Welcome to CS61A!

This week

- Please see the course web site (`http://cs61a.org`), esp. the announcements and *Syllabus* link.
- The CS61A Piazza is another source you should consult regularly.
- In particular, the Q&A for today's lecture will be conducted on @27.
- If you need to be added to OkPy, use the form on Piazza message @23.
- If you change your CalCentral email, use the form on Piazza message @48.

More This Week

- This week, we will be holding tutorials, about which you should have received information (see Piazza @44). You should be able to move yourselves if necessary.
- This week, there is a Lab 0, which is a self-paced setup and installation guide, due next week. There will be special tech-help office hours (see Piazza) in case the Lab 0 instructions don't work for you.
- There will be an *Ask Me Anything* (AMA) session with the instructors at 1PM PST this Thursday where you can—well—ask us anything.
- Exam prep sessions begin on Friday.
- Lab orientations, discussion orientations, parties, lost sections, and regular office hours begin next week.

What is Computer Science?

- Computer science is the study of a group of subjects related to *computation*, such as
  + Programming: algorithms, software engineering, ...
  + Theory: computational complexity, computability, cryptography, ...
  + Programming Languages: semantics, formal verification, compilation. . . .
  + Graphics: creating, rendering, and manipulating images.
  + Artificial Intelligence: performing tasks typically requiring human intelligence.
  + Systems: combinations of software and hardware (e.g., operating systems, robotics, avionics).

What's This Course About?

- This is a course about *programming,* which is the art and science of constructing artifacts ("programs") that perform computations or interact with the physical world.
- To do this, we have to learn a *programming language* (Python in our case), but programming means a great deal more, including
  - Design of what programs do.
  - Analysis of the performance of programs.
  - Confirmation of their correct operation.
  - Management of their complexity.
- This course is about the "big ideas" of programming. We expect most of what you learn to apply to any programming language.

Other Courses

- If you've had no exposure to programming before, you may want to consider CS 10, The Beauty and Joy of Computing.
  - Designed for students without prior experience.
  - A programming environment created by Berkeley, now used in courses around the world and online.
  - An introduction to fundamentals (and to Python) that sets students up for success in CS 61A.
  - See `http://cs10.org`.
- If you are not actually interested in computer science as an academic subject, but want to learn to program in support of your real interests, you'll find plenty of DIY online resources for learning Python, Java, etc.

Course Organization

- Readings cover the material. Try to do them before...
- Lectures summarize material, or present alternative "takes" on it. zz
- Laboratory exercises are "finger exercises" designed to introduce a new topic or certain practical skills.
- Discussion Orientations are largish meetings that review material.
- Tutorials are small group meetings that follow orientations and allow exercise of the material.
- Homework assignments are more involved than lab exercises and often *require some thought*. Plan is to have them due on Thursdays. Feel free to discuss the homework with other students, but turn in your own solutions.
- Projects are four larger multi-week assignments intended to teach you how to combine ideas from the course in interesting ways. We'll be doing these in pairs.
- Use the discussion board (Piazza) for news, advice, etc.

Mandatory Warning

- We allow unlimited collaboration on labs.
- On homework, feel free to collaborate, but try to keep your work distinct from everyone else's
- Likewise on projects, except that you and your partner submit a joint project.
- You can take small pieces of code within reason (ask if unsure), but you *must* attribute it!
- Otherwise, copying is against the Code of Conduct, and generally results in penalties.
- Most out-and-out copying is due to desparation and time pressure. Instead, see us if you're having trouble; that's what we're here for!

What's In A Programming Language?

- *Values*: the things programs fiddle with;
- *Primitive operations* (on values);
- *Combining mechanisms,* which glue operations together;
- Some predefined names (the *library*);
- *Definitional mechanisms*, which allow one to introduce symbolic names and (in effect) to extend the library.

Python Values (I)

- Python has a rich set of values, including:

Type	Values	Literals (Denotations)

Integers	$0 \quad -1 \quad 16 \quad 13$
	 36893488147419103232
		`0 -1 0o20 0b1101`
		 `0x20000000000000000`

Boolean (truth) values	true, false	`True False`
`<NoneType>`		`None`

Functions	`func f(x): ...`
	 `func` λ`(x): ...`
		`def f(x): ...,`
		 `lambda x: ...,`
		 `operator.add, operator.mul`

- A *denotation* is a Python expression that denotes a value.
- Functions take values and return values (including functions). Thus, the definition of "value" is recursive: definition of function refers to functions.
- They don't look like much, perhaps, but with these values we can represent anything (in fact just booleans and functions would suffice!)

Python Values (II)

- . . . but not conveniently. So now we add more complex types, including:

Type	Values	Denotations
Strings	pear,	`"pear"`
	I ♡ NY	`"I \u2661 NY"`
	Say "Hello"	`"Say \"Hello\""`
Tuples		`(), (1, "Hello", (3, 5))`
Ranges	0–10, 1–5	`range(11), range(1, 6)`
Lists		`[]`
	1, Hello, (3, 5)	`[1, "Hello", (3, 5)]`
	0, 1, 8, 27, 64	`[ x**3 for x in range(5) ]`
Dictionaries		`{ "Paul" : 69, "Ann" : 68}`
Sets	$\{\}, \{1, 2\}$,	`set([]), { 1, 2 },`
	$\{x \mid 0 \leq x < 20$	`{ x for x in range(20) if prime(x) }`
	$\wedge\ x$ is prime$\}$	

Representing Values

- These types give Python the power to *represent* just about anything.
- Much of the art of programming involves doing just that—creating *data structures* that behave in ways that are *analogous to* the relevant characteristics of real-world things we're computing things about.
- This analogizing is known as *abstraction*, and is a major theme of this course.

A Few General Rules

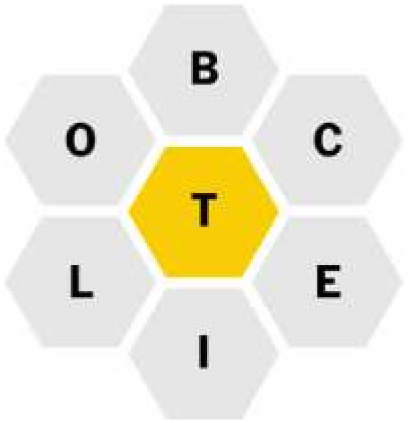
- Whatever the assignment, start now.
- “Yes, that’s really all there is. Don’t fight the problem.”
- Practice is important. Don’t just assume you can do it; do it!
- *ALWAYS* feel free to ask us for help.
- DBC
- RTFM
- Have fun!

A Taste of Python

- Beginning Friday, we'll look at our first important Python construct: the function call.
- Calls on built-in (or library) functions perform the basic computations on values, from which all programs are built.
- Let's take a quick look at some of things we can accomplish by such operations.

A Puzzle

- The New York Times carries a Spelling Bee puzzle. Given a puzzle like this:



one is supposed to form as many words of at least 5 letters using just letters from the given letters (any number of times), with at least one being the center letter.

- With one dictionary, I got the following words:

```
beetle, belittle, betel, billet, blotto, bootee, bootie, bottle,
collect, collectible, cootie, eclectic, elect, elicit, elite, illicit,
ioctl, licit, little, lotto, ocelot, octet, octette, titbit, title,
tittle, toilet, toilette, tootle
```

A Python Puzzle Solver

Here's a Python program that solves these puzzles (in fact, it's more general, as the comment suggests):

```
def find_words(word_list, must_contain, may_also_contain, min_length):
    """Return a list of all words in the list of strings WORD_LIST
    that are at least MIN_LENGTH letters long, contain all the letters
    in the string MUST_CONTAIN at least once, and consist only of
    letters in MUST_CONTAIN and in the string MAY_ALSO_CONTAIN."""

    must_contain_set = set(must_contain)
    may_contain_set = set(must_contain + may_also_contain)
    return [ word for word in word_list
                  if len(word) >= min_length and
                     must_contain_set <= set(word) <= may_contain_set ]
```

```
>>> find_words(words, "t", "bceilo", 5)
['beetle', 'belittle', 'betel', 'billet', 'blotto', 'bootee',
 'bootie', 'bottle', 'collect', 'collectible', 'cootie', 'eclectic',
 'elect', 'elicit', 'elite', 'illicit', 'ioctl', 'licit', 'little',
 'lotto', 'ocelot', 'octet', 'octette', 'titbit', 'title', 'tittle',
 'toilet', 'toilette', 'tootle']
```