Exam 1 Review

16 February 2022
Logistics
75 points / 75 minutes

Aim to budget your time as about one point per minute.
You actually have 120 minutes (the full lab period), so time shouldn’t be an issue.
The exam will be on paper, with the computers put away.

Help me out: Write neatly and make your final answer obvious.
During the exam you can refer to one $8\frac{1}{2} \times 11$-inch piece of paper, double-sided, with any notes you want written or typed on it.

Preparing this sheet of notes is an excellent way to study, encouraging you to consider what's important that you'll want to refer to. They won't be graded, but you'll be asked to turn in your notes with the exam.
During the exam, I’ll decline to answer most questions.

However, if there’s something on the exam that you think is a mistake or is unclear, feel free to ask, and I might give a correction or clarification to everyone.
All the sections of CMPU 101 are taking their exams on Friday, at different times.

You are on your honor not to discuss the contents of the exam with anyone else.
You are generally responsible for the material in

Chapters 1–8 of *A Data-Centric Introduction to Computing*,
Classes 1–8,
Labs 1–4, and
Assignments 1–3.
How to study:

(Re)read the textbook and do the exercises labeled “Do Now!”
(Re)read the lecture slides and code
Redo the labs (without looking at your previous solutions)
Review problems
You should expect approximately six questions, along the lines of the following.
1 Pyret expressions
Given a series of Pyret expressions, indicate either what each expression evaluates to or whether it results in an error.

Expressions may involve:

- values of type Number, String, Boolean (no Image values)
- functions that take and return those types of values
- operations/operators over those types of values
- parenthesized expressions
- named values of the form `name = expression`
- `if–else` expressions
>>> "Vassar"

>>> True

>>> 42
>>> "Vassar"
"Vassar"

>>> True
true
true

>>> 42
42
42
42 + 1

"42" + 1
42 + 1
43

"42" + 1

Error! Does not compute!
>>> 42 + 1
43

>>> string-to-number("42").value + 1
43
```python
>>> 42 < 43

>>> 42 > 43

>>> 42 < 43 and 42 == 43
```
42 < 43
true

42 > 43
false

>>> 42 < 43 and 42 == 43
Error!
>>> 42 < 43
true

>>> 42 > 43
false

>>> (42 < 43) and (42 == 43)
false
>>> n = 42

>>> n == 42
>>> n = 42

>>> n == 42

Defining a name to have a value
>>> n = 42  \hspace{2cm} \text{Defining a name to have a value}

>>> n == 42  \hspace{2cm} \text{Asking if two values are equal}
When a name occurs in an expression, Pyret looks it up and replaces it with a value!
>>> n = 42

>>> 42 == 42

When a name occurs in an expression, Pyret looks it up and replaces it with a value!
>>> n = 42

>>> 42 == 42
true
if "Rose" == "rose":
    "Rose is a rose is a rose is a rose."
else:
    "Gertude Stein condemns Pyret."
end

Cf. Wikipedia:
Rose is a rose is a rose is a rose.
If "Rose" == "rose":  
   "Rose is a rose is a rose is a rose."
else:  
   "Gertude Stein condemns Pyret."
end

Cf. Wikipedia: 
Rose is a rose is a rose is a rose.
if false:
    "Rose is a rose is a rose is a rose is a rose."
else:
    "Gertude Stein condemns Pyret."
end

Cf. Wikipedia:
Rose is a rose is a rose is a rose.
"Gertude Stein condemns Pyret."

Cf. Wikipedia:
Rose is a rose is a rose is a rose.
if true:
  ⟨x⟩
else:
  ⟨y⟩
end

if false:
  ⟨x⟩
else:
  ⟨y⟩
end
What is the value of the following expression?

```python
if (2 < 3) or (2 > 3):
    2 == 3
else:
    not(2 == 3)
end
```
2 Fixing a function
Given a function, identify and fix its problems.

Identifying the problems may involve carefully reading the function header, docstring, examples, and body.

If the body and/or tests don’t match the behavior described in the docstring, fix whatever is wrong so they do.
Part 1: Warm up

You've been asked to review a program written by a coworker. The problem they were asked to solve was to turn a numeric grade into its corresponding letter grade.

Your coworker shows you this solution:

```haskell
fun grade(percent :: Number) -> String:
  doc: "Return the letter grade for a percent."
  if percent > 90:
    "A"
  else if percent >= 80:
    "B"
  else if percent >= 70:
    "C"
  else if percent >= 60:
    "D"
  else:
    "F"
end
end
```

**TASK:** Answer the following questions:

- Why doesn't this solution work as intended?
- Why wouldn't your coworker realize this function isn't right? What should they have done?
3 Incomplete function: missing examples
Given a function that doesn’t have any examples, fill in the \textit{where} clause, ensuring that you choose examples that fully test the function.
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
end

where:
  # Fill in the where block with sufficient examples.
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:
    # Fill in the where block with sufficient examples.
    water-state(-2) is "solid"
    water-state(50) is "liquid"
    water-state(1000) is "gas"
4 Incomplete function: missing body
This time you’re given the `where` clause for a function, but the rest of the function is missing!

Write a function header, docstring, and body that match the examples.
fun
doc:

where:

  grade-to-school(0) is "elementary"
  grade-to-school(5) is "elementary"
  grade-to-school(6) is "middle"
  grade-to-school(8) is "middle"
  grade-to-school(9) is "high"
  grade-to-school(12) is "high"

end
fun grade-to-school(grade :: Number) -> String:
    doc: "Return the type of school someone is in given their grade number"
    if grade <= 5:
        "elementary"
    else if grade <= 8:
        "middle"
    else:
        "high"
end
where:
    grade-to-school(0) is "elementary"
    grade-to-school(5) is "elementary"
    grade-to-school(6) is "middle"
    grade-to-school(8) is "middle"
    grade-to-school(9) is "high"
    grade-to-school(12) is "high"
end
5  Reading and understanding code
Given a function,

indicate which tests pass or fail

Briefly describe (in words, not code) what you’d need to change to make sure all the tests pass.
6 Table processing
Given a table and a function that takes a row from the table:

- Fill in the expected return values in the `where` clause.
- Write the docstring for the function.
- Write an expression to access the value in a particular row and column of the table.
- Write an expression that involves some form of table processing to compute/locate a particular value within the table.
- Given a table expression show the resulting table.
- Write a predicate function that takes a row of the table and determines whether that row satisfies (returns `true`) the given criteria.
You’ve been hired as a consultant for a major political candidate. The campaign keeps track of donations using a table of the form

```
table:
  donor :: String, amount :: Number
...
end
```

Where each value in the amount column is the number of dollars donated, and the donor column has blank entries (that is, the string "") when the donor is anonymous.

The rules of campaign donation declare that a single contributor may only donate $2,800 to a given campaign. Furthermore, anonymous donations are limited to $50.
a. The first task you are given is to design a program \texttt{any-bad-donations} that takes a table of donations as input and returns \texttt{true} if any of the donations in the table are illegal ones.

To make things simpler, you should first design a helper function \texttt{is-bad-donation} that answers this question for a single row.
test-table =
  table:
    donor :: String, amount :: Number
    row: "Lynn Burke", 3000
    row: "Robert Wilkins", 50
    row: "", 100
    row: "", 10
  end

We can’t design the functions if we don’t have some test data to write examples with. So, first we define a test table.
test-table =

table:
  donor :: String, amount :: Number
  row:  "Lynn Burke", 3000
  row:  "Robert Wilkins", 50
  row:  "", 100
  row:  "", 10
end

fun is-bad-donation(r :: Row) -> Boolean:
  doc: "Return true if a donation exceeds the applicable limit"

  ...

where:
  is-bad-donation(test-table.row-n(0)) is true
  is-bad-donation(test-table.row-n(1)) is false
  is-bad-donation(test-table.row-n(2)) is true
  is-bad-donation(test-table.row-n(3)) is false
end

We were told to write a helper function. It makes sense to do that next.
test-table =

  table:
    donor :: String, amount :: Number
  row: "Lynn Burke", 3000
  row: "Robert Wilkins", 50
  row: "", 100
  row: "", 10
  end

fun is-bad-donation(r :: Row) -> Boolean:
  doc: "Return true if a donation exceeds the applicable limit"
  ((r["donor"] == "") and (r["amount"] > 50))
  or
  (r["amount"] > 2800)
where:
  is-bad-donation(test-table.row-n(0)) is true
  is-bad-donation(test-table.row-n(1)) is false
  is-bad-donation(test-table.row-n(2)) is true
  is-bad-donation(test-table.row-n(3)) is false
end

We were told to write a helper function. It makes sense to do that next.
test-table =
  table:
    donor :: String, amount :: Number
  row: "Lynn Burke", 3000
  row: "Robert Wilkins", 50
  row: 
  row: 
end

fun is-bad-donation(r :: Row) -> Boolean:
  doc: "Return true if a donation exceeds the applicable limit"
  ...
end

fun any-bad-donations(donations :: Table) -> Boolean:
  ...
where:
  any-bad-donations(test-table) is true
  any-bad-donations(table: donor, string end) is false
end

And next we need to write the main function, using our helper.
test-table =
  table:
    donor :: String, amount :: Number
    row: "Lynn Burke", 3000
    row: "Robert Wilkins", 50
    row: "", 100
    row: "", 10
  end

fun is-bad-donation(r :: Row) -> Boolean:
  doc: "Return true if a donation exceeds the applicable limit"
  ...
end

fun any-bad-donations(donations :: Table) -> Boolean:
  bad-donations = filter-with(donations, is-bad-donation)
  bad-donations.length() > 0
where:
  any-bad-donations(test-table) is true
  any-bad-donations(table: donor, string end) is false
b. Some nefarious special-interest groups are attempting to get around the laws on campaign donations by making multiple donations. For example, the wealthy but unscrupulous donor “Netochka” might try to make two donations of $2,800 each:

```table
  donor :: String, amount :: Number
  row: "Netochka", 2800
  row: "Netochka", 2800
end```

Design a program `donor-total` that will take as input the name of a donor and a table of donors and produce the total amount donated by that donor.

For example, it would tell us that the total amount donated by donor “Netochka”, given the list of donors above, is $5,600, which violates the donation limits.
test-table2 =
table: donor :: String, amount :: Number
  row: "Netochka", 2800
  row: "Netochka", 2800
  row: "Alice", 50
end
test-table2 =
  table: donor :: String, amount :: Number
    row: "Netochka", 2800
    row: "Netochka", 2800
    row: "Alice", 50
end

fun donor-total(donor :: String, donors :: Table) -> Number:
  doc: "Return the sum of all contributions from the specified donor"
  ...
  where:
    donor-total("Alice", test-table2) is 50
    donor-total("Netochka", test-table2) is 5600
end
fun donor-total(donor :: String, donors :: Table) -> Number:
  doc: "Return the sum of all contributions from the specified donor"

  fun is-right-donor(r :: Row) -> Boolean:
    r["donor"] == donor
  end

  all-by-donor = filter-with(donors, is-right-donor)
  sum(all-by-donor, "amount")

where:
  donor-total("Alice", test-table2) is 50
  donor-total("Netochka", test-table2) is 5600
end
fun donor-total(donor :: String, donors :: Table) -> Number:
  doc: "Return the sum of all contributions from the specified donor"

  all-by-donor = filter-with(
    lam(r): r["donor"] == donor end, is-right-donor)

  sum(all-by-donor, "amount")

where:
  donor-total("Alice", test-table2) is 50
  donor-total("Netochka", test-table2) is 5600
end

Alternatively, using a lambda expression:
Acknowledgments

This class incorporates material from:

Marc Smith, Vassar College
Jason Waterman, Vassar College