What is a type?

It is what a piece of data represents. Formally a type is

1. a set of legal values.
2. It is the more primitive types that make up the type
3. The legal operations you can do on the type

String:

1. {“”, “a”, “b”, “c”, …, “z”, “ab”, …}
2. Array of type char
3. charAt, length, substring, concatenation, …

A type system is made up of

1. How to define a type
2. Type equivalence: when two different types are treated as the same
3. Type compatibility: when you can use type B in places expecting type A
4. Type inference: What is the type of different program operations?

Type equivalence:

1. Strict name equivalence: 2 types are equivalent if they have the same name
2. Loose name equivalence: 2 types have the same name or a name that is an alias of the other type
3. Structural equivalence: 2 types are equivalent if they have the same structure

Type compatibility:

Issue here is when you have to convert a type.  
Type casting: an explicit conversion  
Type coercion: an implicit type conversion

Type casting can be converting or non-converting  
Ex: Java typecasts a type coersions of non-primitive types, do not change the data  
Converting is when you change the data, non-converting is when you dont

Parameterized Types: types that contain types  
ex: String[] 🡪 array parameterized by String  
ex: Integer[] 🡪 array parameterized by Integer  
ex: LinkedList<Integer> 🡪

In Java, we can use String[] where Object[] is expected.   
We cannot use LinkedList<String> where LinkedList<Object> is expected.

Covariance: If A is subtype of B, then C<A> is a subtype of C<B>  
We can use C<A> wherever C<B> is expected

Contravariance: If A is a subtype of B, the we can use C<B> wherever C<A> is expected.

Invariance: C<A> and C<B> are not compatible

String[] and Object[] are covariant  
LinkedList<String> and LinkedList<Object> are invariant

public A myMethod(B b) {}

type is B->A

overriding in java: Same name, same parameter signature, same or narrower return type

Functions are covariant in the return type and invariant on the parameters

First class values:

1. values that can be assigned to variables
2. input or returned from functions
3. created in any context

Java: primitives, classes/objects  
Scheme: atoms, lists, functions  
C: primitives, pointers, structures