Programming & Computational Thinking

Abstraction & Pair Programming

What is programming?

- Who does programming?
- Where do programmers work?
- How do you think programming has changed in the last 5, 10, or 20 years?
- How do you think it will change in the future?

https://youtu.be/dU1xS07N-FA

Computational Thinking

A bit of *a* computing *buzz-word* right now.

It's not going to be *a* term you're likely to define in *a* job interview, but the concepts that you'll learn *as* we learn to do it, those might be...

So what is it?

A process that generalizes a solution to open ended problems. Open-ended problems encourage full, meaningful answers based on multiple variables, which require using decomposition, data representation, generalization, modeling, and algorithms found in Computational Thinking. ~Wikipedia

Key Concepts

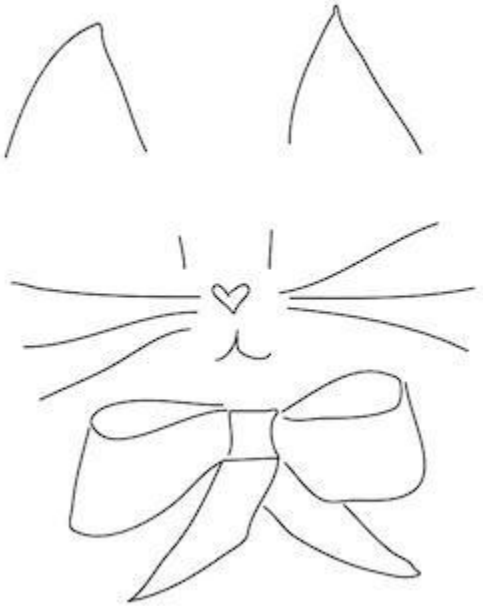
- abstraction
- generalization
- composition & decomposition
- creativity
- data and information
- algorithms

Abstraction

Our first key topic!

What do you think it means?

Meet Bart



Abstract Bart

- My cat sketch is an abstraction.
- It's a cat, but you probably wouldn't know it was Bart and not some other cat.
- The details have been removed.

Can you think of some other examples?

Abstraction in Computing

In computer science, abstraction is a technique for managing complexity of computer systems. It works by establishing a level of complexity on which a person interacts with the system, suppressing the more complex details below the current level.
~Wikipedia

Two areas of focus

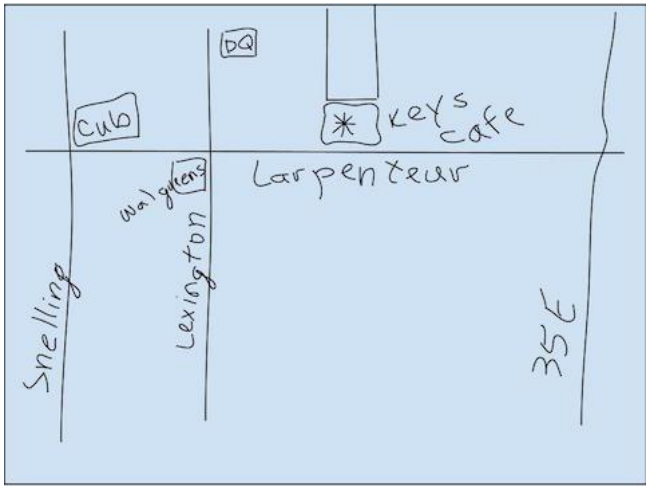
- Data representation
  - Examples: network diagrams, icons on a desktop, dots on a radar
- Systems interactions - interfaces
  - Examples: communication protocols, APIs (application programming interfaces), project development methodologies

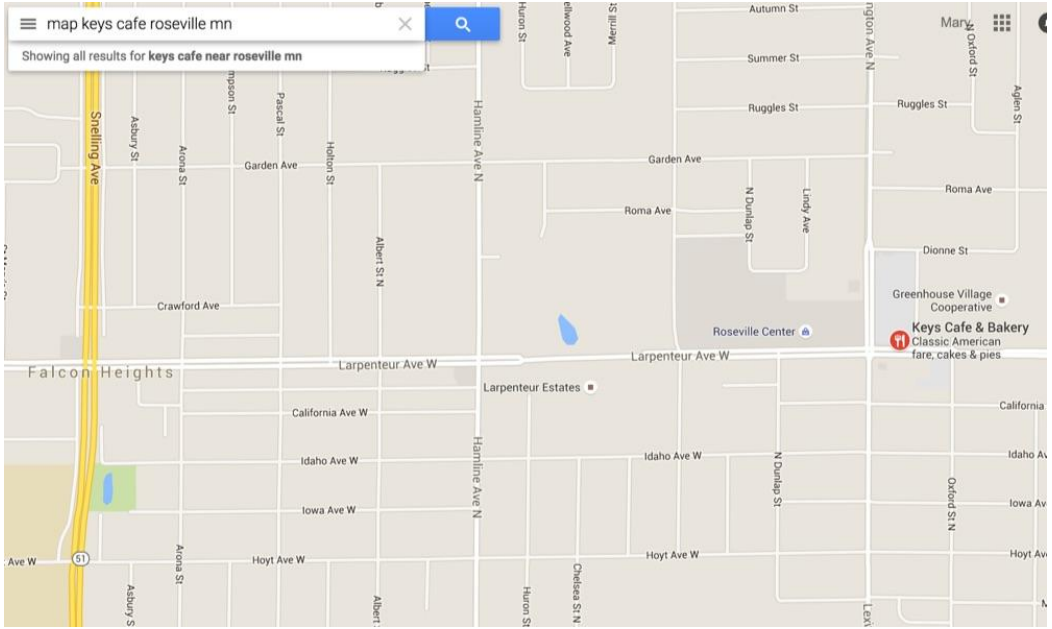
Data Abstraction

Quick exercise! 60 seconds

Draw me a map to your favorite restaurant.

Data Abstraction on Maps





Systems Abstraction

Think about driving a car...

Car Interface

Turn key - start car

Gas Pedal - go

Brake Pedal - stop

Steering Wheel - turn right or left

But what really happens when you do these things? What happens under the hood?

Hiding details is important

Cars have changed a lot over time, but how much has the way we drive them changed?

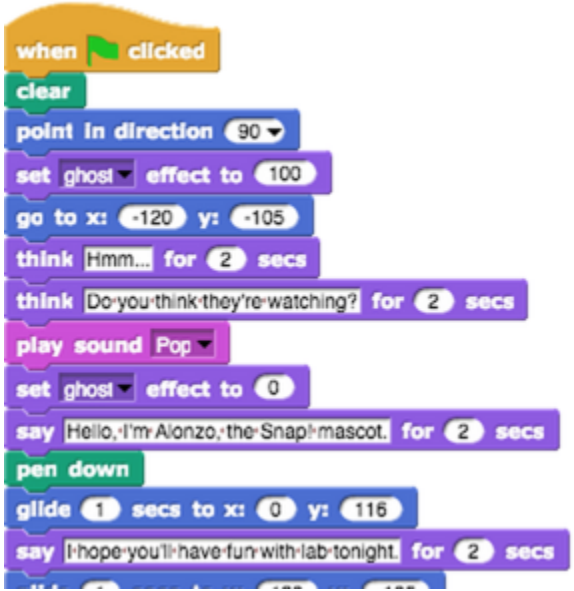
Abstraction allows you to change the underlying details, *the implementation*, without affecting the way the system is used.

Almost time to code...

We'll be learning to code in Snap!

It is a graphical, block-based language from UC Berkeley.

Many of the labs and material we will use is borrowed and adapted from their Beauty and Joy of Computing curriculum.



Why Snap?

- It's easy to learn (Welcome Program)
- Everything is right there waiting for you to use it.
- Because it is interactive, you can do more interesting things.
- It offers fast, visual feedback.
- No syntax errors.

But will it get me a job?

No, you won't get *a* job using in *Snap!*

It's *a* learning tool, but that doesn't negate its value. We are here to learn after all.

Later in the course we will move to using Python, which is *a real-world*, text-based language that you might use on the job.

Polyglot Programmers

A polyglot is a person who knows how to use many languages.

Once you've learned the basics of how to think logically, learning programming languages is easy.

After this class, you can take this knowledge and learn any language you want.

Learn by doing

- I'm not going to lecture on how to code.
- Labs will introduce you to the topics as you put them into practice.
- Take your time and make sure you understand the material as you go.

Be curious. Experiment. Ask "what if".

Pair Programming

- An industry practice popular in Agile development.
- Simply put, you take
  - two people
  - give them one computer, keyboard and mouse,
  - and they work together on a single assignment

Why?

- Two heads are better than one.
- There is a sounding board for ideas
- Fewer mistakes or bugs in the final code
- Better, easier to read code
- More people know more of the code
- “Pair pressure” encourages good performance, and less slacking off

Driver & Navigator

There are two roles:

- The Driver's role is tactical, to focus on the mechanics of operating the computer and entering code.
- The Navigator's role is strategic, to think about what needs to be done and plan ahead to get there.

https://youtu.be/vgkahOzFH2Q

Switching roles

- In the early labs, there are reminders.
- Later, the reminders go away, but you still need to switch, ideally every 10-15 min.
- Do it as it feels natural, but do it.
- It might seem awkward at first, but you'll learn to feel comfortable switching roles at any time.

Be patient & respectful

- Constant communication is key.
- You will learn by explaining your thoughts and rational to someone else.
- Ask questions, challenge your pair.
- The best pairs are closely matched in skill
- But even with a master and newbie, both can gain a lot from the experience.

Take breaks

- It's a long night, step away for a bit.
- Do not work ahead without your pair!
- Stop together and agree on when to return and start up again.
- Be respectful of your partner and do not be late coming back.

Don’t fret over mistakes

You’ll see me make mistakes.

Your pair will make mistakes.

You will make mistakes.

If this freaks you out, acknowledge it and work to get over it. We are here to learn, and mistakes are part of learning.

Disagree? Talk it through

- Conflicts will occur, but you need to learn to work through them.
- Sometimes you need to follow through on a bad idea to just try it out. Maybe it turns out to be a fine idea and you were wrong; maybe it doesn't. Be OK with it either way.

Charm your pair

Whether you know more or less, whether you are outgoing or shy, believe in yourself and believe in your pair.

Be confident in what you know, be honest about what you don't.

Most important, be open to new ideas, discussion and learning.

Do I have to?

- Not pairing is not an option, at least not if you want full-credit for labs.
- Part of your participation grade is based on how well you work together in lab.
- Why? This is a job skill too.

Questions in lab

- Before you ask me a question about the lab, you must first ask two other pairs.
- Be respectful about interrupting. Let them know you have a question and allow them to finish a thought if needed.
- If you have no answer after asking two other pairs, put up the red flag.

Lab check-off

- When you are ready for me to check-off (grade) your lab, put up the green flag.
- I'll look at at least one of the "show me" items from the lab sheet, and each person will answer a question from the lab.
- See another flag? Review your answers to worksheet with another pair while waiting.

Create a Snap Account

Go to: snap.berkeley.edu/run

Click on the cloud button & fill out the form.

Check email for your password.

Lab Time!

- Access the labs from the course web site.
- It might be helpful to use one computer as the instruction screen & work on the other.
- DO NOT work “together” (or side-by-side) on two computers. That’s not pairing.
- You can use a laptop for instructions, but code on the classroom computers only.

Lab Advice

- Don’t skip stuff!
- One person should log into Snap! to start.
- Save often to the cloud.
- It’s OK to save everything in one program.
- To get *a clean* script, make a new sprite.
- At the end of lab, log out, have the other person log in and save to their cloud.