Computer Language Type Systems

Definition of Type Systems

- **Definition:** A type system is a set of rules and constraints that govern the assignment and use of types in a programming language.
- **Purpose:**
  - Ensure program correctness.
  - Enhance code readability.
  - Facilitate maintenance and debugging.

Type Checking

- **Type Checking:**
  - Definition: The process of verifying that a program is type-safe.
  - Static Type Checking: Performed at compile-time.
  - Dynamic Type Checking: Performed at runtime.

Static vs. Dynamic Typing

- **Static vs. Dynamic Typing:**
  - Static Typing: Types are checked at compile-time. Examples: C, C++, Java.
  - Dynamic Typing: Types are checked at runtime. Examples: Python, JavaScript.
  - Trade-offs:
    - Static type checking ensures you are aware of issues before code is shipped. May be overkill for small proofs-of-concept.
    - Dynamic is easier to write, so you can prototype quickly. Much extra code is needed to ensure safety in production.

Strong vs. Weak Typing

- **Strong vs. Weak Typing:**
  - Strong Typing: Type conversions are not implicitly performed. Examples: Java, Python.
  - Weak Typing: Type conversions are implicitly performed. Examples: C, JavaScript.
  - Code Examples

Type Inference

- **Type Inference:**
  - Definition: The ability of a compiler to deduce the types of expressions without explicit type annotations.
  - Benefits: Reduces verbosity, enables concise code.
  - Examples: Anything in Python. C++ auto keyword.

Data Types

- **Data Types:**
  - Primitive Types: Basic data types provided by the language. Examples: int, float, char.
  - Composite Types: Types composed of primitive or other composite types. Examples: real strings (not char arrays), arrays, structs, classes.

Type Safety

- **Type Safety:**
  - Definition: The degree to which a programming language prevents type errors.
  - Static Type Safety: Enforced at compile-time.
  - Dynamic Type Safety: Enforced at runtime.

Polymorphism

- **Polymorphism:**
  - Definition: The ability of a type to represent values of multiple types.
  - Compile-time Polymorphism (Static): Achieved through function overloading or generics.
  - Runtime Polymorphism (Dynamic): Achieved through inheritance and virtual functions.