Programming Language Concepts

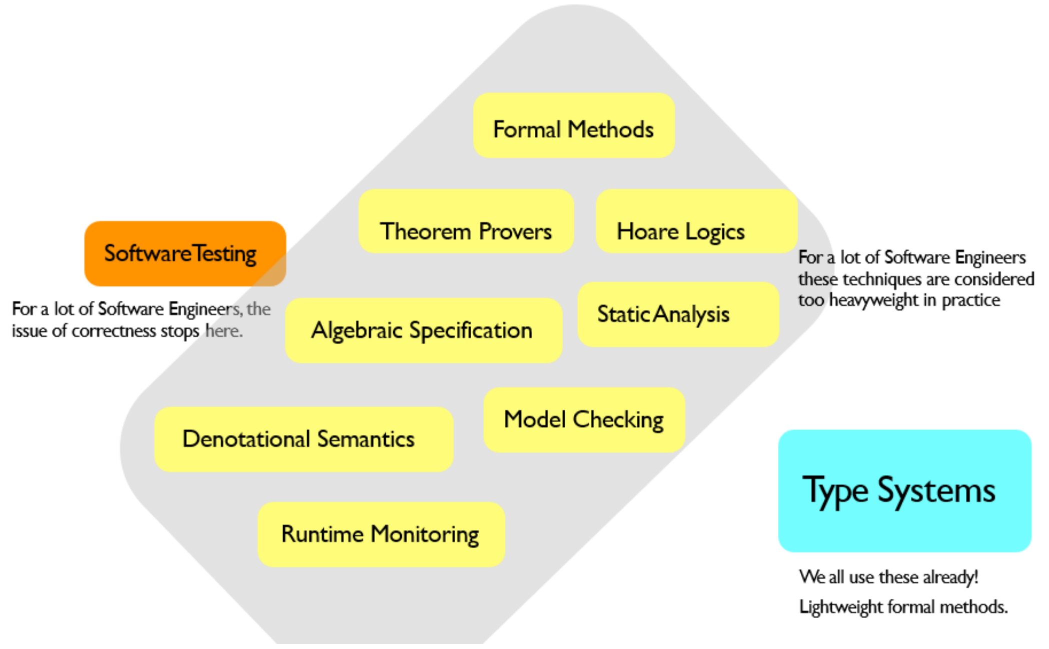
Dr. Imran

Lecture 5

Programming Language Concepts
Introduction to Type Systems

When Programs Go Right?

- A major concern of Software Engineers should be the correctness of the software that they produce.
- However, correctness can mean many different things:


Formal Methods
Theorem Provers
Hoare Logics
Software Testing
For a lot of Software Engineers, the issue of correctness stops here.
Algebraic Specification
Static Analysis
For a lot of Software Engineers these techniques are considered too heavyweight in practice
Denotational Semantics
Model Checking
Runtime Monitoring
Type Systems
We all use these already!
Lightweight formal methods.

Programming Language Concepts

Introduction to Type Systems

What is a Type System?

- A type system in a programming language is a **set of rules**.

- Govern how types (such as integers, strings, arrays, etc.) can be used in the language.

- It ensures that variables, expressions, and functions are used consistently and safely.

- The type system helps catch errors early in the development process, improves code readability, and enhances program reliability.

Programming Language Concepts
Introduction to Type Systems

So, what are types?

- They are the particular classifications of program behaviours that one makes.
- For example, in simple type systems for C functions we classify programs according to the number of and primitive type of the arguments along with the type of the result of the function.

```
def add(a,b):
    return a + b;
```

Add takes 2 arguments a & b,
not concerned of types of a and b or operations
involved in adding them

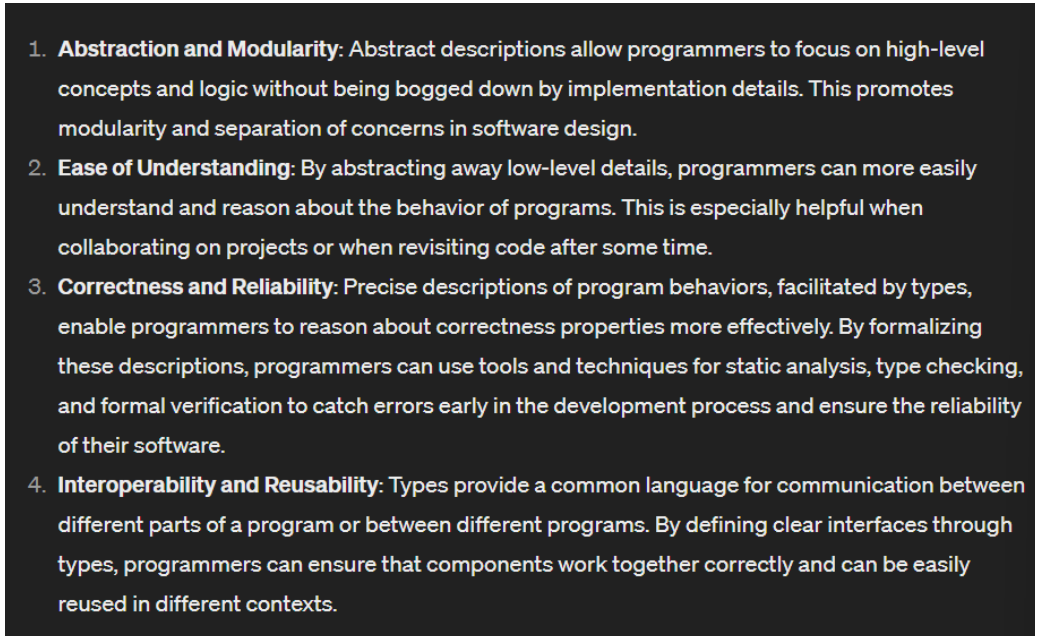
- Types are **abstract** descriptions of programs
    - We can study the correctness properties of interest by abstracting away all of the low-level details.

Programming Language Concepts
Introduction to Type Systems

So, what are types?

- Types are **precise** descriptions of program behaviours
    - We can use mathematical tools to formalise and check these interesting properties

Can describe 'add' that takes 2 arguments & spports addition (int, float, str) and return result of same type



1. **Abstraction and Modularity**: Abstract descriptions allow programmers to focus on high-level concepts and logic without being bogged down by implementation details. This promotes modularity and separation of concerns in software design.
2. **Ease of Understanding**: By abstracting away low-level details, programmers can more easily understand and reason about the behavior of programs. This is especially helpful when collaborating on projects or when revisiting code after some time.
3. **Correctness and Reliability**: Precise descriptions of program behaviors, facilitated by types, enable programmers to reason about correctness properties more effectively. By formalizing these descriptions, programmers can use tools and techniques for static analysis, type checking, and formal verification to catch errors early in the development process and ensure the reliability of their software.
4. **Interoperability and Reusability**: Types provide a common language for communication between different parts of a program or between different programs. By defining clear interfaces through types, programmers can ensure that components work together correctly and can be easily reused in different contexts.

Programming Language Concepts
Introduction to Type Systems

What do types do for us?

- We mentioned above that Types Systems are concerned with correctness and that we use type systems to guarantee the absence of certain behaviours: e.g.
    - application of an arithmetic expression to the wrong kind of data
    - existence of a method or field when invoked on a particular object
    - array lookups of the correct dimension

- Type Systems can also be used to enforce higher-level modularity properties
    - maintain integrity of data abstractions
    - check for violation of information hiding

Programming Language Concepts
Introduction to Type Systems

What do types do for us?

- In doing this,Type Systems enforce disciplined programming
    - type systems form the backbone of module-based languages for large-scale composition types are the interfaces between the modules
    - encourages abstract design
    - types also are a form of documentation

Programming Language Concepts
Introduction to Type Systems

What do we use types?

- Most programming languages use at least some notion of types for programs or the data that programs manipulate. But there are several different approaches to using types.

Static vs. Dynamic Typing:

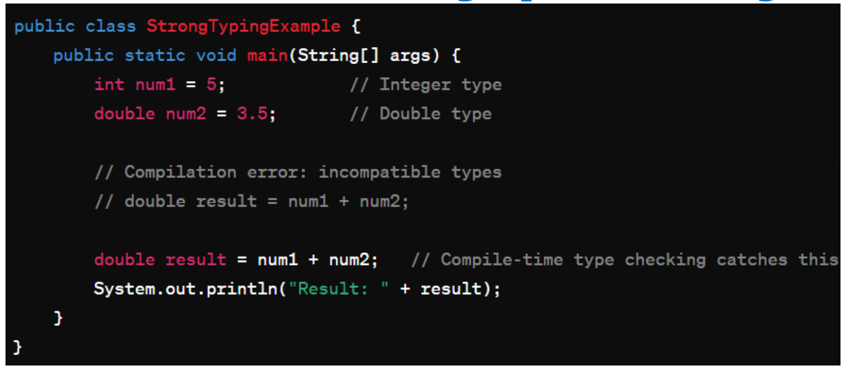
- **Static Typing:** In a statically typed language, variable types are determined at compile time and cannot change during execution. Examples include Java, C, and C++.
- **Dynamic Typing:** In a dynamically typed language, variable types are determined at runtime and can change during execution. Examples include Python, JavaScript, and Ruby.

Programming Language Concepts

Introduction to Type Systems

What do we use types?

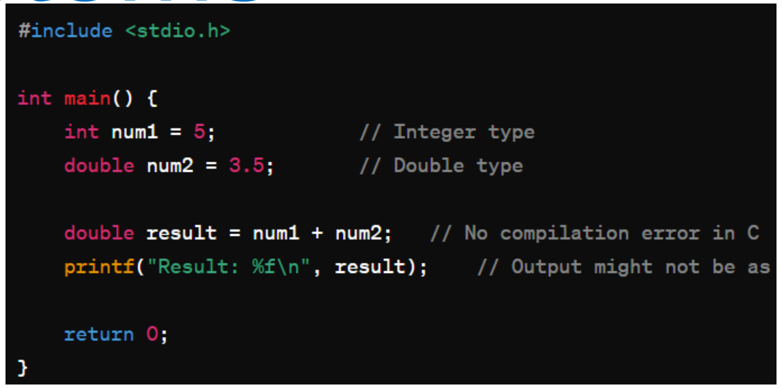
Strong vs. Weak Typing:



```
public class StrongTypingExample {
    public static void main(String[] args) {
        int num1 = 5;           // Integer type
        double num2 = 3.5;      // Double type

        // Compilation error: incompatible types
        // double result = num1 + num2;

        double result = num1 + num2;    // Compile-time type checking catches this
        System.out.println("Result: " + result);
    }
}
```



```
#include <stdio.h>

int main() {
    int num1 = 5;               // Integer type
    double num2 = 3.5;          // Double type

    double result = num1 + num2;    // No compilation error in C
    printf("Result: %f\n", result);     // Output might not be as

    return 0;
}
```

Strong Typing: In a strongly typed language, implicit type conversions are not allowed unless explicitly defined by the programmer. This helps prevent unexpected behaviors and errors. Examples include Java and Python.

Weak Typing: In a weakly typed language, implicit type conversions can occur automatically, which can lead to unexpected behaviors and errors if not handled carefully. Examples include C and JavaScript.

Programming Language Concepts
Introduction to Type Systems

What do we use types?

Type Inference: Some languages, like Haskell and Swift, support type inference, where the compiler can automatically deduce the types of expressions and variables without explicit type annotations from the programmer. This can reduce verbosity while still maintaining type safety.

Type Safety: Type safety ensures that operations are performed only on compatible types, preventing unintended consequences such as type errors or memory corruption. For example, in a statically typed language, you cannot add an integer to a string without explicitly converting one of them.

Programming Language Concepts
Introduction to Type Systems

Static vs Dynamic Typing

- Statically typed languages necessarily use an approximation of the run time types of values.
    - Why ? Because statically determining control flow is undecidable (in sufficiently rich languages).
    - Compile time checking can avoid costly run time errors though.
    - Where types are used for memory layout, static typing is appropriate (e.g. C).

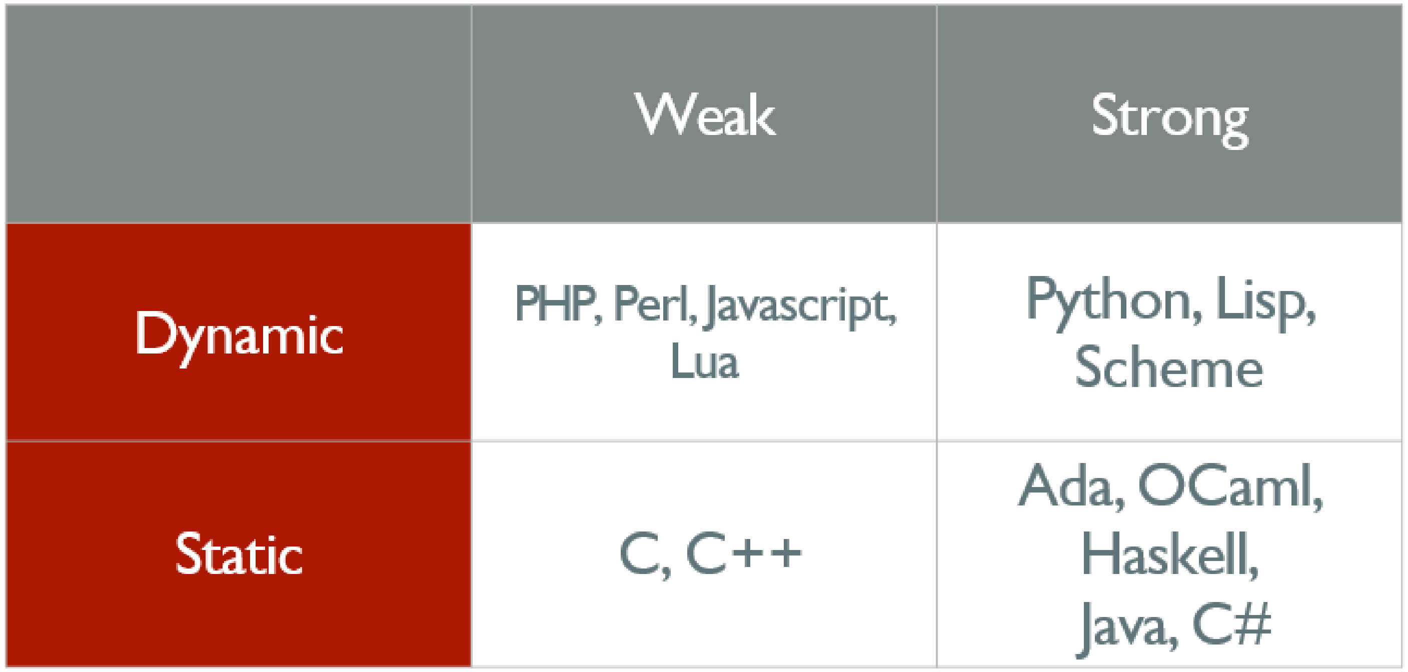
Programming Language Concepts
Introduction to Type Systems

Static vs Dynamic Typing

- Dynamically typed languages check the types of data at point of use in run time.
  - Exact types can be used. This implies no false negative type errors.
  - It is quite common to allow variables to change the type of data they store, or objects to dynamically grow new methods. Some programmers find this convenient.
  - Very common in scripting languages and web programming.
  - Should not be used for Critical Systems as errors may be detected too late.

Programming Language Concepts

Introduction to Type Systems



	Weak	Strong
Dynamic	PHP, Perl, Javascript, Lua	Python, Lisp, Scheme
Static	C, C++	Ada, OCaml, Haskell, Java, C#

↖sort of ↗

Visual Basic (.NET) ?!??!!?

Future programming languages are likely to feature controllable Dynamic / Statically typed regions of code - these are likely to be able to interact in safe ways.

Programming Language Concepts
Introduction to Type Systems

Strong vs Weak Typing

- Liskov and Zilles described "Strong" typing by the requirement that "whenever an object is passed from a calling function to a called function, its type must be compatible with the type declared in the called function."
  - This 'definition' generalizes to the same requirement on any consumer of data.
  - Let's consider a language to have "Weak" typing otherwise.
- One implication of Strong Typing is that intended types must be declared or inferred with functions/methods.
  - This can make languages verbose - Java suffers from this

Programming Language Concepts
Introduction to Type Systems

Strong vs Weak Typing

- An implication of Weak Typing is that data of the ‘wrong’ type may be passed to a function - and the function is free to choose how to behave in that case.
  - Typically, weakly typed languages attempt to implicitly coerce data to be of the required type. This can go disastrously wrong : Software Engineers favourite example is Ariane 5.

“The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error.“ (from the Ariane 501 Inquiry Board Report)

Programming Language Concepts
Introduction to Type Systems

Examples

JAVA

- Java is a statically typed and strongly typed language. Variables must be explicitly declared with their types, and type checking is performed at compile time.

```
int num1 = 5;

String str = "Hello";

// int result = num1 + str; // Error: incompatible types
```

Programming Language Concepts
Introduction to Type Systems

Examples

PYTHON

- Python is a dynamically typed and strongly typed language. Variable types are determined at runtime, and implicit type conversions are not allowed.

```
num1 = 5
str = "Hello"
# result = num1 + str  # TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Programming Language Concepts
Introduction to Type Systems

Examples

JavaScript

- JavaScript is a dynamically typed and weakly typed language. Variable types can change dynamically, and implicit type conversions can occur.

```
var num1 = 5;
var str = "5";
var result = num1 + str;  // result will be "55"
```

Programming Language Concepts
Introduction to Type Systems

Examples

C

- C is a statically typed and weakly typed language, meaning that variables must be explicitly declared with their types, and implicit type conversions can occur.

Explicit Declarations

```
#include <stdio.h>
int main() {
    int num1 = 5;
    float num2 = 3.14;
    char letter = 'A';
    }
```

Programming Language Concepts

Introduction to Type Systems

Examples

C

Implicit Type Conversion

```
#include <stdio.h>
int main() {
    int num1 = 5;
    float num2 = 3.14;
    float result = num1 + num2; // Implicit conversion of num1 to float
}
```

The concepts of pointer types, type casting, arrays and strings are of high significance in C.