# Quiz: Solutions

Narayani Vedam

2023-10-20

## I. True or False

1. Lists in R can store variables of different data types, including functions and other lists - **TRUE**
2. `select()` can be used to remove variables from a Tidy dataset - **TRUE**

```
select(-column_or_variable_name)
```

3. If you have a pipe operation with three functions: `A() %>% B() %>% C()`, reversing the order to `C() %>% B() %>% A()` will always produce the same result - **FALSE**
4. R functions can have default values for their arguments, making it optional for users to provide values when calling the function - **TRUE**
5. A data frame in R is a two-dimensional data structure where all columns must have the same data type - **FALSE**
6. In R, you can pass functions as arguments to other functions - **TRUE**
7. `%>%` can be used to add layers to a ggplot - **FALSE**; we use `+`
8. We can add an observation to a Tidy dataset using `mutate()` - **FALSE**

```
mutate(new_column_name=operation())
```

9. `select()` and `filter()` are equivalent and can be used interchangeably - **FALSE**; `select()` is for columns/variables and `filter()` is for rows/observations
10. The output of the command, `1+"1"` is `"2"` - **FALSE**

```
1+"1"
```

```
## Error in 1 + "1": non-numeric argument to binary operator
```

11. In `x<-c(1,"4",7,10.00,6L)`, the type of the fifth entry of `x` is different from the type of `x` - **FALSE**

```
x<-c(1,"4",7,10.00,6L)
print(typeof(x))
```

```
## [1] "character"
```

```
print(typeof(x[5]))
```

```
## [1] "character"
```

12. You can multiply a logical vector with a vector of type double, the output will be of type logical - **FALSE**

```
x<-c(TRUE,FALSE,TRUE)
y<-c(5,6,0)
```

```r
z<-x*y
print(z)
```

```
## [1] 5 0 0
```

```r
print(typeof(z))
```

```
## [1] "double"
```

13. In R, the `&` operator is used for element-wise logical AND, and it returns a vector of the same length as the input vectors - **TRUE**

```r
x<-c(TRUE,FALSE,TRUE)
y<-c(FALSE,FALSE,TRUE)
z<-x&y
print(z)
```

```
## [1] FALSE FALSE  TRUE
```

```r
print(typeof(z))
```

```
## [1] "logical"
```

14. `x|y` evaluates to `TRUE` if either x or y is `TRUE` - **TRUE**

```r
x<-c(TRUE,FALSE,TRUE)
y<-c(FALSE,FALSE,TRUE)
z<-x|y
print(z)
```

```
## [1]  TRUE FALSE  TRUE
```

```r
print(typeof(z))
```

```
## [1] "logical"
```

15. Variables defined in the global environment can be modified or reassigned within a function using the `<<-` operator - **TRUE**

```r
x<-1
example <- function() x<<-5
example()
print(x)
```

```
## [1] 5
```

16. In R, variables declared within a function have global scope and can be accessed from outside the function - **FALSE**

```r
rm(list=ls())
example <- function() x<-1
print(x)
```

```
## Error in print(x): object 'x' not found
```

17. You can use the `as.logical()` function to explicitly coerce a numeric value to a logical value - **TRUE**

```r
x<-c(1,23,56,0)
z<- as.logical(x)
print(z)
```

```
## [1]  TRUE  TRUE  TRUE FALSE
```

```
print(typeof(z))
```

```
## [1] "logical"
```

18. In `x<-c('a',1,3)`, using `as.numeric(x)` will replace the non-numeric entry of `x` with its numeric equivalent - **FALSE**

```
x<-c('a',1,3)
print(typeof(x))
```

```
## [1] "character"
```

```
z<- as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

```
print(z)
```

```
## [1] NA  1  3
```

```
print(typeof(z))
```

```
## [1] "double"
```

19. In R, the order in which you pass arguments to a function does not matter as long as you provide all the required arguments - **FALSE**

```
x<-10
y<-5
example<-function(x,y) x/y
example(y,x)
```

```
## [1] 0.5
```

20. In R, you can use single quotation marks (') to define character strings instead of double quotation marks (") - **TRUE**

```
z<-'a'
print(typeof(z))
```

```
## [1] "character"
```

```
z<-"a"
print(typeof(z))
```

```
## [1] "character"
```

# II. Short Answers-1

1. Explain what is meant by local scope of a variable and illustrate it with an example code

A variable with local scope cannot be accessed outside the block - functions, iterations and conditional blocks - that they are defined in

Example:

```
example <- function() x<-1
example()
print(x)
```

```
## [1] 10
```

2. Give one reason why it is generally not a good idea to use global variables in your functions. Illustrate it with an example code

It is not a good idea to use global variables because they can be inadvertently changed by functions used in the same program

Example:

```
x<-0
example <- function() x<<-x+5
# First function call
example()
print(x)
```

```
## [1] 5
```

```
# Second function call
example()
print(x)
```

```
## [1] 10
```

# III. One-line Code

1. Imagine you have a dataset, "student_info", with information about students, including "StudentID"(type:`dbl`), "FirstName"(type:`chr`), "LastName"(type:`chr`), "Gender"(type:`chr`) and "Score"(type:`dbl`). Write a line of code for each of the following data wrangling tasks:

a. Filter the dataset to include only Female students

```
student_info %>% filter(Gender=="Female")
```

b. Calculate the average score of female students

```
student_info %>% filter(Gender=="Female") %>% summarise(average_score=mean(Score))
```

2. You are working with a dataset, "customer_orders", containing information about customer orders, including "OrderID"(type:`dbl`), "CustomerID"(type:`dbl`), "ProductID"(type:`dbl`), "Quantity"(type:`dbl`) and "Price"(type:`dbl`). Write a line of code to perform each of the following data wrangling tasks:

a. Filter the dataset to include only orders with a price greater than 100

```
customer_orders %>% filter(Price>100)
```

b. Calculate the total price of all these high-value orders

```
customer_orders %>% filter(Price>100) %>% summarise(Total_price=sum(Price))
```

3. Imagine you have a dataset, "product_info", containing information about products, including "ProductID" (type:`dbl`), "ProductName" (type:`chr`), "Category"(type:`chr`) and "Price"(type:`dbl`). Write a line of code for each of the following data wrangling tasks

a. Filter the dataset to include only products in the Electronics category.

```
product_info %>% filter(Category=="Electronics")
```

b. Calculate the average price of products in this category

```
product_info %>% filter(Category=="Electronics") %>% summarise(Average_value=mean(Price))
```

4. Imagine you have a dataset containing information about customers and their purchases. The dataset is called "customer_purchases", and includes columns for "CustomerID" (type:`dbl`), "CustomerName"

(type:`chr`), "PurchaseDate" (type:`Date`), and "PurchaseAmount" (type:`dbl`). Write a line of code for each of the following tasks

a. Filter the dataset to include only purchases whose amount is greater than or equal to 100

```
product_info %>% filter(PurchaseAmount>=100)
```

b. List the filtered data in the decreasing order of their amount

```
product_info %>% filter(PurchaseAmount>=100) %>% arrange(desc(PurchaseAmount))
```

# IV. Short Answers-2

1. What will be the output of the following code? Why is the value of `y` different from `x`?

```
# Initialise x
x<-10
# Assign the value of x to y
y<-x
# Change the value of x
x<-5
cat("x is", x, "y is", y)
```

```
## x is 5 y is 10
```

The initial value of `x` is 10 and is assigned to `y`. `x` is reassigned a value 5, but `y` is not reassigned to `x`. Hence, its value remains 10.

2. Identify the name of the function, arguments and body of the function below

```
# Definition of function

print_updated_value <- function(x,y='100'){

# Printing sum

print(x+y)

}
```

- Name: `print_updated_value()`
- Arguments: `x, y='100'`
- Body: `print(x+y)`

3. How many functions are there in all in the following code snippet? List the functions

```
# Function definition
Calc_sample_mean <- function(size,input_mean,input_sd=10){
# Generate the sample
sample<- rnorm(size,mean_value=input_mean, sd=input_sd)
# Mean of the sample
mean(sample)
}
```

There are three functions;

- `Calc_sample_mean()`
- `rnorm()`

- `mean()`

4. Debug the code snippet below to print the sum of `x` and `y` using appropriate type conversions wherever deemed necessary. Provide the error-free version of the code below

```
# Definition of function
print_updated_value <- function(x,y='100'){
# Printing sum
print(x+y)
}
# Function call
print_updated_value(2)
```

```
## Error in x + y: non-numeric argument to binary operator
```

The question requires us to use **type_conversions**; The corrected code is below.

```
# Definition of function
print_updated_value <- function(x,y='100'){
# Printing sum
print(x+as.numeric(y))
}
# Function call
print_updated_value(2)
```

```
## [1] 102
```

# V. Debug/Evaluate

1. Evaluate the following code in RStudio and debug it. Explain what was wrong with the code, what the error-free code does and why the output of the function is different from carName?

```
carName <- 3
print_car_name <- function(){
names<-c("Volvo", "Mercedes", "Audi", "BMW")
carName<-names[carName]
return(carName)
}
cat("the name of the car you chose is", print_car_Name(),", but the value of the variable carName is",
```

```
## Error in print_car_Name(): could not find function "print_car_Name"
```

The error is in the capitalization of the name of the function, "print_car_Name()", it should be "print_car_name()". The output of the corrected code is below

```
carName <- 3
print_car_name <- function(){
names<-c("Volvo", "Mercedes", "Audi", "BMW")
carName<-names[carName]
return(carName)
}
cat("the name of the car you chose is", print_car_name(),", but the value of the variable carName is",
```

```
## the name of the car you chose is Audi , but the value of the variable carName is 3
```

This example demonstrates the idea of global and local variables. Initially `carName` is assigned a value of 3, globally. Inside the function, the global variable is used as the index to assign a value from the vector or car names, "names", to the local variable with the same name, "carName".

Therefore, the function returns the value of the local variable, "Audi", while the value of "carName" outside the function is the same as the global value, 3.

2. Evaluate the code in RStudio and debug it to obtain the output. Explain what was wrong with the code, what the error-free code does and the reason for the output

```
multiply <- function(value){
function_inside_function<-function(multiplier){
return(multiplier*value)
}
return(function_inside_function(10))
}
Multiply(50)
```

```
## Error in Multiply(50): could not find function "Multiply"
```

The error is in the capitalized "m" in the name of the function, "Multiply". The error-free code is below.

```
multiply <- function(value){
function_inside_function<-function(multiplier){
return(multiplier*value)
}
return(function_inside_function(10))
}
multiply(50)
```

```
## [1] 500
```

The function `multiply()` has another function, `function_inside_function()` defined locally. This function, takes `value` which is the argument passed to `multiply()` and multiplies it with `multiplier`, an argument of `function_inside_function()`. When we call this local function with the argument `multiplier=10`, it multiplies 10 with the assigned `value=50` in the function call `multiply(50)`, yielding the result 500.

3. Evaluate the code in RStudio. Explain what the code below does and the reason for the output.

```
multiply <- function(value){
function_inside_function<-function(multiplier){
return(multiplier*value)
}
return(function_inside_function(10))
}
function_inside_function(50)
```

```
## Error in function_inside_function(50): could not find function "function_inside_function"
```

The function `multiply()` has another function, `function_inside_function()` defined locally. This function, takes `value` which is the argument passed to `multiply()` and multiplies it with `multiplier`, an argument of `function_inside_function()`.

The code above throws an error because the function being called, `function_inside_function(50)` is of local scope, inside the main function `multiply()`

# VI. Working with Datasets

1. Download the titanic.csv Download `titanic.csv` file into a folder on your computer, say "Quiz" inside your NM2207 folder, and

a. Read the contents of the dataset (only code required)

```
library(tidyverse)
titanic <- read_csv("titanic.csv")
```

b. List all the passengers who survived in class 3 (only code required)

```
titanic %>% filter(Survived==1,Pclass==3)
```

c. Among those who survived in class 3, how many are female? (only code required)

```
titanic %>% filter(Survived==1,Pclass==3,Sex=="female") %>% summarise(count=n())
```

d. What is the mean value of the age of survivors in class 3? (only code required)

```
titanic %>% filter(Survived==1,Pclass==3) %>% summarise(Average_value=mean(Age))
```

e. List the survivors in class 3 in the decreasing order of their age (only code required)

```
titanic %>% filter(Survived==1,Pclass==3) %>% arrange(desc(Age))
```

f. Obtain a plot of age of the survivors in class 3 versus their passenger id (only code required)

```
titanic %>% filter(Survived==1,Pclass==3) %>% ggplot(aes(x=PassengerId,y=Age)) + geom_point() + theme_bu
```

2. Download the gapminder.csv Download gapminder.csvfile into a folder on your computer, say "Quiz" inside your NM2207 folder, and

a. Read the contents of the dataset (only code required)

```
library(tidyverse)
gapminder <- read_csv("gapminder.csv")
```

b. Select the columns, country, continent and year (only code required)

```
gapminder %>% select(country,continent,year)
```

c. Find the number of times each country is repeated (only code required)

```
gapminder %>% group_by(country) %>% summarise(Number=n())
```

d. List only the rows corresponding to Singapore in the decreasing order of their gdpPercap (only code required)

```
gapminder %>% filter(country=="Singapore") %>% arrange(desc(gdpPercap))
```

e. Find the mean value of life expectancy of Singapore (only code required)

```
gapminder %>% filter(country=="Singapore") %>% summarise(Average_value=mean(lifeExp))
```

f. Obtain a plot of Singapore's population from 1952 to 2007 (only code required)

```
gapminder %>% filter(country=="Singapore") %>% ggplot(aes(x=year,y=pop)) + geom_point() + theme_bw()
```

## VII. Working with Functions

1. Write a function, `country_name_length`, to identify names of countries longer than 8 characters in the global variable, `country_names`

Only final code is required.

a. Initialise a global variable, `country_names`, with the names of five countries of your choice

```
country_names<-c("India","Australia","Philippines","Indonesia","Malaysia")
```

b. Pass it as an argument to the function, `country_name_length`

```
country_name_length <- function(country_names){}
```

c. In the local scope, initialise the variable `name_length` with the lengths of the countries

```
country_name_length <- function(country_names){
  name_length <- nchar(country_names)
}
```

d. Retrieve the names of the countries longer than 8 characters and store them in local scope in `long_names`

```
country_name_length <- function(country_names){
  name_length <- nchar(country_names)
  long_names <- name_length>8
}
```

e. Print the longest name and the length of the name

```
country_name_length <- function(country_names){
  name_length <- nchar(country_names)
  long_names <- name_length>8
  print(country_names[name_length==max(name_length)])
}
```

f. Write a function call to execute the function

```
country_name_length(country_names)
```

```
## [1] "Philippines"
```

2. Write a function called `guessNum` for a guessing game where the players must guess the value of a number stored in a variable answer,

Only final code is required.

a. In the first step, take the user's guess as the argument

```
guessNum <- function(guess){}
```

b. In the local scope of guessNum, initialize the value of the variable answer to be 15

```
guessNum <- function(guess){
  answer <- 15
}
```

c. Compare the user's guess with the variable answer

```
guessNum <- function(guess){
  answer <- 15
  if(guess>answer){

        }
  else if(guess==answer){

      }
  else

}
```

d. Print out an appropriate message to the console, telling them if guess was above, below, or right on the number stored in answer

```
guessNum <- function(guess){
  answer <- 15
  if(guess>answer){
    print("Guess is high")
    }
  else if(guess==answer){
    print("Guess is on point")
  }
  else print("Guess is low")
}
```

e. In the global scope, initialize the value of a variable guess to be 16. Write the function call for the guessNum function, using guess as the argument.

```
guessNum <- function(guess){
  answer <- 15
  if(guess>answer){
    print("Guess is high")
    }
  else if(guess==answer){
    print("Guess is on point")
  }
  else print("Guess is low")
}

guess <- 16

guessNum(guess)
```

```
## [1] "Guess is high"
```