```
(ns anonymous-function.core
  (:gen-class))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))
```

```
#'anonymous-function.core/-main
```

```
; this is an anonymous function. It can only be
called where it is defined.
#(* % %)

(map #(* % %) (range 11))
```

```
(0 1 4 9 16 25 36 49 64 81 100)
```

```
; you can also define a function by just using (def
name #(anonymous-function %))
(def square #(* % %))

(square 2)
```

```
4
```

```
; you can also declare an function by using (fn
[varname] expression)
(def square-fn (fn [x] (* x x)))

(square-fn 2)
```

```
4
```

```
(square-fn 2)
```

```
4
```

```
(defn square-defn
  "this function is defined by the defn macro, takes
one number, and returns it's square."
  [num]
  (* num num))
```

```
#'anonymous-function.core/square-defn
```

```
(square-defn 2)
```

```
4
```

```
;In clojure functions are "first class". That means
they can be passed to other functions as arguments.
Here we use that principle to test if all the
previously defined squaring functions return the
same correct answer.
(map #(%1 %2) [#(* % % ) square square-fn square-
defn] (repeat 4 4))
```

```
(16 16 16 16)
```

```
; in the last expression we used an anonymous
function to do a one to one mapping where each
function is given the number 4 and returns 16. Let's
break that expression down to learn more about
multivariable anonymous functions.
```

```
(map #(+ %1 %2) (range 1 10) (range 1 10))
```

```
(2 4 6 8 10 12 14 16 18)
```

```
; the above expression maps over two ranges and sums
the nth element of each range and returns a new
range of doubled numbers. The %1 %2 tags specify
which nth element from a given list will be used.
For example


(map #(/ %1 %2) (range 2 10 2) (reverse (range 2 10
2)))
(map #(/ %2 %1) (range 2 10 2) (reverse(range 2 10
2)))
```

```
(1/4 2/3 3/2 4)

(4 3/2 2/3 1/4)
```

```
; both expressions above are almost identical except
the arguments in the anonymous function are switched
around. For a multiple-arity mapping over several
lists each list is passed to the anonymous function.
so there can be %1 %2 %3 ... %N numbers of input to
an anonymous function.
```