisible by 2 or 3 or 5

def divisible\_by\_2\_or\_3\_or\_5(x):

y = x % 2

### desired invariant A: y is 0 iff x is divisible by 2

z = y % 3

### desired invariant B: z is 0 iff x is divisible by 3

w = x % 5

### desired invariant C: w is 0 iff x is divisible by 5

### desired invariant: return True iff x is divisible by 2,

### or by 3, or by 5

return (y == 0 and z == 0 and w == 0)

Question: When invoked with the argument 6, this function returns False, which is not correct. Use this value to identify the problem in the code and make the fewest number of changes to the code shown above so that the desired invariants hold.

ANSWER: Two changes are necessary:

1. the line z = y % 3
2. the Boolean connectives in the expression (y == 0 and z == 0 and w == 0)

Email about this

Ravi, Michelle:

Following up on the committee meeting this morning: here's a link to the document I sent Janalee with some initial thoughts about teaching/assessing debugging as part of a progression of "reasoning about programs" skills.

I'd be curious to hear your thoughts.

--s

---------- Forwarded message ----------  
From: **Saumya Debray** <[debray@cs.arizona.edu](mailto:debray@cs.arizona.edu)>  
Date: Sun, Oct 29, 2017 at 8:00 PM  
Subject: Re: 120 Slides for tomorrow  
To: Janalee O'Bagy <[jobagy@cs.arizona.edu](mailto:jobagy@cs.arizona.edu)>

Janalee,

I've been slowly (time-slicing with various other things) trying to write down some ideas for exercises for teaching debugging -- your slides finally got me to try and get them to a point where I could show them to you.  I've put a draft [here](https://docs.google.com/document/d/1mxTWPlKERJVKV61X_JZMQA_KU0YTQ8ZYhRhhv2civmE/edit?usp=sharing).  Let me know see what you think.

The more I think about this topic, the more it seems to me that debugging should mesh smoothly with other aspects of "reasoning about programs", which is what I've tried to do.  The document I've shared has just one example for each kind of reasoning problem, but in reality of course it would not be so cut and dried.  What I envisioned was something like the following:

* for each of the four kinds of reasoning mentioned, use a series of problems of increasing complexity;
* slowly merge them together, so that after a few weeks students see different kinds of reasoning problems mixed together;
* assign a few such problems every week, so that students get practice from the repetition.

I'd be curious to hear your reaction.

--s