

CM20167 Answers to the quiz about Lisp

Guy McCusker

1W2.1

Question 1: truth-valued functions

- ▶ The “p” in `zerop` etc. is for “predicate”.
- ▶ The function `null` returns `t` when applied to the empty list and `()` on all other values.
- ▶ The function `not` returns `t` when applied to the empty list (because it represents “false”) and `()` on all other values (because everything that is not the empty list represents “true”).
- ▶ You should notice that they’re the same.

I wonder why there are two functions with the same behaviour? Someone suggested it would improve program readability, which is true... but do you really think the designers of Lisp had readability in mind? Look at all those brackets!

Question 2: Lists

Consider the list

```
( ( quizzes ( are fun ) ) )
```

- ▶ The `car` is `(quizzes (are fun))`.
- ▶ The `cdr` is `()`.
- ▶ The list has no `cadr`. The `cadr` would be the `car` of the `cdr`, but the `cdr` is empty.
- ▶ The `cadar` is `(are fun)`.

Question 3: Evaluation

Consider the expression

```
'(car (list '(list '+ 1 2) 3 4))
```

- ▶ This evaluates to (car (list (quote (list (quote +) 1 2)) 3 4)).
- ▶ That evaluates to (list (quote +) 1 2)
- ▶ That evaluates to (+ 1 2)
- ▶ That evaluates to 3
- ▶ That evaluates to 3

Question 4: The empty list

Give three expressions, in three characters or fewer each, that evaluate to the empty list.

Answer: (), '() and nil.

() does not count!

Question 5: higher-order functions

- ▶ A function is called higher-order if it takes other functions as its arguments.
- ▶ The filter function can be written using foldr as follows.

```
(defun keeper (p)
  (lambda (a l)
    (if (p a)
        (cons a l)
        l)
  )
)

(defun foldrfilter (p l)
  (foldr (keeper p) () l)
)
```

Question 6: currying

- ▶ Currying is the process of transforming a function of $n + 1$ arguments into a function of n arguments, which returns a function of one argument, so that `(f a1 a2 ... anplus1)` gives the same as `((curried-f a1 a2 ... an) anplus1)`.

- ▶ The curried plus function is

```
(defun cplus (a)
  (lambda (b)
    (+ a b)
  )
)
```

Question 7: higher-order plumbing with lambda

```
(defun spice (f)
  (lambda (a)
    (lambda (b)
      (f a b)
    )
  )
)
```

```
(defun yoghurt (f)
  (lambda (a b)
    ((f a) b)
  )
)
```

- ▶ If `myfun` is a function of two arguments, the function `(yoghurt (spice myfun))` does exactly the same as `myfun`.
- ▶ `spice` curries functions, and `yoghurt` un-curries functions.