Spring 2022 courses

DONE cc Test 2

SETTINGS

Introduction to Advanced Data Science (DSC 205) test 2:

- 10 questions from quiz 4-6 (some modified)
- 10 new questions inspired by DataCamp practice questions
- Partial credit enabled
- You can change answers before submitting
- You can resume an incomplete submission
- It is mentioned if > 1 answer is correct

Good luck!

What is the output of this code?

```
TRUE == FALSE
```

FALSE

SOLUTION:

```
TRUE == FALSE
```

```
[1] FALSE
```

Complete the code to return the output

Calculate the sum for each element of the `myList` list, ensuring that missing values are removed.

Replace `[1]`, `[2]` and `[3]` in the code block to return the result shown below.

```
myList <- list(x = c(2,4), y=c(8,NA))
sapply(X = myList, FUN = [1], [2] = [3])
```

Results:

```
x y
6 8
```

TRUE:

- [1] sum [2] na.rm [3] TRUE

FALSE:

- [1] summary [2] rm.na [3] TRUE
- [1] mean [2] na.rm [3] FALSE
- [1] sum [2] NaN [3] TRUE

SOLUTION:

```
myList <- list(x = c(2,4), y=c(8,NA))
sapply(myList, sum, na.rm = TRUE)
```

```
x y
6 8
```

Complete the code to return the output

Determine which of these fruits have an `"a"` in their name.

Replace the `???` in the code block to return the result shown below.

```
fruits <- c("apple", "peach", "orange", "lemon")
???(pattern = "a", x = fruits)
```

Results:

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

TRUE:

- grepl

FALSE:

- grep
- sub
- gsub

SOLUTION:

```
fruits <- c("apple", "peach", "orange", "lemon")
grepl(pattern = "a", x = fruits)
```

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

True or false?

> "Simplification in `sapply` is only attempted if `X` has length greater than zero and if the return values from all elements of `X` are all of the same (positive) length."

TRUE

Complete the code to return the output

You want to compute the median of the values of a vector defined in the code block below.

Replace `[1]`, `[2]` and `[3]` in the code block to return the result shown below.

```
values [1] c(1,2,3,3,4,NA)
median (x [2] values, na.rm [3] TRUE)
```

RESULTS:

```
[1] 3
```

TRUE:

- [1] <- [2] = [3] =

FALSE:

- [1] %>% [2] = [3] =
- [1] <- [2] == [3] =
- [1] -> [2] |> [3] =

SOLUTION:

```
values <- c(1,2,3,3,4,NA)
median (x = values, na.rm = TRUE)
```

```
[1] 3
```

Complete the code to return the output

Replace [1] and [2] in the code block by the correct expression to get the result shown below.

```
x <- -6

if([1]) {
  print("is is a positive number")
} [2] {
  print("x is a negative number")
}
```

TRUE:

- [1] x>0 [2] else

FALSE:

- [1] x<0 [2] else

- [1] x |> 0 [2] else
- [1] x>0 [2] elseif

RESULTS:

```
[1] "x is a negative number"
```

SOLUTION

```
x <- -6

if(x>0) {
  print("is is a positive number")
} else {
  print("x is a negative number")
}
```

```
[1] "x is a negative number"
```

Complete the code to return the output

Convert the vector in the code block to a list by replacing `???` with the correct function to get the output shown below.

Tip: storing a vector as a list element is not the same as converting a vector to a list.

```
???(c(1,3,4))
```

RESULTS:

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

[[2]]
[1] 3

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

TRUE:

- as.list

FALSE:

- unlist
- is.list
- list

SOLUTION:

```
as.list(c(1,3,4))
```

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

[[2]]
[1] 3

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

Complete the code to return the output

We're looking for the right function from the `apply` family to generate two samples from a normal distribution.

In the code block below, the function `rnorm` is applied to the vector `1:2`, and `n=5` values are generated.

Replace `???` by the `apply` function that generates the output shown below.

```
???(X=1:2, FUN=rnorm, n=5)
```

```
[[1]]
[1] -0.3983099  0.3844952  2.4857206 -2.4099111  0.3299106

[[2]]
[1] 2.942309 1.276206 3.224599 1.956562 3.672690
```

TRUE:

- lapply

FALSE:

- sapply
- tapply
- vapply

SOLUTION:

```
lapply(X=1:2, FUN=rnorm, n=5)
```

```
[[1]]
[1] 1.1590621 0.2943868 1.1132802 1.9783626 1.9728533

[[2]]
[1] 2.2789812 2.1153327 2.7591436 1.2062976 0.2188382
```

Complete the code to the return the output

Replace `???` in the code block below to get the results shown.

SOLUTION:

```
???(c(1, -4.2, 5))
```

RESULTS:

```
[1] 1.0 4.2 5.0
```

TRUE:

- abs

FALSE:

- round
- rep
- seq

SOLUTION:

```
abs(c(1, -4.2, 5))
```

```
[1] 1.0 4.2 5.0
```

What is the output of this code?

```
p <- as.Date("2015-01-06")
format(p, "%d-%m")
```

TRUE:

- "06-01"

FALSE:

- "06-Tuesday"
- "Jan-2015"
- "06-2015"

RESULTS:

```
[1] "06-01"
```

SOLUTION:

```
p <- as.Date("2015-01-06")
format(p, "%d-%m")
```

```
[1] "06-01"
```

6 Complete the code!

`cities` is a `character` vector of city names. Complete the code parts [1] [2] below to obtain the number of characters of each city in `cities` using `sapply`, a user-friendly version of `lapply`.

```
cities <- c("Jonesboro", "Batesville", "Conway", "Searcy")

sapply(cities, nchar)
```

```
Jonesboro Batesville     Conway     Searcy
        9         10          6          6
```

More than one answer is correct.

True:

- [1] cities [2] nchar
- [1] cities[1:4] [2] nchar
- [1] X=cities [2] FUN=nchar

False:

- [1] cities() [2] nchar()

Feedback: `sapply` is a simple version of `lapply` for vectors. It requires the format (check `help(sapply)`): `sapply(X = v, FUN= fun)` for a vector `v` and a function `fun`. The names `X` and `FUN` can be omitted. `cities[1:4]` is an index version of `cities`. The function `cities()` is not known, and `nchar()` is missing an argument, and cannot be used like that inside `sapply`.

```
cities <- c("Jonesboro", "Batesville", "Conway", "Searcy")

sapply(cities, nchar)
sapply(cities[1:4], nchar)
sapply(X=cities, FUN=nchar)
```

```
Jonesboro Batesville     Conway     Searcy
        9         10          6          6
Jonesboro Batesville     Conway     Searcy
        9         10          6          6
Jonesboro Batesville     Conway     Searcy
        9         10          6          6
```

6 Which function turns a list `l` into a vector?

TRUE:

- unlist(l)

FALSE:

- as.vector(l)
- is.vector(l)
- vector(l)

Feedback: `as.vector()` converts a matrix into a vector. `is.vector` checks if its argument is a vector or not. The function `vector()` generates a (by default logical) empty vector.

```
cities <- c("Jonesboro", "Batesville", "Conway", "Searcy")
l <- lapply(cities,nchar)
u <- unlist(l)
v <- as.vector(l)
identical(v,l)
identical(u,l)
```

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

```
li <- list(1, "foo")
unlist(li)
is.matrix(as.matrix(li))
class(as.matrix(li))
vector(li)
as.vector(matrix(1:4))
vector()
```

```
[1] "1"   "foo"
[1] TRUE
[1] "matrix" "array"
Error in vector(li) : invalid 'mode' argument
[1] 1 2 3 4
logical(0)
```

6 CHANGED Complete [1] [2] and [3] in the code below to get the output.

- `addr` is a function that adds its two arguments.
- `foo` is a numeric vector

Identify [1] [2] [3] to obtain the output below!

```
addr <- function(x, y) x + y      # define function
foo <- c(1.45, 4.4, 2.33, 5.0)    # define list

bar <- lapply([1], [2], y = [3])  # run function
unlist(bar)                       # print as vector
```

```
Error: unexpected '[' in "bar <- lapply(["
Error in unlist(bar) : object 'bar' not found
```

Output:

```
: [1]  7.45  10.40  8.33 11.00
```

TRUE:

- [1] foo [2] addr [3] 6

FALSE:

- [1] bar [2] foo [3] x
- [1] foo [2] TRUE [3] bar
- [1] foo [2] addr [3] TRUE

Feedback: `lapply` has the arguments `(X, FUN, FUN.VALUE)` where `FUN.VALUE` is a vector of return values from `FUN`, which is applied to the list `X`. In this case, `FUN.VALUE = y = 5`.

```
addr <- function(x, y) x + y       # define function
foo <- list(1.45, 4.4, 2.33, 5.0)  # define list

bar <- lapply(foo, addr, y = 6)    # run function
unlist(bar)                        # print as vector
```

```
[1]  7.45 10.40  8.33 11.00
```

6 CHANGED What is the output of this code?

Tip:

- `strsplit` splits its argument vector according to its `split` attribute.
- `length` returns the number of elements of a vector
- `sum` sums the numeric elements of a vector

```
cities <- c("Jonesboro:large", "Batesville:small",
            "Searcy:medium", "Conway:medium")

citySize <- strsplit(x=cities, split=":")
len <- sapply(citySize,length)
sum(len)
```

```
[1] 8
```

TRUE:

- 8

FALSE:

- TRUE
- FALSE
- 6

Feedback: After the split, `citySize` is a list of three elements - each element is a character vector like `"Jonesboro" "large"`. `len` is the vector `2 2 2`, with the `length` of each list element, and `sum` sums these values to obtain `6`.

```
cities <- c("Jonesboro:large", "Batesville:small",
            "Searcy:medium", "Conway:medium")

citySize <- strsplit(x=cities, split=":")
len <- sapply(citySize,length)
sum(len)
citySize
```

```
[1] 8
[[1]]
[1] "Jonesboro" "large"

[[2]]
[1] "Batesville" "small"

[[3]]
[1] "Searcy" "medium"

[[4]]
[1] "Conway" "medium"
```

5 CHANGED `search()` returns a list of all packages loaded in the current R session

TRUE

5 CHANGED Which of these four vectors is of the type `logical` (Boolean)?

```
1: c(True, False, False)
2: c(TRUE, FALSE, FALSE)
3: c(TRUE, False, False)
4: c(T, F, F)
```

More than one answer is correct.

TRUE:

- 2
- 4

FALSE:

- 1 and 3
- 3

5 Complete the function definition

> "The function `round(x,digits=0)` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`."

E.g. `round(2.3412)` results in `2`.

Complete the code `???` in the code block below to return the output shown at the end for the two function calls.

```
y <- 2.3412

new_round <- function(p, ??? ) {
  round(p, digits)
}
c(new_round(y), new_round(y, 3))
```

```
[1] 2.300 2.341
```

TRUE:

- `digits = 1`

FALSE:

- `digits`
- `1`
- `TRUE`

Feedback: the second argument is needed to set the number of decimal places `digits`. The function is called twice inside an unnamed vector. The first call, `new_round(y=2.3412)`, uses the default `digits=1`, the second call, `new_round(y=2.3412,3)` sets `digits=3`.

5 Select the code to return the output

Below is a preview of the `mat` matrix.

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Select the code to create this matrix.

TRUE:

- `matrix(c(1,3,5,2,4,6),nrow=2,byrow=TRUE)`

FALSE:

- `matrix(c(1:6),ncol=2,byrow=TRUE)`
- `matrix(c(1:6),nrow=2,byrow=TRUE)`
- `matrix(c(1:6),byrow=TRUE)`

```
matrix(c(1,3,5,2,4,6),nrow=2,byrow=TRUE)
matrix(c(1:6)) ## default is ncol=1
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
     [,1]
```

```
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

4 CHANGED Be the interpreter!

The function `oddcount` is defined in the code block blow. What is the output of the last command (without an explicit argument), `oddcount()`?

```
oddcount <- function(x=1) {
  k <- 0
  for (n in x) {
    if (n %% 2 == 1) k <- k+1
  }
  return(k)
}
oddcount()
```

TRUE:

- `1`

FALSE:

- `0`
- `ERROR`
- `<bytecode: 0x00000000051a7cf8>`

Feedback: Feedback: If no argument is given, the default value 1 used as the function argument - x=1 means n=1 , and 1 %% 2 = 1, therefore k=0+1=1 is returned. The bytecode is the byte-compiled version of the function. These versions run faster than the non-compiled versions. oddcount is not byte-compiled, but the built-in functions like mean() are. Test it by entering mean without an argument. You may know the bytecode concept from Java whose bytecode is executed by the Java virtual machine.

4 CHANGED Extract column vectors

`ToothGrowth` contains the factor vector `supp`, which indicates if a test subject received Vitamin C (`"VC"`) or Orange Juice (`"OJ"`). The data frame looks like this:

```
'data.frame':      60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
 $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

Which command confirms that 30 subjects received Vitamin C?

TRUE:

- `sum(ToothGrowth$supp=="VC")`
- `sum(ToothGrowth[,2,]=="VC")`
- `length(ToothGrowth[ToothGrowth=="VC"])`

FALSE:

- `length(which(ToothGrowth=="OJ"))`

```
str(ToothGrowth)
sum(ToothGrowth$supp=="VC")
sum(ToothGrowth[,2,]=="VC")
length(ToothGrowth[ToothGrowth=="VC"])
length(which(ToothGrowth=="OJ"))
```

```
'data.frame':   60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
 $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
[1] 30
[1] 30
[1] 30
[1] 30
```

Author: Marcus Birkenkrahe
Created: 2022-03-27 Sun 22:08
Validate