**Types**
#A collection of values that share common properties - usually a common set of operations
#Tells you what is legal to do with some value in the language.

**Type Error**
Attempt to use a value in an operation inconsistent with the value’s type.

**Compile-time (static) type checking:**
#it may flag an error as something that would not ever cause a run-time error.
Advantages:
- Less runtime overhead
- Whole program is checked

**Run-time (dynamic) type checking:**
#run-time type checking is expensive - must be done each time program is executed
Advantages:
- Allows certain programming styles not possible with compile time type checking
- More flexible data structures
  - Lists in python may contain values of any type unlike haskell which only has one type

**Type Safety: Strong Typing**

Static String Typing = compile error

Dynamic Strong Typing = error at the point of misuse

In type-safe languages, values are managed “from the cradle to the grave”
- #Objects are created and initialized in a type-safe way
- #An object cannot be corrupted during its life time
- #Objects are destroyed, and their memory reclaimed, in a type-safe way.
- #Any  change of type requires explicit conversion

Python: 5 + “5” #not allowed error
Javascript: 5 + “5” = “55”
C: Does not have type safety
Heap values are created in a type-unsafe way.
Casts and unchecked array accesses can corrupt memory during its lifetime
C deallocation is unsafe, and can lead to dangling pointers (pointer that points to invalid data)