Problem 1

For each of the following Pyret expressions, write what it will evaluate to or, if it will produce an error, write “error”.

a. "Good luck!" → "Good luck!"  
   1 pt

b. (3 + 3) * 7 → 42  
   2 pts

c. "4" + 2 → Error. Can’t add a string to a number.  
   2 pts

d. "o" == 0 → false  
   string-substring(greeting, 0, 4) → "ahoy"  
   2 pts

e. greeting = "ahoy, world"  
   2 pts

f. if 3 > 2 or 2 < 3:  
   "Good!"
   else:
   "Oh no!"
   end  
   → Error. Missing parentheses in the condition.

g. food-order = "apple cider donut"  
   if food-order == "apple":  
   "$0.25"
   else if string-contains(food-order, "donut"):
   "$2.50"
   else if string-contains(food-order, "apple cider"):
   "$1.50"
   else:
   "Free food!"
   end  
   → "$2.50"
Problem 2

The function below does not work as intended – and it is missing an example that would have exposed this flaw. Read the code carefully and fix both problems.

10 points

You don't need to copy the provided code; just show the correction and additional example.

```plaintext
fun biggest-num(n1 :: Number, n2 :: Number, n3 :: Number) -> Number:
  doc: "Return the biggest of the three input numbers"
  if (n1 > n2) and (n2 > n3):
    n1
  else if (n2 > n1) and (n1 > n3):
    n2
  else:
    n3
end

where:
  biggest-num(3, 2, 1) is 3
  biggest-num(2, 3, 1) is 3
  biggest-num(1, 2, 3) is 3
end

Change the first two questions to be:

(n1 > n2) and (n1 > n3)

and

(n2 > n1) and (n2 > n3)

The case that won't work in the starting function is

biggest-num(2, 2, 1) is 2

since n1 or n2 are the largest, but it's not the case that n1 > n2.
```
Problem 3

Here is a function, `different`, that takes three strings as inputs and checks whether they are all different. However, the programmer forgot to write the examples in the `where` clause. Fill in enough examples to fully test this function.

```haskell
fun different(s1 :: String, s2 :: String, s3 :: String) -> Boolean:
  doc: "Return true if all three strings are different from each other"

  not((s1 == s2) or (s1 == s3) or (s2 == s3))

where:

different("a", "b", "c") is true
different("a", "a", "b") is false
different("a", "b", "b") is false
different("a", "b", "a") is false

end
```
Problem 4

Write a function that takes a Number representing a year from 1980–2022 and returns a String giving the last name of the US president who was in office on January 1st of that year. The relevant election results are to the right.

**Note:** New presidents are normally sworn in on January 20th of the year following their election. See the examples provided in the where block below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Elected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carter</td>
<td>1976</td>
</tr>
<tr>
<td>Reagan</td>
<td>1980</td>
</tr>
<tr>
<td>Reagan</td>
<td>1984</td>
</tr>
<tr>
<td>Bush</td>
<td>1988</td>
</tr>
<tr>
<td>Clinton</td>
<td>1992</td>
</tr>
<tr>
<td>Clinton</td>
<td>1996</td>
</tr>
<tr>
<td>Bush</td>
<td>2000</td>
</tr>
<tr>
<td>Bush</td>
<td>2004</td>
</tr>
<tr>
<td>Obama</td>
<td>2008</td>
</tr>
<tr>
<td>Obama</td>
<td>2012</td>
</tr>
<tr>
<td>Trump</td>
<td>2016</td>
</tr>
<tr>
<td>Biden</td>
<td>2020</td>
</tr>
</tbody>
</table>

fun president-name(year :: Number) -> String:
  doc: "Return the name of the president as of January 1 of the specified year"
  if year <= 1980:
    "Carter"
  else if year <= 1988:
    "Reagan"
  else if year <= 1992:
    "Bush"
  else if year <= 2000:
    "Clinton"
  else if year <= 2008:
    "Bush"
  else if year <= 2016:
    "Obama"
  else:
    "Biden"
end

where:
  president-name(1977) is "Carter"
  president-name(1985) is "Reagan"
  president-name(1992) is "Bush"
  president-name(2000) is "Clinton"
  president-name(2001) is "Bush"
  president-name(2010) is "Obama"
  president-name(2022) is "Biden"
Problem 5

Consider the following function:

fun first-is-last(s :: String) -> Boolean:
  doc: "Return true if the first character of s is the same as the last character"

  len = string-length(s)
  string-substring(s, 0, 1) == string-substring(s, len - 1, len)

where:
  first-is-last("abba") is true
  first-is-last("abbot") is false
end

Write a function first-is-last-n, which generalizes first-is-last so it takes an additional input, n, which is the number of characters to compare at the beginning and end of the string.

That is, calling first-is-last-n("abba", 1) is equivalent to calling first-is-last("abba"), and both are true. However, calling first-is-last-n("abba", 2) would make the function compare "ab" to "ba" and return false.

fun first-is-last-n(s :: String, n :: Number) -> Boolean:
  doc: "Return true if the first n characters of s are the same as the last n characters"

  len = string-length(s)
  string-substring(s, 0, n) ==
  string-substring(s, len - n, len)

where:
  first-is-last-n("abba", 1) is true
  first-is-last-n("abbot", 1) is false
  first-is-last-n("aacaa", 2) is true
  first-is-last-n("abba", 2) is false
end
Problem 6

After the “election” of 2024, the US Government seized all non-profit institutions and sold them off to commercial enterprises. Vassar was acquired by Rump University, who forced the CS department to weaken its major requirements to: 101, 102, 203, 334, and one more 300-level course. Letter grades were eliminated. A transcript was then a table with one column, indicating the courses the student had passed, e.g.,

```
transcript =
  table: course :: Number
    row: 101
    row: 102
    row: 203
    row: 334
    row: 375
end
```

For this problem, design a function `major-complete` that takes a transcript as a parameter and returns a Boolean value indicating whether the corresponding student completed the major requirements. Be sure to include type annotations, docstrings, and examples.

**Hint:** This problem will require you to write multiple functions (including functions defined inside other functions). You’ll find it easier if you make a task plan before you try writing code!

```plaintext
fun passed(cn :: Number, t :: Table) -> Boolean:
  doc: "Return true if the transcript shows the student passed the given course"

  fun row-matches-cn(r :: Row) -> Boolean:
    doc: "Return true if the row matches the course"
    r["course"] == cn
  end

  cn-table = filter-with(t, row-matches-cn)
  cn-table.length() > 0
where:
  passed(101, transcript) is true
  passed(240, transcript) is false
end
```
fun major-complete(t :: Table) -> Boolean:
  doc: "Return true if the student's transcript shows they have completed the CS major"

fun advanced(r :: Row) -> Boolean:
  doc: "Return true if a course is 300-level"
  r["course"] >= 300
end

advanced-courses = filter-with(t, advanced)

passed(101, t) and passed(102, t) and passed(203, t) and passed(334, t) and (advanced-courses.length() >= 2)
where:
  major-complete(transcript) is true
  major-complete(table: course end) is false
end
You can use this page for scratch work or, if you need extra space for any of your answers, you may write them here, circle them, clearly label them, and add a note next to the problem saying that the answer is on the last page.