CS61A Lecture 1

Wednesday, August 28th, 2019

Every live lecture will start with announcements, which will be on the course website. Overflow room for the rest of the semester in the evening, until people stop coming to class. You can come live or watch video lectures on the course website.

Extra OHs throughout this and next week, and big OH session on Friday to set up your computer.

About the Course

The CS61A Community

Most of the teaching is done by the course staff that have been trained over many semesters.

- 57 teaching assistants this semester with drop-in OHs,
- 50 mentors for small group (4 people) optional weekly mentoring sessions, after the first midterm, who also do OHs as well as extra sessions.
- More than 200 academic interns who want to know what teaching the course is like.
- 2000+ students with a few more on the waitlist. Biggest 61A course ever.

Parts of the Course

- Lecture: Videos posted to cs61a.org before each live lecture.
- Lab sections: Sit in front of a computer and practice your coding experience.
- Discussion sections: Sometimes it's time to close the computer and solve problems without using the computer to give you feedback.
- Staff office hours: 1-on-1 interactions with the staff if you're stuck ona specific part or so.

Which one is the most important? It depends on the student. Some people will find different parts of the course more important than others.

- Online course: http://composingprograms.com, read it before you come to lecture
- Weekly homework assignments, 4 larger projects, 2 midterms and a final with dates posted.
- Lots of optional special events to help you complete all the work, sometimes on weekends and evenings.

An Introduction to Computer Science

What is CS?

The study of the most flexible machine ever created - the computer. What problems can be solved? How can we solve those problems? And what techniques are the most effective?

If you ask any CS prof whatg they study, they will always answer with the special subgroup.

- Systems - The study of large systems like the one your computer runs on, or Facebook

- Artificial Intelligence - Do what living things are good at
- Graphics - Cool FX and movies.
- Security - Hide from the world.
- Networking
- Programming Languages
- Theory
- Scientific Computing

Most people who are studying CS work in some specialization of this. For example, within AI, there are branches Decision Making, Robotics, Natural Language Processing, many sub-sub fields. For example, NLP, answering questions, translations, and other sub-sub-sub-fields.

No matter where you are in this vast hierarchy, most computer scientists have a common enemy: complexity. It's hard to think of many programs at once.

This course is about managing complexity, which is a topic called abstraction. Taking some complex thing with many steps, giving it a name, and not worrying about the little details. It's a very important skill in computer science.

What is CS61A about?

61A is also a course about programming paradigms so that all the pieces fit together nicely.

61A is an introduction to actually programming.

- Full understanding of Python fundamentals, of all the details and how they relate to the concepts of computer science.
- Combining multiple ideas into larger projects. So that we experience how to wrestle with complexity and master abstraction.
- How computers interpret programming languages.
- Different types of languages in Scheme and SQL.

At least 12 to 15 hours a week, but manageable if you dedicate the time.

Alternatives to CS61A

61A is not a perfect intro to CS.

- CS10: Beauty and Joy of Computing, gets people excited about computer science. Designed for students who have never programmed before; 61A was not designed for students who have not programmed before. 10 has easy programming languages and Python multimedia projects.
- Data 8: Foundations of Data Science, works in any combination (before, together or after).

Course Policies

- The point of this course is for you to learn the things you haven't already learnt, not to show what you have already learnt.
- Building a community: CS is not about one person sitting in front of a computer and building something themselves. Talk to others about the problems.

- Course staff: here to support you and help you succeed, not to judge you and see if you're cut out for CS. If you're stuck, please ask.

More details on the website.

Collaboration

- Asking questions is highly encouraged
  - Discuss everything with other students and learn from others.
  - Some projects can be completed with a partner
  - Choose partner from your discussion section
- The limits of collaboration
  - One simple rule: Don't share your code, except with your project partner
  - Copying project solutions causes people to fail the course.
  - We do catch people who violayte because:
    - We also know how to search the webs for solutions
    - We use computers to check your work.
- Copying only works in college because the answers are set but this isn't always the case in life.

Expressions

An expression describes a computation and evaluates to a value.

In Python, you type an expression, press return and the computer calculates the value of the expression. Python is a pretty great calculator, you can add, divide, use parentheses.

Types of expressions

1 + 2 is a primitive expression.

This is not exclusive to CS, mathematicians do that too. Over the years, they have invented many methods of expressing computations:

- 3+3
- $\sqrt{111}$
- $\frac{3}{6}$
- $\sum_{n=1}^{3} n^2$
- $f(x)$ - the function call

One of these forms is more general than others: the function call.

As computer scientists, instead of expressing computations with subscripts and vertical bars, we write down everything using the function call notation:

```
operator(operand, operand)
```

Call functions can have operators and operands. Programming languages interpret your expressions by using the following evaluation procedures.

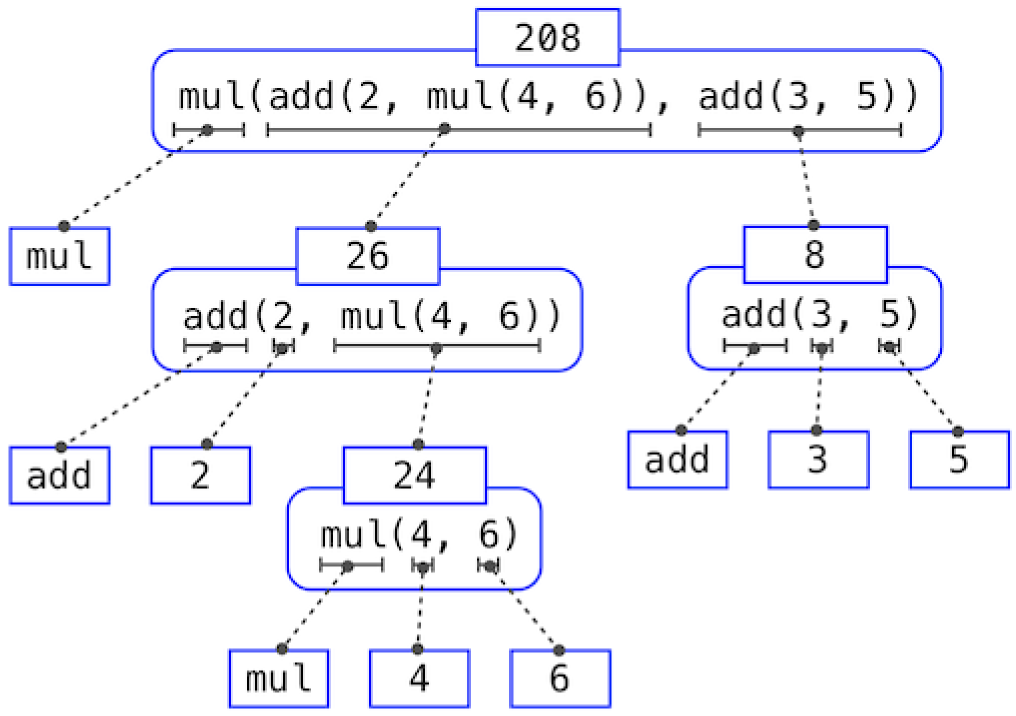
Evaluation procedure:

- Evaluate the operator and the operand subexpressions
- Apply the function that is the value of the operator.

Evaluating nested expressions

Call expressions use parentheses. We always evaluate the inner values first before working our way outwards.

Expression trees help us figure out the way in which the Python interpreter calculates, and these diagrams let us figure out the way in which Python interpreter works.



You begin evaluating the whole expression first, before finishing the evaluating the whole expression last. The process goes evaluate the whole thing, the pieces, then finish the whole thing again to wrap up.

This sort of describes how you evaluate call expressions: the trees grow from the top-down. Evaluating the root, the full expression at the top, requires first evaluating the branches that are its sub-expressions. The leaf expressions (nodes with no branches stemming from them), are either functions or expressions . Viewing the tree, we can imagine working our way upwards, beginning at the terminal nodes until the full expression is evaluated.

Functions, Values, Objects, Interpreters and Data

YouTube

You will not understand all the things done in this demo for weeks to come, but you eventually will as we delve into the ideas of computer science.