Anonymous functions

- can have code build and evaluate expressions, e.g. build and run (f (+ a b) (* 10 c))

```
(eval (cons ‘f (append ‘(+ a b) ‘(* 10 ‘c))))
```

- what about building and returning a function?
- can build functions “on the fly”, store them in variables or pass as parameters, run them sometime later
- don’t need to give them a name, since we’ll access them through the variable or parameter they’re stored in
- hence we often call them anonymous functions

Lambda

- Theory of functional computing generally used lambda as the symbol for a function, so many languages use lambda to refer to anonymous functions
- In lisp, a function called lambda is used to build anonymous functions
- Syntax is (lambda parameterlist statements-for-body)

```
(lambda (x y)
    (if (and (numberp x)(numberp y)) (* x y)))
```

Running lambda functions

- Lambda returns the function it built, so we can store it in variables or pass it as a parameter

```
(defvar f (lambda (x) (if (numberp x) (- x))))
```

- We can then use higher order functions to run it, e.g.

```
(apply f '(10))          ; returns -10
(funcall f -3)           ; returns 3
(mapcar f '(10 -3 6)) ; returns (-10 3 -6)
```

Construction using lambda

- Can get as complex/creative as desired when using lambda to build a function, as long as the end result is syntactically correct
- Soon we'll look at writing functions that build lambda functions, where the parameters we pass to the "writer" tell it what customizations to use in the lambda function it creates

Sample use

- Sometimes we have a simple function that we're only going to use once, as a parameter to some higher order function, e.g. take sqrt of all numbers in a list
- Might be sensible to simply pass the lambda expression to the higher order function, i.e. (mapcar lambdafunc list)

```
(mapcar
   (lambda (i) (if (numberp i) (sqrt i) i)
    L)
```