**What is subtyping?**
- Subtyping = subset
- Subtyping = subclasses
- Subtyping = substitutability
- Subtyping = polymorphism

All valid perspectives, but not all the same meaning.

**Subtyping Semantically = implicit coercions**

- Types `

- Subtype relation

- Function converting values of \( A \) to values of \( B \)

**How can we take advantage of this?**

Some languages let programmers define their own coercions to extend the subtyping relation.

- C++
- Scala

**What can go wrong?**

**Ambiguity**

In C++ and Scala, how the compiler inserts coercions can affect the semantics of a program.

**Preventing Ambiguity**

- \( A \)
- \( B \)
- \( C \)

- Coerce \( A \) to \( B \) is due to

- Coerce \( A \) to \( C \) is due to

- Coerce \( A \) to \( A \) is due to

- Coerce \( A \) to \( B \) is due to

- Coerce \( A \) to \( C \) is due to
Number Conversion Example

```
int isDouble double isInt we fail to coerce
```

Convenient, but what goes wrong?

```
2.5 \rightarrow 2 \rightarrow 2.0 \rightarrow int \rightarrow double
```

```
\nequivalence \neq \text{identity} +
```

```
\text{losses of transforming like floor}
```

---

Subtyping Categorically

```
But First!
```

```
What's a category??
```

---

A Category is:

1. A collection of objects, e.g. ABC
2. For each pair of objects A and B, a collection of morphisms, e.g. \( f: A \rightarrow B \) or \( A \rightarrow B \)
3. For every object A, \( A \rightarrow A \) identity
4. For this looks like a graph, but there's more!

---

A Category also has:

3. For every object \( A \), a "special" morphism \( A \rightarrow A \)
4. For every \( A \rightarrow B \rightarrow C \), a morphism \( A \rightarrow C \) called composition
5. More to come later...

---

Example: Category of Sets (called \( \mathbf{Set} \))

1. The objects are all possible sets
2. The morphisms from set \( A \) to set \( B \) are all possible functions from \( A \) to \( B \)
3. \( \text{id} \) is the identity function: \( \text{id} \)
4. morphism composition is function composition

```
Classic example, but not all morphisms are functions!
```

---

Example: Subtyping as a category

1. The objects are the types
2. There exists a unique morphism from \( A \rightarrow B \) if \( A \) is a subtype of \( B \)
3. \( \text{id} \) exists because of reflexivity
4. composition is defined because of transitivity

The categories correspond to their
(i.e. a set with \( \text{morphisms} \) to \( \text{relation} \) between any two objects)
A Category (lastly) also has:

5. For all \( f : A \to B \), \( \forall g : f = f \circ g \cdot id_B \)
   i.e. composition of identities does nothing

6. For all \( f : A \to B \), \( \forall g, h : f = g \circ h \cdot (f, id_B) \)
   i.e. composition is associative

Thus, give a (possibly empty) path of morphisms
From \( A \) to \( B \), there is no unambiguous
way to compose that path into a
morphism from \( A \) to \( B \)!

A Functor from \( C \) to \( D \) is:

1. A Function from objects of \( C \)
   to objects of \( D \)

2. A Function from \( A \to B \)
   to \( F(A) \to F(B) \)
   preserves more!

3. \( F(\text{id}_A) = \text{id}_{F(A)} \)
   (preserves identities)

4. \( F(f \circ g) = F(f) \cdot F(g) \)
   (preserves composition)

In particular for subtyping

3) \( \text{coerce}_A \cdot A = \text{id}_{\text{exs}} \)

4) \( \text{coerce}_A \cdot \text{coerce}_B \cdot \text{coerce}_C = \text{coerce}_C \cdot \text{coerce}_B \cdot \text{coerce}_A \)

so inambiguity is related to
categorical structure

A Factor also has:

Sneak Peek:

What does

"Hello" + 1+2

evaluate to?