CSCI 0111: Sample midterm questions

Oct 16, 2020

These questions have been used on past CSCI 0111 exams. They are somewhat more programming-focused than our midterm will be this year, but should still be useful for studying!

Question	Points
Erroneous expressions?	30
The `in-order` function	20
The `eye` function	20
Understanding operations	10
The `order-by-total` function	20
Total:	100

Erroneous expressions? (30 points)

Examine each of the programs below. Next to each program, write either (1) the output the program produces in Pyret's "Interactions" window when it is run, or (2) the word "ERROR" if the program produces an error. These programs are not designed to trip you up; if they are wrong, they are not wrong for "trivial" reasons such as missing colons or the like.

(a) [5 points] Program 1

```
x = 1
x + 3
```

(a) _____

(b) [5 points] Program 2

```
x = 1
if x:
  "a"
else:
  "b"
end
```

(b) _____

(c) [5 points] Program 3

```
x = false
if x:
  "a"
else:
  "b"
end
```

(c) _____

(d) [5 points] Program 4

```
# This function's definition is the same
#  in Programs 4, 5, and 6
fun greater-than-five(x :: Number) -> Boolean:
  x > 5
end

greater-than-five(6)
```

(d) _____

(e) [5 points] Program 5

```
# This function's definition is the same
#  in Programs 4, 5, and 6
fun greater-than-five(x :: Number) -> Boolean:
  x > 5
end

3 + greater-than-five(6)
```

(e) _____

(f) [5 points] Program 6

```
# This function's definition is the same
#  in Programs 4, 5, and 6
fun greater-than-five(x :: Number) -> Boolean:
  x > 5
end

if greater-than-five(6):
  "a"
else:
  "b"
end
```

(f) _____

The `in-order` function (20 points)

We want to define a function `in-order` that takes three numbers and determines whether its arguments are in sorted order. Here's what the function looks like, with some `placeholders`.

```
fun in-order(
  x :: Type of x,
  y :: Type of y,
  z :: Type of z)
-> Output type:
  Function body
where:
  in-order(1, 1, 1) is true
  in-order(1, 2, 2) is true
  in-order(1, 2, 5) is true
  in-order(3, 3, 2) is false
  in-order(3, 2, 1) is false
  in-order(3, 1, 4) is false
end
```

What should we replace the placeholders with so that the function satisfies its specification and passes all of the tests in the `where` block?

(a) [2 points] `Type of x`

(a) _____

(b) [2 points] `Type of y`

(b) _____

(c) [2 points] `Type of z`

(c) _____

(d) [4 points] `Output type`

(d) _____

(e) [10 points] `Function body`

(e) _____

The `eye` function (20 points)

You've written a program that draws a complex scene, with multiple interacting characters. The program is done, but you're looking for ways to clean it up. You notice that you've written similar code in a few places to draw eyes. For instance, in one location you've written

```
overlay(
  circle(15, "solid", "green"), # iris
  circle(50, "solid", "white")  # eyeball
)
```

In another, you've written

```
overlay(
  circle(15, "solid", "brown"), # iris
  circle(50, "solid", "white")  # eyeball
)
```

Write an `eye` function that can be used to reduce this repetition. You do *not* need to include a docstring, comments, or tests for this function.

The next two questions use the following table:

```
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 (10 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.

(a) [5 points] Expression 1

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

(a) _____

(b) [5 points] Expression 2

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

(b) _____

The `order-by-total` function (20 points)

You've been asked to write a function called `order-by-total`, which takes 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.