WELCOME!

(download slides and .py files and follow along!)

6.0001 LECTURE 1

TODAY

- course info
- what is computation
- python basics
- mathematical operations
- python variables and types
- NOTE: **slides and code files up before each lecture**
  - highly encourage you to download them before lecture
  - take notes and run code files when I do
  - bring computers to answer **in-class practice exercises!**

COURSE INFO

- Grading
  - approx. 20% Quiz
  - approx. 40% Final
  - approx. 30% Problem Sets
  - approx. 10% MITx Finger Exercises

COURSE POLICIES

- Collaboration
  - may collaborate with anyone
  - required to write code independently and write names of all collaborators on submission
  - we will be running a code similarity program on all psets
- Extensions
  - **no extensions**
  - **late days**, see course website for details
  - **drop and roll** weight of max two psets in final exam grade
  - should be EMERGENCY use only

RECITATIONS

- not mandatory
- two flavors

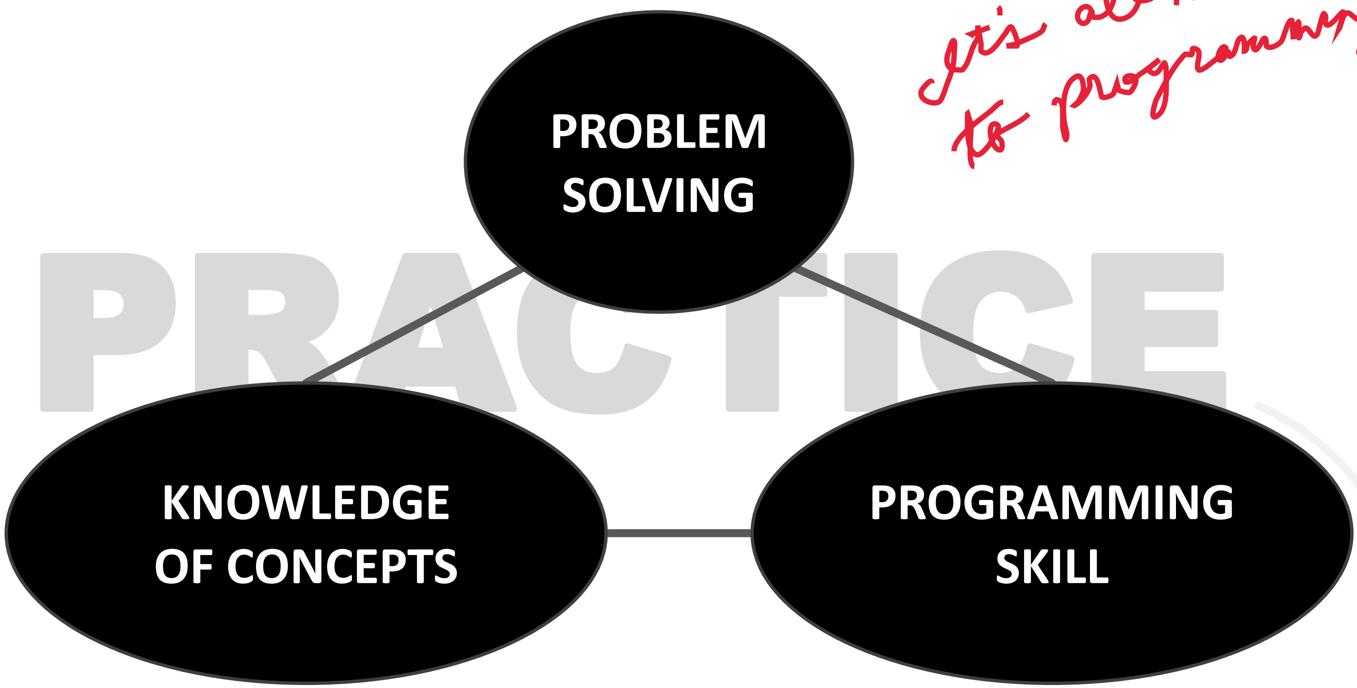
1) Lecture review: **review** lecture material
   - if you missed lecture
   - if you need a different take on the same concepts

2) Problem solving: teach you **how to solve** programming problems
   - useful if you don’t know how to set up pseudocode from pset words
   - we show a couple of harder questions
   - walk you through how to approach solving the problem
   - brainstorm code solution along with the recitation instructor
   - will post solutions after

FAST PACED COURSE

- Position yourself to succeed!
  - **read psets when they come out** and come back to them later
  - use late days in emergency situations
- New to programming? **PRACTICE. PRACTICE? PRACTICE!**
  - can't passively absorb programming as a skill
  - download code before lecture and follow along
  - do MITx finger exercises
  - don't be afraid to try out Python commands!

it's all relevant to programming



TOPICS

- represent knowledge with **data structures**
- **iteration and recursion** as computational metaphors
- **abstraction** of procedures and data types
- **organize and modularize** systems using object classes and methods
- different classes of **algorithms**, searching and sorting
- **complexity** of algorithms thinking algorithmically

WHAT DOES A COMPUTER DO

- Fundamentally:
  - performs **calculations**
    a billion calculations per second!
  - **remembers** results
    100s of gigabytes of storage!
- What kinds of calculations?
  - **built-in** to the language
  - ones that **you define** as the programmer
- computers only know what you tell them

computers = powerful + dumb

TYPES OF KNOWLEDGE

- **declarative knowledge** is **statements of fact**.
  - someone will win a Google Cardboard before class ends
- **imperative knowledge** is a **recipe** or “how-to”.
  1) Students sign up for raffle
  2) Ana opens her IDE
  3) Ana chooses a random number between 1^{st} and n^{th} responder
  4) Ana finds the number in the responders sheet. Winner!

A NUMERICAL EXAMPLE

- square root of a number `x` is `y` such that `y*y = x`
- recipe for deducing square root of a number `x` (16)
  1) Start with a **guess**, `g`
  2) If `g*g` is **close enough** to `x`, stop and say `g` is the answer
  3) Otherwise make a **new guess** by averaging `g` and `x/g`
  4) Using the new guess, **repeat** process until close enough

g	g*g	x/g	(g+x/g)/2
3	9	16/3	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

WHAT IS A RECIPE

1) sequence of simple **steps**
2) **flow of control** process that specifies when each step is executed
3) a means of determining **when to stop**

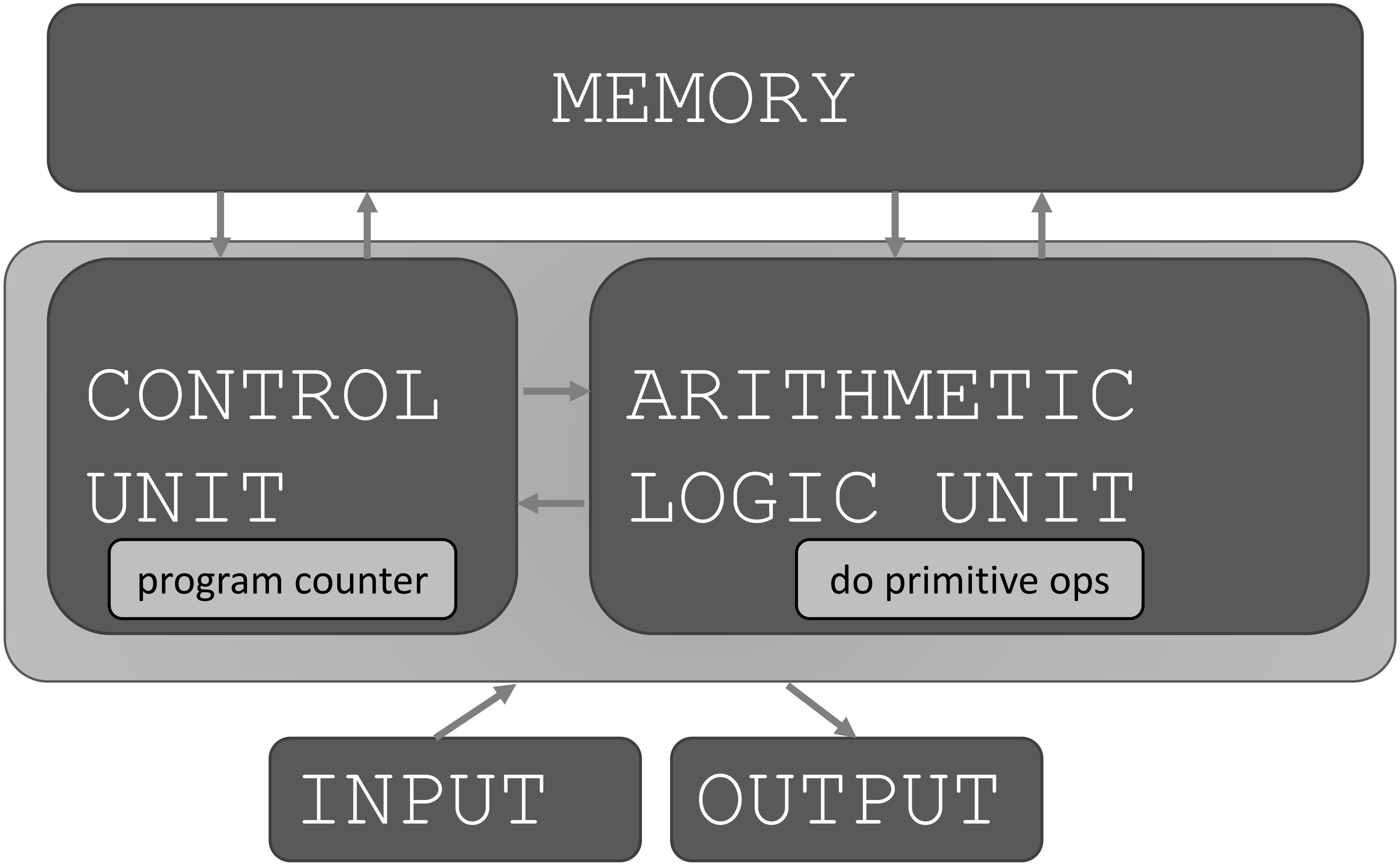
1+2+3 = an **algorithm**!

COMPUTERS ARE MACHINES

- how to capture a recipe in a mechanical process
- **fixed program** computer
  - calculator
- **stored program** computer
  - machine stores and executes instructions

von neumann architecture

BASIC MACHINE ARCHITECTURE



STORED PROGRAM COMPUTER

- sequence of **instructions stored** inside computer
  - built from predefined set of primitive instructions
    1) arithmetic and logic
    2) simple tests
    3) moving data
- special program (interpreter) **executes each instruction in order**
  - use tests to change flow of control through sequence
  - stop when done

BASIC PRIMITIVES

- Turing showed that you can **compute anything** using 6 primitives
- modern programming languages have more convenient set of primitives
- can abstract methods to **create new primitives**
- anything computable in one language is computable in any other programming language

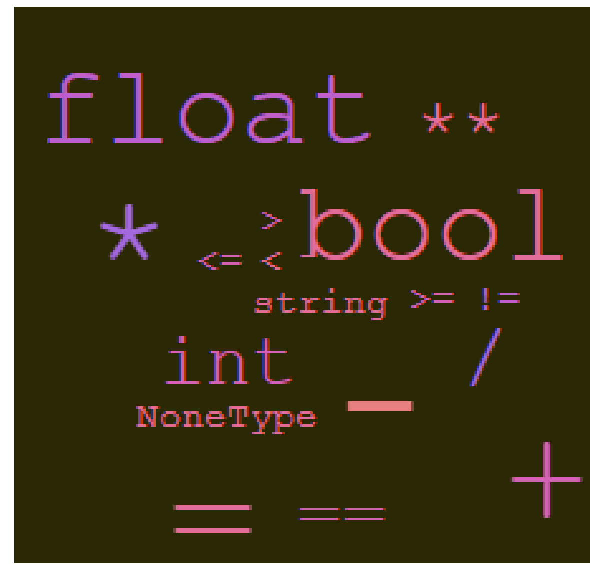
CREATING RECIPES

- a programming language provides a set of primitive **operations**
- **expressions** are complex but legal combinations of primitives in a programming language
- expressions and computations have **values** and meanings in a programming language

ASPECTS OF LANGUAGES

- **primitive constructs**
  - English: words
  - programming language: numbers, strings, simple operators

Word Cloud copyright Michael Twardos, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.



Word Cloud copyright unknown, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.

ASPECTS OF LANGUAGES

- **syntax**
  - English: `"cat dog boy"` → not syntactically valid
    `"cat hugs boy"` → syntactically valid
  - programming language: `"hi"5` → not syntactically valid
    `3.2*5` → syntactically valid

ASPECTS OF LANGUAGES

- **static semantics** is which syntactically valid strings have meaning
  - English: `"I are hungry"` → syntactically valid
    but static semantic error
  - programming language: `3.2*5` → syntactically valid
    `3+"hi"` → static semantic error

ASPECTS OF LANGUAGES

- **semantics** is the meaning associated with a syntactically correct string of symbols with no static semantic errors
  - English: can have many meanings `"Flying planes can be dangerous"`
  - programming languages: have only one meaning but may not be what programmer intended

(powerful + dumb!)

WHERE THINGS GO WRONG

- **syntactic errors**
  - common and easily caught
- **static semantic errors**
  - some languages check for these before running program
  - can cause unpredictable behavior
- no semantic errors but **different meaning than what programmer intended**
  - program crashes, stops running
  - program runs forever
  - program gives an answer but different than expected

PYTHON PROGRAMS

- a **program** is a sequence of definitions and commands
  - definitions **evaluated**
  - commands **executed** by Python interpreter in a shell
- **commands** (statements) instruct interpreter to do something
- can be typed directly in a **shell** or stored in a **file** that is read into the shell and evaluated
  - Problem Set 0 will introduce you to these in Anaconda

OBJECTS

- programs manipulate **data objects**

not just "what kind of data is this?"

- objects have a **type** that defines the kinds of things programs can do to them
  - Ana is a human so she can walk, speak English, etc.
  - Chewbacca is a wookie so he can walk, “mwaaarhrhh”, etc.
- objects are
  - scalar (cannot be subdivided)
  - non-scalar (have internal structure that can be accessed)

SCALAR OBJECTS

- `int` – represent **integers**, ex. `5`
- `float` – represent **real numbers**, ex. `3.27`
- `bool` – represent **Boolean** values `True` and `False`
- `NoneType` – **special** and has one value, `None`
- can use `type()` to see the type of an object

```
>>> type(5)
int
>>> type(3.0)
float
```

what you write into the Python shell

what shows after hitting enter

TYPE CONVERSIONS (CAST)

- can **convert object of one type to another**
- `float(3)` converts integer `3` to float `3.0`
- `int(3.9)` truncates float `3.9` to integer `3`

PRINTING TO CONSOLE

- to show output from code to a user, use `print` command

```
In [11]: 3+2
Out[11]: 5
```

"Out" tells you it's an interaction within the shell only

```
In [12]: print(3+2)
5
```

No "Out" means it is actually shown to a user, apparent when you edit/run files

EXPRESSIONS

- **combine objects and operators** to form expressions
- an expression has a **value**, which has a type
- syntax for a simple expression

```
<object> <operator> <object>
```

OPERATORS ON ints and floats

- `i+j` → the **sum**
- `i-j` → the **difference**
- `i*j` → the **product**
- `i/j` → **division**

if both are ints, result is int
if either or both are floats, result is float

result is float

modulo

- `i%j` → the **remainder** when `i` is divided by `j`
- `i**j` → `i` to the **power** of `j`

SIMPLE OPERATIONS

- parentheses used to tell Python to do these operations first
- **operator precedence** without parentheses
  - **
  - *
  - /
  - + and – executed left to right, as appear in expression

use parentheses

BINDING VARIABLES AND VALUES

- equal sign is an **assignment** of a value to a variable name

variable value

```
pi = 3.14159
pi_approx = 22/7
```

- gives a float value

- value stored in computer memory
- an assignment binds name to value
- retrieve value associated with name or variable by invoking the name, by typing `pi`

ABSTRACTING EXPRESSIONS

- why **give names** to values of expressions?
- to **reuse names** instead of values
- easier to change code later

```
pi = 3.14159
radius = 2.2
area = pi*(radius**2)
```

PROGRAMMING vs MATH

- in programming, you do not "solve for x"

```
pi = 3.14159
radius = 2.2
# area of circle
area = pi*(radius**2)
radius = radius+1
```

an assignment
* expression on the right, evaluated to a value
* variable name on the left
* equivalent expression to `radius = radius + 1`
is `radius += 1`

CHANGING BINDINGS

- can **re-bind** variable names using new assignment statements
- previous value may still stored in memory but lost the handle for it
- value for area does not change until you tell the computer to do the calculation again

```
pi = 3.14
radius = 2.2
area = pi*(radius**2)
radius = radius+1
```

+=