

# **CSCI 275: Programming Abstractions**

**Lecture 36: Practical Concerns (cont.) & Wrap Up (you did it!)  
Fall 2024**

**Stephen Checkoway  
Slides from Molly Q Feldman**



# Practical Concerns

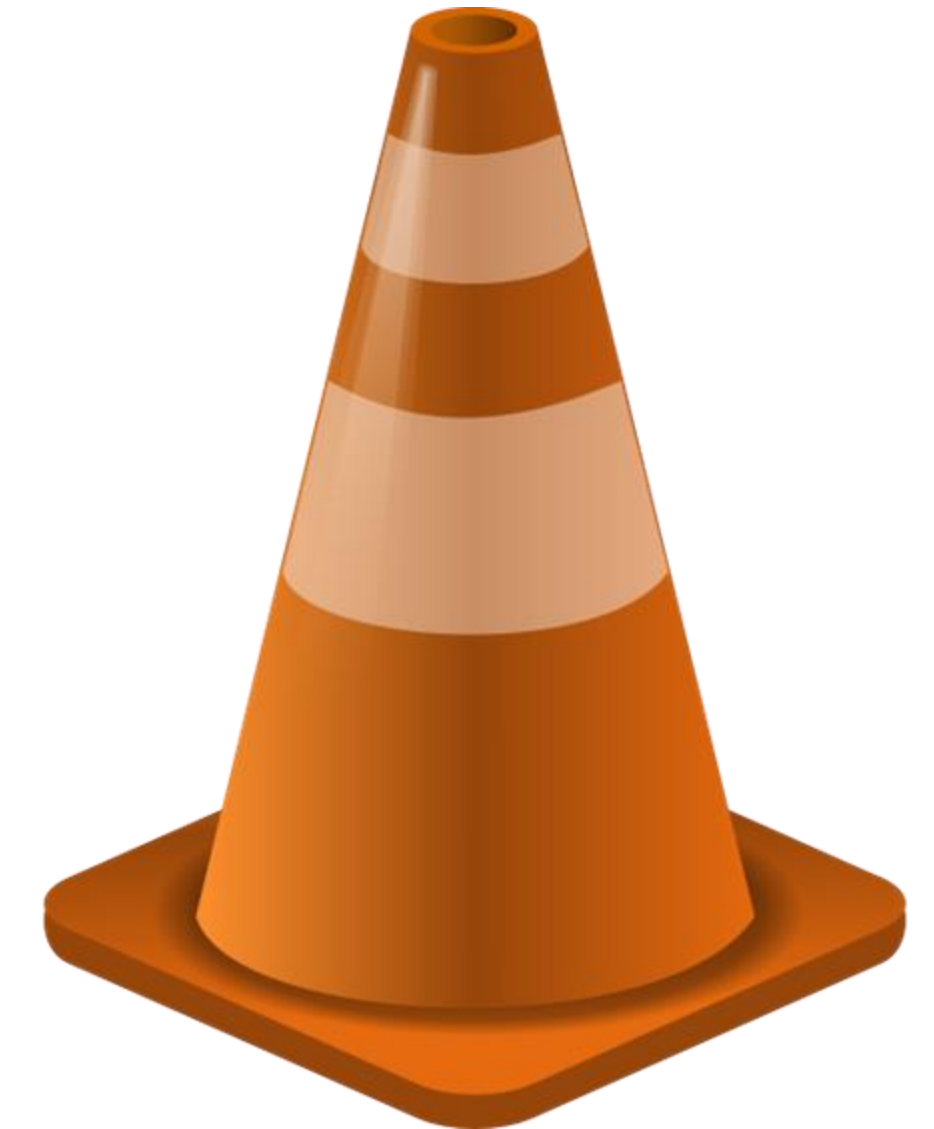
Partial answers to three big questions:

(1) What PL should I learn for what task?

(1) Why do we make new ones?

(1) Why do we have the languages that we have?

*I'll give you my (and others') opinions on these! Not definitive, but hopefully fun/interesting. Also, I'll hopefully provide some helpful links.*



Why do we have the languages  
we have?

What is your preferred single line comment symbol?

A. //

B. %

C. #

D. /\* \*/

E. Something else

# Language & Hardware Coupling

In ~1960s, the development of languages tracked with the development of computer hardware and the systems stack

- They were still working on the abstractions that would lead to ideas like modern Unix

# A (Very) Short History

Most modern programming languages can trace their roots back to either (or both) of **Fortran** and **Algol 60**

**Algol 60** circa 1960, built for specific hardware, implemented call-by-name

**Fortran** circa 1957

“The IBM Mathematical Formula Translating System FORTRAN is an automatic coding system for the IBM 704 EDPM. More precisely, it is a 704 program which accepts a source program written in a language - the FORTRAN language - closely resembling the ordinary language of mathematics, and which produces an object program in 704 machine language, ready to be run on a 704.” (Fortran Manual)

# Language & Hardware Coupling

In ~1960s, the development of languages tracked with the development of computer hardware and the systems stack

- They were still working on the abstractions that would lead to ideas like modern Unix
- Languages were built very low level – languages before C (e.g. PL/I and BCPL) are even closer to the file system, etc.

C's origins start with BCPL and are most closely related to the B programming language

# How do languages evolve?

“Other fiddles in the transition from BCPL to B were introduced as a matter of taste, and some remain controversial, for example **the decision to use the single character = for assignment instead of :=**. Similarly, B **uses /\* \*/ to enclose comments, where BCPL uses //, to ignore text up to the end of the line**. The legacy of PL/I is evident here. **(C++ has resurrected the BCPL comment convention).**” [pg, 3]

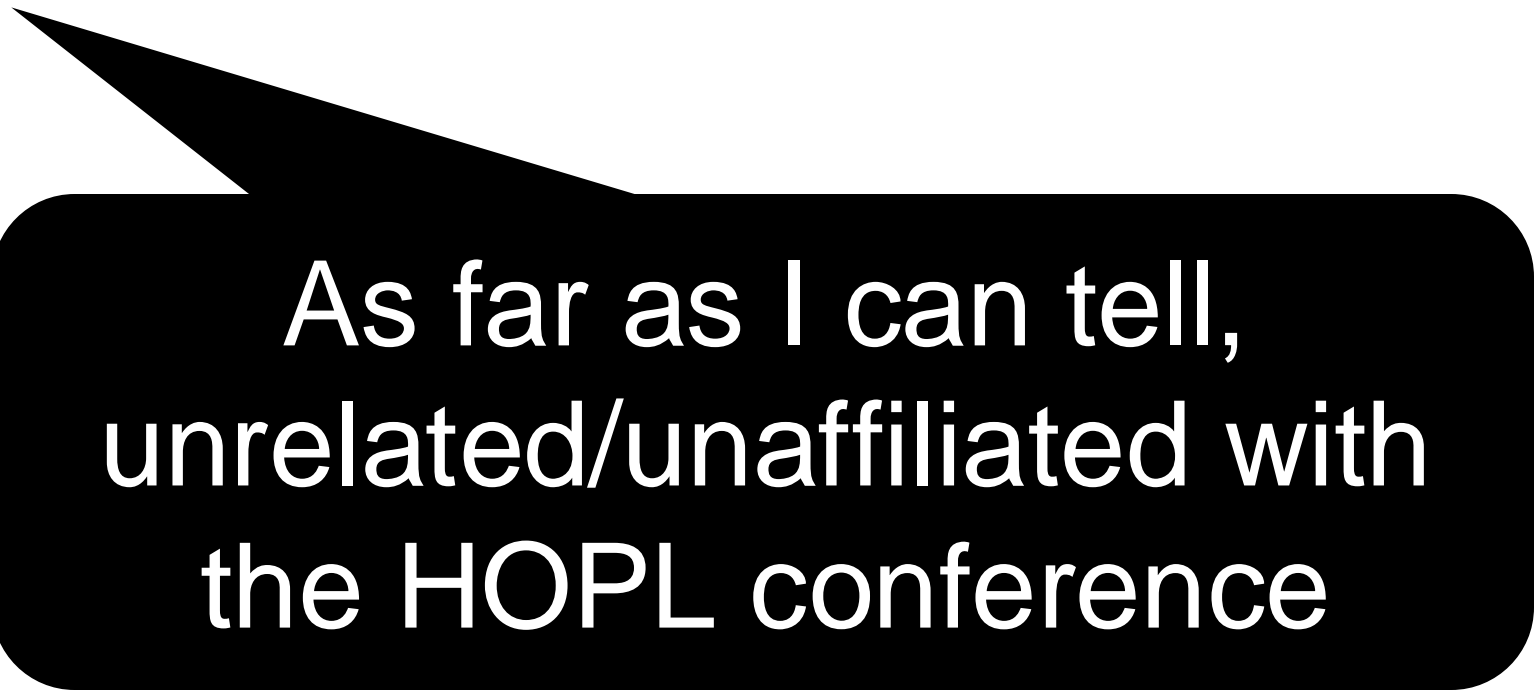


# **The real question answered: Why is it called C?**

**“After creating the type system, the associated syntax, and the compiler for the new language, I felt that it deserved a new name; NB seemed insufficiently distinctive. I decided to follow the single-letter style and called it C, leaving open the question whether the name represented a progression through the alphabet or through the letters in BCPL.”**

# Some Folks Have Tried to Graph This

<https://hopl.info/>



As far as I can tell,  
unrelated/unaffiliated with  
the HOPL conference

Let's say I want to print out the string that contains hello world. How do I do that in Python?

A.`print 'hello world!'`

B.`print ("hello world!")`

C.`print ('hello world!')`

D.`print "hello world!"`

E. Depends / More than one of the above

# Even if it's the same language, it changes

There are new language releases *all the time*

- [Python 3.12.3](#), documentation released on 9 April 2024.
- [Python 3.12.2](#), documentation released on 6 February 2024.
- [Python 3.12.1](#), documentation released on 8 December 2023.
- [Python 3.12.0](#), documentation released on 2 October 2023.
- [Python 3.11.9](#), documentation released on 2 April 2024.
- [Python 3.11.8](#), documentation released on 6 February 2024.
- [Python 3.11.7](#), documentation released on 4 December 2023.
- [Python 3.11.6](#), documentation released on 2 October 2023.
- [Python 3.11.5](#), documentation released on 24 August 2023.
- [Python 3.11.4](#), documentation released on 6 June 2023.
- [Python 3.11.3](#), documentation released on 5 April 2023.

# Big Changes to Existing Languages

But proper changes between core versions are much less common (e.g. Python 2 and Python 3)

Two questions you might have:

1. What is different between Python 2 and Python 3?
2. When did this transition happen?

# Python 2 to Python 3

A lot of Python 3 is *backwards compatible* to Python 2

Notably, **not all of it!** <https://snarky.ca/why-python-3-exists/>

Many changes are to make the language more modern, more efficient, and more consistent

Good example is the `print` statement

[https://python-future.org/compatible\\_idioms.html](https://python-future.org/compatible_idioms.html)

How many years do you think it was between Python 3 starting and Python 2 being “sunset” (i.e. no longer supported in favor of Python 3)?

A. 2 years

B. 4 years

C. 10 years

D. 14 years

E. Something else



# It took 14 YEARS to transition to Python 3



# So, why do we have the languages we have?

- Languages, once heavily adopted, are hard to get rid of
  - Fortran had a stable release in 2023!
  - Still used heavily in applied mathematics, to my knowledge
- Many languages we have *fundamentally* changed the language landscape (e.g., C)
- Many of the languages that exist, but are not used, were middle steps, did not get adopted, etc.
  - HOPL PDF is strong evidence of how many languages exist versus how many languages we know

Wrap up

# Why learn this material?

“The best preparation for quickly learning and effectively using new languages is **understanding the fundamentals underlying all programming languages** and to have some prior experience with a variety of computational models.

Such knowledge will **endure** longer than today’s “hot” languages, which will undoubtedly become obsolete and give way to new languages in the future.

In addition, this knowledge will enable students to quickly look beyond an unfamiliar language’s surface-level details (such as syntax) and **grasp the underlying computational model’s design principles.**”

# Molly's Hypothesis

Learning **functional programming** and **programming languages concepts** makes you a better programmer. Period.

**Turn to your neighbor and discuss the following question:**

*What is your favorite punctuation mark?*

Do you have strong feelings about parentheses?

A. Yes

B. No

C. I still don't get this attempt at a joke

# Summary of this semester's topics

No expected knowledge of functional programming → writing an interpreter for MiniScheme

Key takeaways from the course

- Recursion!
- Functional programming
  - accumulators
  - tail recursion
- Higher-order functions (`map`, `filter`, `foldl`, `foldr`)
- Parsing and interpreting a language
- Lambda calculus



# Things to remember

- Never write a loop when you can just `map`!
- Programming languages are built by *people*
  - Design is a choice
  - There are fundamental building blocks
- Elements of building an Interpreter
  - Concrete vs. Abstract Syntax
  - Parsing
  - Interpreting
  - Evaluation order
  - Scope