CS61A Lecture 7

Wednesday, September 11th, 2019

Announcements

- Guerilla section is on Saturday, focus on control (`while` and `if` statements), extra practice for midterm.
- Midterm 1: you can bring a magnifying glass!
- Friday will be solving old exam problems.

Design

Functional Abstractions

```
def square(x):
    return mul(x,x)

def sum_squares(x,y):
    return square(x) + square(y)
```

What does `sum_squares` need to know about `square`?

- `square` takes one argument.
- `sum_squares` does **not** need to know its intrinsic name, just what name it has now.
- `square` computes the square of a number.
- It does not need to know how the square is computed.

If the name `square` were bound to a built-in function, `sum_squares` would work identically. Functional abstraction is the process of describing what a function takes in and outputs, but not how it does it.

Choosing Names

This is a practical matter that has no effect on your actual code, but on the people reading your code (composition).

You are giving a gift to yourself by naming stuff well since you often have to reread your code. Names should convey meaning or purpose of the values to which they are bound.

The type of a value bound to the name is best documented in a function's docstring. Function names typically convey their effect (`print`), behavior (`triple`) or the value returned (`abs`).

For example:

- Instead of `true_false`, you could write `rolled_a_one` in Hog.

- Instead of single letters like `d`, write full words like `dice`.
- If you must call a function within another function, it is a technically a helper function. But that doesn't mean you should call helper functions only by `helper`. Instead, for example in Hog, write `take_turn`, which helps the `play` function.
- Don't call values by `my_int` or `my_str`, but what they are in the real word.
- If you must single letters, don't use `l`, `I`, or `O`, but instead letters that are distinct from numbers.
- Which values deserve new names? Anytime you're repeating some big compound expression, it probably deserves a name. It saves you from repeating yourself, and marginally speeds up your program by not having to calculate the expression twice.
- Names can be long if they help document your code.
- Names can be short if they represent generic quantities, such as counts, arguments, arbitrary functions.
- Names typically used in mathematical functions are recommended to be used:
    - n, k, i usually integers
    - x, y, z usually real numbers
    - f, g, h usually functions

Demo: Sounds

Watch video