Answer the questions in the spaces provided on the question sheets. You should have plenty of room for the answers, but if you run out of room because you need to make a correction, you can use the back of the page (and indicate that you are doing so). The last page of the quiz is a reference sheet for list and table operations. Unless otherwise noted in a question, feel free to use any of the operations defined on the sheet. Please do not use operations that are not defined on the sheet.

Anonymous grading ID: ____________________________________________

Name: ___________________________________________________________

(Please include your name if and only if you don’t remember your anonymous ID)

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding operations</td>
<td>20</td>
</tr>
<tr>
<td>The order-by-total function</td>
<td>20</td>
</tr>
<tr>
<td>The good-list function</td>
<td>25</td>
</tr>
<tr>
<td>Total:</td>
<td>65</td>
</tr>
</tbody>
</table>
The first two questions use the following table:

```plaintext
orders = table: product :: String, unit-price :: Number,
        quantity :: Number, discount-code :: String
    row: "Warm hat", 10, 17, ""
    row: "Winter coat", 50, 2, ""
    row: "Scarf", 12, 10, "CHEAPSCARF"
end
```

Understanding operations  (20 points)
Examine each of the expressions below. Next to each expression, write the output it produces when it is entered in Pyret’s “Interactions” window. None of the programs produce errors.
The orders table, above, has been defined in Pyret’s “Definitions” window.
If an expression returns a list, write it out: for instance, an answer might be

```
[ list: 1, 2, 3 ]
```

(a) 5 points  Expression 1

```plaintext
orders.row-n(0)["product"]
```

(b) 5 points  Expression 2

```plaintext
filter-by(orders, lam(r):
    (r["discount-code"] == ") and
    (r["quantity"] < 15)
end).row-n(0)["unit-price"]
```

(c) 5 points  Expression 3

```plaintext
filter-by(orders, lam(r):
    (r["quantity"] > 5)
end).get-column("quantity")
```

(d) 5 points  Expression 4

```plaintext
L.map(lam(x): x + 2 end,
    [ list: 1, 2, 3 ])
```
The order-by-total function  (20 points)

You've been asked to write a function called order-by-total over a table with the same structure as the orders table (as defined above). It should add a column to the table called total-price, which contains the unit-price and quantity columns multiplied together. It should return the resulting table, sorted by the values in the new column in descending order. It should pass the following test:

orders-answer = table: product :: String,
               unit-price :: Number,
               quantity :: Number,
               discount-code :: String,
               total-price :: Number

   row: "Warm hat", 10, 17, ",", 170
   row: "Winter coat", 50, 2, ",", 100
   row: "Scarf", 12, 10, "CHEAPSCARF", 120
end

order-by-total(orders) is orders-answer

Your function doesn't need to include a docstring or tests, but it should be correctly annotated with types.
The good-list function  (25 points)
You are working for a company with a bug in their database software: it will crash if
any of the numbers 10, 25, or 2019 appear in an input. As such, you’ve been asked to
write a function to ensure that lists do not contain any of these troublesome numbers.

(a) 10 points  bad-number
Write a function bad-number from Numbers to Booleans. It should return true
is the number is 10, 25, or 2019 and false otherwise. Hint: this function should
not require any list or table functions.
(b) The good-list function

Fill in the placeholders in the following definition of the good-list function, which checks to see if any of the bad numbers are present in the list. The function should return false if a bad number is present, and true otherwise. The function should call bad-number in order to determine whether a number is bad.

fun good-list(lst :: List<Number>) -> Boolean:
    doc: "determines whether lst is safe for the database"
    cases (List) lst:
      | empty => Empty case
      | link(fst, rst) => Link case
    end
end

i. 5 points Empty case

ii. 10 points Link case
Table operations in Pyret

- `filter-by(t :: Table, f :: (Row -> Boolean)) -> Table`
  - returns a table of rows in `t` on which `f` returns `true`

- `sort-by(t :: Table, colname :: String, ascending :: Boolean) -> Table`
  - returns a table of the rows in `t` sorted by the value in the `colname` column (in ascending order if `ascending` is true, in descending order otherwise)

- `build-column(t :: Table, colname :: String, f :: (Row -> Value)) -> Table`
  - returns `t` with a new column named `colname` of type `Value`, with values computed by `f`

- `transform-column(t :: Table, colname :: String, f :: (A -> B)) -> Table`
  - returns `t` where the values in column `colname` are the values returned by `f` when it is applied to the original values

- `sum(t :: Table, colname :: String) -> Number`
  - sums the values in column `colname`

- `mean(t :: Table, colname :: String) -> Number`
  - gets the average of the values in column `colname`

- `table.row-n(n :: Number) -> Row`
  - gets the `n`-th row of `table`

- `table.get-column(colname :: String) -> List`
  - gets the values in column `colname` as a list

- `row["colname"]`
  - gets a particular column’s value from a row
List operations in Pyret

- \texttt{L.filter(f :: (A \to \text{Boolean}), lst :: \text{List}<A>) \to \text{List}<A>}
  
  \begin{itemize}
  \item returns a list of the values in \texttt{lst} where \texttt{f} returns \texttt{true}
  \end{itemize}

- \texttt{L.map(f :: (A \to B), lst :: \text{List}<A>) \to \text{List}<B>}
  
  \begin{itemize}
  \item returns a new list with elements obtained by applying \texttt{f} to the elements of \texttt{lst}
  \end{itemize}

- \texttt{L.append(lst1 :: \text{List}, lst2 :: \text{List}) \to \text{List}}
  
  \begin{itemize}
  \item returns a list with the elements of \texttt{lst} followed by those of \texttt{lst2}
  \end{itemize}

- \texttt{link(fst :: A, rst :: \text{List}) \to \text{List}}
  
  \begin{itemize}
  \item returns a list with \texttt{fst} followed by the elements of \texttt{rst}
  \end{itemize}