- Overloading — symbols. e.g static, +, *, /
  — user defined operator overloading (C++)
- **Function Overloading**

```
c = max(a, b);
```

In C, will only work for values defined in max()

```
int max (int a, int b) {

} //max()
```

all uses of max() have to provide int values.

Also, once defined, we cannot define another version (say, for float values):

```
float max (float a, float b) {

}
```

In Java, we can define both versions!!
i.e. Java allows function overloading.

- as long as we use different function signatures.

function signatures: the first line/header of a function definition.

```
int max(int a, int b);
float max (float a, float b);
```

In C, we have to use diffent names:

```
int maxint (int a, int b);
float max_float(float a, float b);
```

Polymorphism: "many forms"

has many different definitions in PL world, making it confusing.

Original: There is ONE definition for an overloaded function (i.e. same code is executed for parameters of different types)

e.g. reversing a list w/o regards to type of elements int.

e.g. In Python,

```
def max(a,b):
    if a>b:
        return a
    else
        return b
```

In Python >, <, etc are defined for all primitive types.

use

```
c=max(3,8)          # integer
c=max(3.14,Math.sqrt(8.12))   # float
c=max("Hello","Nihao"))       # string
```

This is called Parametric Polymorphism.

Most functional PLs allow polymorphism.

Q: "Does Java have polymorphism"?

Internet says YES. But strictly speaking it is OVERLOADING and not POLYMORPHISM.

Example Java.

```
abstract class shape {
    public abstract float area();

}//shape
public class Triangle extends shape {

    public float area(){

    }//area()

}//Triangle
public class Square extends shape {

    public float area() {

    }//area()

}//square
public class MyProgram{
        Shape T = new Triangle(—);
        Shape S = new Square(—);

        T.area();
        S.area();
```

This is called subtype polymorphism (!!)

First-class Values

e.g. int / 5

are values in a program that can be:

1. assigned to a variable e.g. a=5
2. passed as a parameter of a function f(5)
3. returned as a value of a function return 5;
5. included in other data structures A[i]=5;

Q. What about functions?

In Java : NO
In C : NO (but you can pass a pointer to a function)

In Python : YES

```
def double(a):
    return 2*a
```

```
double(5)
→ 10
```

We can also do:

① x = double
② f(double, ...)
③ return double
④ l[0] = double

In Python, functions are also first-class objects!

Higher-Order Functions: $h = f(g(x))$

requires functions as first-class objects.

e.g. Python

map(f, L1, L2, ...LN)

where f is a function that takes N arguments
L1, .. LN are lists

returns

```
[ f(L1[0], L2[0], ..., LN[0]),
  f(L1[1], L2[1], .., LN[1]),
  :
]
```

e.g ① l = [1, 2, 3, 4]

```
map(double, l)
→ [2, 4, 6, 8]
```

②

```
def power(a, b):
    return a**b
```

power(2, 3)

→ 8   ~~8~~ $\equiv 2^3$

```
l1 = [1, 2, 3, 4]
l2 = [2, 3, 4, 5]
map(power, l1, l2)
```

→ [1, 8, 81, 3125] $\equiv [1^2, 2^3, 3^4, 4^5]$

Also ~~f~~ filter(f, l)

l = [2,3,4,5,6,7]

```
def even(n):
    return n%2==0;
```

```
even(2)
→ True
even(11)
→ False
```

filter(even, l)

→ [2,4,6]

Ex: prime(n) → True, if n is prime
False o/w

range(n) → [0,1,2, ...n-1]
range(a,b) → [a,a+1, ... b]
range(1,5) → [1,2,3,4,~~5~~]

try filter(prime, ~~[illegible]~~ range(2,100))

→ [all prime #'s in [2,100]]

λ-Calculus – 1930's by Alonzo Church

$\lambda x.x^2$ λ-expressions [Notation for defining functions.

$\equiv$

def ___ $(x) = x^2$

[Any computable function can be written as a λ-expression.

function application

$$(\lambda x.x^2)(2) \Longrightarrow 4$$

Essentially, you are defining a function, say f

definition $f = \lambda x.x^2$ **Python** def double(x):

use $f(2) \Longrightarrow 4$ return x*x

λ-functions in a PL can be used to define in-line, anonymous functions.

e.g. **Python**

```
l = [1,2,3,4]
map(double, l) => [2,4,6,8]
```

or, using λ-function/anonymous function

```
map(lambda x: 2*x, l)
```

Q: How to associate a stack frame??

Uses a **closure**: function + referencing environment

Needed when functions are first-class objects.