CSCI 0111: Quiz 1

Sep 27, 2019

Answer the questions in the spaces provided on the question sheets. You should have plenty of room for the answers, but if you run out of room because you need to make a correction, you can use the back of the page (and indicate that you are doing so).

Anonymous grading ID: ________________________________

Name: ________________________________

(Please include your name if and only if you don’t remember your anonymous ID)

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erroneous expressions?</td>
<td>30</td>
</tr>
<tr>
<td>The in-order function</td>
<td>20</td>
</tr>
<tr>
<td>The eye function</td>
<td>20</td>
</tr>
<tr>
<td>Just for fun: Testing image functions</td>
<td>0</td>
</tr>
<tr>
<td>Total:</td>
<td>70</td>
</tr>
</tbody>
</table>
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) 4

(b) 5 points Program 2

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

(b) ERROR

(c) 5 points Program 3

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

(c) "b"
(d) **5 points** Program 4

```plaintext
# 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

```plaintext
# 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

```plaintext
# 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

(b) 2 points Type of y

(c) 2 points Type of z

(d) 4 points Output type

Number

(a) ____________________________

(b) ____________________________

(c) ____________________________

(d) ____________________________

Boolean

(x <= y) and (y <= z)

(e) 10 points Function body
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

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

In another, you’ve written

```plaintext
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.

```plaintext
fun eye(color :: String) -> Image:
    overlay(circle(15, "solid", color),
            circle(50, "solid", "white"))
end
```
Just for fun: Testing image functions  (0 points)

In general, in CSCI 0111 you aren’t asked to write tests for functions that produce images. Take a minute to think about why this is.

Now, try to design a system for testing image-producing functions. How would a programmer use the system? Would it require pixel-perfect accuracy?

Describe your system below.