Class 06: R Functions

Eric Wang PID: 17678188

Table of contents

Background

Functions are at the heart of using R. Everything we do involves calling and using functions (from data input, and analysis to results output).

All functions in R have at least 3 things:

- A **name** the thing we use to call the function.
- One or more input **arguments** that are comma separated
- The **body**, lines of code between curly brackets {} that does the work of the function.

A first function

Let's write a silly wee function to add some numbers:

```
add <- function(x) {
  x + 1
}
```

Let's try it out

```
add(100)
```

```
[1] 101
```

Will this work

```
add( c(100,200,300))
```

```
[1] 101 201 301
```

Modify to be more useful and add more than just 1

```
add <- function(x, y = 0) {
  x + y
}
```

```
add(60,50)
```

```
[1] 110
```

Will this work?

```
add(100)
```

```
[1] 100
```

```
log(10, base = 10)
```

```
[1] 1
```

N.B. input arguments can be required or optional. Optional ones have a default. You can specifiy it in the function code with an equals sign.

```
#add(11,200,300)
```

A second function

All functions in R look like this

```
Name <- function(arg) {
  Body
}
```

The `sample()` function in R will randomize the numbers within a certain data set.

- By default, the function samples without replacement (no repeats).
- If you want to have repeats you can have `sample(x, size, replace = True)`
- If you want to find the probability for a value to show up you can also include `sample(x,size,replacement = True, prob = )`

```
sample(1:100)
```

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

Q. Return 12 numbers picked randomly from input 1:10

```
sample(1:10, 12, TRUE)
```

```
 [1] 1 9 2 1 7 9 5 5 3 5 5 1
```

Q. Write the code to generate a random 12 nucleotide long DNA sequence

sample("A,G,T,C", 12, TRUE)

```
x <- c("A", "T","C","G")
sample(x, 12, TRUE)
```

```
 [1] "G" "A" "T" "A" "G" "A" "T" "G" "G" "T" "T" "T"
```

Q. Write a first version function called `generate_dna()` that generates a user specified length `n` random DNA sequence.

```
generate_dna <- function(n=6) {
  sample(x, n, replace = TRUE)

  }
generate_dna(40)
```

```
 [1] "C" "G" "C" "G" "G" "A" "C" "C" "G" "G" "G" "C" "T" "C" "A" "G" "G" "A" "T"
[20] "G" "A" "T" "C" "C" "G" "T" "A" "T" "T" "G" "G" "C" "T" "G" "C" "T" "T" "G"
[39] "A" "G"
```

Q. Modify your function to return a FASTA like sequence so rather than [1] “T” “C” “T” “C” “C” “G” “C” “A” “T” “C” “C” “T” we want “GCAAT”

```
bases <- c("G","C","A", "T")
generate_dna <- function(n=6) {
  paste(sample(bases, n, replace = TRUE),collapse = "" )
  }
generate_dna(40)
```

```
[1] "CGCAGCATCGCACGTGCTCGCTTCTCTGGTAGGTGCTGTA"
```

Q. Give the use an option to return FASTA format output sequence or standard multi-element vector format.

```
bases <- c("G","C","A", "T")
generate_dna <- function(n=6, fasta = TRUE) {
  bases <- c("G","C","A", "T")
  x <- sample(bases, n, replace = TRUE)

  if(fasta){
  x <- paste(x,collapse = "" )
  }
  return(x)
  }
generate_dna(40, FALSE)
```

```
 [1] "T" "G" "G" "A" "C" "T" "A" "G" "A" "T" "C" "A" "A" "C" "C" "T" "G" "A" "T"
[20] "G" "A" "T" "A" "A" "G" "A" "C" "A" "G" "A" "G" "C" "A" "A" "C" "T" "A" "C"
[39] "G" "T"
```

A new cool function

Q. Write a function called `generate_protein()` that generates a user specied length protein sequence in FASTA like format

```
generate_protein <- function(n=6){
  aa <- c("A","R","N","D","C",
          "Q","E","G","H","I",
          "L","K","M","F","P",
          "S","T","W","Y","V")
  fasta.aa <- sample(aa, n, replace = TRUE)
  fasta.aa <- paste(fasta.aa, collapse = "")
  return(fasta.aa)
}
generate_protein(6)
```

```
[1] "MKPGVT"
```

Q. Use your new `generate_protein()` function to generate sequences between length 6 and 12 amino-acids in length and check of any of these are unique in nature.

```
lengths <- 6:12
proteins <- lapply(lengths, generate_protein)
proteins
```

```
[[1]]
[1] "FYTKLQ"

[[2]]
[1] "DGGVTTH"

[[3]]
[1] "QPQNQVMY"

[[4]]
[1] "IVSIVGPHK"

[[5]]
[1] "QWDLMLNNEW"
```

```
[[6]]
[1] "VLWIHPDWYGV"

[[7]]
[1] "SLSGLNWDHVHW"
```

```
#lapply(x,function)
```

- applies a function to each element of x
- always returns a list

```
#sapply(x,function)
```

- applies a function to each element of x
- always returns a vector or matrix

Or we could do a `for()` loop:

```
for(i in 6:12) {
  cat(">", i, sep="", "\n")
  cat(generate_protein(i), "\n")
}
```

```
>6
EVVFGA
>7
ELKQIET
>8
FAKQKCTW
>9
IQRVRKFRM
>10
DEEIPVAPHM
>11
NENGIIQDRYN
>12
PSANISELVTDM
```

Identical matches with 100% coverage were only found in sequences of 6 and 7.