Programming

- it's hard to do the programming to get something done
- details are hard to get right, very complicated, finicky
- not enough skilled people to do what is needed
- therefore, enlist machines to do some of the work
  - leads to programming languages

- it's hard to manage the resources of the computer
- hard to control sequences of operations
- in ancient times, high cost of having machine be idle
- therefore, enlist machines to do some of the work
  - leads to operating systems
Evolution of programming languages

- **1940's: machine level**
  - use binary or equivalent notations for actual numeric values

- **1950's: "assembly language"**
  - names for instructions: ADD instead of 0110101, etc.
  - names for locations: assembler keeps track of where things are in memory; translates this more humane language into machine language
  - this is the level used in the "toy" machine
  - needs total rewrite if moved to a different kind of CPU

```assembly
loop  get           # read a number
    ifzero  done  # no more input if number is zero
    add     sum   # add in accumulated sum
    store   sum   # store new value back in sum
    goto    loop  # read another number

done  load    sum   # print sum
      print
      stop

sum   0   # sum will be 0 when program starts
```
Evolution of programming languages, 1960's

- "high level" languages: Fortran, Cobol, Basic
  - write in a more natural notation, e.g., mathematical formulas
  - a program ("compiler", "translator") converts into assembler
  - potential disadvantage: lower efficiency in use of machine
  - enormous advantages:
    - accessible to much wider population of users
    - portable: same program can be translated for different machines
    - more efficient in programmer time

```
sum = 0
10 read(5,*) num
if (num .eq. 0) goto 20
   sum = sum + num
   goto 10
20 write(6,*) sum
   stop
end
```
Evolution of programming languages, 1970's

- "system programming" languages: C
  - efficient and expressive enough to take on any programming task
    writing assemblers, compilers, operating systems
  - a program ("compiler", "translator") converts into assembler
  - enormous advantages:
    accessible to much wider population of programmers
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```c
#include <stdio.h>
main() {
    int num, sum = 0;

    while (scanf("%d", &num) != -1 && num != 0)
        sum += num;
    printf("%d\n", sum);
}
```
C code compiled to assembly language  (x86, Mac)

```c
#include <stdio.h>
main() {
    int num, sum = 0;
    while (scanf("%d", &num) != -1 && num != 0)
        sum = sum + num;
    printf("%d\n", sum);
}
(You are not expected to understand this!)
```

```assembly
Ltmp2:
    movl $0, -8(%rbp)
    movl $0, -12(%rbp)
    jmp LBB1_2
LBB1_1:
    movl -12(%rbp), %eax
    movl -8(%rbp), %ecx
    addl %eax, %ecx
    movl %ecx, -8(%rbp)
LBB1_2:
    leaq -12(%rbp), %rax
    xorb %cl, %cl
    leaq L_.str(%rip), %rdx
    movq %rdx, %rdi
    movq %rax, %rsi
    movb %cl, %al
    callq _scanf
    movl %eax, %ecx
    cmpl $-1, %ecx
    je LBB1_4
    movl -12(%rbp), %eax
    cmpl $0, %eax
    jne LBB1_1
LBB1_4:
```
C code compiled to assembly language  (ARM64)

```c
#include <stdio.h>

main() {
    int num, sum = 0;
    while (scanf("%d", &num) != -1 && num != 0)
        sum = sum + num;
    printf("%d\n", sum);
}
```

(You are not expected to understand this!)
Evolution of programming languages, 1980's

• "object-oriented" languages: C++
  – better control of structure of really large programs
    better internal checks, organization, safety
  – a program ("compiler", "translator") converts into assembler or C
  – enormous advantages:
    portable: same program can be translated for different machines
    faster, cheaper hardware helps make this happen

```cpp
#include <iostream>
main() {
  int num, sum = 0;

  while (cin >> num && num != 0)
    sum += num;
  cout << sum << endl;
}
```
Evolution of programming languages, 1990's

- "scripting", Web, component-based, ...:
  - Java, Perl, Python, Ruby, Visual Basic, Javascript, ...
    - write big programs by combining components already written
    - often based on "virtual machine": simulated, like fancier toy computer
    - enormous advantages:
      - portable: same program can be translated for different machines
      - faster, cheaper hardware helps make this happen

```javascript
var sum = 0; // javascript
var num = prompt("Enter new value, or 0 to end")
while (num != 0) {
    sum = sum + parseInt(num)
    num = prompt("Enter new value, or 0 to end")
}
alert("Sum = " + sum)
```
I just typed `import antigravity`.

That’s it?

...I also sampled everything in the medicine cabinet for comparison.

But I think this is the Python.

I dunno... dynamic typing? whitespace?

Come join us! Programming is fun again! It’s a whole new world up here!

But how are you flying?

I learned it last night! Everything is so simple!

Hello, world is just print "Hello, world!"
Why so many programming languages?

- every language is a tradeoff among competing pressures
  - reaction to perceived failings of others; personal taste
- notation is important
  - "Language shapes the way we think and determines what we can think about."
    Benjamin Whorf
  - the more natural and close to the problem domain, the easier it is to get the machine to do what you want
- higher-level languages hide differences between machines and between operating systems
- we can define idealized "machines" or capabilities and have a program simulate them -- "virtual machines"
  - programming languages are another example of Turing equivalence
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.