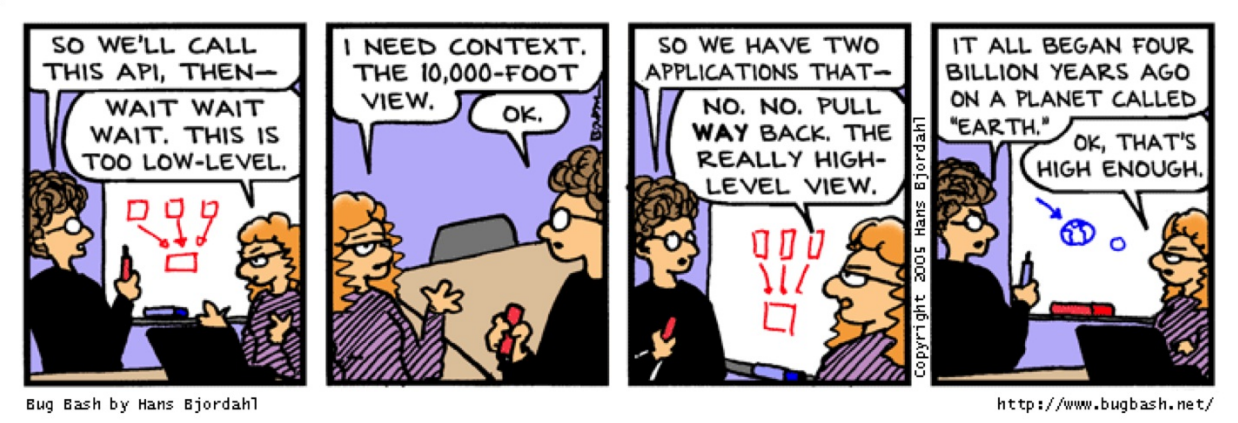
Meeting 22: Objects



Announcements

- HW6 due 11/17
- Project status due 11/17 (in your project repo)

Homework 5

- Time: mean 20.0 hours, stdev 9.7, median 20
- Hardness: mean 5.1, stdev, 0.8, median 5

Congratulations, you've made it!

Comments

- Having a better idea of how the homework assignments go, this assignment went a bit more smoothly. The modules (esp. uniquify and closure conversion) were a bit tricky to code correctly, but I had a better sense of writing good tests and what to watch out for.

  I found this advice given early on this assignment hugely helpful: grow the P2 subset your compiler supports gradually; first only closed functions, then functions with free immutable vars, then immutability. Having a stable end-to-end baseline to work on top of is very nice.

  One painful bug came at the end, when I wasn't quite handling stack references correctly within function calls. These kinds of bugs are super nasty to debug (when they crop up only in final assembly)

- I refactored a bunch, so there was a lot of sunk time in that.

- When we came across errors they didn't really give any indication what actually went wrong, making debugging extremely challenging. For example, we initially forgot to add back to rsp after calling functions, and the errors we got were 1. nondeterministic due to graph coloring sometimes picking callee-save registers and 2. misleading since we'd usually get a segmentation fault with no further description

- Trying to debug where some bad data (like something that should be a list, but is not) came from, especially now with the added indirection of function calls was the hardest part. It may have been easier while still educational to add function calls and closures before polymorphism.

  Overall, I love this course. It has been one of my most challenging and interesting. I would have liked to be exposed to more static AST analysis techniques such as type checking or theory as opposed to focusing on difficult implementation homework assignments like explication, though.

- There are so many circuitous steps that could break. If you catch 95% of bugs in each step during programming, the 5% bugginess per step results in a compiler that almost never compiles anything correctly.

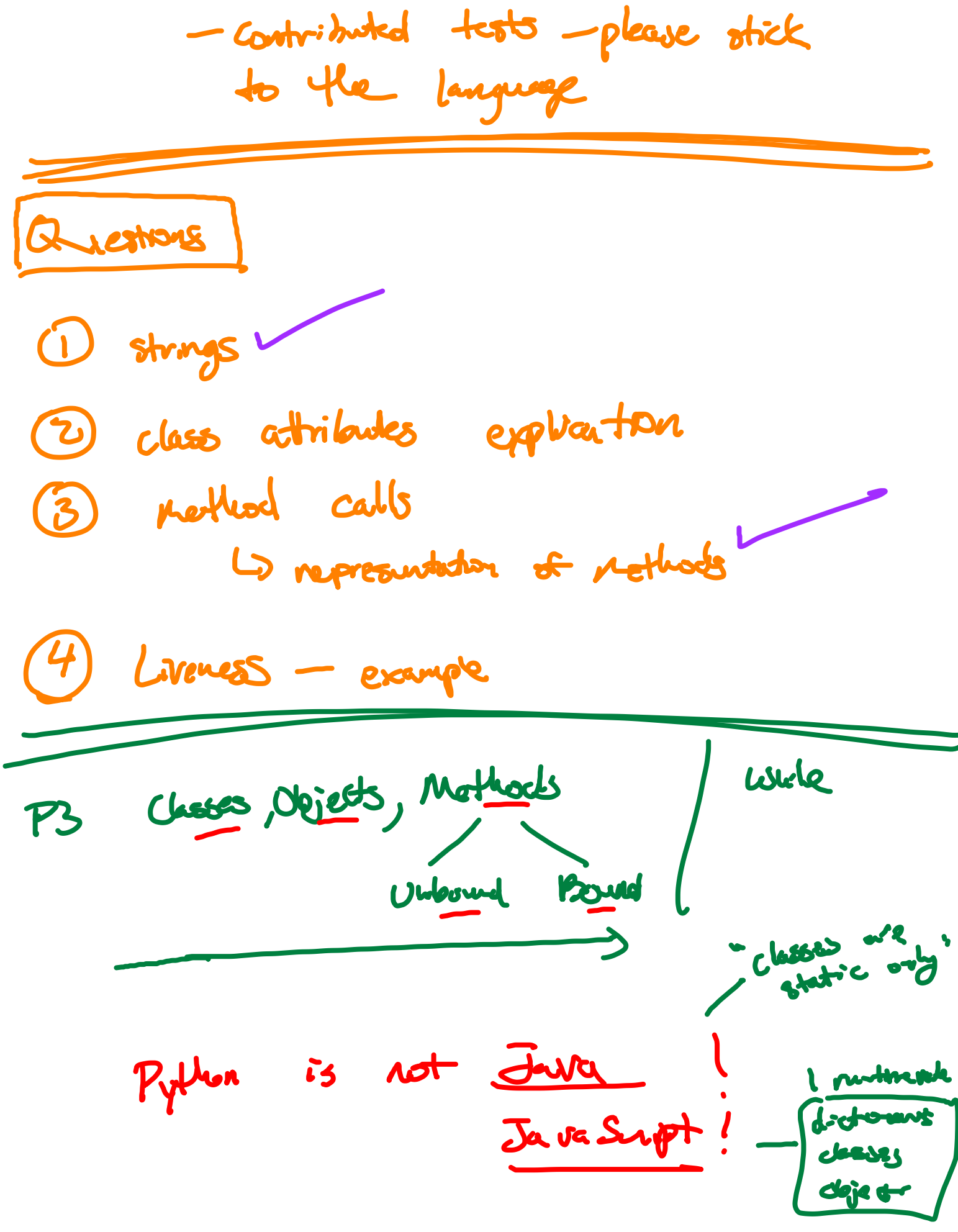
  Python was never designed to be compiled and reasoning about the consistency of the program in the absence of any type information at compile time is impossibly frustrating

- We hadn't fully and successfully implemented HW4. We started to work on HW5 and realized that, for our implementation, we really needed lists which we had never gotten to work. So after making some progress on uniquification, we had to roll back to working on HW4. While we now have lists partially working, we didn't have time to get functions fully up and running. Hopefully we can make some headway on that before moving into objects.

  I think that is the hardest part of the course: if you fall behind, there really isn't a good mechanism to get caught up. If a later homework relies on success on a previous homework it can be daunting to have to go back. Maybe it would be good to highlight the things in each assignment that really must be done in order to complete the next homework. And it might also be good to offer some advice on what things can be ignored or worked around for the current homework. I know that will be somewhat variable depending on our specific implementations, but it can be frustrating to hit a wall and feel that we cannot progress.

- Implementation is once again the most difficult part. It seemed like all the instructor tests are very complex this time, since I have 7 passing personal tests of varying complexity,

yet still only pass 1 of the instructor tests. No matter how many errors I got rid of, I never passed more tests.



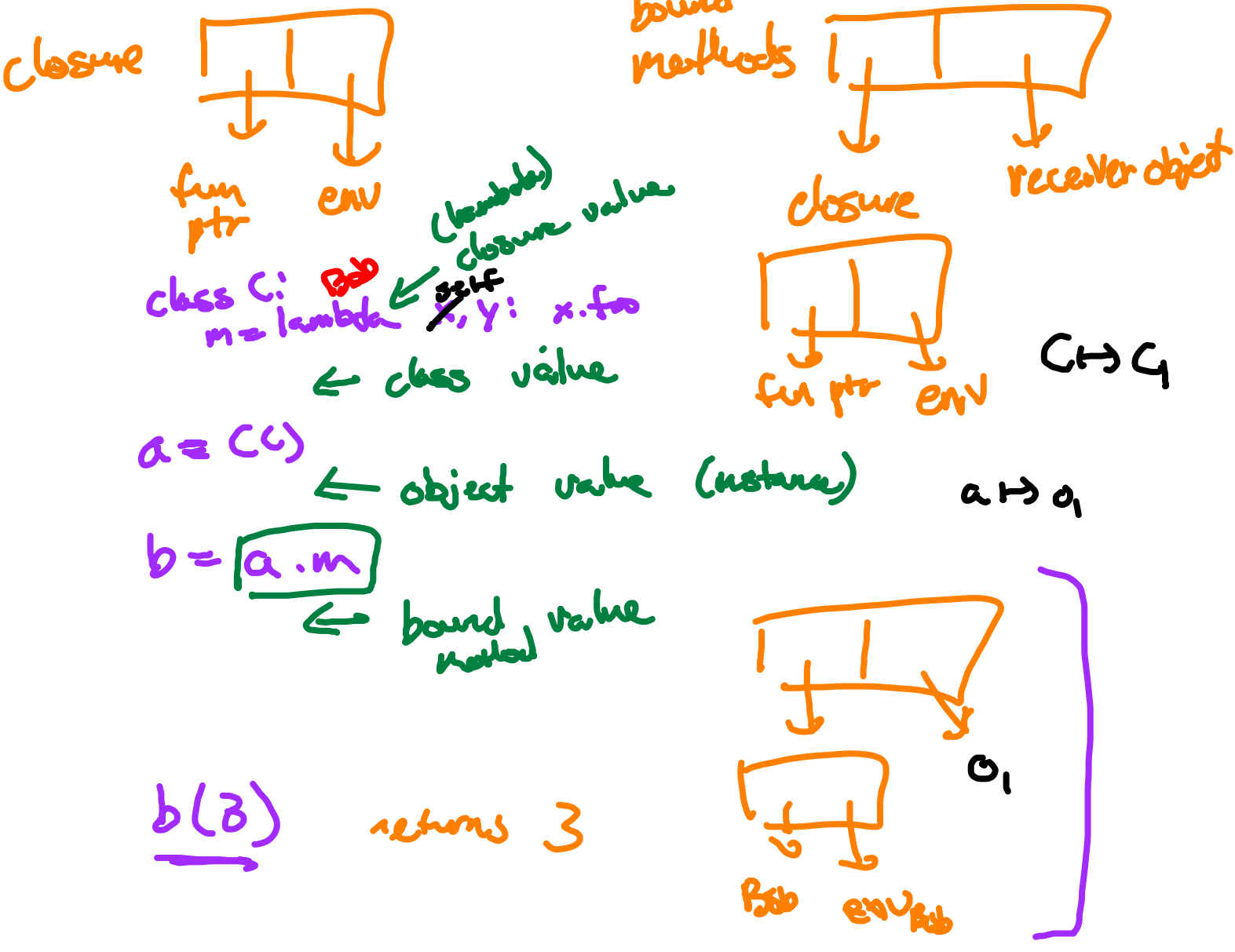
→ Attributes of both classes & objects can be dynamically extended

⇒ classes live at run time

⇕

classes are first-class values

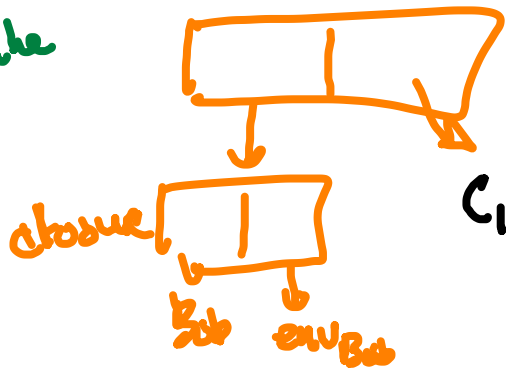
→ methods are first-class



~~d(3)~~ ×

d(a, 3)

~~d(None, 3)~~ ×



Strings

x.f

⇒

get_attr(x, 'f')

source code

f — identifier

↳ 'f' — at x86 at runtime

Static data area of your assembly

.asciz f'attr 'f'

```
class C:
    x = 0
```

← this is a attribute assignment / declaration

```
o = C()
o.f = 10
```

← this is an object attribute