CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 19 Assignment 3
Nov 3, 2022

Evaluate This

```
(defn foo
  [n]
  (println "Start foo")
  (+ n 10))

(defn bar
  [n]
  (println (str "Start bar " n))
  (+ n 100))
```

```
(println (bar (foo 1)))
```

Output

```
Start foo
Start bar 11
111
```

Now Evaluate This

```
(def page (future (slurp "http://eli.sdsu.edu/")))
```

Question 1

```
(defn bill-total
  [bill]
  (apply + (map #(* (get % :price 0) (get % :quantity 0)) bill)))



  (deftest question1
    (testing "Normal case"
    (are [result input] (= result (bill-total input))
      6 [{:price 2 :quantity 3}]
      8 [{:price 2 :quantity 3} {:price 1 :quantity 2}]
      0 []))

  (testing "Error case"
    (are [result input] (= result (bill-total input))
      0 [{:price 2}])))
```

Question 2

```
(defn add-item [bill item]
  (if (some #(= (:name %) (:name item)) bill)
    (map (fn [x] (if (= (:name x) (:name item))
                   (assoc x :quantity (+ (:quantity x) (:quantity item)))
                   x)) bill)
    (conj bill item)))
```

```
(defn add-items [bill items]
  (reduce add-item bill items))
```

Question 2

```
(deftest question2
  (testing "Normal case"
    (are [result bill items] (= (set result) (set (add-items bill items)))
      [{:name "a" :quantity 5}] [{:name "a" :quantity 3}] [{:name "a" :quantity 2}]
      [{:name "a" :quantity 5} {:name "b" :quantity 1}] [{:name "a" :quantity 3} {:name "b" :quantity 1}]
[{:name "a" :quantity 2}]
      [{:name "a" :quantity 5} {:name "b" :quantity 2}] [{:name "a" :quantity 3} {:name "b" :quantity 1}]
[{:name "b" :quantity 1} {:name "a" :quantity 2}]
      [{:name "a" :quantity 5} {:name "b" :quantity 1}] [{:name "a" :quantity 3}] [{:name "b" :quantity 1}
{:name "a" :quantity 2}]))

  (testing "Error case"
    (are [result bill items] (= (set result) (set (add-items bill items)))
      [{:name "a" :quantity 2}] [] [{:name "a" :quantity 2}]
      [{:name "a" :quantity 3} {:name "b" :quantity 1}] [{:name "a" :quantity 3} {:name "b" :quantity 1}] []
      [{:name "a" :quantity 3} {:name "b" :quantity 2} {:name "c" :quantity 2}] [{:name "a" :quantity 3}
{:name "b" :quantity 1}] [{:name "b" :quantity 1} {:name "c" :quantity 2}]
      [] [] [])))
```

Question 3

```
(defn divisors [n]
  (filter #(zero? (mod n %)) (range 1 (inc n))))


(deftest question3
  (testing "Normal case"
    (is (= [1 2] (divisors 2)))
    (is (= [1 5] (divisors 5)))
    (is (= [1 2 5 10] (divisors 10)))
    (is (= [1 2 4 5 10 20] (divisors 20)))))
```

Question 4

```
(defn abundance
  [n]
  (let [num (apply + (divisors n))]
    (- num (* n 2))))
```

```
(defn abundance
  [n]
  (as-> n ?
    (divisors ?)
    (apply + ?)
    (- ? (* 2 n))))
```

```
(deftest question4
  (testing "Normal Case"
    (is (= 4 (abundance 12)))
    (is (= -4 (abundance 5)))
    (is (= -2 (abundance 10)))
    (is (= 2 (abundance 20)))))
```

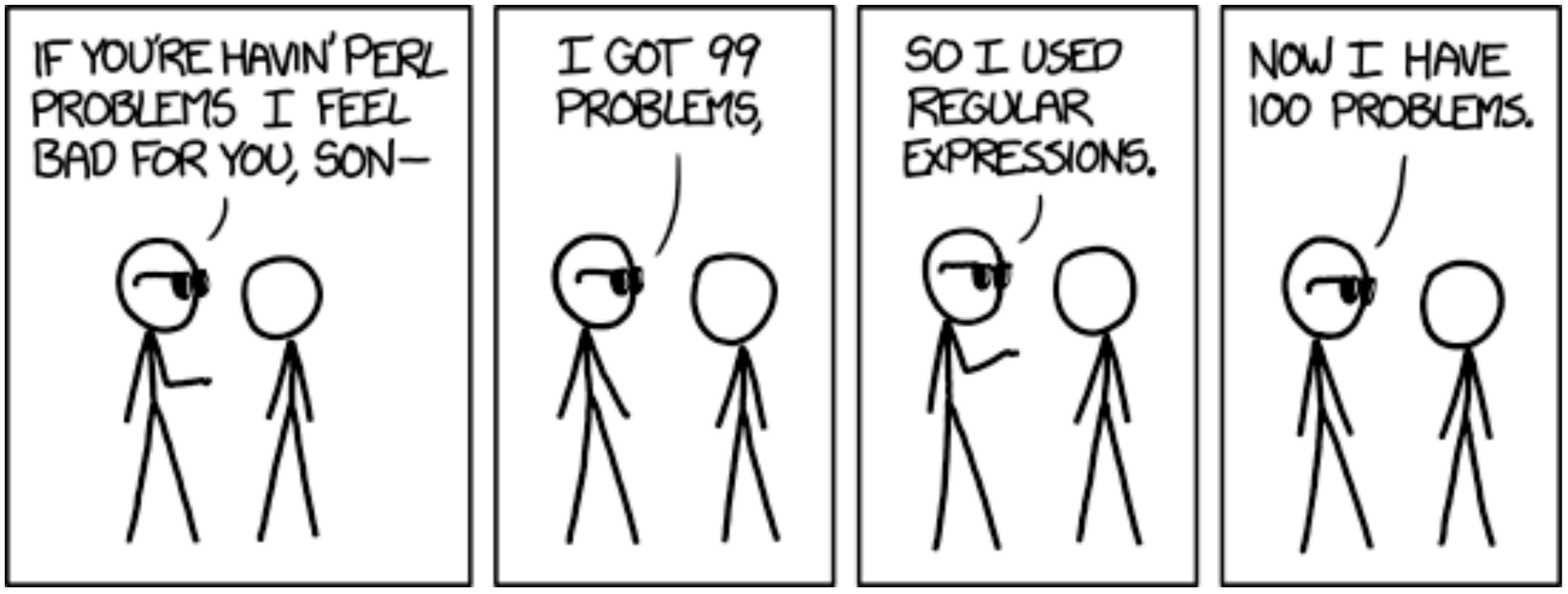
```
(defn divisors [n]
  (for [x (range 1 (inc n))
        :when (= (mod n x) 0)] x))

;; calculate the difference of the sum of diviser and twice the number

(defn calculate-difference [n]
  (let [sum-of-diviser (reduce + (divisors n)) twice-the-number (* n 2)]
    (- sum-of-diviser  twice-the-number)))

;; check is there is a difference, and if it is positive, it returns that the number is abundant

(defn abundance [n]
  (let [difference (calculate-difference n)]
    (cond
      (pos? difference) (str n " is an abundant number and the difference is " difference)
      :else (str n " is not an abundant number"))))
```

Question 5

```
(filter #(> (abundance %) 0) (range 1 300))


(defn abundance?
  [n]
  (pos-int? (abundance n)))

(defn abundance-numbers
  [n]
  (->> (range 1 (inc n))
       (filter abundance?)))


(def question5-answer '(12 18 20 24 30 36 40 42 48 54 56 60 66 70 72 78 80 84 88
90 96 100 102 104 108 112 114 120 126 132 138 140 144 150 156 160 162 168
174 176 180 186 192 196 198 200 204 208 210 216 220 222 224 228 234 240 246
252 258 260 264 270 272 276 280 282 288 294 300))
```

```
(defn count-substring [text substring]
  (count (re-seq (re-pattern (str "(?=" substring ")")) text)))


 (defn pattern-count [text substring]
   (count (re-seq (re-pattern substring) text)))

     (deftest question6
      (testing "normal case"
        (is (= 3 (count-substring "abababa" "aba")))
        (is (= 3 (count-substring "abababa" "ba")))
        (is (= 4 (count-substring "abababa" "a")))
        (is (= 4 (count-substring "baaaaa" "aa")))
        (is (= 3 (count-substring "baaaaa" "aaa")))
        (is (= 3 (count-substring "baaaaa" "aaa")))
        ))
```

```
(defn count-substring [text pattern]
  (count
    (filter
      #{(seq pattern)} (partition (count pattern) 1 text))))
```


IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.