Class06: R Functions

Paul Brencick (PID:A17668863)

Table of contents

Background

Function are important for everything that we do in R. Everything we do involves calling and using functions (from data input, analysis, to results)

All functions in R have at least 3 things:

1. A **name**; the thing that you use to call a function.
2. One or more input **arguments** that are comma separated.
3. The **body**; lines of code between curly brackets { } which do the work of the function.

Our First Function

We are going to write a dumb function to add some numbers even though it already exist:

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

Lets try it out!

```
add(100)
```

```
[1] 101
```

Will this work??

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

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

Modified version that is more useful to add more than just 1:

```
add <- function(shlop,gap=1) {
  shlop + gap
}
```

```
add(100, 10)
```

```
[1] 110
```

Will this still work??

```
add(100)
```

```
[1] 101
```

N.B Input arguments can either be **required** or **optional**. Optional ones contain a default, which is a fall-back specified in the function code with an equal signs.

```
#add(30,100, 400)
```

Lets do another function :)

All functions in R look like this:

```
name <- function(argument) {
 body
}
```

The `sample()` function in R randomly selects a value from within a collection or a range. It randomly shuffles the elements of x and picks the amount specified by size.

```
sample(1:10, size = 4)
```

```
[1] 8 7 5 4
```

Q. Return 12 numbers picked randomly from the imput 1:10

```
sample(1:10, size = 12, replace = T)
```

```
 [1] 9 7 9 6 8 2 3 7 6 2 7 9
```

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

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

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

Q. Write a first verion function called `generate_dna()` which generates a user specified length `n` random DNA sequence?

```
generate_dna <- function(size=6) {
 bases <- c("A","T","C","G")
 sample(bases,size,T)
}
```

Lets try it out:

```
generate_dna(20)
```

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

Q. Modify your function to return a FASTA like sequence rather than “G” “A” “C” “A”, we want “GTACT”

```
generate_dna <- function(size=6) {
 bases <- c("A","T","C","G")
 ans <- paste(sample(bases,size,T), collapse="")
 return(ans)
}
```

```
generate_dna(6)
```

```
[1] "AATCGG"
```

Q. Give the user an option ot return FASTA format output sequence or standard multi-element vector format?

```
generate_dna <- function(size=6, fasta=T) {
 bases <- c("A","T","C","G")
 ans <- paste(sample(bases,size,T), collapse="")

 if (fasta) {
   return(ans)
 } else {
   return(sample(bases,size,T))
 }
 return(ans)
}
```

```
generate_dna(10,T)
```

```
[1] "GACGGCCGAC"
```

```
generate_dna(10, F)
```

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

A new cool function

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

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

```
generate_protein(10)
```

```
[1] "WLRGPEWGMW"
```

Q. Use your new `generate_protein()` function to generate sequences between length 6 and 12 amino-acids and check if any of there are unique in nature (i.e found in the NR database at NCBI)?

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

proteins <- data.frame(
  length = 6:12,
  sequence = sapply(6:12, generate_protein)
)

proteins
```

```
  length     sequence
1      6       WTPVED
2      7      GGIRRKG
3      8     KGPHQDGS
4      9    NRNTNLCVE
5     10   YPIVAPVEWD
6     11  NKNHRPEWGNQ
7     12 DNCDCHTHKCML
```

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

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

```
>6
SWKWMD
```

```
>7
PIMIVHA
>8
WCYNWVGF
>9
VAQTLTQRV
>10
KVSDQAFVRN
>11
VNFFSIFHNNM
>12
MTGYLRARMCKS
```