UNIVERSITY OF MANITOBA
COURSE: STAT 2150
DATE & TIME: Feb. 6, 2025, 10:00–11:15 a.m.
Term Test 1
DURATION: 75 minutes

I understand that academic dishonesty is a serious offence:

Signature: ______________________________

INSTRUCTIONS

I. This test contains 6 pages of questions for a total of 20 marks. The number of marks for each question or question part is shown in the margins on the following pages.

II. Answer all questions on the exam paper in the space provided beneath the question. If you need more space, put a note in the space beneath the question and continue on page 8.

III. **Do not use any functions or methods that we have not discussed in class.** Your code must be R code, not any other computer programming language.

IV. Do not obscure the QR code.

V. Please do not turn the page until it is time to begin.

UNIVERSITY OF MANITOBA
COURSE: STAT 2150
Term Test 1
DATE & TIME: Feb. 6, 2025, 10:00–11:15 a.m.
DURATION: 75 minutes

1.

[½] (a) Suppose the following two lines of code are run. What would be the output?

```
x = c(-3,1,5,9,13)
x[mean(x)]
```

Solution:

Since `mean(x)` equals 5, we need to find `x[5]`, which is 13.

[1] (b) You have a vector `x` with 99 components. Write R code to extract the following components into a single output vector: (1) elements at positions that are multiples of 4, (2) the 99th component.

Solution:

```
x[c(seq(4,96,4),99)]
```

(Get a vector of indices, and then extract those components from `x`.)

Alternatively:

```
c(x[seq(4,96,4)],x[99])
```

(Extract the relevant components from `x` and then combine the two extracted vectors into one.)

[2] 2. Write an R function that takes three arguments:

- `bill`, the total amount of a restaurant bill, before tip
- `percent`, the percentage a customer will tip (for example, 15 for 15% tip), which is added to the bill
- `split`, the number of people that will split the bill, with a default value of 1

and calculates and returns the amount each person who is splitting the bill, will have to pay, after tip.

```
myfunc = function(bill,percent,split=1){
  total = bill*(1+percent/100)
  answer = total/split
  return(answer)
}
```

[3] 3. Write an R function called `myseq` with arguments `n`, `times1`, and `times2` that returns a sequence of numbers based on the following rules:

1. Create a sequence of numbers from 1 to `n`.
2. Repeat each number in that sequence `times1` times.
3. Repeat the entire sequence `times2` times.

The function should use the `rep()` function and should return the final sequence as a vector.

For example, if we call the function as `myseq(4,3,2)`, it should return:

```
[1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3 3 4 4 4
```

Hints:

- you can initalize an empty vector `vec` with the code `vec = numeric(0)`
- you may find it helpful to use the `append()` function, where `append(x,values)` appends `values` to a vector `x`

Method 1:

```
myseq = function(n,times1,times2){
  vec = numeric(0)
  for(i in 1:n){
    vec = append(vec,rep(i,times1))
  }
  ans = rep(vec,times2)
  ans
}
```

Method 2:

```
myseq = function(n, times1, times2){
  vec = numeric(0)
  for(i in 1:n){
    vec = append(vec,rep(i,times1))
  }
  ans = numeric(0)
  for(i in 1:times2) {
    ans = append(ans,vec)
  }
  ans
}
```

Method 3:

```
myseq = function(n,times1,times2){
  vec = numeric(0)
  for(i in 1:times2){
     for(j in 1:n){
        vec = append(vec,rep(j,times1))
     }
  }
  vec
}
```

Method 4:

```
myseq = function(n, times1, times2){
  vec =  numeric(0)
  for(i in 1:times2){
    for(j in 1:n){
      for(k in 1:times1){
        vec = append(vec,j)
      }
    }
  }
  vec
}
```

4. Suppose a zoo has a large vector called `animals` in R consisting of all of their animals. It is a character vector, where every element is “lion”, “giraffe”, “elephant”, “monkey”, or “zebra”.

   Create a new vector called `food` that consists of the animal’s food, so that each component of `food` contains the food for the corresponding component of the `animals` vector. The values in the `food` vector should be “meat” for lions, “leaves” for giraffes and elephants, and “standard” for any other animal.

   Regarding giraffes and elephants, you must make your code efficient so that you check if an animal is either one of those in one line of code, rather than handling the cases of giraffes and elephants separately. Similarly, you must handle the cases of monkeys and zebras in one line of code, rather than handling the cases of monkeys and zebras separately.

[2] (a) Create the `food` vector with a `for()` loop.

Hint: in order to initialize a numeric vector of length 10, for example, we use `numeric(10)`. In order to initialize a character vector of length 10, for example, we use `character(10)`.

```
food = character(length(animals))
for(i in 1:length(food)){ # or length(animals)
  if(animals[i] == "lion"){
    food[i] = "meat"
  } else if(animals[i] == "giraffe" | animals[i] == "elephant"){
    food[i] = "leaves"
  } else{
    food[i] = "standard"
  }
}
```

[1½] (b) Create the `food` vector using `sapply()`.

```
foodfn = function(x){
  if(x == "lion"){
    out = "meat" # or return("meat")
  } else if(x == "giraffe" | x == "elephant"){
    out = "leaves" # or return("leaves")
  } else{
    out = "standard" # or return("standard")
  }
  out
}
food = sapply(animals,foodfn)
```

5. Suppose we have a data frame called `pizza` with a large number of pizza orders and two columns: `size` (“small”, “medium” or “large”) and `spicy` (spiciness level of 1 – 10).

[1] (a) Write the R code that creates a subset of `pizza` (without using the `subset()` function) showing only the pizza orders that are “large” size and have a spiciness level of 8 or higher.

Solution:

```
pizza[which(pizza$sizes == "large") & which(pizza$spicy >= 8),]
```

Alternatively, `which()` can be removed.

[2] (b) Suppose we categorize the spiciness level into three groups:

- low (spiciness ≤ 3)
- moderate (spiciness between 4 and 6, inclusive)
- high (spiciness ≥ 7)

Write the R code that makes a contingency table showing how many pizza orders fall into each combination of pizza size and spiciness level.

```
spicy_category = character(nrow(pizza)) # or length(pizza$spicy)
for(i in 1:nrow(pizza)){  # or length(pizza$spicy)
  if(pizza$spicy[i] <= 3){
    spicy_category[i] = "low"
  } else if(pizza$spicy[i] <= 6 & pizza$spicy[i] >= 4){
    spicy_category[i] = "moderate"
  } else{
    spicy_category[i] = "high"
  }
}
table(spicy_category,pizza$sizes)
```

Alternatively, `spicy_category` can be created with `sapply()`.

[1] (c) Write the R code that sorts the `pizza` data frame so that the spiciness levels are in order from least to greatest.

```
pizza[order(pizza$spicy),]
```

[1] (d) Write the R code that finds the spiciness level of the third to last order in the `pizza` data frame.

Method 1:

```
pizza$spicy[length(pizza$spicy) - 2]
```

Method 2:

```
pizza$spicy[nrow(pizza) - 2]
```

Method 3:

```
spicy = pizza$spicy
spicy_new = spicy[-length(spicy)]
spicy_new2 = spicy_new[-length(spicy_new)]
spicy_new2[length(spicy_new2)]
```

[2] (e) Suppose we would like to create a new column in the `pizza` data frame called `orderID`, order numbers. Small size pizzas get order numbers starting at 2000 (2000, 2001, . . .), medium size pizzas get order numbers starting at 3000 (3000, 3001, . . .), and large size pizzas get order numbers starting at 4000 (4000, 4001, . . .). Create this `orderID` vector and add it to the `pizza` data frame.

Hint: it may be useful to create "counting" variables that keep track of how many pizzas of each size have been encountered so far in the data frame:

```
small_counter = 0
medium_counter = 0
large_counter = 0
```

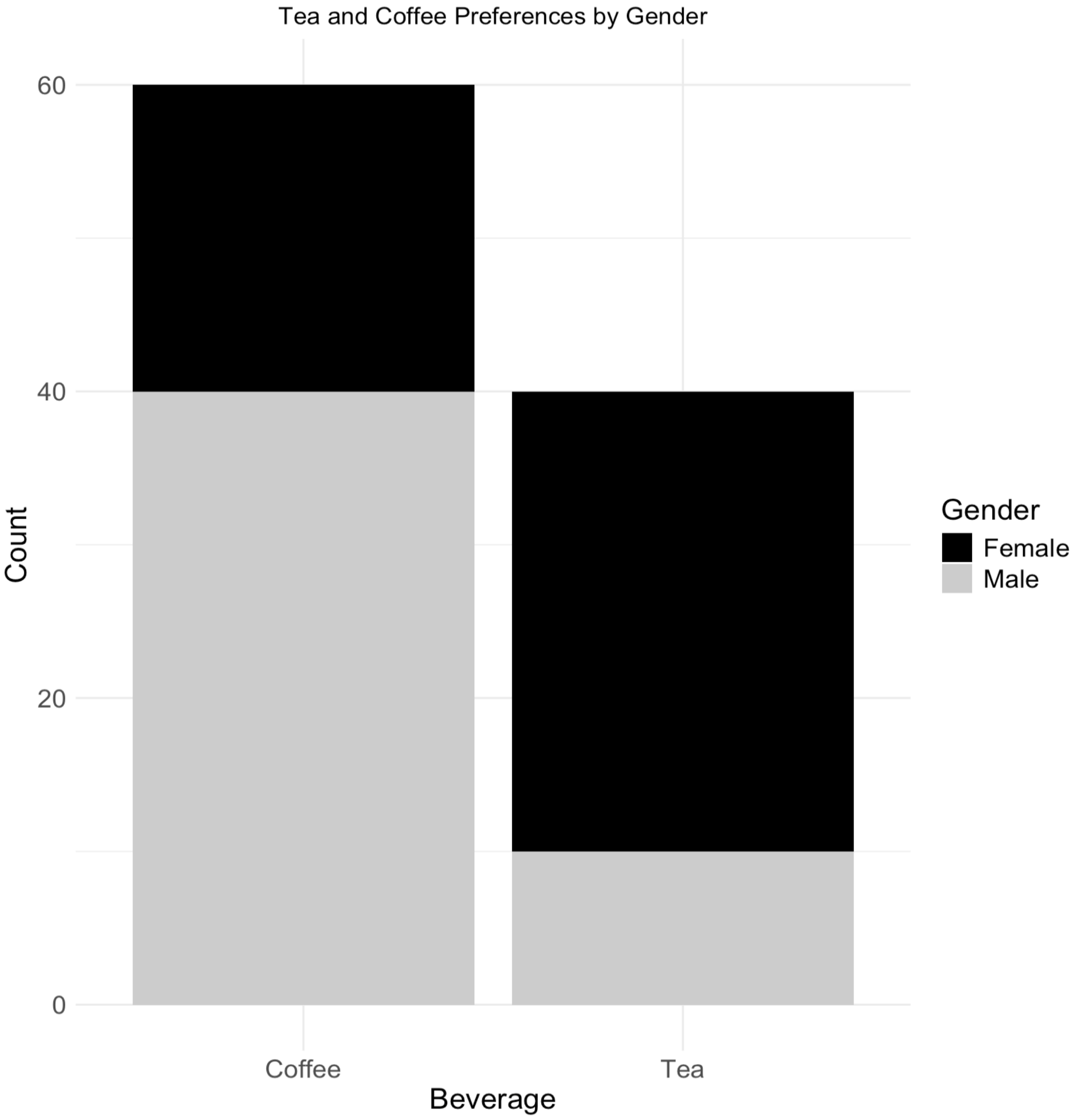
Solution:

```
orderID = numeric(length(pizza$sizes)) # or nrow(pizza)
small_counter = 0
medium_counter = 0
large_counter = 0
for(i in 1:length(pizza$sizes)){
  if(pizza$sizes[i] == "small"){
    orderID[i] = 2000 + small_counter
    small_counter = small_counter + 1
  } else if(pizza$sizes[i] == "medium"){
    orderID[i] = 3000 + medium_counter
    medium_counter = medium_counter + 1
  } else{
    orderID[i] = 4000 + large_counter
    large_counter = large_counter + 1
  }
}
pizza = data.frame(pizza,orderID)
```

[2] 6. Write a `while()` loop that prints all of the whole numbers from 10 to 0. Each time the code inside the loop gets executed, the next number (counting down) gets printed. Also, when 0 is printed, an additional message is printed: "Blast off!".

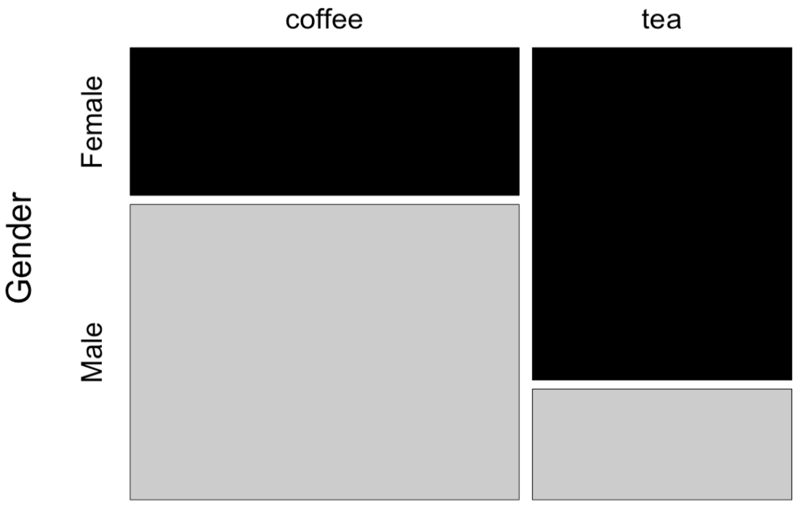
```
x = 10
while(x >= 0){
  print(x)
  if(x == 0){
    print("Blast off!")
  }
  x = x - 1
}
```

[1] 7. Suppose we ask 100 people if they prefer tea or coffee and we construct the below stacked bar chart.



Carefully draw a sketch of what a mosaic plot of the same data would look like.

Solution:



The mosaic plot shows that within the “coffee” column, 67% is male (40 out of 60 in the stacked bar chart) and 33% is female (20 out of 60 in the stacked bar chart). Similarly, within the “tea” column, 75% is female and 25% is male.

[blank]