ERIC BAILEY

PAIP EXERCISES

Contents

Introduction to Common Lisp

Using Functions

1 ⟨*titles* 1⟩≡ (7)

```
(defparameter *titles*
  '(Mr Mrs Miss Ms Sir Madam Dr Admiral Major General)
  "A list of titles that can appear at the start of a name.")
```

Defines:
titles, used in chunk 4.

2 ⟨*abstract first-name* 2⟩≡

```
(⟨function first-name(name): 3⟩
  (⟨if the first element of name is a title 4⟩
    ⟨then return the first-name of the rest of the name 5⟩
    ⟨else return the first element of the name 6⟩))
```

3 ⟨*function first-name(name):* 3⟩≡ (2)

```
defun first-name (name)
 "Select the first name from a name represented as a list."
```

4 ⟨*if the first element of name is a title* 4⟩≡ (2)

```
if (member (first name) *titles*)
```

Uses *titles* 1.

5 ⟨*then return the first-name of the rest of the name* 5⟩≡ (2)

```
    (first-name (rest name))
```

6 ⟨*else return the first element of the name* 6⟩≡ (2)

```
(first name)
```

Exercises

7 ⟨*intro.lisp* 7⟩≡

```
(ql:quickload :lisp-unit)

(defpackage #:intro
  (:use #:common-lisp #:lisp-unit))

(in-package #:intro)

⟨titles 1⟩

;; Exercise 1.1
⟨Exercise 1.1 8⟩

;; Exercise 1.2
⟨Exercise 1.2 17⟩

;; Exercise 1.3
⟨Exercise 1.3 24⟩
```

Exercise 1.1

Define a version of `last-name` that handles "Rex Morgan MD," "Morton Downey, Jr.," and whatever other cases you can think of.

8 ⟨*Exercise 1.1* 8⟩≡ (7)

```
⟨suffixes 13⟩

⟨last-name 9⟩

⟨Exercise 1.1 tests 14⟩
```

9 ⟨*last-name* 9⟩≡ (8)

```
(defun last-name (name)
  "Select the last name from a name represented as a list."
  (if ⟨the last element of a name is a suffix 10⟩
      ⟨then return the last-name of all but the last element of the name 11⟩
    ⟨else return the last element of the name 12⟩))
```

Defines:
`last-name`, used in chunks 11, 15, and 16.

First, we check to see if the last element of the `name` is a suffix, i.e. whether it's a member of `*suffixes*`.

10 ⟨*the last element of a name is a suffix* 10⟩≡ (9)

```
(member (first (last name)) *suffixes*)
```

Uses `*suffixes*` 13.

If it is, then drop it from the `name` and return the `last-name` of the result.

11 ⟨*then return the* `last-name` *of all but the last element of the name* 11⟩≡ (9)

```
(last-name (butlast name))
```

Uses `last-name` 9.

Otherwise, it's the last name, so return it.

12 ⟨*else return the last element of the name* 12⟩≡ (9)
```
(first (last name))
```

13 ⟨*suffixes* 13⟩≡ (8)
```
(defparameter *suffixes*
  '(MD Jr)
  "A list of suffixes that can appear at the end of a name.")
```
Defines:
suffixes, used in chunk 10.

14 ⟨*Exercise 1.1 tests* 14⟩≡ (8)
```
(define-test test-last-name
  ⟨Rex Morgan MD 15⟩
  ⟨Morton Downey, Jr 16⟩)
```

15 ⟨*Rex Morgan MD* 15⟩≡ (14)
```
(assert-equal 'Morgan (last-name '(Rex Morgan MD)))
```
Uses last-name 9.

Assert that the last-name of *Rex Morgan MD* is *Morgan*.

16 ⟨*Morton Downey, Jr* 16⟩≡ (14)
```
(assert-equal 'Downey (last-name '(Morton Downey Jr)))
```
Uses last-name 9.

Exercise 1.2

Write a function to exponentiate, or raise a number to an integer power. For example `(power 3 2)` $= 3^2 = 9$.

17 ⟨*Exercise 1.2* 17⟩≡ (7)
```
⟨square 22⟩

⟨power 18⟩

⟨Exercise 1.2 tests 23⟩
```

18 ⟨*power* 18⟩≡ (17)
```
(defun power (x n)
  "Raise x to the power of n."
  (cond ⟨if n is zero return 1 19⟩
        ⟨if n is even return x to the power of n over two, squared 20⟩
        ⟨otherwise return x times x to the power of n minus one 21⟩))
```
Defines:
power, used in chunks 20, 21, and 23.

$$x^n = \begin{cases} 1 & \text{if } n = 0, \\ (x^{n/2})^2 & \text{if } n \text{ is even}, \\ x \times x^{n-1} & \text{otherwise.} \end{cases}$$

19 ⟨*if n is zero return 1* 19⟩≡ (18)
```
((zerop n) 1)
```

$x^0 = 1$

20 ⟨*if n is even return x to the power of n over two, squared* 20⟩≡ (18)
```
((evenp n) (square (power x (/ n 2))))
```
Uses power 18 and square 22.

21 ⟨*otherwise return x times x to the power of n minus one* 21⟩≡ (18)
```
(t (* x (power x (- n 1))))
```
Uses power 18.

22 ⟨*square* 22⟩≡ (17) $\text{square}(x) = x^2$
```
(defun square (x) (expt x 2))
```
Defines:
square, used in chunk 20.

23 ⟨*Exercise 1.2 tests* 23⟩≡ (17)
```
(define-test test-power
  (assert-equal 9 (power 3 2)))
```
Uses power 18.

Exercise 1.3

Write a function that counts the number of atoms in an expression. For example: `(count-atoms '(a (b) c))` = 3. Notice that there is something of an ambiguity in this: should `(a nil c)` count as three atoms, or as two, because it is equivalent to `(a () c)`?

24 ⟨*Exercise 1.3* 24⟩≡ (7)
```
(defun count-atoms (exp)
  "Return the total number of non-nil atoms in the expression."
  (cond ⟨if exp is nil there are no atoms 25⟩
        ⟨if exp is an atom there is only one 26⟩
        ⟨otherwise add the count of the atoms in the first and rest of exp 27⟩))
```
Defines:
count-atoms, used in chunk 27.

25 ⟨*if exp is nil there are no atoms* 25⟩≡ (24)
```
((null exp) 0)
```

26 ⟨*if exp is an atom there is only one* 26⟩≡ (24)
```
((atom exp) 1)
```

27 ⟨*otherwise add the count of the atoms in the first and rest of exp* 27⟩≡ (24)
```
(t (+ (count-atoms (first exp))
      (count-atoms (rest exp))))
```
Uses count-atoms 24.

Chunks

Index

Bibliography