Unit 1 Practice Questions with Solutions

1. For this question, choose ALL of the answers that are correct. Suppose we wish to create a vector **x** of length 10 whose elements are the numbers 1 to 10, in order. Which lines of code would produce such a vector?

   √ **x = seq(1,10,1)**

   √ **x = double(10)**
   for(i in 1:10){x[i] = i}

   √ **x = c(1:10)**

   ○ x = vector(1:10)

   √ **x = 1:10**

2. Suppose I have a vector `x = c(1,2,3)` and a vector `y = c(4,5,6)`. What would be the final output after the following two lines of code are run?

```
temp = cbind(x,y)
temp[2,]
```

   Solution:

   `[1] 2 5` (that is, the vector `c(2,5)`)

3. Suppose a data frame called `animalDF` is created in R consisting of a large number of animals and their weights (in pounds). All animals in the data frame are chickens, snakes or turtles. A preview of the data frame is shown below:

animal	**weight**
chicken	2
snake	50
chicken	3
turtle	12
turtle	4
snake	4.5
chicken	6
⋮	⋮

   (a) Write the R code that creates a new data frame called `heavyAnimals` that consists of only animals with a weight of at least 5 pounds (that is, 5 pounds or greater).

```
heavyAnimals = animalDF[which(animalDF$weight >= 5),]
```

   or

```
heavyAnimals = animalDF[animalDF$weight >= 5,]
```

(b) Write the R code that obtains counts of the numbers of chickens, snakes, and turtles in the `heavyAnimals` data frame.

```
table(heavyAnimals$animal)
```

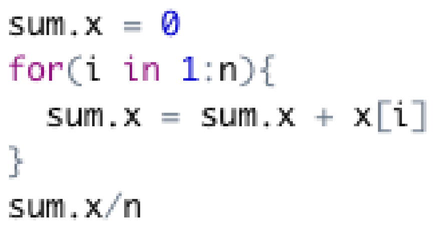
4. Consider calculating the following in R:

   $2e^{-2} + \ln(4)$

   Which line of code would correctly calculate this?

   - [✓ **2*exp(-2) + log(4)**]
   - ○ 2*exp(-2) + ln(4)
   - ○ 2*e∧(-2) + ln(4)
   - ○ 2exp(-2) + log(4)
   - ○ 2*e∧(-2) + log(4)

5. Assume a vector **x** has been defined and its length has been assigned to the object **n**. Which line of code would NOT compute the mean of **x**?

   - ○ mean(x)
   - ○ 1/n*sum(x)
   - [✓ **average(x)**]
   - ○ sum(x)/n
   - ○ 

```
sum.x = 0
for(i in 1:n){
  sum.x = sum.x + x[i]
}
sum.x/n
```

6. Consider the vector `x = c(51,78,60,36,42)`. What would be the output of `order(x)`?

```
[1] 4 5 1 3 2
```

7. Suppose I have a vector **x** with 50 numeric values. Write the R code that obtains all elements of **x** that are either greater than 37 or less than 7.

   `x[x > 37 | x < 7]` or `x[which(x > 37 | x < 7)]`

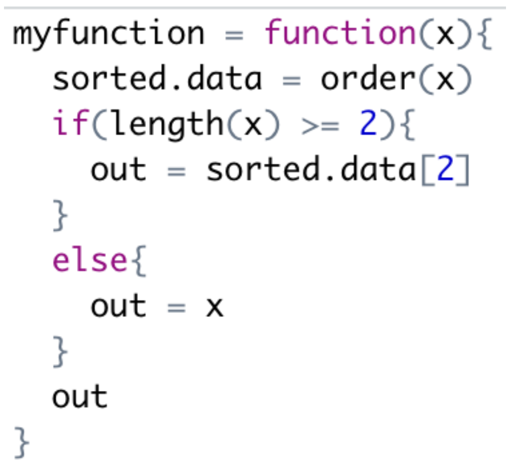
   or

   `c(x[which(x > 37)], x[which(x < 7)])`

8. At an arts and crafts fair, a vendor sells goods that cost \$5 per item. If a buyer purchases more than 25 items, however, the vendor gives a 15% discount on the total cost. Write an R function that takes in the number of items purchased by a buyer and returns the total cost.

```
costfunc = function(items){
   totalcost = items*5
   if(items > 25){
      totalcost = totalcost*0.85
   }
   return(totalcost)
}
```

9. Consider the following function in R:



```
myfunction = function(x){
  sorted.data = order(x)
  if(length(x) >= 2){
    out = sorted.data[2]
  }
  else{
    out = x
  }
  out
}
```

It is supposed to sort the data from smallest to biggest. Then, if there is more than one component to **x**, report the second smallest value. If there is only one component, it should report that value. What is wrong with the code?

- ○ A { is missing.
- ○ A (is missing.
- [✓ **The function order() should be sort()**]
- ○ There is no default for **x**
- ○ There should be an ifelse() statement

10. Two matrices A and B are compatible for matrix multiplication if the number of columns of matrix A is the same as the number of rows of matrix B. Write a function that takes two matrices and outputs their product if the matrices are compatible and outputs "Not possible" if the matrices are not compatible.

    Note: suppose you have two matrices: A and B. In order to multiply the matrices in R (assuming the dimensions of the matrices match that it makes sense to multiply them), we use the matrix multiplication operator `%*%`. That is, `A%*%B` multiplies A and B.

    Solution:

```
mat.mult = function(A,B){
A.col = dim(A)[2]
B.row = dim(B)[1]
if(A.col == B.row){
   out = A%*%B
} else{
   out = "Not possible"
}
out
}
```

11. We know that the median of (1,6,3), a vector with an odd number of values, is 3 and the median of (2,8,4,6), a vector with an even number of values, is 5. If there are an odd number of values, the median is in position $\frac{n+1}{2}$ of the sorted data vector. If there are an even number of data values, we go to position $\frac{n+1}{2}$ of the sorted data vector and then take the average of the corresponding data values.

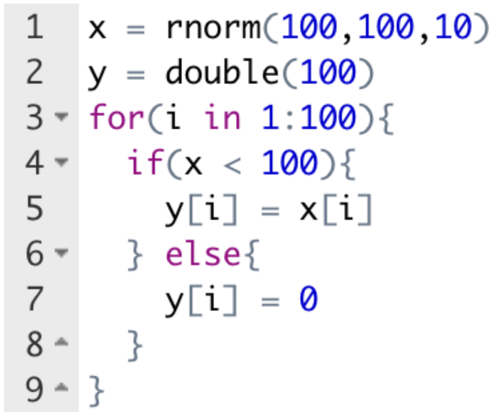
    Of course, R has a function called `median()` that can calculate the median of a dataset. Create your own R function called `myMedian()` that takes a vector of data values and returns the median of the dataset, *without* using the `median()` function.

    Hint: we can use modular arithmetic to determine if there are an odd or even number of data values in a vector `x`. If `x%%2` equals 1, then there are an odd number of data values in `x`. If `x%%2` equals 0, then there are an even number of data values in `x`.

    ```
    myMedian = function(x){
      length = length(x)
      medianpos = (length + 1)/2
      sortedx = sort(x)
      if(length %% 2 == 0){
        ans = mean(c(sortedx[medianpos-0.5],sortedx[medianpos+0.5]))
      } else{
        ans = sortedx[medianpos]
      }
      ans
    }
    ```

    Or: `medianpos-0.5` can be replaced by `length/2`; `medianpos+0.5` can be replaced by `length/2+1`.

12. Consider the following code in R:



```
1 x = rnorm(100,100,10)
2 y = double(100)
3 for(i in 1:100){
4   if(x < 100){
5     y[i] = x[i]
6   } else{
7     y[i] = 0
8   }
9 }
```

It is supposed to generate 100 values from a normal distribution with mean 100 and standard deviation 10 and store those values in x. Then a vector y is initialized. For each element of y, we fill it in with the value of x in the same position if the value is less than 100; otherwise, we fill it in with a 0. Fix the code. Reference the line numbers in your response. For example, “line 15 should say …” or “between lines 15 and 16 …”.

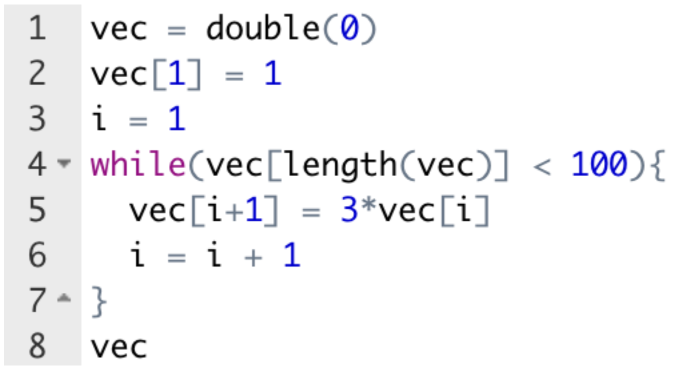
Solution:

In line 4, x $<$ 100, should be x[i] $<$ 100.

13. Write R code which completes the following tasks:

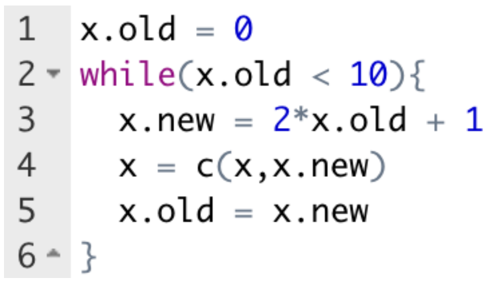
    1. Creates an empty vector; call the vector **vec**.
    2. Fills in the first element of **vec** with the value 1.
    3. Using a while loop, the subsequent elements of **vec** should be filled in with 3 times the previous value.
    4. The loop should end as soon as an element of **vec** is greater than or equal to 100.
    5. Print **vec**.

Solution:



```
vec = double(0)
vec[1] = 1
i = 1
while(vec[length(vec)] < 100){
  vec[i+1] = 3*vec[i]
  i = i + 1
}
vec
```

14. Consider the following code in R:



```
1  x.old = 0
2  while(x.old < 10){
3    x.new = 2*x.old + 1
4    x = c(x,x.new)
5    x.old = x.new
6  }
```

It is supposed to create a vector **x** such that (1) the first element is 0, (2) each subsequent element is twice the previous value plus 1, and (3) once the largest element is larger than 10, no more elements are added. Fix the code. Reference the line numbers in your response. For example, "line 15 should say ..." or "between lines 15 and 16 ...".

Solution:

In line 4, the variable **x** is being concatenated with **x.new**, but **x** has never been declared. Before line 1, add **x** = 0.

15. The Fibonacci sequence starts with the numbers 0 and 1. Each subsequent number is the sum of the two preceding numbers. Write R code that starts with a vector of length 2 with the values 0 and 1 and fills in subsequent elements. When the sequence reaches a number greater than 100, no more elements are added. Finally, the vector of the resulting sequence is printed.

Solution:

```
vec = c(0,1)
i = 3
while(vec[length(vec)] < 100){
  vec[i] = vec[i-1] + vec[i-2]
  i = i + 1
}
vec
```

16. Suppose we want to make a vector, call it evens, of all the even integers between 0 and 100. For this, consider the following code:

```
evens = double(0)
i = 1
evens[1] = 0
while(max(evens) <= 100){
  i = i+1
  evens[i] = 2*(i-1)
}
```

What is wrong with the code?

- ○ There is nothing wrong.
- ✓ **<= 100 should be < 100**
- ○ evens[i] should be evens[i+1]
- ○ i = i + 1 should move down one line in the while loop
- ○ while(max(evens) <= 100) should be for(i in 1:100)

When `i` is set to 50 (when it was previously 49), we then get evens[50] $= 2 * (i - 1) = 2 * (50 - 1) = 98$. So the next time through the `while()` loop, max(evens) = 98, which is `<= 100`, so the code inside the loop gets executed again. Now, `i` is set to 51 and evens[51] = 100. At this point, we should be done. However, max(evens) is still `<= 100`, so we have to execute the code inside the loop again, which means `i` will get set to 52 and evens[52] = 102. Changing `<=` to `<` prevents this extra even number from being populated.

17. An elementary school child is asked to find the smallest two consecutive whole numbers such that their product exceeds 800. Write a `while()` loop that tests every integer starting from 1 to see if its product with the next higher integer exceeds 800. After the `while()` loop stops, print the smaller of the two consecutive whole numbers.

```
i = 1
j = 2
prod = i*j
while(prod < 800){
   i = i + 1
   j = j + 1
   prod = i*j
}
print(i)
```

Alternatively:

```
i = 1
prod = i*(i+1)
while(prod < 800){
   i = i + 1
   prod = i*(i+1)
}
print(i)
```

Alternatively:

```
i = 1
prod = 0
while(prod < 800){
   prod = i*(i+1)
   i = i + 1
}
print(i-1)
```

18. Recall that we can obtain the z-score corresponding to a value of x through the formula $z = \dfrac{x-\mu}{\sigma}$, where μ is the mean of the normal distribution and σ is the standard deviation of the normal distribution. Suppose exam scores in a large course are known to follow a normal distribution with mean 70 and standard deviation 12. Three students in the class had the exam scores that appear in the below **scores** object.

```
scores = c(70,80,60)
```

Use the R function **sapply()** to obtain the z-scores of these three students.

Solution:

```
myfunction = function(x){
   (x - 70)/12
}
sapply(scores,myfunction)
```

19. Consider the following function:

$$f(x) = e^{2x} + 1, -\infty < x < \infty.$$

Now consider the following values of x: $-2.5, 1.4, -3.6, 0.8$ and 4.5.

(a) Write R code that uses a loop to evaluate this function at the above values of x and stores the results in a vector called **x.out**.

Solution:

```
x = c(-2.5, 1.4, -3.6, 0.8, 4.5)
n = length(x)
x.out = numeric(n)
for(i in 1:n){
 x.out[i] = exp(2*x[i])+1
}
```

(b) Repeat part (a) but without using a loop.

Solution:

```
x = c(-2.5, 1.4, -3.6, 0.8, 4.5)
x.out = sapply(x,function(y){exp(2*y)+1})
```

Alternatively,

```
x = c(-2.5, 1.4, -3.6, 0.8, 4.5)
x.out = exp(2*x) + 1
```

20. Suppose a vector `ages` exists, of length 100, with the ages of 100 children ages 12 to 17. We would like to create a vector called `driving`, of the same length, that will contain “yes” or “no” in each position, depending on if the child is old enough to drive in Manitoba. (You need to be at least 16 years old to drive in Manitoba.) Create the vector `driving` using:

    (a) a `for()` loop

    ```
    driving = character(100)
    for(i in 1:100){
       if(ages[i] >= 16){
          driving[i] = "yes"
       } else{
          driving[i] = "no"
       }
    }
    ```

    (b) the `sapply()` function

    ```
    myfunc = function(x){
       if(x >= 16){
          yesno = "yes"
       } else{
          yesno = "no"
       }
       return(yesno)
    }
    driving = sapply(ages,myfunc)
    ```