Exercise - Making Change

Do you remember physical money? In olden times I used to collect small change in a jar and periodically sort through it so I could do my laundry. I always seemed to have too many of the coins I *didn't* need, and to few of the ones I *did* need! This problem is about making change; it's also about making *functions*. To answer it you'll need two pieces of outside information. First: US coinage consists of pennies ($0.01), nickels ($0.05), dimes ($0.10), quarters ($0.25), and dollar coins ($1.00). Second: UK coinage consists of pence, 2p coins, 5p coins, 10p coins, 20p coins, 50p coins, 1 pound coins, and 2 pound coins.

You'll also need to use the R operators `%/%` and `%%`. These represent *integer division* and *modulo*. For example `16` divided by `3` equals `5` with a remainder of `1` so `16 %/% 3` returns `5` while `16 %% 3` returns `1`. Before beginning this exercise, experiment with `%/%` and `%%` to make sure you understand how they work. If necessary, search the internet to find out more about them or read the R help file "Arithmetic Operators."

For technical reasons that I won't delve into here, the simplest reliable way of representing monetary values in a computer is by using *whole numbers* rather than decimals. Rather than representing fourteen pounds and thirty-two pence as `14.32`, for example, we'd store this value as `1432`. In short, we store the number of *pence* (or cents) as an *integer*. Please follow this convention throughout.

1. The following code block outlines a simple algorithm for making change. The customer is owed a certain number of `cents`, an integer between `1` and `99`, and your task is to calculate how many `quarters`, `dimes`, `nickels`, and `pennies` to remove from the cash register. The rule is to always give as many *high denomination* coins as you can before moving on to lower denomination coins. If the customer is owed $0.75, for example, you remove three quarters rather than seven dimes and a nickel. Fill in the gaps in the code, run it, and display the resulting vector `change`. Check the result by hand to make sure your logic is correct.

```
cents <- 73
quarters <- cents %/% 25
cents <- cents %% 25

dimes <- ______ # Delete the underscores & fill in the gap!
cents <- ______ # ditto!

nickels <- ______ # ditto!
cents <- ______ # ditto!

change <- c('quarters' = quarters,
            'dimes' = dimes,
            'nickels' = nickels,
            'pennies' = cents)
```

2. You've written some handy code! But now suppose you wanted to reuse it to make change for a different amount, say $0.37 or $0.19. Copying-and-pasting your existing code is tedious and error prone. Instead of doing this, write a *function* called `make_change_US()`. It should take a single input argument `cents`, a integer between `1` and `99`, and return a vector called `change` listing the number of quarters, dimes, nickels, and pennies formatted as above. Run your function to make change for $0.37 and $0.19 and check the results against your hand calculations.
3. The function `make_change_US()` has some limitations. Suppose we wanted to make change for *UK* currency rather than US currency. This would require us to write a completely new function despite the underlying logic of the problem being identical. There's a better way forward. Write a new function called `make_change()` that it takes *two* input arguments. The first argument is `cents`. As before, this is a integer between `1` and `99`. The second argument is new: `currency` is a named vector that stores the denominations between `1` and `99` cents along with their labels. For example we would set `currency` equal to `c('quarter' = 25, 'dime' = 10, 'nickel' = 5, 'penny' = 1)` for US currency and `c('50p' = 50, '20p' = 20, '10p' = 10, '5p' = 5, '2p' = 2, '1p' = 1)` for UK currency. As above, `make_change()` should return a named vector called `change` indicating how many of each coin to give as change, but now this vector should take its names from `currency`. The logic is the same as above, but the implementation is a bit trickier. While there are various ways to solve this, the simplest is probably to use a `for` loop. Test `make_change()` by making change for 78 cents/pence in US/UK currency. Compare your results to `make_change_US(78)` to double-check.

Solutions

Part 1 Solution

```
cents <- 73

quarters <- cents %/% 25
cents <- cents %% 25

dimes <- cents %/% 10
cents <- cents %% 10

nickels <- cents %/% 5
cents <- cents %% 5

change <- c('quarters' = quarters,
            'dimes' = dimes,
            'nickels' = nickels,
            'pennies' = cents)
change
```

```
quarters    dimes  nickels  pennies
       2        2        0        3
```

Part 2 Solution

```
make_change_US <- function(cents) {

  quarters <- (cents - cents %% 25) / 25
  cents <- cents %% 25

  dimes <- (cents - cents %% 10) / 10
  cents <- cents %% 10
```

```
  nickels <- (cents - cents %% 5) / 5
  cents <- cents %% 5

  pennies <- cents

  change <- c('quarters' = quarters, 'dimes' = dimes, 'nickels' = nickels,
              'pennies' = pennies)
  return(change)
}

make_change_US(37)
```

```
quarters    dimes  nickels  pennies
       1        1        0        2
```

```
make_change_US(19)
```

```
quarters    dimes  nickels  pennies
       0        1        1        4
```

Part 3 Solution

```
make_change <- function(cents, currency) {

  change <- rep(NA, length(currency))
  names(change) <- names(currency)

  for(i in 1:length(currency)) {
    change[i] <- (cents - cents %% currency[i]) / currency[i]
    cents <- cents %% currency[i]
  }
  return(change)
}

US <- c('quarter' = 25, 'dime' = 10, 'nickel' = 5, 'penny' = 1)
UK <- c('50p' = 50, '20p' = 20, '10p' = 10, '5p' = 5, '2p' = 2, '1p' = 1)
make_change_US(78)
```

```
quarters    dimes  nickels  pennies
       3        0        0        3
```

```
make_change(78, US)
```

```
quarter    dime  nickel   penny
      3       0       0       3
```

```
make_change(78, UK)
```

```
50p 20p 10p  5p  2p  1p
  1   1   0   1   1   1
```