Generic Functions

Intro: Class-based OOP

```
class OutputStream {
   void println(Object obj) { ... }
   ...
}
```

... allows you to say:

```
out.println(pascal);
```

Intro: Class-based OOP

```
out.println(pascal);
```

...or, in Lisp syntax:

```
(send out ‘println pascal)
```

The receiver is just another argument.

So let’s change...

(send receiver message args ...)

...to...

(call message receiver args ...)

“Call” is redundant.

So let’s change...

(call message receiver args ...)

...to...

(message receiver args ...)

So now we have generic functions!

```
(send out 'println pascal)
```

...is now...

```
(println out pascal)
```

Let’s define methods.

```
(defmethod println ((out output-stream)
                    (p person))
  ...)
```

This is a method with multiple dispatch!

Generic functions.

- Invented when Lispers implemented OOP.
  Function calls appear more natural in Lisp.
  (LOOPS, New Flavors, CLOS)
- Generic functions were already needed.
  Mathematical operations are generic!
  They work on ints, floats, complex, etc.

Mathematical ops as generic functions.

```
(defmethod + ((x int) (y int))
  ...)

(defmethod + ((x float) (y float))
  ...)

(defmethod + ((x complex) (y complex))
  ...)
```

...but how does it work?

- Let’s implement generic functions!

Further notes.

- Efficiency:
  Cache everything!
- Multiple dispatch:
  Consider all the args when selecting applicable and most specific methods!
- Advice:
  Add qualified methods that are called before, after or around everything else!

Further information.

- Pick a good Common Lisp tutorial.
  - Peter Seibel, Practical Common Lisp, http://www.gigamonkeys.com/book/
  - David Lamkins, Successful Lisp, http://www.psg.com/~dlamkins/sl/

Further information.

- Papers about CLOS: http://www.dreamsongs.com/CLOS.html