Statically v/s dynamically typed languages

- **Statically typed languages are those in which type checking is done at the compile time, so this also means that in statically typed languages each variable has a type and it doesn't change over the course.** Now, in contrast, **dynamically typed languages are those in which type checking is done at runtime, and there is no type checking at compile time, so this also means that in dynamically typed languages there may or may not be a type associated with a variables**, and if a type is associated then it could be a generic type like "var" in JS which hold good for both a string and number.
    - "Implementations of dynamically type-checked languages generally associate each runtime object with a type tag (i.e., a reference to a type) containing its type information. This runtime type information (RTTI) can also be used to implement dynamic dispatch, late binding, down casting, reflection, and similar features."
- Even if language is statically typed, still it could have some dynamically typed feature, which basically means that some sort of type checking at runtime as well. This is useful in casting of types.
    - "A number of useful and common programming language features cannot be checked statically, such as down casting. Thus, many languages will have both static and dynamic type checking; the static type checker verifies what it can, and dynamic checks verify the rest."
- "Some languages allow writing code that is not type-safe. For example, in C, programmers can freely cast a value between any two types that have the same size."
- **Advantage of "statically" typed languages are that:**
    - Since most of the type checking is done at compile time so interpreter or runtime can run at full speed, without worrying about the types.
    - It leads to lesser number of runtime exception or errors related to type, because most of the type checking is done at compile time.
- **Advantage of "dynamically" typed languages are that:**
    - They could help in extremely fast prototyping, since developer need not to understand the type system so dev can loosely create variables and run it, and this leads to very fast prototyping.
- **List of statically and dynamically typed languages**:
    - Statically:
        - Java
        - C (C is a statically typed language but lesser "strongly" typed as compared to Java because it allows more implicit conversions)
        - C++
        - C#
    - Dynamically:
        - PERL
        - PHP
        - Python
        - JavaScript
        - Ruby
- **Type checking is an important security feature.** Suppose, there is no type checking, and a method accepts an object of type "BankAccount" which has a method called as "creditAccount(BankAccountDetails)", now at runtime if there is no type checking then I can pass an object of my own class which has same method "creditAccount(BankAccountDetails)" and it will get executed, considering we are talking about object oriented language because OOP supports "polymorphism" and here what we are discussing is nothing but "polymorphism". So, basically an object oriented language (which basically means it supports "polymorphism") which doesn't have strong type checking can lead to security issues.

Strongly v/s weakly typed languages

- **Strongly typed languages are those in which implicit conversions are not allowed if there is loss of precision. For example, in Java, you can cast an "int to long" because there is no loss of precision but you cannot "implicitly" cast a "long to int" because there would be loss of precision. In contrast, in weakly typed languages, implicit conversions are allowed even if there is loss of precision.**
- I think dynamically typed language can also be a strongly typed language if "at runtime" it doesn't allow implicit conversions in which there is loss of precision.

Good further readings

- Type_system
- Strong_and_weak_typing
- Category:Statically_typed_programming_languages
- Category:Dynamically_typed_programming_languages