A closer look at types....

polymorphism ≡ comes from Greek meaning ‘many forms’

In programming:

Def: A function or operator is polymorphic if it has at least two possible types.
Polymorphism

i) Overloading

Def: An overloaded function name or operator is one that has at least two definitions, all of different types.

Example: In Java the ‘+’ operator is overloaded.

String s = “abc” + “def”;

int i = 3 + 5;
Polymorphism

Example: Java allows user defined polymorphism with overloaded function names.

```java
bool f (char a, char b) {
    return a == b;
}

bool f (int a, int b) {
    return a == b;
}
```

Note: ML does not allow function overloading
ii) **Parameter Coercion**

**Def:** An implicit type conversion is called a **coercion**.

Coercions usually exploit the type-subtype relationship because a widening type conversion from subtype to supertype is always deemed safe → a compiler can insert these automatically → type coercions.

**Example:** type coercion in Java

```java
double x;
x = 2;
```

the value 2 is coerced from int to double by the compiler
Polymorphism

Parameter coercion is an implicit type conversion on parameters. Parameter coercion makes writing programs easier – one function can be applied to many subtypes.

Example: Java

```java
void f (double a) { ... }
```

```plaintext
int ⊆ double
float ⊆ double
short ⊆ double
byte ⊆ double
char ⊆ double
```

all legal types that can be passed to function ‘f’.

Note: ML does not perform type coercion (ML has no notion of subtype).
iii) Parametric Polymorphism

**Def:** A function exhibits **parametric polymorphism** if it has a type that contains one or more **type variables**.

**Example:** ML

```ml
- fun f(x,y) = (x = y);
val f = fn : 'a * 'a -> bool
```

**Example:** C++ and Java

C++ and Java have templates that support parametric polymorphism.
iv) Subtype Polymorphism

**Def:** A function or operator exhibits *subtype polymorphism* if one or more of its *constructed types* have subtypes.

Note: one way to think about this is that this is type coercion on constructed types.
Polymorphism

Example: Java

```java
class Cup { ... }
class CoffeeCup extends Cup { ... }
class TeaCup extends Cup { ... }

TeaCup t = new TeaCup();
TeaCup t = new TeaCup();
Cup c = t;  // type coercion: TeaCup -> Cup

void fill (Cup c) {...}

TeaCup t = new TeaCup();
CoffeeCup k = new CoffeeCup();

{ fill(t);
  fill(k);  // subtype polymorphism
```