**Making Change Narrative**

The assignment was to write a program using hashes to make change. I set up a test file called “make\_change\_test2.rb”first to test different change amounts. Next, I set up a different file called “change\_maker2.rb” where I entered the code that would actually make the change using quarters, dimes, nickels and pennies. To begin the test I entered:

require “minitest/autorun”

require\_relative “change.maker2.rb”, then I set up the class:

class TestChangeMaker < Minitest::Test

Once this was completed, then I could begin to add in my different tests to see if my “change\_maker2.rb” was working properly. To begin my “change\_maker2.rb”, I defined my function like this:

def change\_maker(cents), then I set up the hash for my coins, setting them to zero for now.

total = {“quarters” => 0, “dimes” = > 0, “nickels” => 0, “pennies” => 0}

then, ending the function with “end”.

Now, to test if change was zero in my test program:

def\_test\_change\_is\_0

assert\_equal({“quarters” => 0, “dimes” => 0, “nickels” => 0, “pennies” => 0}, change\_maker(0)), then end my function of course with

end

This first test passed. Now, I moved on to test for anything under 5 cents, so my change would be anywhere from 1-4 pennies. Within the test program, I made new tests to pass for change of 1 and 3 pennies, so basically:

def\_test\_change\_is\_1

assert\_equal({“quarters” => 0, “dimes” => 0, “nickels” => 0, “pennies” => 1}, change\_maker(1)), then end my function of course with

end

def\_test\_change\_is\_3

assert\_equal({“quarters” => 0, “dimes” => 0, “nickels” => 0, “pennies” => 3}, change\_maker(3)), then end my function of course with

end

If the tests pass for 1 and 3, it seemed logical they would pass for 2 and 4 pennies as long as long as I passed in the correct values. Then, within my “change\_maker2.rb” file, I set “pennies” => cents within the hash for total. So now, if I pass in 1,2,3,or 4, it should pass the tests. It did.

Next, I would move on to nickels. So, first I will set up a new test within my test program:

def\_test\_change\_is\_5

assert\_equal({“quarters” => 0, “dimes” => 0, “nickels” => 1, “pennies” => 0}, change\_maker(5)), then end my function of course with

end

So, then to change my “change\_maker2.rb” to bring in nickels, I added if/else to the code:

if cents >= 5

total[“nickels”] = 1

else

total[“pennies”] = cents

end

Now, I add in total after my if/else end, then another end for the function change\_maker.

So, now, the test should work for 5 and still to continue to pass for 1-4, as if my cents are set as 5, it will change a nickel, and if it’s less than that, it will proceed to the else statement. I did get this to pass.

Moving on to test change greater than a nickel but less than a dime, I made up a test for change of 6:

def\_test\_change\_is\_6

assert\_equal({“quarters” => 0, “dimes” => 0, “nickels” => 1, “pennies” => 1}, change\_maker(6)),

end

My code will now need to look at combining coins to make change. So, I altered the code to this with nothing changing yet above this point in the code:

if cents >= 5

total[“nickels”] = 1

cents = cents – 5

total{“pennies”} = cents #so this should produce the pennies change left

else

total[“pennies”] = cents

end

total

end

If this code passed, I assumed it would also pass for 7,8, and 9 as long as I made no errors when passing in the correct amount to the test like for 7: 1 nickel, 2 pennies for change\_maker(7), etc. That passed as well, so now I can move on to dimes.

For the test program, I added the test when change was 10:

def\_test\_change\_is\_10

assert\_equal({“quarters” => 0, “dimes” => 1, “nickels” => 0, “pennies” => 0}, change\_maker(10)),

end

Initially, I changed my if statement:

if cents == 10

total[“dimes”] = 1

By making this change, I then turned my if cents => 5, into and elsif statement:

elsif cents >= 5

total{“nickels”} = 1

cents = cents – 5

total {“pennies”} = cents

Staying the same:

else

total[“pennies”] = cents

end

total

end

Moving on to test change of 12, I made a new test once again:

def\_test\_change\_is\_12

assert\_equal({“quarters” => 0, “dimes” => 1, “nickels” => 0, “pennies” => 2}, change\_maker(12)),

end

Back to changing the code to make this test pass, I changed the if cents == 10 statement to this::

if cents >= 10

total[“dimes”] = 1

cents = cents – 10

total[“pennies”] = cents

I didn’t change anything below this point, so my elsif and else statements stayed put. The change\_of\_12 test also passed.

Moving on to test for 15 in change, I once again entered a new test very similar to all the others:

def\_test\_change\_is\_15

assert\_equal({“quarters” => 0, “dimes” => 1, “nickels” => 1, “pennies” => 0}, change\_maker(15)),

end

Heading back to the code, I changed it to pretty much the same as I did when I tested for a dime:

if cents == 15

total[“dimes”] = 1

total[“nickels”] =1

Then, changing the elsif to:

elsif cents >= 10

Keep code same below this line

Now, to get the test to pass for change from essentially 16-19 cents. Again, right a new test for say 18:

def\_test\_change\_is\_18

assert\_equal({“quarters” => 0, “dimes” => 1, “nickels” => 1, “pennies” => 3}, change\_maker(18)),

end

Code changes to:

if cents >= 15

total [“dimes”] = 1

total[“nickels] = 1

cents = cents -15

total [“pennies”] = cents

Code remains below the last elsif line.

Moving on to test 20 as change:

def\_test\_change\_is\_20

assert\_equal({“quarters” => 0, “dimes” => 2, “nickels” => 0, “pennies” => 0}, change\_maker(20)),

end

When I got to this point, I refactored the code with these changes:

After my hash, I entered these values:

dimes = 10

nickels = 5

pennies = 1

changing the if to:

if cents >= dimes (replacing the if cents>=15)

total[“dimes]” = 1 (no change)

cents = cents – dimes

end

if cents >= nickels (this is a change)

total[“nickels”] = 1 (no change)

then remove all those lines all the way down through else and add:

cents = cents – nickels

end

if cents >= pennies

total[“pennies”] = cents

end

total these last 3 lines stay the same from before

end

Getting close to finishing, I changed the if’s in to a while loop. So this refactoring was made:

Getting rid of the if cents>=dimes

And total[“dimes”] = 1:

while cents >= dimes

total[“dimes”] = total[“dimes”] +1

then proceed with the next line of cents = cents – dimes

This code passed with the while loop too.

Last part was to add in the quarters, so above my dimes = 10, I just added

quarters = 25

then under the pennies = 1, I put in a while loop for the quarters as well:

while cents >= quarters

total[“quarters”] = total[“quarters”] + 1

cents = cents = quarters

end

No changes after this point in the code.

Now to test something where all 4 different coins are needed to make change:

def\_test\_change\_is\_94

assert\_equal({“quarters” => 3, “dimes” => 1, “nickels” => 1, “pennies” => 4}, change\_maker(94)),

end

Basically here, I put in another hash my “total” hash to push a value to the coins. So like this:

Coins = {“quarters” => 25, “dimes” => 10, “nickels” => 5, “pennies” => 1}

Then, I was able to eliminate all the lines from quarters = 25 all the way through the last if statement. So, my new function became:

coins.each do|coin,value|

while cents >= value

total[coin] = total[coin] +1

cents = cents – value

end

end (first end for the each.do, second end for the change\_maker function)

total

end

The 94 test also passed.

So, after much manipulation and refactoring, I felt I was able to get this into a much ore concise form using the each do function and the while loop. If not, I would have had an endless amount of elsifs between my if and else. So barring, I didn’t make too many errors typing up this explanation, I hope I explained as detailed as possible what I did in this assignment.

Dolly Throckmorton