Class06: R Functions

Leah Johnson (PID: A17394690)

Table of contents

Background

All functions in R have at least 3 things:

- A **name** that we use to call the function.
- One or more input **arguments**.
- The **body** the lines of R code that do the work.

Our first function

Let's write a silly wee function called 'add()' to add some numbers (the input arguments).

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

Now we can use this function:

```r
add(100, 1)
```

```
[1] 101
```

```
add(x=10, y=10)
```

```
[1] 20
```

```
add(x=c(11, 1, 100), y=1)
```

```
[1]  12   2 101
```

> Q. What if I give a multiple element vector to ‘x’ and ‘y’?

```
add(x=c(100, 1), y=c(100, 1))
```

```
[1] 200   2
```

> Q. What if I give three inputs to the function?

```
#add(x=c(100, 1), y=1, z=1)
```

> Q. What if I give only one input to the add function?

```
addnew <- function(x, y=1) {
  x + y
}
```

```
addnew(x=100)
```

```
[1] 101
```

```
addnew(c(100,1), 100)
```

```
[1] 200 101
```

If we write our function with input arguments having no default value, then the user will be required to set them when they use the function. We can give our input arguments “default” values by setting them equal to some sensible value - e.g. y=1 in the ‘addnew()’ function?

A second function

Let’s try something more interesting: Make a sequence generating tool:

The ‘sample()’ function can be a useful starting point here:

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

```
[1] 3 1 4 9
```

> Q. Can you generate 9 random numbers taken from the input vector x=1:10?

```r
sample(1:10, size=9)
```

```
[1] 10  9  6  3  4  5  1  8  2
```

> Q. Can you generate 12 random numbers taken from the input vector x=1:10?

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

```
 [1] 2 8 9 8 7 5 9 9 2 4 9 9
```

> Q. Write code for the ‘sample()’ function that generates nucleotide sequences of length 6?

```r
sample(x=c("A", "C", "T","G"), size=6, replace=TRUE)
```

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

> Q. Write a first function ‘generate_dna()’ that returns a *user specified length* of DNA sequence:

```r
generate_dna <- function(len) {
  sample(x=c("A", "C", "G", "T"), size=len, replace=TRUE)
}
```

```r
generate_dna()
```

```
[1] "A" "C" "C" "C"
```

```
generate_dna <- function(len=100) {
  sample(x=c("A", "C", "G", "T"), size=len, replace=TRUE)
}
```

```
generate_dna(len = 100)
```

```
  [1] "A" "C" "T" "G" "A" "C" "A" "A" "G" "A" "G" "T" "A" "A" "C" "G" "G" "C"
 [19] "C" "T" "A" "G" "C" "A" "T" "T" "G" "A" "T" "G" "C" "G" "C" "A" "C" "C"
 [37] "C" "C" "C" "G" "C" "A" "A" "T" "A" "C" "A" "A" "A" "T" "C" "T" "T" "G"
 [55] "T" "C" "C" "A" "T" "A" "C" "C" "G" "C" "G" "G" "A" "A" "G" "A" "G" "G"
 [73] "C" "C" "A" "T" "C" "T" "C" "T" "A" "C" "C" "G" "G" "T" "G" "G" "T" "T"
 [91] "C" "A" "A" "C" "G" "G" "C" "G" "C" "C"
```

Key points Every function in R looks fundamentally the same in terms of its structure. Basically three things: name, input, and body

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

Functions can have nultiple inputs. These can be **required** arguments or **optional** arguments. With optional arguments having a set default value.

Q. Modify and improve our 'generate_dna()' function to return its generated sequence in a more standard format like "AGTAGTA" rather than the vector "A", "C", "G", "T".

```
generate_dna <- function(len=6, fasta=TRUE) {

  ans <- sample(x=c("A", "C", "G", "T"), size=len, replace=TRUE)
  if(fasta) {
    cat("Single-element vector output")
    ans <- paste(ans, collapse = "")
  } else {
    cat("Multi-element vector output")
  }
  return(ans)
}
generate_dna(fasta=TRUE)
```

```
Single-element vector output
```

```
[1] "TGTCTG"
```

```r
generate_dna(fasta=FALSE)
```

```
Multi-element vector output
```

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

The 'paste()' function - its job is to join up or stick together (a.k.a paste) input strings together.

```r
paste("alice", "loves R", sep=" ")
```

```
[1] "alice loves R"
```

Flow control means where the R brain goes in your code.

```r
good_mood <- TRUE

if(good_mood) {
  cat("Great!")
} else {
  cat("Bummer!")
}
```

```
Great!
```

A Protein generating function

> Q. Write a function that generates a user specified length protein sequence.

There are 20 natural amino acids.

```r
aa <- c("A", "R", "N", "D", "C", "Q", "E", "G", "H", "I", "L", "K", "M", "F", "P", "S", "T",
generate_protein <- function(len=100) {

  # The amino-acids to sample from
  aa <- c("A", "R", "N", "D", "C", "Q", "E", "G", "H", "I", "L", "K", "M", "F", "P", "S", "T

  # Draw n=len amino acids to make our sequence
  ans <- sample(aa, size=len, replace=TRUE)
  ans <- paste(ans, collapse= "")
  return(ans)
}
```

```r
myseq <- generate_protein(42)
myseq
```

```
[1] "TMWETDLRFWHPNLEKFVWHWRAAVNQLTNWLHMHLLGPSKA"
```

> Q. Use that function to generate random protein sequences between length 6 and 12.

```r
generate_protein(6)
```

```
[1] "DMMADG"
```

```r
generate_protein(7)
```

```
[1] "LNPYEFH"
```

```r
generate_protein(8)
```

```
[1] "RVANCSES"
```

```r
generate_protein(9)
```

```
[1] "YENDCMQSK"
```

```r
generate_protein(10)
```

```
[1] "AFQQLSQDGM"
```

```r
generate_protein(11)
```

```
[1] "YHMSFQHMIFI"
```

```r
generate_protein(12)
```

```
[1] "WDFCRWGFYFKV"
```

```r
for(i in 6:12) {
  # FASTA ID line ">id"
  cat(">", i, sep="",  "\n")
  # Protein sequence line
  cat(generate_protein(i), "\n")
}
```

```
>6
QECAHQ
>7
AHTMFQH
>8
CLIYSKKP
>9
PCMCYLCYA
>10
RKHFRILCAV
>11
ETMVDSVMLSN
>12
CPKYMLKVYVHY
```

> Q. Are any of your sequences unique i.e. not found anywhere in nature?

My sequences 9-12 are unique. Beginning at my sequence 9, there is 90% identity and 100% coverage. There are 9 amino acids on the MHC class 1 binding groove.