Question 0: AG: Knowing how to move beyond the simplest representations allows you do more things, such as riding a bike without understanding the physics behind it (Ask your physics GSI for details, it's surprisingly complicated.)

**ABSTRACTION.** This is not only the big idea here, but like the question says, this is the big idea in every class in CS.

GS: 2 pts, NPC.

Question 1: AG: Details of words and sentences, as well as or and and as a true and false finder, respectively.

a) The first one returns c. So the way to do these things is to go from the inside out. (butlast cheers) returns cheer, the first of which is c. The last of c is still c. For this one, we also accepted 'c and "c". For future reference, however, 'c will be considered incorrect because 'c is the same as (quote c). This actually does not make sense, since the first procedure returns a word, not a call to another procedure.

**Error!** Remember, bf returns a sentence with the number 7, although first returns the number 10. So, the call is (- 10 (7)). This is a not a number error, since sentences are not numbers.

**maybe.** This was a very commonly missed question. One mistake many students made is that they forgot that or was a true finder. That means, it stops execution as soon as it finds true. (or #f #f #t BIGERROR) returns #t, and (or #f #f 'big-word BIGERROR) returns big-word, since 'big-word is a true value, which gets returned. This accounts for not being a procedure with no calls, as well as for maybe returned instead of true or false.

GS: 1 pt each, NPC, except the second one which was 1/2 point each for answer and explanation.

b) (sentence (bl (sentence (word 'word (quote s)) 'bf)))
(sentence (bl (sentence (word 'word 's ) 'bf))))
(sentence (bl (sentence 'words 'bf)))
(sentence (bl '(words bf)))
(sentence '(words))
'(words)

so it returns (words)

GS: 3 pts for all correct.
-2 for not using all of the words provided
-2 for major parens mistake
-1 for minor parens mistake
-1 for something like ('word ... That open parenthesis doesn't have a quote in front of it, which means that the next thing should evaluate to a procedure. However 'word is a word, not a procedure.
-1 for quoting the s (quote 's)
-1 for no quote on the second word or on bf
-1 for (bf '()) There isn't a first item in the sentence () and so the (bf '()) results in an error
-1 for returning the wrong answer
-1 for calling the procedure word on a sentence. For example (word 'word (quote (s)) is like saying (word 'word '(s)) but the procedure word can only take in words as arguments.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Question 2: AG: Logic, debugging, cond
Calling (exactly-one? false false) should return false but instead returns true.
Changing line # 4 to (else false) fixes the bug so it now works as advertised. There are also many other solutions that are possible.

The big question that most people had was what did line 1 did. For each of the cond cases, people are used to a call like ((number? 3) 'do-something). Remember, scheme interprets that into (true 'do-something). Here, the a simply gets evaluated to true or false. Also, line 2 never gets called, since if a were true, it would evaluate line 1 instead. For that reason, line 3 is only evaluated if b is true (and a is false), which should return true.

It should also be noted that many students treated and as if it were equivalent to equal?, which it is not: (and false false) is false, whereas (equal? false false) is true.

GS:
6 points if entirely correct.
1 point for each blank

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Question 3: AG: Abstraction, data structures
(define (make-bag items weight) (sentence 'bag-containing items 'weighs weight))
(define (bag-items bag) (bl (bl (bf bag))))
(define (bag-weight bag) (last bag))
We got a lot of questions on what it each of them should do. This question calls on you to write a constructor as well as two accessors (or getters). Many people used item to get the items, but remember, since items is a SENTENCE objects, you must use something that returns a sentence, like `butlast` and `butfirst`.

Here are some example calls.

```lisp
> (define *sample-bag* (make-bag '(banana pineapple) 20) )
*sample-bag*
> (bag-items *sample-bag*)
(banana pineapple)
> (bag-weight *sample-bag*)
20
```

Another common mistake was using `make-bag` in the accessors.

```lisp
(define (bag-items bag) (bl (bl (bf (make-bag items weight) ))) )
```

In this case, `bag` is unused, so you aren't accessing the information about items from `bag`. Only one point was deducted for this mistake.

GS:
2 points for `make-bag`.
3 points for `bag-items`, one for each of the `butlast` and `butfirsts`
1 point for `bag-weight`.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

**Question 4:**
AG: The Case Study and the valuable lesson that one can re-use software, you only need converters (written using constructors & selectors)?

```lisp
;; Solution 1
;;-----------------
(define (iso-century-day-span iso-date1 iso-date2)
  (century-day-span
   (iso-date->date iso-date1)
   (iso-date->date iso-date2)))
```

```lisp
;; QUICK AND DIRTY WAY
;; with data abstraction violations abounding
;; (you'll learn later about these) and no constructors or selectors
(define (iso-date->date iso-date)
  (se (bf iso-date) (first iso-date)))
```

```lisp
;; Solution 2
;;-----------------
;; CORRECT WAY
(define (make-date month day year) (se month day year)) ;; constructor
(define (iso-month-name iso-date) (first (bf iso-date))) ;; selector
(define (iso-date-in-month iso-date) (first (bf (bf iso-date)))) ;; selector
```
(define (iso-year iso-date) (first iso-date)) ;; selector

;; INPUT : iso-date (e.g., (90 january 3))
;; RETURNS : date (e.g., (january 3 90))
;; EXAMPLE : (iso-date->date '(90 january 3)) ==> (january 3 90)
(define (iso-date->date iso-date)
    (make-date (iso-month-name iso-date)
               (iso-date-in-month iso-date)
               (iso-year iso-date)))

GS: (6 pts)
6 pts : perfect
5 pts : one trivial mistake
3 pts : get "the idea" but bigger mistakes
0 pts : trying to write century-day-span