Edurekaloops and functions R

SAGAR MEHTA

07/05/2020

1.Given a vector “First_Hundred”, which comprises of sequence of first hundred natural numbers:•Change all the odd numbers to the string “ODD”•Change all the even numbers to the string “EVEN”

```
first_hundred <- 1:100
first_hundred
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

```
for(number in first_hundred)
{
  if(number %% 2 == 0)
    print("even")
  else
    print("odd")
}
```

```
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
```

```
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
## [1] "odd"
## [1] "even"
```

2.From the “iris” dataset, find the number of observations whose “Sepal.Length” is greater than ‘6.5’ by using only loops and conditional statements

```
data("iris")
sepal_6.5 <- ifelse(iris$Sepal.Length>6.5,"Less than 6.5","More than 6.5")
sepal_6.5 <- as.factor(sepal_6.5)
summary(sepal_6.5)
```

```
## Less than 6.5 More than 6.5
##            30           120
```

3.“CO2” is a preloaded data-set in R. From the “CO2” data-set, find the mean ‘uptake’ of only those observations where Type is “Mississippi” and Treatment is ‘chilled’. You canuse only loops and conditional statements.

```
data("CO2")
mean_1 <- ifelse(CO2$Type == 'Mississippi' & CO2$Treatment == 'chilled',mean(CO2$uptake[64:84]),'Na')
unique(mean_1)[2]
```

```
## [1] "15.8142857142857"
```

4.On the “CO2” data-set, use ‘tapply()’ function to obtain mean, median, minimum and maximum values of ‘uptake’ with respect to the ‘Treatment’ column

```
tapply(CO2$uptake,CO2$Treatment,mean)
```

```
## nonchilled    chilled
##   30.64286   23.78333
```

```
tapply(CO2$uptake,CO2$Treatment,median)
```

```
## nonchilled    chilled
##       31.3       19.7
```

```
tapply(CO2$uptake,CO2$Treatment,min)
```

```
## nonchilled    chilled
##       10.6        7.7
```

```
tapply(CO2$uptake,CO2$Treatment,max)
```

```
## nonchilled    chilled
##       45.5       42.4
```

5.‘swiss’ is a preloaded data-set in R. Using the ‘invoke_map()’ function, find out the minimum ‘Fertility’ and maximum ‘Infant.Mortality’ from the ‘swiss’ data-set.

```
data("swiss")
library(purrr)
```

```
## Warning: package 'purrr' was built under R version 3.5.2
```

```
invoke_map(list(min_fertility = 'min',max_infant = 'max'),list(swiss$Fertility,swiss$Infant.Mortality))
```

```
## $min_fertility
## [1] 35
##
## $max_infant
## [1] 26.6
```

6.Create a custom function “dice()” which will give a random number between 1-6 every time the function is invoked.

```
dice <- function() {
  sample(1:6)
}
dice()
```

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

```
dice()
```

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