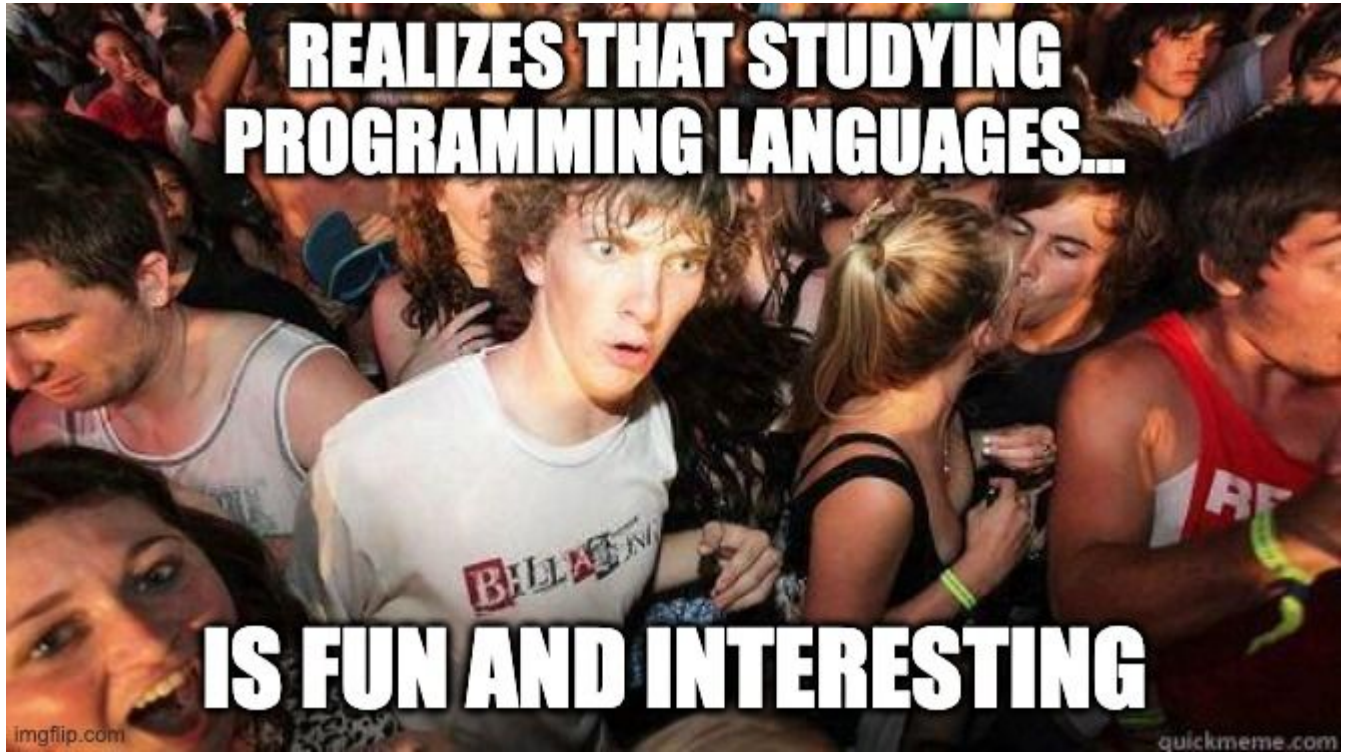
CS131: Programming Languages
Fall 2023

Carey Nachenberg
climberkip@gmail.com; ENG VI 299



Useful Information

Mon: 4-5pm, Wed: 1-2pm
Location: Eng VI 364

Our Class Hub

https://bruinlearn.ucla.edu/courses/168646

Syllabus, Assignments, PPTs, Lectures Notes, Solutions

https://ucla-cs-131.github.io/fall-23-website

Your Questions – Our Answers

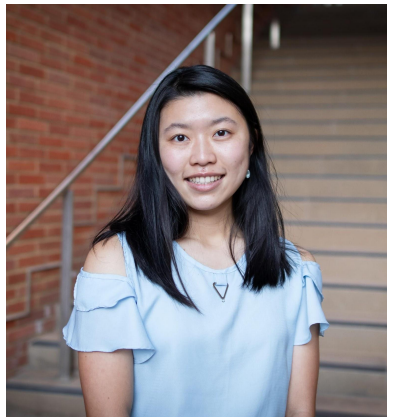
https://campuswire.com/p/G8DA58F59

Verification Code: 5216

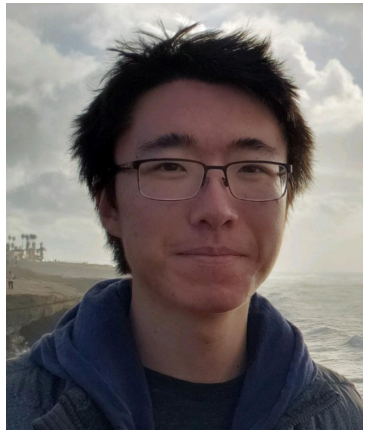
Meet Our Team

Carey Nachenberg

Andrey Storozhenko
Discussions 1D and 1G



Bonnie Liu
Discussions 1B and 1H

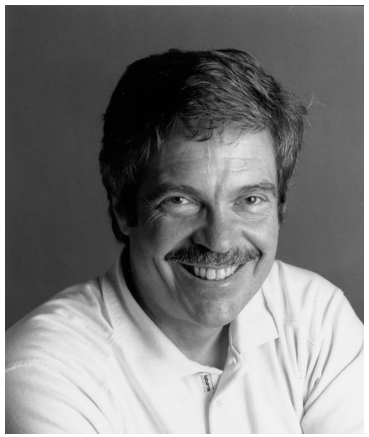


Brian Wang
Discussions 1C and 1F

Justin Cui
Discussions 1A and 1E

Contributors

I couldn't have created this course without the hours of explanations/content/feedback from the following folks:


Alan Kay

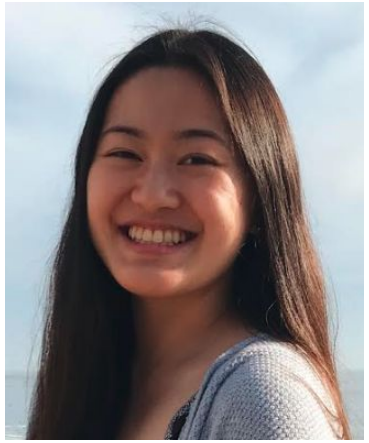

Angelina Lue


Devan Dutta


James Wu


Kevin Tan

Kyle Chui


Maggie Li


Matthew Wang


Paul Eggert


Timothy Gu


Todd Millstein

Guess what?

I love Comic Sans soooo much...

I'm gonna use it for 131 as well! 

OK, just kidding... We'll use Corbel!

What's CS131 About?

Languages come and go, but they're all built upon a common set of paradigms and building-blocks.

Our goal in CS131 is to learn these paradigms and building-blocks, understanding the implications of each.

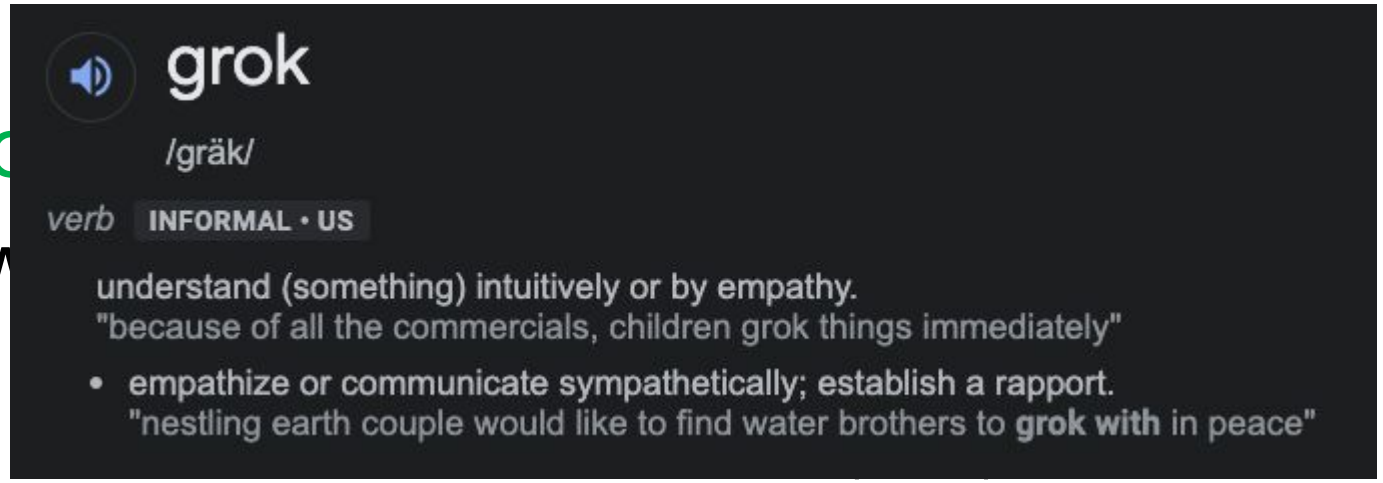
This will enable you to pick up new languages fast, select the best language for a problem, and even invent your own language!

Your Goal by Week 10?

Be able to explain – in your own words - the major language paradigms and building blocks, and their tradeoffs.

Given any new language*, be able to grok how it works and write correct, efficient code in a matter of hours.

Gain fluenc
basic proficiency w



Assignments and Exams

10% 30% 25% 35%

CS131 has homework, projects, a midterm and a final exam.

Homewo arn.

ew language!

Our Midterm is on Thursday, Nov 7th from 6 to 8pm!

your understanding of key programming language concepts

your ability to analyze languages you've never seen before

your ability to write correct code in our focus languages

Be Wary of Online Resources

13 Answers

Sorted by: Highest score (default)

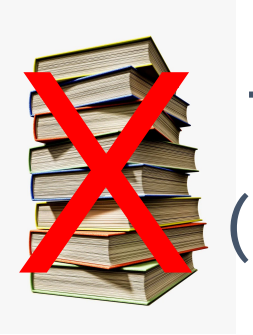
Python is strongly, dynamically typed.

513

- **Strong** typing means that the type of a value doesn't change in unexpected ways. A string containing only digits doesn't magically become a number, as may happen in Perl. Every change of type requires an explicit conversion.

Warning: Stack Overflow, Medium, etc. have lots of incorrect explanations!

Learning Resources



There is no required textbook
(I couldn't find any good ones!)

But… we'll be posting slides and lecture notes, and…

Side-by-side Language Comparisons

Rosetta Code
http://rosettacode.org/

Hyperpolyglot
https://hyperpolyglot.org/

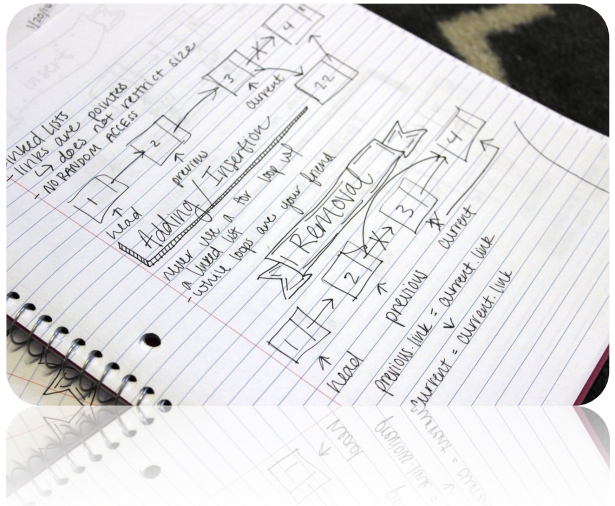
Gentle Language Introductions

Learn You a Haskell for Great Good
http://learnyouahaskell.com/

W3 Schools Python Intro
https://www.w3schools.com/python/

tutorialspoint
https://www.tutorialspoint.com/prolog

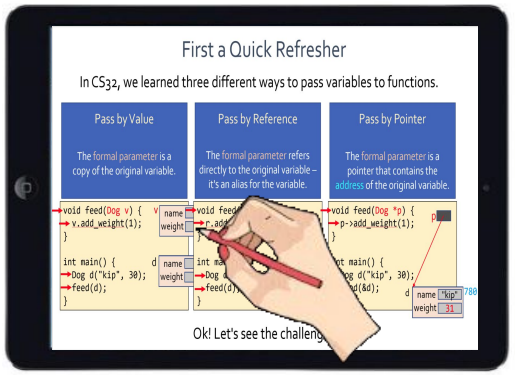
Attention Note Takers!



Many students are used to taking detailed notes during lecture…

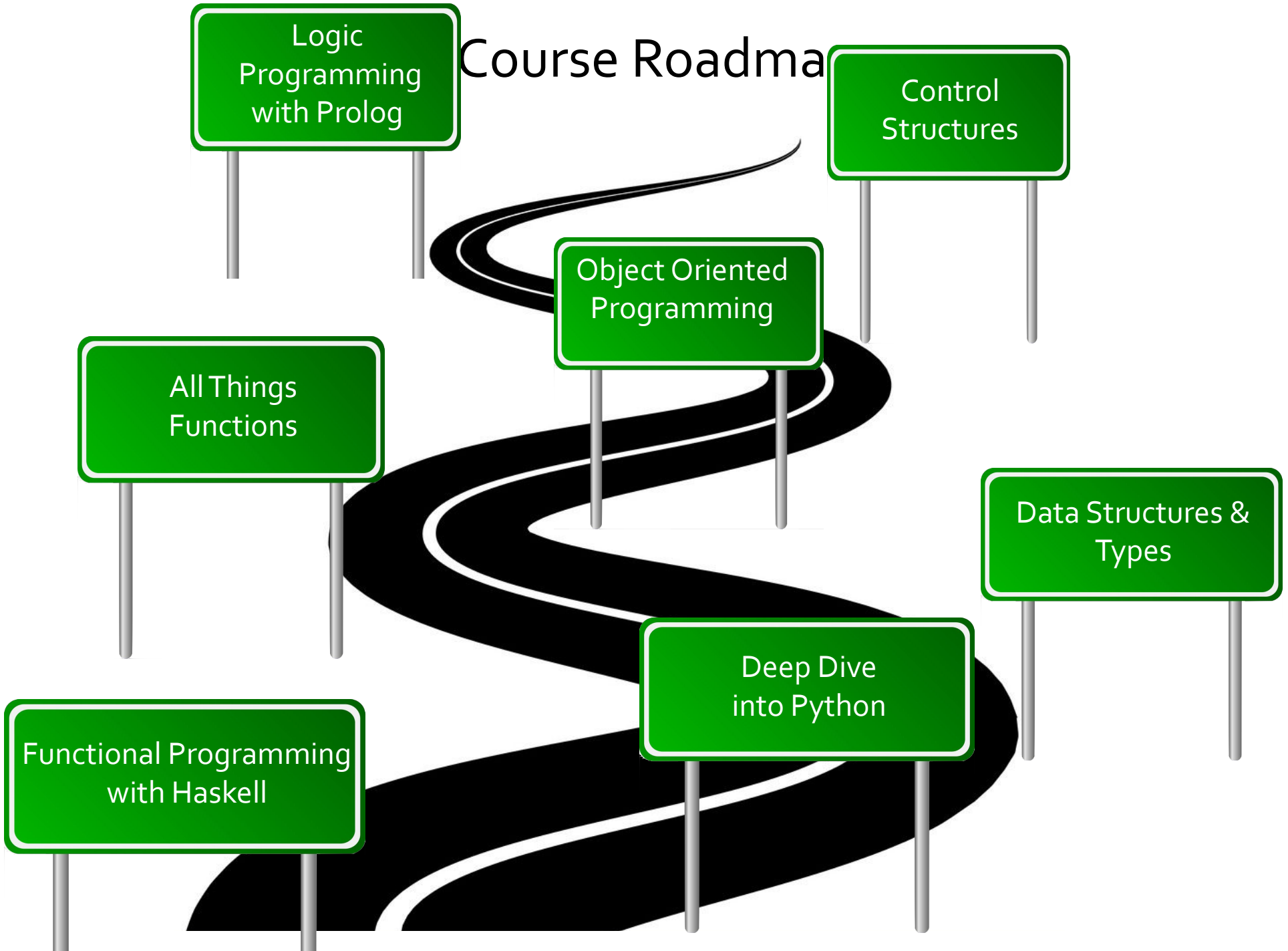
If that's you, you may find it difficult to keep up since I show lots of code examples.

Code examples

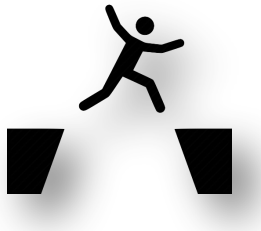


Suggestion: Download our lecture notes or slides ahead of time and take notes on them…

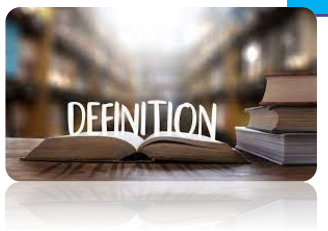
Course Roadma



Ok, let's go!



What is a Programming Language?



A programming language is a st[illegible]ctured system of communication designed to express com[illegible]

A programming language can be use[illegible] that specify the behavior of a computer.

When you see this bar, it means:

You have 1 minute to discuss with your neighbor and come up with an answer.

In CS131, we'll be focusing on high-level, "universal" languages like C++ and Python – that is, those capable of implementing pretty much any valid program.

What were the world's first *high-level* programming languages?

First Interpreted Language:

Shortcode

John Mauchly '49

A program was a set of statements – each a math expression you wanted to compute.

First Compiled Language:

A-0

Grace Hopper '51

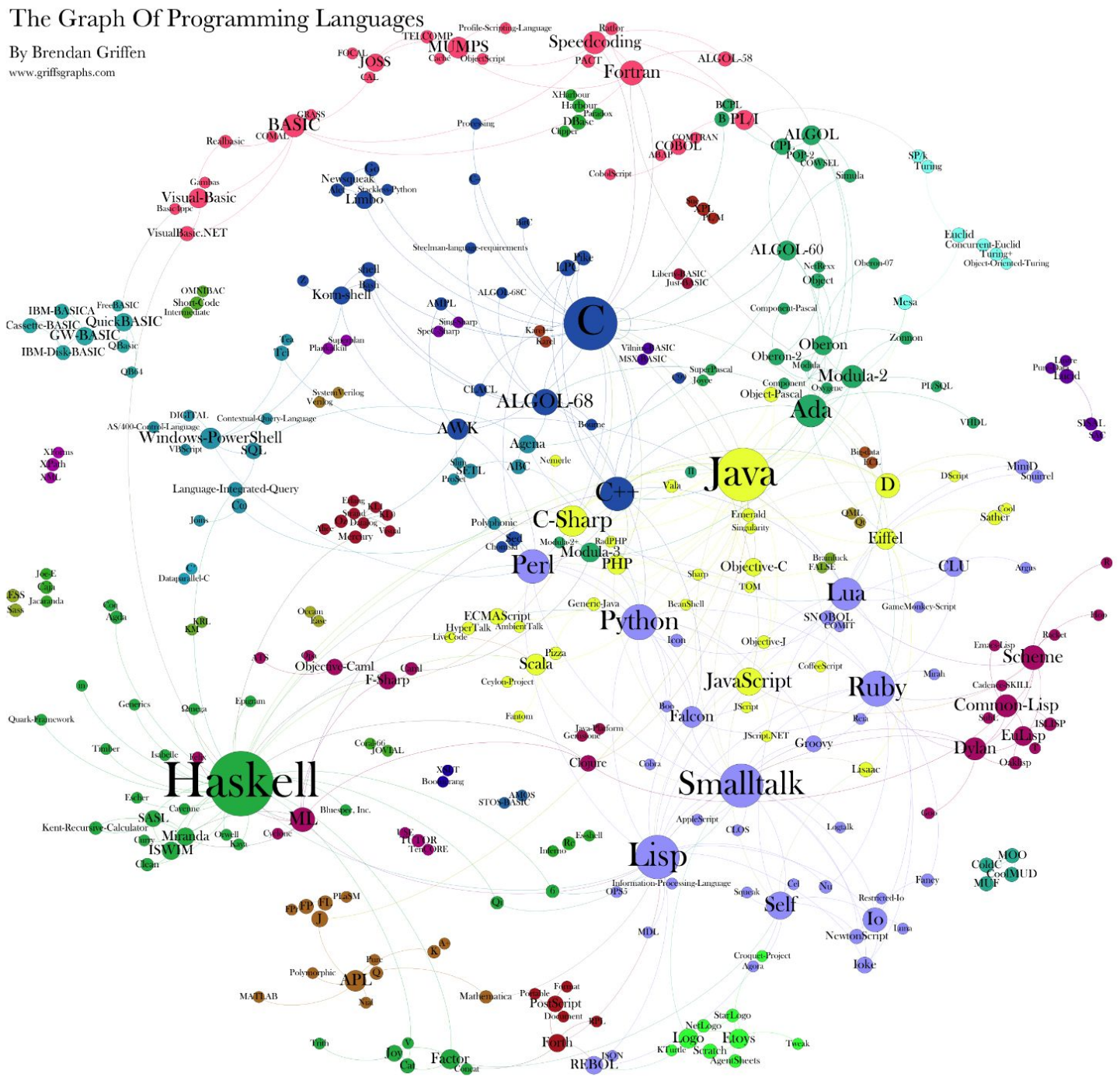
A program was a sequence of subroutine calls with arguments. The compiler converted the program into machine code.

How many different languages exist?

Wikipedia lists 687 different programming languages!

But there are thousands!

Here's a nice visualization showing how interconnected they are!



Why So Many Languages?

Each language is designed to address a particular category of problem, research area, or business domain.

JavaScript was designed for front-end use cases

Simula was developed to build simulations

SQL was designed to query databases

There are also thousands of Domain-Specific Languages (DSLs) that folks have designed to accelerate problem-solving for specific use cases!

How much can a language's design impact problem-solving speed?

Let's find out!

Challenge: How Small is the Smallest Spell Checker?

Given an array of sentences and an array holding an English dictionary, what's the minimum number of lines of code it would take to build a spell checker?

You can choose any language you like!

Assume standard sentence delimiters: comma, period, space, …

```
results = spell_check(s, d);
// results should hold: “admyt” and “lyt”
```

s

“I must admyt, it is lit”
“So LYT… I can't take it!”
"Lyt; AdMyT it!"

d

“a”

Answer: 3 lines!

Assume the dictionary is all lower case and in alphabetical order.

```
# Python spell checker using list comprehensions
import re

def spell_check(s, d):
 return {w for w in [w for ln in s for w in re.split(r'[. ,!;"]',ln)]
                 if w.lower() not in d and w != ''}
```

Writing an essay in different languages

PYTHON JAVA

Programming and Starbucks?!?!

&

Programming Languages are Like Starbucks Drinks

Just as we have different classes of drinks…

Coffee Drinks

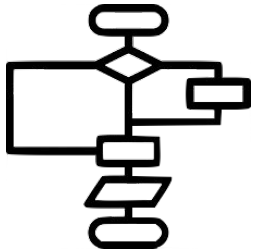
Hot Chocolate

Teas

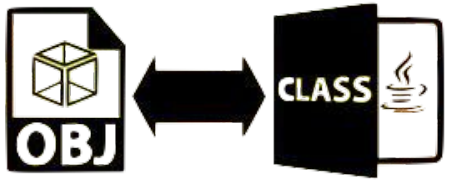
Ciders

There are different programming language paradigms:

Imperative

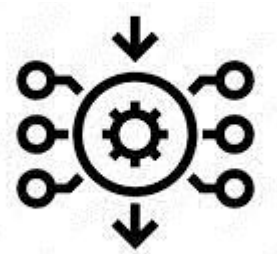


Object Oriented



Functional

Logic



The Major [illegible]ms

This quarter, we're going to [illegible]uage paradigms!

And just like coffee drinks, many languages are hybrids!

C

Imperative Languages

What you're used to - programs are a made up of statements, loops, and mutable variables.

```
int a = 0;
while (a++ < 100)
    cout << a << endl;
```

C++

Smalltalk

OOP Languages

Programs are organized into classes and objects which send messages, e.g., obj.msg(x), to each other to get things done.

```
class Dog { ... };
```

Scala

Haskell

Functional Languages

Programs are made up of math-like functions. No statements or iteration, only expressions, functions, constants and recursion!

```
f(n) = if n>0 then n*f(n-1)
                else 1
```

Prolog

Logic Languages

Programs define a set of facts, e.g.:
like(dogs, meat), category(beef, meat).

Programs also define rules:
likes(A,C): if like(A,B) and category(C, B)

You then query: likes(dogs, beef)?

Programming Languages are Like Starbucks Drinks

Coffee Choices

Coffee: light, medium, dark-roast

Milk: 2%, skim, whole, soy, almond, oat

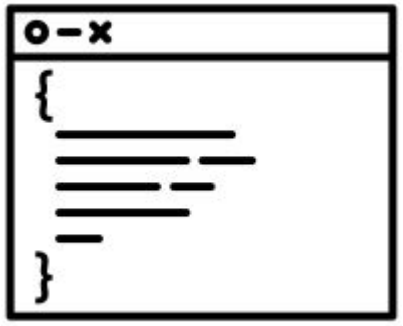
Flavoring: chocolate, caramel, vanilla

Sweetener: sugar, Splenda, stevia

Starbucks lets you customize coffee across key dimensions…

For instance, C++ made the following design choices…

But Python made a totally different set of choices!



Lang[illegible]hoices

Type checking: static typing, dynamic typing

Parameter passing: by-value, by-reference, by-pointer, by-object reference, by-name

Scoping: lexical scoping, dynamic scoping

Memory management: manual, automatic

…

[illegible]ming language makes choices across dozens of dimensions.

Language Dimensions: More Examples

Pro: Detects typing bugs at compile time, when they're easy to fix
Con: More verbose code

```
int a;  // variable declared first!
a = 5;  // then it can be assigned
```

vs.

Pro: Faster dev, simpler code
Con: Less readable code, may detect errors only at runtime

```
a = 5   # 1st assignm't creates var
```

Pro: Makes for simpler code
Con: May hide conversion bugs

```
int a = 5;
double b = a;  // OK!
```

vs.

Pro: Detects bugs more easily
Con: More verbose code

```
int a = 5;
double b = (double)a;  // OK
```

Pro: Catches out-of-bounds bugs
Con: Runtime checks slow execution!

```
int arr[] = new i...;

cout << arr[10]; // Blocked!
```

vs.

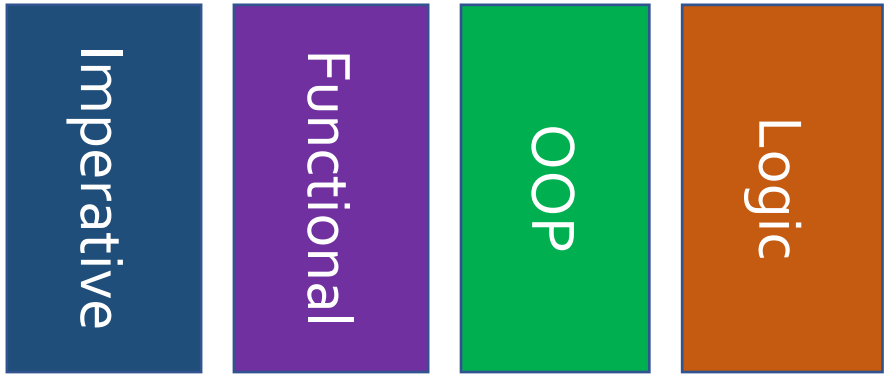
Pro: Faster
Con: Permits nasty memory bugs!

```
int arr[10];

cout << arr[10]; // Allowed! Eek!
```

Summarizing Our Learning Approach

We'll deep-dive into the major paradigms



We'll go wide and survey the spectrum of design options

Variable Scoping	Static	Dynamic			
Parameter Passing	By-value	By-reference	By-object ref	By-name	By-pointer
Memory Management	Manual	Automatic			
...					
Typing Approach	Static	Dynamic	Weak	Strong	Gradual

An important skill is the ability to logically deduce which design choices each language has made.

So throughout the quarter, we'll be training ourselves to do this with challenges.

Let's see an example.

First a Quick Refresher

In CS32, we learned three different ways to pass variables to functions.

Pass by Value

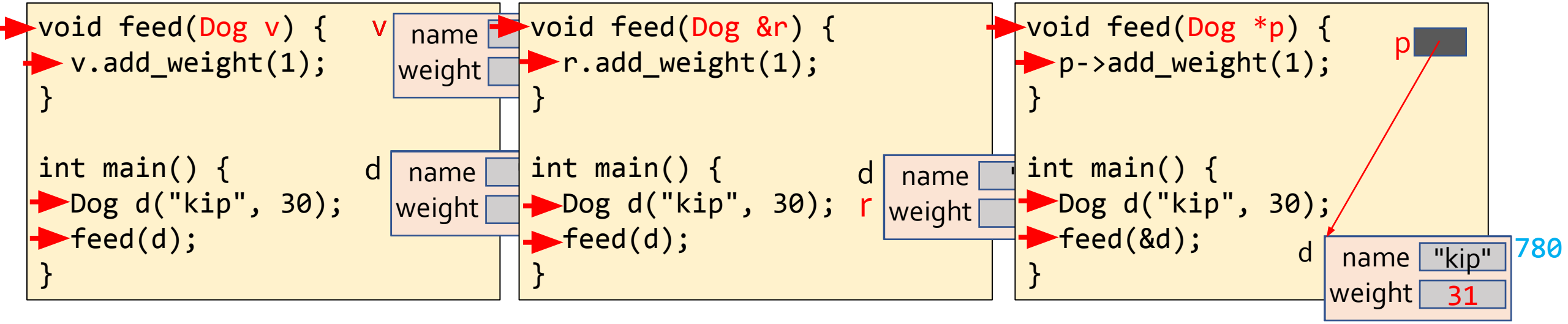
The formal parameter is a copy of the original variable.

Pass by Reference

The formal parameter refers directly to the original variable – it's an alias for the variable.

Pass by Pointer

The formal parameter is a pointer that contains the address of the original variable.



Ok! Let's see the challenge!

Classify That Langua

That didn't change the original variable's value!

The code below ... pri... the following:

```
Carey's major is Econ.
```

Clearly, change_major() ...fully changes ...or.

Reassigning s only changes the local pointer, not the original object!

...igning ...er to a new object?

So, this language must NOT be passing arguments by reference, or student1's name *and* major would have changed.

Tech... by ... passing by pointer in C++!

```
# ... changes
def ...
  s ... _major('Econ')
  # ...es life choices, so starts from scratch
  ... new('Carence', 'Music')
  e...

# main program
student1 = Student.new('...
life_changes(student1)
student1.print_my_detail...
```

Let's see how it works!

s

student1

So, this language must NOT be passing arguments by value, or student1's major would still be CompSci.

By valu...

By p...

This changes student1's major.

name	'Carey'
major	'CompSci' / 'Econ'

This is Ruby!

We'll Deep Dive Into Three Languages

Of the thousands of languages, this quarter we'll deep-dive into Haskell, Prolog, and Python.

While you're familiar with imperative programming and OOP (via C++), we need to introduce you to the other major language paradigms.

Haskell is one of the simplest and easiest-to-grok functional languages.

Prolog is one of the few examples of logic languages.

Python is a dynamically-typed language (and its super useful).

Next, let's learn what it means to specify the details of a programming language!

eded to specify the details of a ogramming Language?

Some language specs, like the one for C++, are written in plain English!!!

Perl

Patterns

$$\frac{longcon = strid_1.\cdots.strid \quad E, v' \vdash at}{E, v \vdash longexcon}$$

$$\frac{E(longexcon) = en \qquad v \notin \{en\} \times \mathrm{Val}}{E, v \vdash longexcon\ atpat \Rightarrow \mathrm{FAIL}} \quad (157)$$

$$\frac{s(a) = v \qquad s, E, v \vdash atpat \Rightarrow VE/\mathrm{FAIL}, s}{s, E, a \vdash \texttt{ref}\ atpat \Rightarrow VE/\mathrm{FAIL}, s} \quad (158)$$

$$\frac{E, v \vdash pat \Rightarrow VE/\mathrm{FAIL}}{E, v \vdash var\ \texttt{as}\ pat \Rightarrow \{var \mapsto v\} + VE/\mathrm{FAIL}} \quad (159)$$

8.5.2.2 Incremen

1 The operand of prefi operand shall be an a result is the updated ++x is equivalent to for information on c

2 The operand of pref the properties of its decrement, see 8.5.1.

type of the type. The expression ors (8.5.18)

he operand of prefix -- and e: For postfix increment and

Most modern languages use an encoding called Extended Backus–Naur Form (EBNF) to define the language's syntax.

A languag area

Oth

And some languages have NO written spec at all (like Perl v1-v4)!

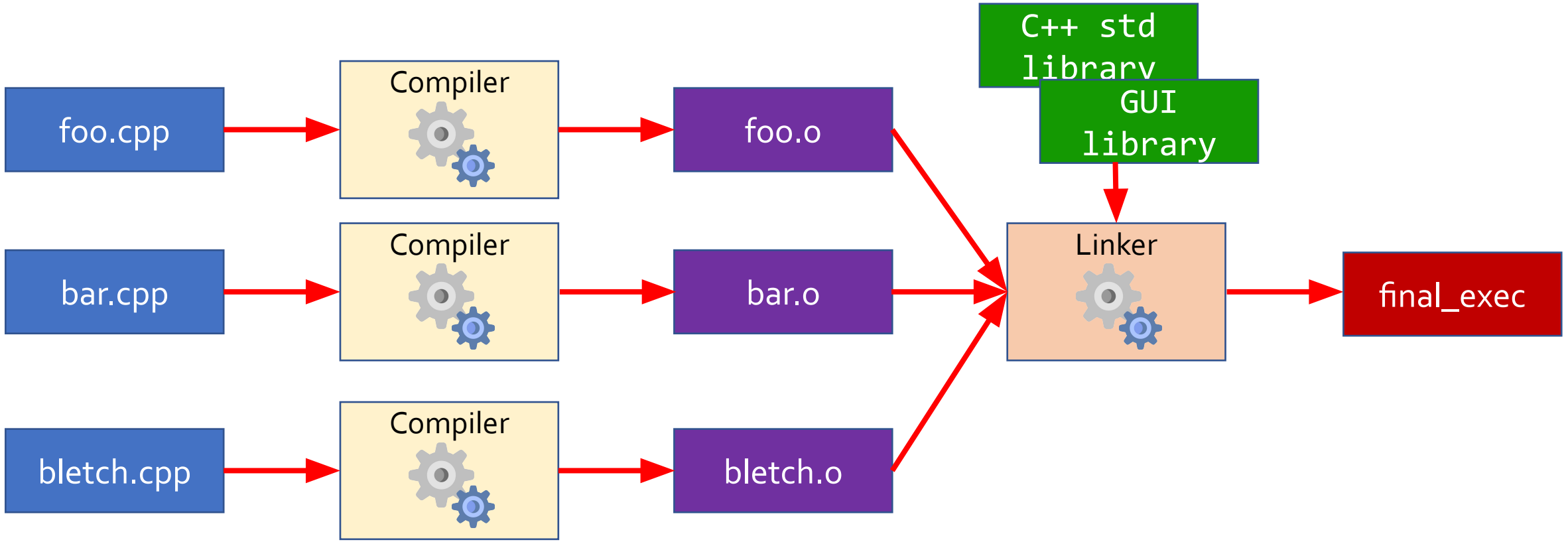
Whatever the compiler/ interpreter does is canonical!

Our Tools

What's a Compiler? What's a Linker?

A compiler is a program that translates program source into object modules (which are either machine language, or bytecode targeted at an interpreter).

A linker is a program that combines multiple object modules and libraries into a single executable file or library.



Does a Compiler W

e.g., that x and i are
both the
and comp
+ and

If the syntax is valid, the parser converts the tokens into
represents program's

The IR generator produces an abstract representation of the program – it's totally independent of the original language.

The representation could be a tree structure, or CPU-agnostic instructions.

Bytecode is a binary encoding – like a machine language, but not for a real CPU.

It is processed by an interpreter or just-in-time (JIT) compiler.

file.cpp

```
while(i<5)
  x = x + i*3;
```

```
func:  type func_name '(' params ')'
       '{'  { type var_name ';' } { stmt } '}'
stmt:  'if' '(' expr ')' stmt [ 'else' stmt ]
                                  [ assg ] ')' stmt
                                  or } ] ')' ';'
                                  pr | ...
```

Machin
or byt

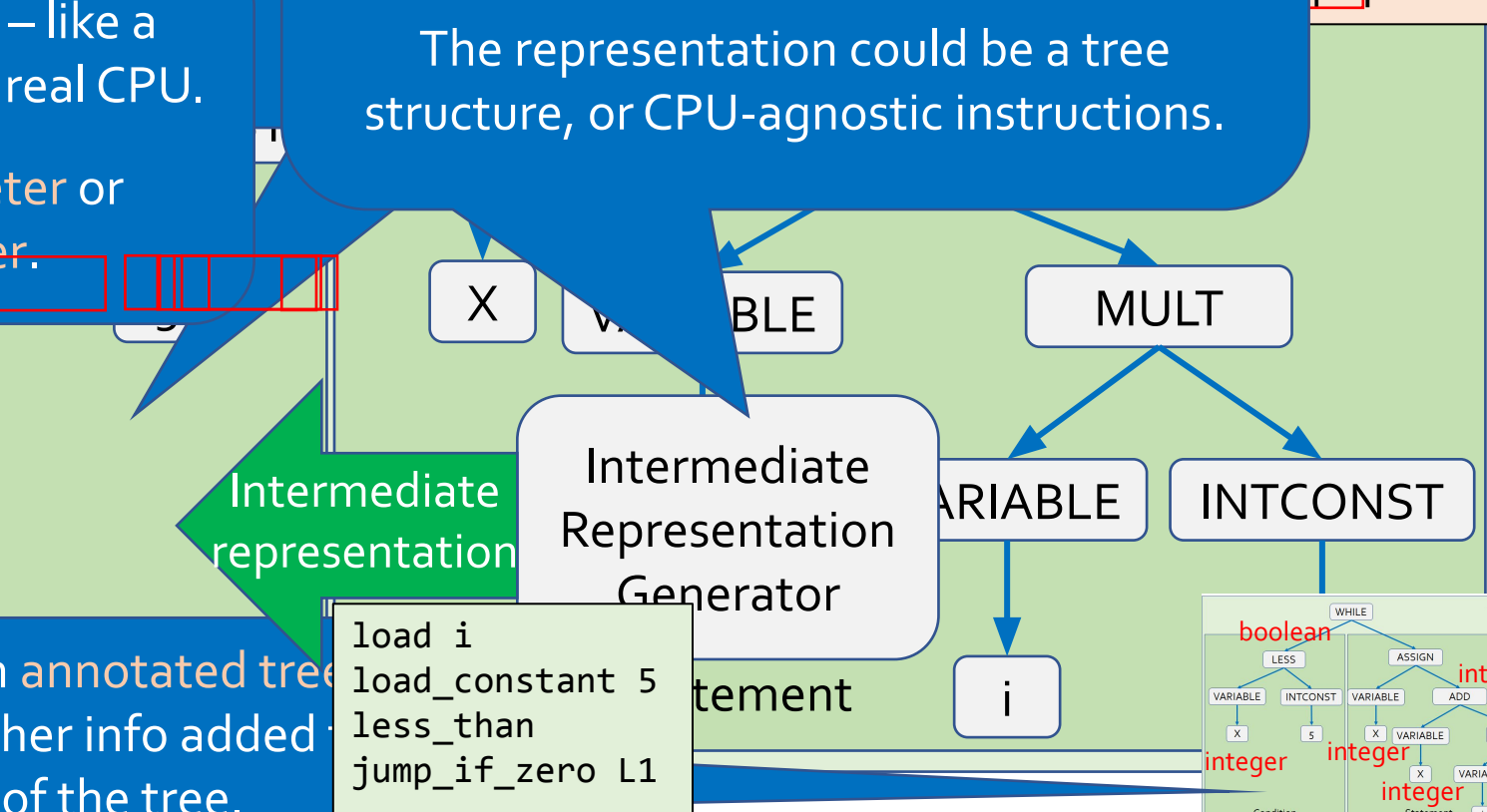
Intermediate representation

Intermediate Representation Generator

```
55 48 8
00 00 6
```

The output is an annotated tree with type and other info added to the nodes of the tree.

```
load i
load_constant 5
less_than
jump_if_zero L1
...
```

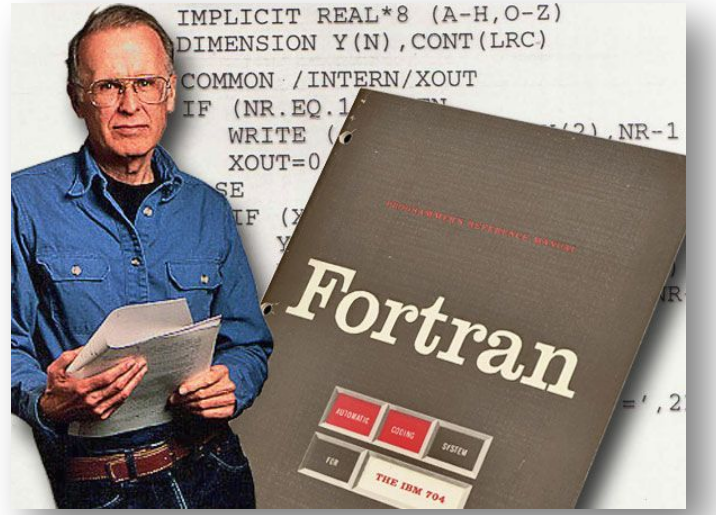


For Funzies

What was the world's first mainstream compiler (for Fortran) written in?

Answer: Assembly language.

Makes sense, right?
What else would it be written in?



John Backus

For Funzies, Part 2

What language are most C++ compilers written in?

Answer: C++?!?!? Through a process called "bootstrapping"

```
// C++ compiler in C++!
class Compiler {
 public:
   Compiler();
   void compile();
};
```

c++-compiler.cpp

to-c-translator.c

```
// C++ compiler in C
struct Compiler
  { ... };
void init(Compiler& c)
void compile(Compiler& c);
```

First you build a C++ to C translator… in some existing language like C.

c++-compiler.cpp

c++-to-c-translator.exe

C Compiler

c++compiler.exe

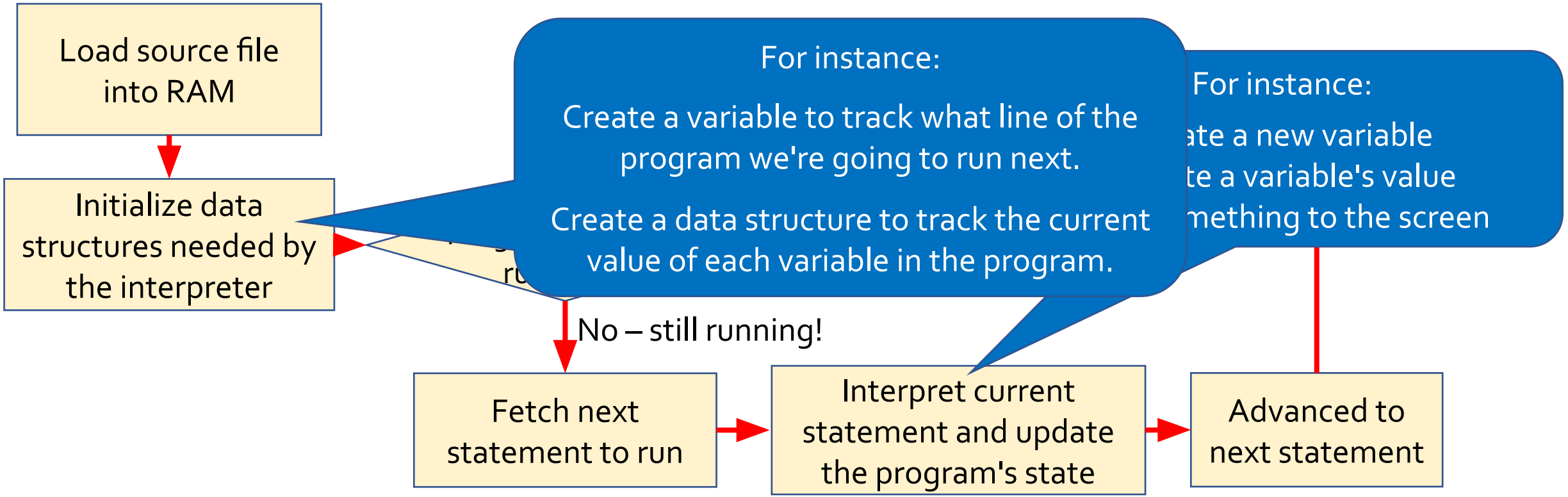
Now we have a working C++ compiler!

c++compiler.exe

What's an Interpreter?

An interpreter is a program that directly executes program statements, without requiring them to first be compiled into machine language.

How does an interpreter work?



What's an Interpreter?

An interpreter is a program that directly executes program statements, without requiring them to first be compiled into machine language.

Ok, let's build an interpreter for a simple programming language!

Consider the following simple program in our new language which:

initializes a variable called x to 10,
adds 20 to the variable,
prints the value of x,
and finally terminates.

```
set x 10
add x 20
print x
end
```

An Interpreter In One Slide!

```
// sp
vector<str
void inter et(const vec
  int cur_line = 0;
  bool termina d =
    vector<strin
    if (tokens[0] ==  end )
     terminated = true;
    else if (tokens[0] == "set")
     variable_to_value[
    else if (tokens[0] == "add")
     variable_to_value[tokens[1]] +=
    else if (tokens[0] == "print")
      cout << variable_to_value[tokens[1]] << endl;
    ++cur_line;
  }
}
```

```
n() {
  vector<string> program =
    {"set x 10",
     "add x 20",
     "print x",
     "end"};

  interpret(program); // 30
}
```

We'll need to track wh
our program we're

We also
track wh
has f

Let's use a map to associate each variable name, e.g., x, with

Let's assume we have a function that can split a
ring up into its parts.

er in a

which causes execu
to terminate.

First we'll parse our statement into separate tokens:
"set x 10" → {"set","x","10"}

Here's how we'll use it…

We'll start executing at line 0.

toi() converts
to an integer

while our program hasn't terminated itself, keep executing the next statement.

s see the details!

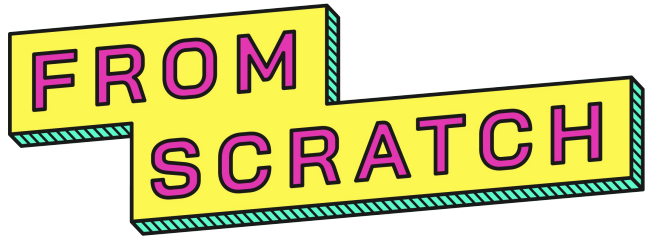
dd", we update the
g the specified amount.

variable's value
map and print it.

"x"

Finally, we advance to the next line in our program.

Interpreters are Soooo Cool

that you'll be building your own 

Our four class projects require you to build an interpreter for a new programming language!

And as we learn new language concepts, you'll add them to your interpreter!

Giving you a deep, hands-on understanding.

A Few Final Things…

Your first homework will be posted Wednesday of *this* week on Gradescope.

Until then, please install the Haskell compiler and Python 3.x interpreter.

A Few Final Things…

CS131 is a notoriously difficult class.

So ask questions…

Stop me if something's unclear…

Don't be afraid to ask for help.

Our goal is not just to cover the material, but for you to deeply understand it and have fun in the process.

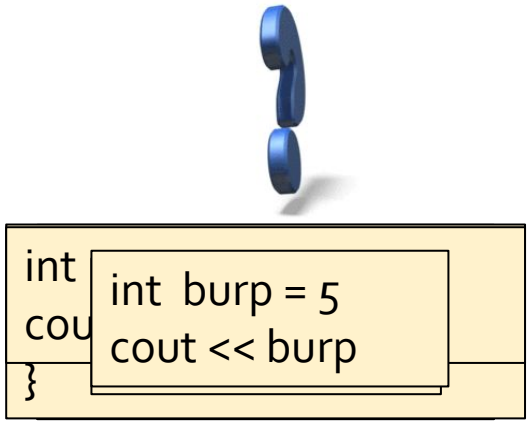
Onward to Functional Programming With Haskell!

Appendix

Syntax or Semantic Error?

Which errors will be detected by the parser vs. the semantic analyzer?

Detected by the Parser	Detected by the Semantic Analyzer



An Early History of Modern Languages

Machine Language

A program is a sequence of numbers which represent instructions, e.g., (0x6b=multiply); the instructions are directly interpreted by the CPU.

```
55 48 89 e5 c7 45 fc
05 00 00 00 6b 45 fc
03 89 45 f8
```

ENIAC '40s

They're hard to read!

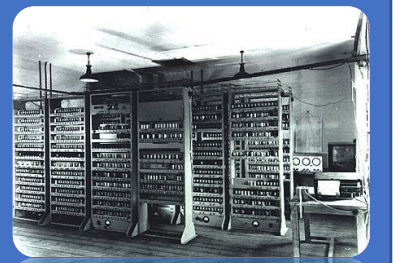
They're not portable to other CPUs!

Assembly Language

A program is a sequence of barely human-readable operations and operands ("add eax, 5") which have a 1-to-1 correspondence with machine language instructions.

```
pushq %rbp
movq  %rsp, %rbp
movl  $5, -4(%rbp)
imull $3, -4(%rbp), %eax
movl  %eax, -8(%rbp)
```

ESDAC '49



t machine and assembly languages?

Development is super slow

They're error-prone!

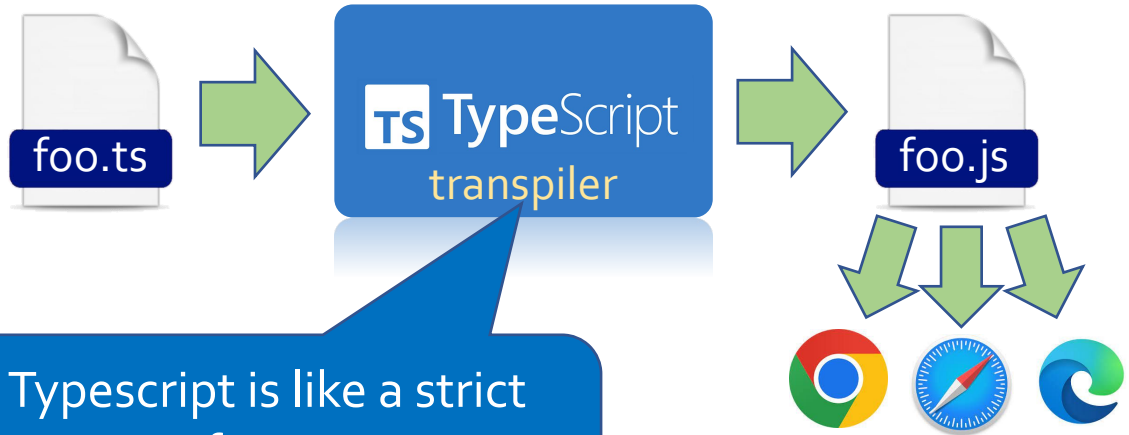
There's no error checking!

What's a Transpiler?

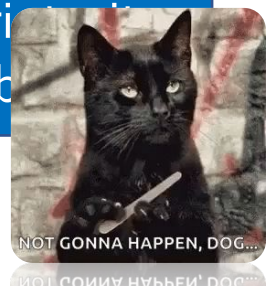
A transpiler is a program that converts source code from one language into another with the same level of abstraction.

Why/when would we want to do that??

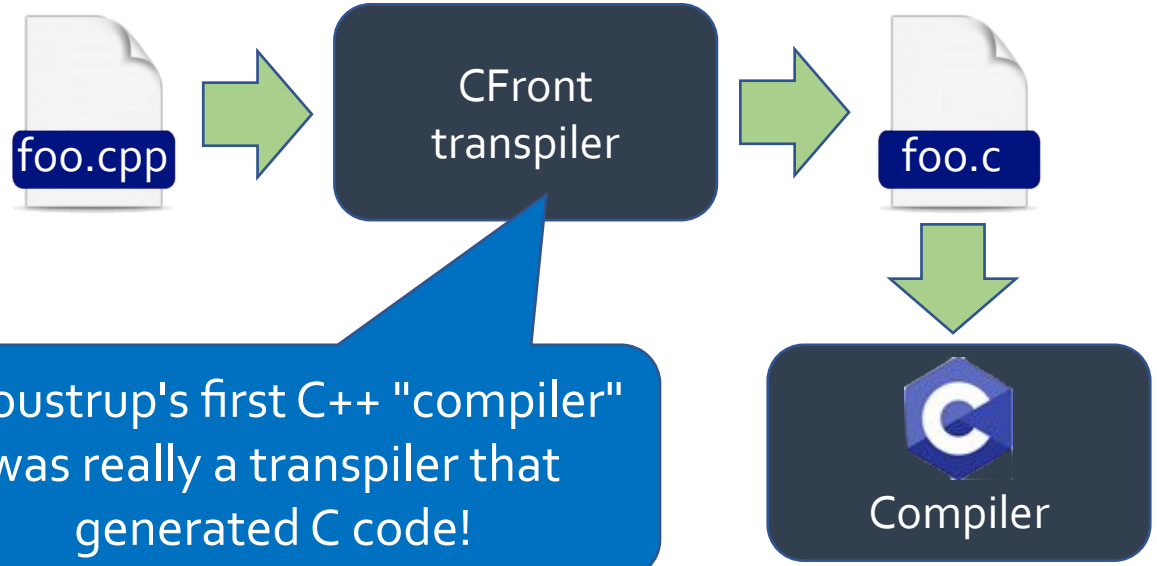
Enable migration to a "safer" version of a language, yet ensure compatibility with existing platforms



Typescript is like a strict version of Javascri… greatly reduces b…



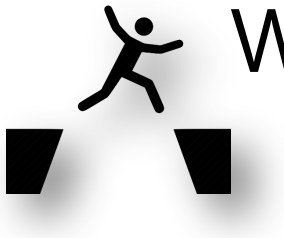
Support a new language's syntax before you have a full compiler for it



Stroustrup's first C++ "compiler" was really a transpiler that generated C code!

OK, now let's see a transpiler in one slide!

Language Choices: Your Turn



What other big language design choices can you think of?

This quarter, we're going to learn all the major language design choices and options for each!

What's CS131 About?

To do this, we'll:

Dive deep into each language paradigm:
Imperative, Functional, OOP, Logic

Go wide, surveying the common building blocks that every language is based on

Be hands-on – you'll be implementing your own programming language from scratch this quarter!

But First… A Throwback to CS33!

Machine Language

A program is a sequence of numbers which represent instructions, e.g., (0x6b=multiply); the instructions are directly interpreted by the CPU.

```
55 48 89 e5 c7 45 fc
05 00 00 00 6b 45 fc
03 89 45 f8
```

ENIAC '40s

They're hard to read!

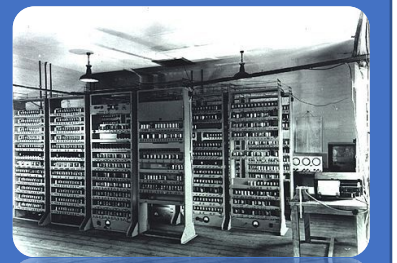
They're not portable to other CPUs!

Assembly Language

A program is a sequence of barely human-readable operations and operands ("add eax, 5") which have a 1-to-1 correspondence with machine language instructions.

```
pushq %rbp
movq  %rsp, %rbp
movl  $5, -4(%rbp)
imull $3, -4(%rbp), %eax
movl  %eax, -8(%rbp)
```

ESDAC '49



t machine and assembly languages?

Development is super slow

They're error-prone!

There's no error checking!

What are the benefits of a high-level language over assembly/machine language?

Programs are easier to understand!

They're portable to many CPUs!

Development is faster

There's more error checking!

Does a Compiler W

e.g., that x and i are
both the
and comp
+ and

If the syntax is valid, the parser converts the tokens into a tree that represents program's operations.

```
func:  type func_name '(' params ')'
       '{' { type var_name ';' } { stmt } '}'
stmt:  'if' '(' expr ')' stmt [ 'else' stmt ]
     | 'while' '(' expr ')' stmt
     | 'for' '(' [ assg ] ';' [ expr ] ';' [ assg ] ')' stmt
     | 'return' [ expr ] ';'
     | func_name '(' [expr { ',' expr } ] ')' ';'
     | '{' { stmt } '}'
expr: const | variable | expr '<' expr | ...
```

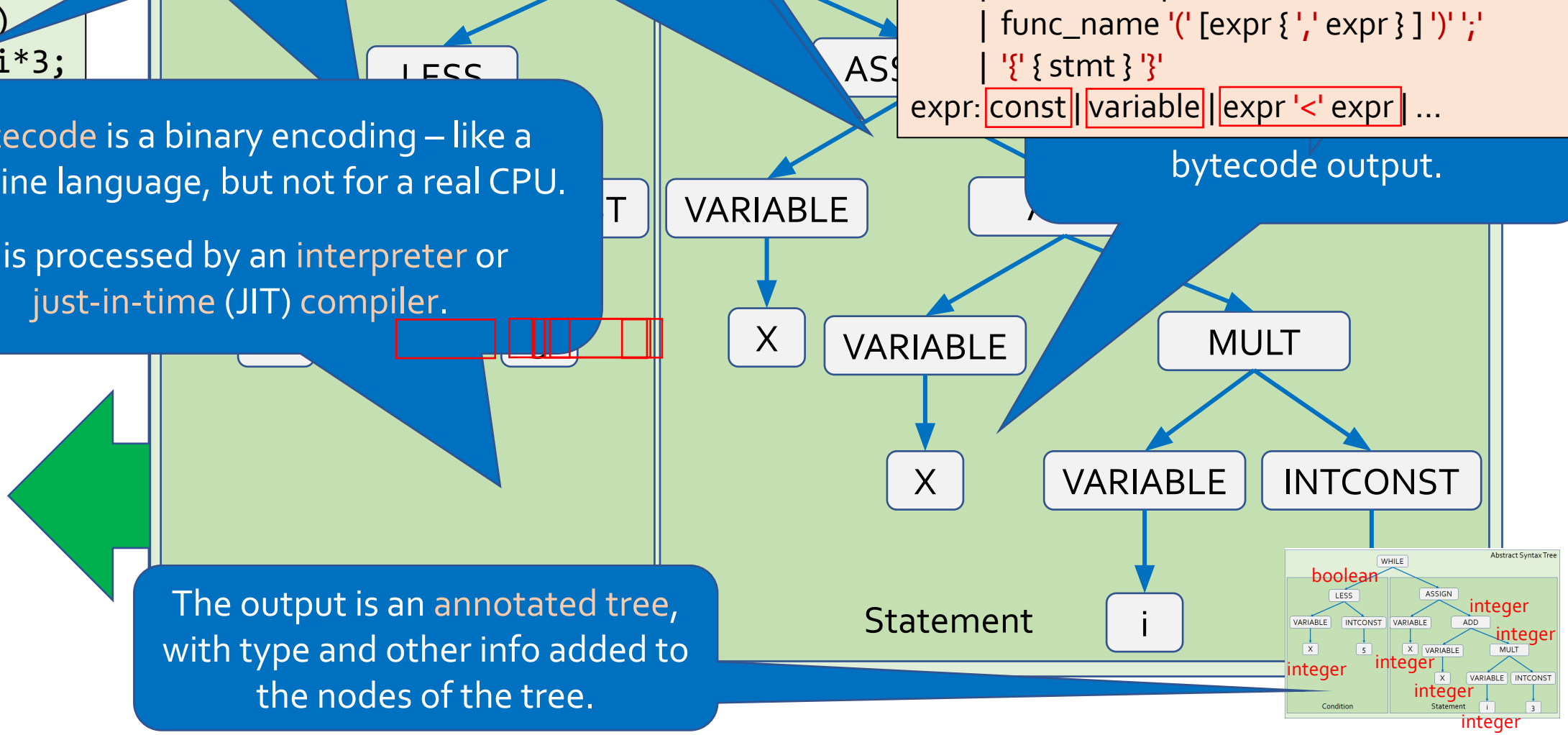
file.cpp

```
while(i<5)
  x = x + i*3;
```

Bytecode is a binary encoding – like a machine language, but not for a real CPU.

It is processed by an interpreter or just-in-time (JIT) compiler.

bytecode output.



The output is an annotated tree, with type and other info added to the nodes of the tree.

Another Example: Sudoku Solver

How many LoC would it take to build a Sudoku solver in Python?

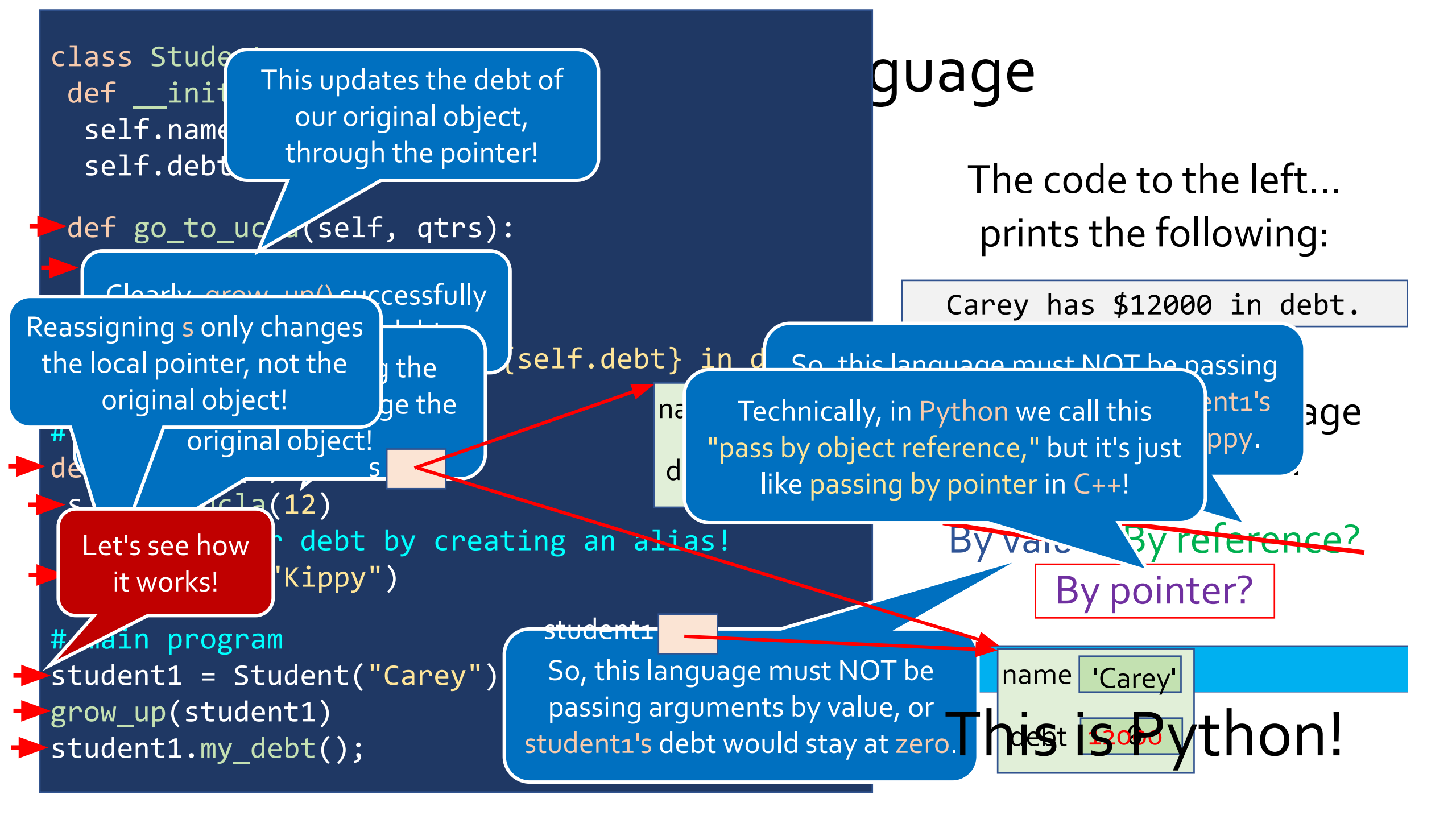
```
% Prolog solution for Sudoku in ~15 lines!
sudoku(Rows) :-
    length(Rows, 9), maplist(same_length(Rows), Rows),
    append(Rows, Vs), Vs ins 1..9,
    maplist(all_different, Rows),
    transpose(Rows, Columns),
    maplist(all_different, Columns),
    Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],
    blocks(As, Bs, Cs),
    blocks(Ds, Es, Fs),
    blocks(Gs, Hs, Is).

blocks([], [], []).
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-
        all_different([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
        blocks(Ns1, Ns2, Ns3).
```

Answer: ~45 lines

Ok, here's the 15-line solution in Prolog!

Clearly, a language can have a massive impact on problem-solving efficiency!


guage
The code to the left...
prints the following:
Carey has $12000 in debt.
This updates the debt of our original object, through the pointer!
Reassigning s only changes the local pointer, not the original object!
Let's see how it works!
Technically, in Python we call this "pass by object reference," but it's just like passing by pointer in C++!
By reference?
By pointer?
So, this language must NOT be passing arguments by value, or student1's debt would stay at zero.
This is Python!
name 'Carey'
debt 12000
student1
grow_up(student1)
student1.my_debt();

The "Right" Answer

What's better? An O($N \cdot \log_2 N$) sorting algorithm or a O(N^2) version?

What's better? Placing a return type before a function's name/parameters or after? Why?

```
// C++
double sqrt(double val)
 { ... }
```

```
// Swift
func sqrt(val: Double) -> Double
 { ... }
```

Guess what? There's no right answer...

just arguments for/ against each approach!

As we compare programming languages, there often won't be a "right" answer.

So what's important is being able to back up our answer with arguments.