Lecture 1

Introduction: CS 151 Computational Thinking

- Goals of the course, lecture notes on web site
- Class protocol
- Office on Roberts 2nd, office hours: MWF 9-10am, Monday afternoons, Monday late, otherwise knock
- TAs: Sunday 4-10, MTW 7-10

Students with some background in CS may want to consider CS 231. They should talk with me or Dale.

What is CS?

- The study of describing processes in a manner such that they can be executed on a computer.
  - Design of computers
  - Design of computer languages
  - Design of algorithms
  - Analysis of algorithms and computational capabilities

Why is it important? 5 years ago no FB, 10 years ago no iPhone, 20 years ago no internet, 40 years ago first IC, 80 years ago no electronic computer. Pace of change is speeding up.

A key component of CS is design

What can a computer do?

- Store data
- Move data
- Manipulate data
- Adjust control flow based on data

Abstraction is a key part of design and the description of processes

Abstraction: face example

Levels of abstraction in a computer and where we are studying.

Abstraction: computer languages

Top-down design as related to abstraction

- square
- building as a set of squares
- city as a set of buildings
Lecture 2

Algorithms: what are they

- Defined by languages: English doesn’t work very well as a programming language
- Rules of programming languages: required, eliminate ambiguity
- Conventions: usage models that make programming, and understanding programs easier
- Interpreted v. compiled

Review the components of a computer and how they interact

- What is a word processing program doing
- What is a mail program doing
- What is a fileserver doing
- What is python doing

Describing algorithms

- Start python interpreter and show some example math expressions

Variables: storing data

- Variables as abstracted memory locations
- Physical example of picking up something and transferring it
- Type: int, float, string, bool, others
- Naming rules

Assignment: moving data

- Operator is the equals sign, described as ‘gets’, or ‘is assigned to’, never ‘equals’
- Show examples in the interpreter
- Assignment is a one-time operation and copies data from one place to another: physical example
- Equality is a test, not an operation in a computer language
- Variable conventions
- Common bugs with variables: using it before it is defined, typos

Types

- Type function, lots of examples
- Look at the type of expressions with mixed types

Casting: telling Python to change the type of a variable when manipulating or assigning the value

Symbol Tables: introduce as bookkeeping in Python; a model for how to predict what will happen

- name, type, value
Lecture 3

Symbol Tables: review
- name, type, value
- create an entry when you assign a value to a variable
- look in the symbol table to find a variable’s value

Operators
- plus, minus, multiply, divide
- parentheses
- modulo as a method for converting a sequence into a repeating pattern

Functions
- Syntax
- Parameters/arguments
- Calling a function
- Symbol table representation

Lab Review
- importing a package into Python
- making the turtle do something: forward, left, right, circle
- making a function to do something like a square, with a parameter

Subdividing a problem using functions
- function to make a square
- function to make a square with four squares inside (x/4.0 in width)
- function to make three blocks stacked
- function to make a city from three buildings
Lecture 4

Calling functions

- Euclid example
- Emphasize pass by copy, usage of parameter and argument
- Walk through symbol table of the function
- Bookkeeping lets us keep track of what variables exist where

Introduce the idea of a drawing as a thing we can place where we want

- Block function (x, y, width, height)
- Equilateral triangle function (x, y, length)
- House(x, y, width, height)
- Focus on parameters
- Create a goto function to simplify the functions
- Demonstrate placement and scale differences
- Demonstrate randomness
- Create a filledBlock function?

How to create a neighborhood?

- Top-down design
- Top level function and coordinate system
- Road
  - Lots of houses along the top side of the road

Introduce a for loop as a fixed repetition tool
Lecture 5

Go over homework

Quiz

Review

- random package
- help in Python
- for loop and the value of the loop variable
- how does it modify the symbol table?

Control flow

- if statement
- if else
- if elif else
- tests: equals, lt, gt, lte, gte, not equals
- boolean expressions: and, not, or
- lower and upper bounds in a single expression
- functionality on differing types: strings, ints, floats

Block function with a fill argument
Lecture 6

Go over quiz

- emphasize that a string is an actual value, not a symbol or a name
- adding a string and an integer doesn’t make sense to a computer (or us)
- variable names don’t have types, data has a type in Python

Control flow

- if statement
- if else
- if elif else
- tests: equals, lt, gt, lte, gte, not equals
- boolean expressions: and, not, or
- lower and upper bounds in a single expression
- functionality on differing types: strings, ints, floats

Do a minimize 3 items function

A function for drawing birds (v’s) up or down
Lecture 7

Review of control flow
- if / elif / else
  - You can have as many elif cases as you want
  - The first test, and only the first test to evaluate as true executes its block of code
  - You do not have to have an else case, but it does have to come last if it exists
  - == is the test for equals
  - != is the test for no equals

Example: find the minimum of three numbers
- consider it as three tests, one will succeed
- consider it as a set of nested tests, or a decision tree
- consider it as a linear search over the parameters

As an exercise in algorithm design, the last is the easiest to code and the easiest to expand to more variables

What if we wanted to create a collection of numbers?
- Sequences are important
- We don’t often know how many items we will have in the sequence when we write the program
- Sequences of characters are a special case (strings), but they are still sequences of data

Lists
- Lists are identified by a comma-separated set of Python objects surrounded by square brackets
- Lists are a sequence of locations for storing data
- Any location in a list can hold any Python type
- You can add to, subtract from, and change the elements of a list; they are \texttt{mutable}

Create a bunch of lists

Index into a bunch of lists

Lists are objects
- The symbol to which we assign an object holds a reference to a symbol table
- Each object has its own symbol table
- An object can have both functions and data in their symbol table
- The functions operate on their own list
- When you assign one list to another, it does not create a new list, only copies the reference to the symbol table
Lecture 8

Homework review
- print versus return
- order of operation

Quiz

Review of Lists
- creating lists
- indexing lists
- adding to lists

Multi-level lists
- Lists containing lists, how to evaluate the expression
- Passing a list to a function (max example)

List slices
- returning part of a list
- everything but the last
- everything but the first
- a copy of the top level

Strings
- Immutable, can’t change the value of a character in a string
- S.capitalize()
- S.lower(), useful for comparisons
- S.find( substring) returns lowest index into S
- S.strip(), remove white space on either end, useful for comparisons
- S.split(), splits a string on white space, optional argument, returns a list

Tuple
- Like a list, but is immutable
- Can hold any type of object
Lecture 9

List/String/Tuple review

- list is mutable, square brackets
- string is immutable, quotes
- tuple is immutable, round brackets
- Each type is an object, with associated functions/methods

Slices

- returns a piece of the sequence
- syntax: beginning index, colon, up to but not included the last index
- If no number provided, goes to the end in that direction

New Concept: Iteration

- The for loop permits you to repeat a block of code
- The loop variable changes each time through the loop
- In Python, all loops are over sequences (list, string, tuple)

Range function:

- The range function returns a list (sequence) of numbers
- The range function can take 1, 2, or 3 arguments
- 1 argument: 0 to end index - 1
- 2 arguments: start index to end index-1
- 3 arguments: start index while less than end index, step is provided

Example looping over a string

- Use the range function
- Use the string itself as the sequence

Example looping over a list

- Use the range function
- Use the list itself as the sequence

Example creating blocks

- Use a list to describe (x, y, width, height, color)
- Use a list of lists to hold all the blocks
- Access each item in the for loop

Emphasize the separation of the specific data and the process
Lecture 10

Review lists and iteration

- Symbol table when you create a list: makes a new one for the object
- Concept of methods on objects: sort, append, etc.
- Slices
  - Iteration: loops over the elements of a list (only the top level)

Program design

1. Define
2. Inputs/outputs
3. Break down the task
4. Organize the steps, or pieces, noting inputs and outputs
5. Identify useful functions, noting repetition
6. Generate an intermediate representation of the algorithm: picture, flow chart, pseudo-code
7. Write code
8. Verify and test
9. Return to step 3, if necessary

Zelle graphics package

- We’ll be using it for at least the next 2 assignments to work with images
- Introduction to using classes/objects
- Lists and strings are also classes/objects: combination of data and functions

Using objects enables object-oriented design

- Nouns in a problem definition become classes/objects
- Verbs in a problem definition become methods for the objects
- Adverbs and adjectives represent data, or parameters for the methods

Images: the primary reason for using Zelle

- 2D array of sensors
- Each location in the array is a pixel
- Pixels exist on a grid
- Each pixels has an address: the row (y) and column (x) of the pixel in the grid
- In the Zelle package, each pixel has three values (red, green, blue) in [0, 255]
- Write example to open and show an image
Lecture 11

Homework:

```python
print a[2] 3
print b[1] ['c', 'd']
print b[1][1] d
print c[2] and
print d [1, 2, 3, 4]
print d[10] error with d[10]
print b[2][4] ?
print a[-1] 4
print e[0] [1, 2, 3, 4]
print e[0][2] 3
print e[-1][-1] 4
print e[1][-2][-1] d
print c[0][0:3] fou
print a[:2] [1, 2]
print c[1:] ['score', 'and']
print d[::1] [1, 2, 3]
```

Quiz: 31

Objects as symbol tables

- An object combines both data and methods/functions
- Notation is identical to calling a function in a package you have imported (turtle.forward)
- Python dot notation indicates a trail of symbol tables left-to-right
- A method has access to the object used to execute it
- list sorting example

Copying an object requires a special operation

- For simple data type (int, float, str) the data is effectively stored in the symbol table
- An object generally has some kind of copy function, often called clone by convention

Why use objects?

- Many objects have different data, but the same functions
- Python can store different kinds of objects in a list
- If all of the objects support the same methods, it is easy to apply a method to many different objects

Create some circles and lines and move them all with a for loop
Lecture 12

Quiz review

- Lists are objects
- When you pass an object into a function, you are passing a reference
- References let you modify the underlying symbol table (data)
- The append method, modifies the list data

Pick up at the end of Friday

- Create a list with several different objects (clone them)
- make a for loop that moves them randomly
- make a main loop that delays (time.time(s))
- make it run for 20s

Review: the sys package

- command line arguments
- sys.argv

Review: cloning objects

- you can create a copy of any object that supports a clone method
- all graphics objects support a clone method

Manipulating objects

- Because we pass a reference to the object to a function, we can modify the object in the function
- swap red/blue
- what about a modulo operator? Try (row + col) % 2 == 1, and then use a divisor on row and col individually
- try out some other things
Lecture 13

More on image manipulation

- The name of the file is not the image, it’s just a string
- Show creating a clone and displaying both images cloning
- Cartoonize the image (e.g. divide each value by 64, then multiply by 64), integer math
- Contrast: if greater than some value, scale up, otherwise scale down, check for saturation
- Add noise, but use the random value to alpha blend in something else (like a constant image)
- How about randomizing the channel order at each pixel? random.shuffle()
- How about a max dependent shuffle? Max red to blue, max green to red, max Blue to green
- Invert the intensity without modifying the color: calc intensity, (255 - I)/255.0 is the scale factor

Default parameters for functions

- Create some simple examples
- How about a putPixMap with an alpha blend that is, by default 1 for the source image?

How about representing a collage as the critical information?

- when making a scrapbook, what information defines an image?
- (x, y, image filename)
- what information defines a page?
- a collection of images and the size of the page
- how could we store this information?
- stick the actual image data at the end of the list [-1], emphasize difference between filename and data
- how could we store lots of images?
Lecture 14

quiz review: look at what functions can do to lists
dynamic collages using lists of filenames/images
  • Reading images by looping over a list of filenames
  • Finding the max height and max width by looping over the images
  • While loop (while 1)
    • Looping over the images, which are stored in a list
    • return on a checkMouse
    • look at the functionality of a break statement
Upgrade with makewhite
Upgrade with blendwhite
Lecture 15

Homework Review

- Focus on ignoring the details: a list is a sequence of things, whatever those things are
- Append/pop modify the lists in place

Quiz

Animation Design Task: a goalie blocking a soccer ball

Objects: each item made of of graphics objects, so we can represent each item as a list of primitives.

- Ground: big box, possibly with white stripes
- Net: bars, possibly with some lines
- Goalie: legs, body, head, arms
- Ball: white circle

Task description: an animation sequence with 20 time steps

- Ball appears on the left side of the screen on frame 2
- Ball reaches the goalie on frame 12, goalie jumps to block it
- Ball falls to the ground during frames 12-18, along with goalie

Design

- Create the window
- Create the objects (lists of primitive objects)
- Draw all of the objects in their initial locations (time step 0)
- Loop for 20 time steps
- Update all of the objects to their positions in time step t
- Hold until a mouse click, then close the window

Need a function that creates each object

Need a function that updates each type of object

- One function may have a big if-statement structure
- Need some kind of information to indicate the type of each object: string

Start the project by making each object really simple: one object
Lecture 16

Quiz review

Animation Design Task: a goalie blocking a soccer ball

Objects: each item made of of graphics objects, so we can represent each item as a list of primitives.

- Ground: big box, 800 wide, 100 tall at (0, 350)
- Net: one box to start 10 wide, 200 tall; at (600, 350)
- Goalie: legs 20 wide, 40 tall, body 30 wide, 60 tall, head (10, -10) radius 10; at (510, 250)
- Ball: white circle, size 10; at (20, 40)

Task description: an animation sequence with 25 time steps

- While < 15, ball is $dx = 35 \times s$, $y(t) = t^2 \times 0.75^2$
- While 15 < $t$ < 24, ball is $dx = -10s$, $y(t) = t^2 \times 0.75^2$
- While 10 < $t$ < 19, goalie is $dx = 0$, $y(t) = 150 - 5t^2$

Design

- Create the window
- Create the objects (lists of primitive objects)
- Draw all of the objects in their initial locations (time step 0)
- Loop for 25 time steps
- Update all of the objects to their positions in time step $t$
- Hold until a mouse click, then close the window

Need a function that creates each object

Need a function that updates each type of object

- One function may have a big if-statement structure
- Need some kind of information to indicate the type of each object: string

Start the project by making each object really simple: one object
Lecture 17

Grammars

- grammar hierarchy: regular, context-free, context-sensitive, unrestricted
- applications: parsing languages, computer and natural

Diagram a sentence / generate a sentence

L-systems

- Astrid Lindenmayer, plant biologist
- developed to model plant growth
- parallel application of the rules
- simple example

Example with Koch snowflake

- base string: F–F–F–
- rule: F → F+F–F+F

Algorithm for string generation

```python
# Given: the input string (e.g. curString)
# Assign to a variable (e.g. newString) the empty string
# for each character in the current string
#   # if there is a rule for the current character
#     # add the replacement string to the end of the new string
#   else
#     # add the current character to the end of the new string
```

What if we attach a graphical meaning to each character?

Concept of an interpreter

- Convert information from one form to another
- Read through the input string
- Execute an action based on each input
- Each action modifies the state of the system
- The system remembers its state from one character to the next

Algorithm for an interpreter: for each character in the string, test it with a big case statement
Lecture 18

Review Homework

Quiz (20 minutes max)

Review of L-system grammars

- vocabulary
- base string
- rules: symbol → replacement

Do a simple example. I’ve done the simple koch snowflake. They have seen brackets, so maybe a simple branching L-system.
Lecture 19

Quiz review: don’t forget loop variables; appending a list to a list

File reading

- open a file: I tend to like the `fp = file( filename, mode )` usage instead of `open`.
- `readlines`: explain it, but also show it by writing code, probably in the interpreter
- close: always need to close a file

File parsing

- looping over the lines
- `split` method as a way of parsing a line of characters into separate words
- looping over the words
- have to be clever to find all of the words: `lower`, `find`, `length`, `isalpha`
- make the function general to find any word in any file

File writing: similar example; demonstrate writing text to a file and opening it in TextWrangler.

- Emphasize the importance of closing the file
Lecture 20

Review of opening/reading and writing/closing a file

L-system example:

- represent an L-system as a list with two items: base string and list of rules
- go over the one-rule replacement algorithm
- write out the string to a file
- make a simple interpreter to draw it

For one-rule L-systems we can use the replace string function

```python
# Given: the input string (e.g. curString)
# for the number of iterations
# curString = curString.replace( symbol, rule )
return curString
```

Reading an L-system from a file

- Design a file format
- Want it to be somewhat flexible
- Want it to be human readable
- Want it to be easy to parse

```python
# Given: the filename (filename)
# Open the file for reading
# Read all the lines of the file
# Close the file
# create an empty L-system (choose a list-based representation) [ '', [] ]
# for each line in the set of lines
# split the line into words
# for each word
  # if the word is the keyword 'base'
  # set the base string of the L-system
  # else if the word is the keyword 'rule'
  # add the rule to the L-system
# return the lsystem (note indentation)
```
Lecture 21

Homework review

- Remind students what the += means in Python
- Remind them what the find function returns

Quiz (20 minutes)

If there is any material from the lab you’d like to spend more time covering, do that.

- branching L-systems and storing/restoring the turtle state

I think it’s worth spending time showing them L-system examples. I have not done any branching L-systems in class except one when I first introduced the concept. Have ABOP open, and demonstrate reading a branching L-system from it and drawing it.

You might show them an example of doing something slightly different with an interpreter, like making the C character create a circle, or the character L make a leaf. Remind them that they should restore the turtle state if they draw some kind of shape.

On Monday I’ll start classes. I don’t think it’s worth starting them after the quiz.
Lecture 22

Quiz review

Classes

- class syntax
- vocabulary: field (variable) and method (function)
- init method
- self variable
- object as symbol table, and self as a reference to the symbol table
- adding fields to the self object (in init method)
- adding methods
- local variables in methods

Student object example

- init
- accessor functions: get/set
- __str__ method as an example of a utility function
- conflicts in naming: fields and methods, symbol table figure

Main code using the Student objects

- Emphasize the automatic inclusion of the self variable
- Emphasize that each object has its own symbol table
- Note aliasing that can occur if you assign one object to another
- Show how you would create a clone method for the student object to do a safe copy
Lecture 23

Classes
- Review syntax: class, class name, init method, other methods
- Review adding fields to a class
- Review accessor functions: get/set
- Symbol tables for the student class
- __str__ method as an example of a utility function
- conflicts in naming: fields and methods, symbol table figure

L-system as a class
- easier to organize than a list: use names to reference a field
- all of the functionality is encapsulated in the class definition
- don’t have to be concerned about how the information is stored in order to use it

Multi-rule L-systems
- still storing the rules as a list of lists
- each rule is a list of 2 items: symbol and replacement
- in the replace function, we have to search the list of rules

```python
def replace( self, cstring ):
    tstring = ''
    for ch in cstring:
        found = False
        for rule in self.rules:
            if rule[0] == ch:
                tstring += rule[1]
                found = True
                break
        if not found:
            tstring += ch
    return tstring
```

Introduce the idea of complexity

Number of operations is related to length of the string and the number of rules
- Number of operations is linear in both cases
- If we double the number of rules, the complexity of the computation doubles
- The break statement can have a practical effect, but does not affect the overall complexity
Lecture 24

Homework review

Quiz
Lecture 25

Default class functions: functions Python uses for certain tasks

- `__init__` is one default class function
- `__str__` is another default class function: give example in student.py

All classes have default versions of these functions. When you define a new version explicitly, it overrides the default version.

Name conflicts in classes: if you have a field and a method with the same name, one wins. Python does not guard the symbol tables closely, you can usually overwrite anything in a symbol table. (In fact, in Python you can override just about anything).

New idea: how could we avoid having to write the same function for lots of different classes?

Inheritance: using another class as a template symbol table, then modifying the new pieces

Create a shape file

- Triangle
- Square
- Star

Then think about eliminating duplication

- the draw function is common to all of the classes, and uses the same arguments
- most of the `__init__` function is common to all of the classes
- how do we call the parent class function? Use the symbol table’s name explicitly
Lecture 26

Dictionaries

- Meaning/Purpose: to store key/value pairs
- Syntax: curly brackets, colon separating key and value
- Key: any immutable type
- Value: any Python type
- Indexing: square brackets with the key value

Examples

Methods

- keys() - return a list of keys
- values() - return a list of all the values
- get(k, [a] ) - return the value of k, if k does not exist return a
- has_key( k ) - return true if the dictionary includes the key
- items() - list of (key, value) pairs
- pop( k ) - remove the key k from the dictionary and return its value

Creating dictionaries from scratch: syntax

Creating dictionaries from lists: a list of lists, with each sublist having 2 items: key, value

Adding to existing dictionaries: index notation

Iterating over dictionaries: loop variable contains the keys

Dictionary ordering: random

Using dictionaries for an L-system

- addRule function: make the symbol the key, make the list of replacements the value
- replace the rules list with a rules dictionary
- replacement algorithm

# with one replacement
def replace(self, curstring):
    newstring = ''
    for c in curstring:
        newstring = newstring + self.rules.get( c, c )[0]

# with multiple possible replacements
def replace(self, curstring):
    newstring = ''
    for c in curstring:
        newstring = newstring + random.choice( self.rules.get( c, c ) )
Lecture 27

Homework review

Quiz

Inspiration image using a tree, shapes, autumn colors, and the gauss function
Lecture 28

Quiz review

Dictionary review

Questions about the project

NPR

- Process of NPR
- Purpose of NPR: visual interest, technical drawing
- Encapsulation and NPR design: forward function is the kernel
Lecture 29

NPR Demonstration

- Draw a set of vertical lines
- Show color manipulation by varying saturation
- Show broken line concept and the idea of jitter
- Show Pollack drawing concept and the idea of stepping along a line
- Any other suggestions from the audience?
Lecture 30

Homework review
Quiz
Introduce the concept of recursion...
Lecture 31

Recursion

- Summation algorithm
- Show function symbol tables
- Show binary search
- Show a graphical example: half squares going right
- Show a graphical example: binary tree
- Show a graphical example: Sierpinski triangle

```python
def sierpinski( x0, y0, size ):
    if size > 2:
        s = shape.Triangle( size )
        s.draw( x0, y0, 1 )
        sierpinski( x0, y0, size/2 )
        sierpinski( x0+size/2, y0, size/2)
        sierpinski( x0+size/4, y0-math.sqrt(3)*size/4, size/2)
```

Show how we can use a filled triangle (use the autumn colors) and how different arrangements of the drawing have different effects on the result.
Lecture 32

Current assignment: modifying the interpreter to take in parameters

- Treat each symbol as a function call with an optional parameter
- Main issue # 1: how does the computer know there is a parameter?
- Main issue # 2: how does the computer figure out the value of the parameter?

Our solution uses parentheses to indicate parameter existence and bound the parameter string.

- The parameter must appear before the symbol it modifies
- An open parenthesis must be followed by a closed parenthesis
- The parameter string must be a number (float or int type)
- The interpreter must temporarily store the parameter string as it parses the input
- The parameter modifies the symbol immediately following the closing parenthesis

Parametric L-systems

- We want to be able to give symbols parameters
- We want to be able to use mathematical expressions in rules
- Example: \((x)F\ (x)F\ [+(x*0.67)F] [-(x*0.67)F]\)
- Makes the branches 2/3 the length of the parent

Four cases that can occur in parametric L-systems

- An unparameterized symbol has an unparameterized rule: same as previous L-systems
- A parameterized symbol has a parameterized rule: substitute the parameter value for X and execute the rule
- An unparameterized symbol has a parameterized rule: odd case, treat as the identity rule
- A parameterized symbol has an unparameterized rule: replace all instances of the symbol in the replacement with a parameterized version

Examples of parametric L-systems: convert systemB into a parametric system and fancy it up
Lecture 33

3D Turtle thinking

- roll, pitch and yaw
- representing the turtle’s orientation (heading)
- characters to represent those operations
- thinking about updating the interpreter
- thinking about updating NPR methods: e.g. position returns (x, y, z) not just (x, y)
- making a cube
- making a pyramid: make a square, turn left 45, turn up 45 forward, down 90, forward, repeat for the other direction
- a house is a pyramid and a cube
- thinking about L-systems in 3D
- show some examples from ABOP
Lecture 34

Homework review: recursion

Quiz

Headings in the 3D turtle: two vectors

Cylinder as a set of polygons: 2 fans and a set of rectangles

Walk through the systemZ example

Subdivision shapes:

Circle
  • make a square with corners on the unit circle
  • for each edge, break it in two and move the midpoint to the unit circle
  • it is the same idea as substitution in L-systems PaPb \rightarrow PaPcPb

Bezier curve
  • Begin with four points \((A_0, B_0, C_0, D_0)\)
  • Calculate the midpoint between each of the original points \((A_1, B_1, C_1)\)
  • Calculate the midpoint between each of the next level points \((A_2, B_2)\)
  • Calculate the midpoint between the last two points \((A_3)\)
  • Make two new sets of points: \((A_0, A_1, A_2, A_3), (A_3, B_2, C_1, D_0)\)
  • Iterate a few more times, then draw the lines
Lecture 35

Brian talked about his high dynamic range microscope research
Lecture 36

Questions about quiz?

Review of topics

Review of other languages
  • Where are they used?
  • What are the differences?
  • What courses would you take to learn them?

What can a computer do?
  • Store data
  • Move data
  • Manipulate data
  • Adjust control flow based on data
Lecture 37

Donuts

Homework review

Review of follow-on CS courses: CS 231, CS 232

Quiz