Cameron Shortt

Informative Speech Outline

General Function: To Inform
Specific Purpose: After my speech my audience will know the differences and trade-offs between the typeless, dynamic, and static type systems.
Central Idea (Thesis Statement): Static types allow limited automated checking of program logic but limit program structure, dynamic types allow easy programming but the set of errors that crash programs is larger, and typeless systems allow both (very) limited automated checking of program logic and easy programming but only for a small class of problems.
Pattern of Organization: Topical
Outline:

I. Introduction
   A. Attention
   B. Ethos
   C. Preview points

II. Body
   A. Static Types
      1. Static types have the programmer explicitly specify the type of a thing
         a. show code snipped (Java) declaring a bunch of variables
      2. This allows the programming language to figure out if something nonsensical is about to happen
         a. show code snippet (Java) adding text to number and associated error
         b. tell story about attempting to microwave different things
         c. source: princeton java textbook, page 14
      3. Means more boilerplate, and simple things become a verbose
         a. show code snippet with multiple functions all adding two things
         b. tell story about defining “food” for microwave
         c. source: java docs

   B. Dynamic Types
      1. Dynamic types have the programming language figure out the type of a thing
         a. show code snippet (Javascript) declaring a bunch of variables and passing them to functions
         b. source: mozilla web glossary
      2. Program structure isn’t restricted
         a. show code snippet adding different types of numbers
         b. tell story about successfully microwaving different things
      3. ...but programs can now explode in weird and nonsensical ways

        a. show code snippet adding text to a number
        b. tell story about microwaving a fork

C. No Types (typeless, not untyped)
    1. Everything is the same, so everything must interoperate
        a. show code snippet (dc) of calculator things
        b. no boilerplate
        c. definition source: type systems paper from inria
    2. There's no way to represent anything other than a number, so no errors!
        a. Add decimal numbers and integers to show everything works (code snippet)
        b. source: dc manual, parameters, strings
    3. Now try doing not math
        a. that's out of the problem space of a calculator

III. Conclusion

A. Static types improve safety at the cost of verbosity, dynamic types improve flexibility at the cost of safety, and a lack of types improve safety and flexibility in the small subset of problems it can solve.
B. Think of the microwave: would you prefer one that will never explode but will only cook food (better hope the manufacturer defined food correctly), one that can only cook ramen, but is really good at making ramen, or one that might explode but is flexible enough for, say, destroying a cd in an interesting and beautiful way.

Works Cited

1. MozDevNet. “Dynamic Typing - Glossary: MDN.” *MDN Web Docs*, developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing.

Dynamically typed languages are infer the value of a variable from context. An example of this is Javascript. The Mozilla foundation contributes to the Javascript standard and developed the original implementation. The MDN Web docs Glossary, where this definition was specified, is maintained alongside the official W3C (working group for the internet) glossary but with examples specific to Mozilla.

2. “GNU DC Manual.” *UNIX General Commands Manual. d*c(1).

The DC language has only one type: the number. The language is thus typeless. The UNIX General Commands Manual is the definitive resource for all of the programs on UNIX. It contains the full specification for this version of DC.

3. Cardelli, Luca. “Type Systems”. http://lucacardelli.name/Papers/TypeSystems.pdf
This paper describes several different definitions of different type theorys, particularly expanding on static typing in functional languages. Luca Cardelli is a programming language researcher at Microsoft Research, which is responsible for many inovations in programming language theory. He has published many papers on programming language theory in reputable journals. His definitions are also consistent with other publicly available resources.

4. “Defining Methods.” *Defining Methods (The JavaTM Tutorials > Learning the Java Language > Classes and Objects)*, Oracle, docs.oracle.com/javase/tutorial/java/javaOO/methods.html.

This documentation entry describes the different ways of defining methods (their name for functions) in Java. Java is a very popular statically typed language, and is one of the most common teaching languages at universites. Oracle is the current developer of the Java language. Oracle has developed many statically typed programming languages, and contributed to research in that area in reputable journals.

5. Chappell, Glenn. “A Primer on Type Systems.” https://www.cs.uaf.edu/~chappell/class/2025_spr/cs331/read/types_primer.html.

This article describes different aspects of type systems, such as strong vs. weak and static vs. dynamic. Glenn Chappell is a professor at the University of Alaska Fairbanks, where he teaches the programming languages course. He has published many papers on programming language and type theory in reputable journals. His descriptions are also consistent with other publicly available resources.

6. MacQueen, David. “Quick Introduction to Type Systems”. https://www.cs.princeton.edu/courses/archive/spring12/cos320/typing.pdf.

This article describes the more formal areas of type theory that are relevant to static typing in functional programming languages. David MacQueen is a professor at Princeton University, where he teaches advanced programming language courses for the graduate school. He has published many papers on programming language and type theory in reputable journals. His descriptions are also consistent with other publicly available resources.