## Question 1. [8 marks]

Beside each set of commands below, write what R will output after those commands are typed into the R console. For each set of commands, only write the output of the **last command**. Note that the ">" shown at the beginnings of each line is the R command prompt, not something typed.
Write your answer after the `[1]`.

| Code | Output |
|---|---|
| `> 20 / 4 + 2 * 2` | `[1] 9` |
| `> z <- c(6.3, 4, 6.7)`<br>`> typeof(z)` | `[1] "double"` |
| `> k <- c("d", "e", "f")`<br>`> k[2] <- "hi"`<br>`> k` | `[1] "d" "hi" "f"` |
| `> v <- c(1, 3, 5, 7, 9)`<br>`> v <- c(v[c(2:4)], 22)`<br>`> v` | `[1] 3 5 7 22` |
| `> a <- 4 * 3`<br>`> t <- 3 > a`<br>`> t` | `[1] FALSE` |
| `> as.character(3 + 4)` | `[1] "7"` |
| `> g <- c(4, 7)`<br>`> g %% 2 == 1` | `[1] FALSE TRUE` |
| `> d <- c(2, 3, 4)`<br>`> d[-1]`<br>`> d` | `[1] 2 3 4` |

## Question 2. [4 marks]

Consider the following function, `MysteryFunc`.

```
MysteryFunc <- function(a, b) {

  p <- 0
  v <- numeric(0)
  while (p < 3) {
    v <- c(v, a, b)
    p <- p + 1
  }

  return(v)

}
```

**Part A:** What is the R console output of the following function calls? Put the result in the box after each function call, directly after the `[1]`.
(*Hint:* Step through the code with the arguments and figure out what is happening to the variables as the function runs. Use the blank pages at the back for rough work if you want more room.)

```
> MysteryFunc(2, 3)
```

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

```
> MysteryFunc(1, 7)
```

```
[1] 1 7 1 7 1 7
```

**Part B:** Based on the above function calls and output, write a proper **docstring** in the box after the function header below that explains what the function `MysteryFunc` does. You do **not** need to write the `#` symbols at the beginning of each line.

```
MysteryFunc <- function(a, b) {
```

```
Returns a vector with the value 'a' followed by the value 'b', repeated three times in a row.
Note: You must use the word 'Returns', not 'Return', and have the arguments in quotes.
```

```
  .
  # rest of function
  .
 }
```

## Question 3. [5 marks]

The follwing are some test cases for the function `SeeingDouble`

```
> SeeingDouble(c("hi", "hello"))
[1] "hihi"  "hellohello"
> SeeingDouble(c("a", "b", "c"))
[1] "aa" "bb" "cc"
```

Complete the function body of the function `SeeingDouble` according to its docstring below. Remember that you can put two strings together using the `paste` function, which is also described on the back of the quiz if you need a reminder. Do not use any functions we haven't talked about in class.

You **must use a loop** of some sort to do the work for this function.

```
SeeingDouble <- function(v) {
  # Returns a copy of string vector 'v', except the string at every index
  # has been doubled.
  #
  # Precodition: length(v) > 0
  ############################################################
  # For loop with index
  for (i in 1:length(v)) {
    v[i] <- paste(v[i], v[i], sep = "")
  }

  return(v)
  ############################################################
  # For loop with elements
  returnVec <- character(0)
  for (e in v) {
    returnVec <- c(returnVec, paste(e, e, sep = ""))
  }

  return(returnVec)
  ############################################################
  # While loop
  i <- 1
  while (i <= length(v)) {
    v[i] <- paste(v[i], v[i], sep = "")
    i <- i + 1
  }

  return(v)

}
```

## Question 4. [5 marks]

Below is a table that gives the amount of time it takes to prepare an order of cupcakes, depending on the amount of cupcakes ordered.

| number of cupcakes | preparation time |
|---|---|
| less than 5 cupcakes | 40 minutes |
| 5 to 7 cupcakes | 60 minutes |
| over 7 cupcakes | 90 minutes |

If the customer wants icing on their cupcakes, an **extra** 15 minutes is added.

The follwing are some test cases for the function `PreparationTime`.

```
> PreparationTime(3, FALSE)
[1] 40
> PreparationTime(6, TRUE)
[1] 75
> PreparationTime(10, FALSE)
[1] 90
```

On the **next page**, complete the function body of the function `PreparationTime` according to its docstring. Do not use any functions we haven't talked about in class.

*Continued on next page...*

```
PreparationTime <- function(numCakes, wantIcing) {
  # Returns the time it takes to prepare 'numCakes' number of cupcakes.
  # 'wantIcing' indicates whether or not the customer wants icing.
  #
  # Precondition: numCakes > 0

  totalTime <- 0

  if (numCakes < 5) {
    totalTime <- 40
  } else if (numCakes < 8) {
    totalTime <- 60
  } else {
    totalTime <- 90
  }

  if (wantIcing) {
    totalTime <- totalTime + 15
  }

  return(totalTime)

}
```