CMPU 101 · Fall 2022

Exam 1: Solutions

Problem 1

For each of the following Pyret expressions, write what it will evaluate to or, if it will produce an error, write *Error*. Remember to include quotation marks for strings. *15 points*

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

b. `(20 + 2) * -1` → `-22` 2 pts

c. `"2" == 2` → `false` 2 pts

d. `"Grace" + " " + "Hopper"` → `"Grace Hopper"` 2 pts

e. 2 pts

```
greeting = "Ahoy, world!"
string-substring(greeting, 0, 1) → "A"
```

f. 2 pts

```
x = 2
y = x + 2
y * 2 → 8
```

g. 2 pts

```
grade = 100

if grade > 90:
  ":-)"
else if grade > 75:
  ":-/"
else:
  ":-("
end

→ ":-)"
```

h. `(6 > 0) and (6 > 10) or (6 < 8)` → Error (missing parentheses) 2 pts

Problem 2

The function design below has two problems: It has a bug and is missing an example that would have revealed it. Read the code carefully and fix both problems.

10 points

```
fun string-longer(str1 :: String, str2 :: String) -> String:
  doc: "Produce the longer of two strings; first string if lengths are equal."

  if string-length(str1) > string-length(str2):
    str1
  else:
    str2
  end

where:
  string-longer("ab", "xyz") is "xyz"
  string-longer("abc", "zy") is "abc"



end
```

Change > to >=

Add test case:

```
string-longer("abc", "xyz") is "abc"
```

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

Problem 3

Consider the following function to return the state of water given a temperature: *10 points*

```
fun water-state(temp :: Number) -> String:
  doc: "Return a string describing the state of water given its
temperature in degrees Celsius"

  if temp <= 0:
    "solid"
  else if temp < 100:
    "liquid"
  else:
    "gas"
  end

where:

  water-state(-2) is "solid"
  water-state(50) is "liquid"
  water-state(1000) is "gas"


end
```

Fill in the `where:` block with enough examples to fully test this function.

Problem 4

Write a function that takes a Number representing a year from 1861 to 1900 and returns a String giving the last name of the person serving as president of Vassar College that year.

10 points

Name	*Dates*
Milo P. Jewett	1861–1864
John H. Raymond	1864–1878
Samuel L. Caldwell	1878–1885
James Monroe Taylor	1886–1914

For a year when two people served as president, return the new (that is, later) president's name. See the examples provided in the `where:` block below.

This problem should be solved *without* using Pyret tables.

```
fun president-name(year :: Number) -> String:
  doc: "Return the name of the president of Vassar College for a given year"
  if year >= 1886:
    "Taylor"
  else if year >= 1878:
    "Caldwell"
  else if year >= 1864:
    "Raymond"
  else if year >= 1861:
    "Jewett"
  else:
    raise("Vassar pre-history")
  end


where:
  president-name(1886) is "Taylor"
  president-name(1885) is "Caldwell"
  president-name(1870) is "Raymond"
  president-name(1861) is "Jewett"
end
```

Problem 5

10 points

The HAL Corporation has a strange password checker in order to enter corporate headquarters. Rather than a string of characters, a user must enter two numbers in the function `passwd`. It will return the string `"you may enter"` or `"not the password"` as follows:

```
fun passwd(a :: Number, b :: Number) -> String:
  doc: "Check numeric password parts a and b"
  if a < b:
    "not the password"
  else if (a - b) <= 5:
    "not the password"
  else if (a - b) > 10:
    "not the password"
  else:
    "you may enter"
  end
where:
  passwd(4, 7) is "not the password"
  passwd(10, 3) is "you may enter"

  passwd(4, 3) is "not the password"
  passwd(50, 0) is "not the password"

end
```

a. While the function includes two test cases in the `where` block, there's one more possibility that should be added. Write the missing test case above.

b. Dave, the lead programmer, believes there is a way to change the `passwd` function so it doesn't require three separate branches of the `if–else if` statement that all return the same result (`"not the password"`). Rewrite the `if` statement with only two branches (`if` and `else`) so that the same tests pass.

```
if (a < b) or ((a - b) <= 5) or ((a - b) > 10):
  "not the password"
else:
  "you may enter"
end
```

This is the straightforward solution, but you can simplify more if you remove the `a < b` check, since whenever `a < b`, it's also the case that `(a - b) <= 5`.

Problem 6

Consider the following `apples` table and `is-expensive` predicate function over individual rows of the `apples` table: *20 points*

```
apples =
  table: variety,        price, is-ripe, availability
    row: "Gala",         1.99,  true,    "high"
    row: "Honeycrisp",   3.99,  true,    "high"
    row: "Macintosh",    1.99,  true,    "medium"
    row: "Autumn Glory", 3.59,  false,   "low"
  end
```

```
fun is-expensive(r :: Row) -> Boolean:
  doc: "Return true if the apple is expensive"
  r["price"] > 2.50
where:
  is-expensive(apples.row-n(0)) is false
  is-expensive(apples.row-n(1)) is true
end
```

a. Fill in the expected return values in the `where` block for `is-expensive`.

b. Fill in the missing docstring above for `is-expensive`.

c. Write an expression to show the price of Macintosh apples found in the `apples` table. Do not use `filter-with`; just access the value directly.

```
apples.row-n(2)["price"]
```

d. Write an expression to show the value in the `variety` column for the most expensive apple in the `apples` table. Your expression should work even if we update the `apples` table to have additional rows.

```
order-by(apples, "price", false).row-n(0)["variety"]
```

e. Show the `expensive-apples` table that results from running this:

```
expensive-apples = filter-with(apples, is-expensive)
```

You don't need to write this as Pyret code; you can draw the table.

variety	*price*	*is-price*	*availability*
"Honeycrisp"	3.99	true	"high"
"Autumn Glory"	3.59	false	"low"

f. Write a predicate function that takes a Row as input and determines whether the kind of apple in that Row is both ripe and has high availability.

We've helped you get started with the first line of the function. Be sure to include a docstring and a `where:` block with enough examples to fully test the function.

```
fun high-ripe(r :: Row) -> Boolean:


  doc: "Determines whether the kind of apple in that row is both ripe and has
high availability"
  r["is-ripe"] and (r["availability"] == "high")
where:
  high-ripe(apples.row-n(0)) is true
  high-ripe(apples.row-n(2)) is false
  high-ripe(apples.row-n(3)) is false
end
```