Problem Solving and Abstraction
(CMPU 101)

Tom Ellman
Lecture 9
Exam 1 Review
Logistics

• Exam is designed to be doable in 75 minutes.

• You actually have 120 minutes.

• Time should not be an issue.
Medium

• Paper only – computer / phones put away.
• Please, please write neatly.
• Make clear what is your final answer.
Notes

• During the exam you can refer to one 8½×11-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.
Questions During Exam

- I will probably decline to answer most questions.

- But, 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.
Honor and Integrity

• 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 who has not yet taken it.
Coverage

- Textbook chapters 1 – 8
- Class meetings 1 – 8
- Labs 1 – 4
- 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 and/or assignments without looking at your previous solutions
Details

• About six questions.

• Along the lines of the following.
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 Images)
- functions that take and return those types of values
- operations/operators over those types of values
- parenthesized expressions
- named values of the form

\[ \text{name} = \text{expression if–else expressions} \]
"Vassar"
true
42
42 + 1
"42" + 1
42 + 1

string-to-number("42").value + 1
42 < 43
42 > 43
42 < 43 and 42 == 43

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

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

if not((2 < 3) or (2 > 3)):    ????
    2 == 3
else:
    not(2 == 3)
end
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.
fun weight-class(n :: Number) -> String:
  doc: "Wrestling classes for Robot Olympics"
  if (n <= 400):
    "middle-weight"
  else if (n <= 300):
    "light-weight"
  else if (n <= 200):
    "feather-weight"
  else:
    "heavy-weight"
end
where:
  weight-class(150) is "feather-weight"
  weight-class(425) is "heavy-weight"
end
Comparisons out of order.
Insufficient test cases.
Incomplete Function: Missing Examples

Given a function that doesn’t have any examples, fill in the `where` clause, ensuring that you choose examples that fully test the function.
Add a **where** block with sufficient examples.

```java
fun water-state(temp :: Number) -> String:
  doc: "Return a string describing a possible state of water given its temperature in degrees Celsius"
  if temp <= 0:
    "solid"
  else if temp < 100:
    "liquid"
  else:
    "gas"
end
end
```
Be careful to test boundary cases.
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 mystery ...
   ...
where:
    mystery(0) is "elementary"
mystery(5) is "elementary"
mystery(6) is "middle"
mystery(8) is "middle"
mystery(9) is "high"
mystery(12) is "high"
end
fun grade-to-school(grade :: Number) -> String:
  doc: "Given grade return type of school."
  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
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.
Table Processing

Given an 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
```

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.
First Task

The first task you are given is to design a program **any-bad-donations** that takes a table of donations as input and returns true if any of the donations in the table are illegal ones. To make things simpler, you should first design a helper function **is-bad-donation** that answers this question for a single row.
First Task

The first task you are given is to design a program `any-bad-donations` that takes a table of donations as input and returns true if any of the donations in the table are illegal ones. To make things simpler, you should first design a helper function `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

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

fun any-bad-donations(donations :: Table) -> Boolean:
  ...
where:
  any-bad-donations(test-table) is true
  any-bad-donations(table: donor, amount end) is false
end
test-table = **table**: donor :: String, amount :: Number

  row: "Lynn Burke", 3000
  row: "Robert Wilkins", 50
  row: "", 100
  row: "", 10

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, amound end) is false

end
Second Task

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>
<thead>
<tr>
<th>donor</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Netochka&quot;</td>
<td>2800</td>
</tr>
<tr>
<td>&quot;Netochka&quot;</td>
<td>2800</td>
</tr>
</tbody>
</table>
```

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
end

fun donor-total(donor :: String, donors :: Table) -> Number:
    doc: "Return the sum of all contributions from donor"
    ...
where:
    donor-total("Alice", test-table2) is 50
    donor-total("Netochka", test-table2) is 5600
end
test-table2 = table: donor :: String, amount :: Number
  row: "Netochka", 2800
  row: "Netochka", 2800
end

fun donor-total(donor :: String, donors :: Table) -> Number:
doc: "Return the sum of all contributions from 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
Acknowledgments

This class incorporates material from:

• Marc Smith, Vassar College
• Jason Waterman, Vassar College
• Jonathan Gordon, Vassar College