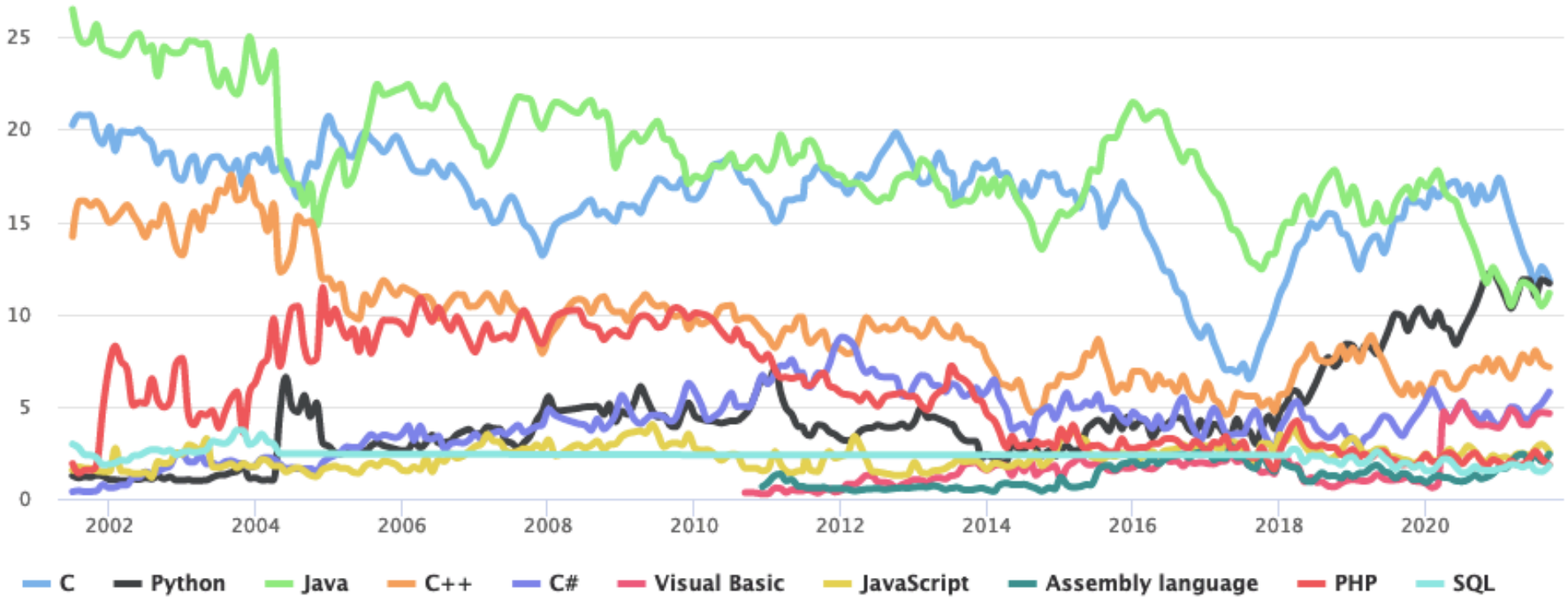
Lecture 8:
Programming Languages



Programming

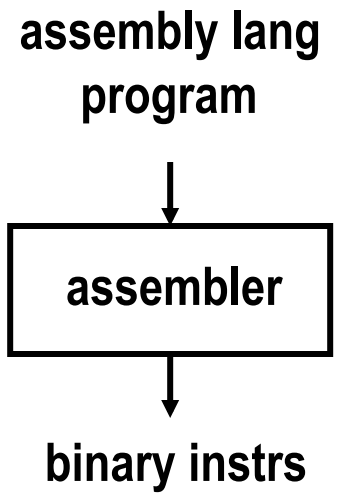
- it's hard to do the programming to get something done
- details are hard to get right, very complicated, finicky
- not enough skilled people to do what is needed
- therefore, enlist machines to do some of the work
  - leads to programming languages

- 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

```
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

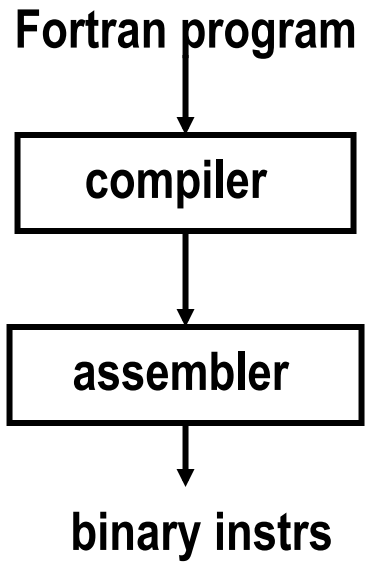
- "high level" languages: Fortran, Cobol, Basic
  - write in a more natural notation, e.g., mathematical formulas
  - a program ("compiler", "translator") converts into assembler
  - potential disadvantage: lower efficiency in use of machine
  - enormous advantages:

    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
```

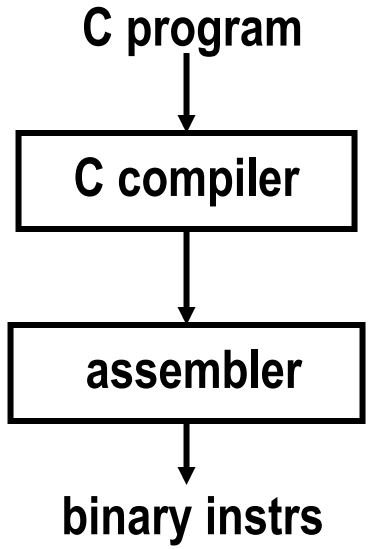


Evolution of programming languages, 1970's

- "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

```
#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)

```
#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!)

```
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:
```

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

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

   while (cin >> num && num != 0)
        sum += num;
   cout << sum << endl;
}
```

Bjarne Stroustrup
1950-

Evolution of programming languages, 1990's

- **"scripting", Web, ...:**

  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

Java (1995)

```
import java.util.*;
class Addup {
   public static void main (String [] args) {
      Scanner keyboard = new Scanner(System.in);
      int num, sum;
      sum = 0;
      num = keyboard.nextInt();
      while (num != 0) {
         sum = sum + num;
         num = keyboard.nextInt();
      }
      System.out.println(sum);
   }
}
```

James Gosling
1955-

JavaScript (1995)

```
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)
```

Brendan Eich
1961-

Python (1990)

```
sum = 0
num = input()
while num != '0':
    sum = sum + int(num)
    num = input()
print(sum)
```

Guido van Rossum
1956-

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