CS 596 Functional Programming & Design
Fall Semester, 2014
Assignment

San Diego State University -- This page last updated 8/26/14

Assignment 1
Due Sept 3 11:59 pm

Each of the questions below has an __ in them. Replace them with the correct answer. Test your answers in Light Table.

What to Turn in

Create one file with all the problems answered. Upload the file to assignment one in the course portal.

Late Penalty

An assignment turned in 1-7 days late, will lose 3% of the total value of the assignment per day late. The eight day late the penalty will be 40% of the assignment, the ninth day late the penalty will be 60%, after the ninth day late the penalty will be 90%. Once a solution to an assignment has been posted or discussed in class, the assignment will no longer be accepted. Late penalties are always rounded up to the next integer value.

The Questions

"We shall contemplate truth by testing reality, via equality."
(= __ true)

"To understand reality, we must compare our expectations against reality."
(= __ (+ 1 1))

"You can test equality of many things"
(= (+ 3 5) __ (+ 2 __))

"Some things may appear different, but be the same"
(= 2 2/1 __)

"You cannot generally float to heavens of integers"
(= __ (= 2 2.0))

"But a looser equality is also possible"
(== 3.0 3 __)

"When things cannot be equal, they must be different"
(not= :fill-in-the-blank __)
"Lists can be expressed by function or a quoted form"
(= '(__ __ __ __ __) (list 1 2 3 4 5))

```
"They are Clojure seqs (sequences), so they allow access to the first"
(= __ (first '(2 3 4 5)))

"As well as the rest"
(= __ (rest '(2 3 4 5)))

"The rest when nothing is left is empty"
(= __ (rest '(10)))

"And construction by adding an element to the front is simple"
(= __ (cons :a '(:b :c :d :e)))

"Conjoining an element to a list can be done in the reverse order"
(= __ (conj '(:a :b :c :d :e) 0))

"You can use a list like a stack to get the first element"
(= __ (peek '(:a :b :c :d :e)))

"Or the others"
(= __ (pop '(:a :b :c :d :e)))

"But watch out if you try to pop nothing"
(= __ (try
        (pop '())
        (catch IllegalStateException e "No dice!")))

"The rest of nothing isn't so strict"
(= __ (try
        (rest '())
        (catch IllegalStateException e "No dice!")))

"You can use vectors in clojure to create an 'Array' like structure"
(= __ (count [42]))

"You can create a vector in several ways"
(= __ (vec nil))

"And populate it in either of these ways"
(= __ (vec '(1)))

"There is another way as well"
(= __ (vector nil))

"But you can populate it with any number of elements at once"
(= [1 __] (vec '(1 2)))

"And add to it as well"
(= __ (conj (vec nil) 333))

"You can get the first element of a vector like so"
```

```
(= __ (first [:peanut :butter :and :jelly]))

"And the last in a similar fashion"
(= __ (last [:peanut :butter :and :jelly]))

"Or any index if you wish"
(= __ (nth [:peanut :butter :and :jelly] 3))

"You can also slice a vector"
(= __ (subvec [:peanut :butter :and :jelly] 1 3))

"Equality with collections is in terms of values"
(= (list 1 2 3) (vector 1 2 __))

"You can create a set in two ways"
(= #{} (set __))

"They are another important data structure in clojure"
(= __ (count #{1 2 3}))

"Remember that a set is a 'set'"
(= __ (set '(1 1 2 2 3 3 4 4 5 5)))

"You can ask clojure for the union of two sets"
(= __ (clojure.set/union #{1 2 3 4} #{2 3 5}))

"And also the intersection"
(= __ (clojure.set/intersection #{1 2 3 4} #{2 3 5}))

"But don't forget about the difference"
(= __ (clojure.set/difference #{1 2 3 4 5} #{2 3 5}))

"There are two ways to create maps"
(= __ (hash-map))

"Maps in clojure associate keys with values"
(= __ (count (hash-map)))

"A value must be supplied for each key"
(= {:a 1} (hash-map :a __))

"The size is the number of entries"
(= __ (count {:a 1 :b 2}))

"You can look up the value for a given key"
(= __ (get {:a 1 :b 2} :b))

"Maps can be used as lookup functions"
(= __ ({:a 1 :b 2} :a))
```

```
  "And so can keywords"
  (= __ (:a {:a 1 :b 2}))

  "But map keys need not be keywords"
  (= __ ({2006 "Torino" 2010 "Vancouver" 2014 "Sochi"} 2010))

  "You may not be able to find an entry for a key"
  (= __ (get {:a 1 :b 2} :c))

  "But you can provide your own default"
  (= __ (get {:a 1 :b 2} :c :key-not-found))

  "You can find out if a key is present"
  (= __ (contains? {:a nil :b nil} :b))

  "Or if it is missing"
  (= __ (contains? {:a nil :b nil} :c))

  "Maps are immutable, but you can create a new, 'changed' version"
  (= {1 "January" 2 __} (assoc {1 "January" } 2 "February"))

  "You can also 'remove' an entry"
  (= {__ __} (dissoc {1 "January" 2 "February"} 2))

  "Often you will need to get the keys (which will be in hash order)"
  (= (list __ __ __)
     (sort (keys {2006 "Torino" 2010 "Vancouver" 2014 "Sochi"})))

  "Or the values"
  (= (list "Sochi" "Torino" __)
     (sort (vals {2006 "Torino" 2010 "Vancouver" 2014 "Sochi"})))



(defn multiply-by-ten [n]
  (* 10 n))

(defn square [n] (* n n))


  "Functions are often defined before they are used"
  (= __ (multiply-by-ten 2))

  "But they can also be defined inline"
  (= __ ((fn [n] (* __ n)) 2))

  "Or using even shorter syntax"
  (= __ (#(* 15 %) __))

  "Short anonymous functions may take multiple arguments"
```

```
  (= __ (#(+ %1 %2 %3) 4 5 6))

  "One function can beget another"
  (= __ ((fn []
           ((fn [a b] (__ a b))
            4 5))))

  "Higher-order functions take function arguments"
  (= 25 (___
         (fn [n] (* n n))))

  "But they are often better written using the names of functions"
  (= 25 (___ square))


(defn explain-defcon-level [exercise-term]
  (case exercise-term
        :fade-out          :you-and-what-army
        :double-take       :call-me-when-its-important
        :round-house       :o-rly
        :fast-pace         :thats-pretty-bad
        :cocked-pistol     :sirens
        :say-what?))

  "You will face many decisions"
  (= __ (if (false? (= 4 5))
          :a
          :b))

  "Some of them leave you no alternative"
  (= __ (if (> 4 3)
          []))

  "And in such a situation you may have nothing"
  (= __ (if (nil? 0)
          [:a :b :c]))

  "In others your alternative may be interesting"
  (= :glory (if (not (empty? ()))
              :doom
              __))

  "You may have a multitude of possible paths"
  (let [x 5]
    (= :your-road (cond (= x __) :road-not-taken
                        (= x __) :another-road-not-taken
                        :else __)))

  "Or your fate may be sealed"
  (= __ (if-not (zero? __)
```

```
        'doom
        'doom))

"In case of emergency, sound the alarms"
(= :sirens
   (explain-defcon-level __))

"But admit it when you don't know what to do"
(= __
   (explain-defcon-level :yo-mama))

"The map function relates a sequence to another"
(= [__ __ __] (map (fn [x] (* 4 x)) [1 2 3]))

"You may create that mapping"
(= [1 4 9 16 25] (map (fn [x] __) [1 2 3 4 5]))

"Or use the names of existing functions"
(= __ (map nil? [:a :b nil :c :d]))

"A filter can be strong"
(= __ (filter (fn [x] false) '(:anything :goes :here)))

"Or very weak"
(= __ (filter (fn [x] true) '(:anything :goes :here)))

"Or somewhere in between"
(= [10 20 30] (filter (fn [x] __) [10 20 30 40 50 60 70 80]))

"Maps and filters may be combined"
(= [10 20 30] (map (fn [x] __) (filter (fn [x] __) [1 2 3 4 5 6 7 8])))

"Reducing can increase the result"
(= __ (reduce (fn [a b] (* a b)) [1 2 3 4]))

"You can start somewhere else"
(= 2400 (reduce (fn [a b] (* a b)) __ [1 2 3 4]))

"Numbers are not the only things one can reduce"
(= "longest" (reduce (fn [a b]
                       (if (< __ __) b a))
                     ["which" "word" "is" "longest"]))
```