Class06: R Functions

Emma Bell

Table of contents

Background

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

All functions in R have at least three things:

1. a **name** the thing we use to call the function.
2. one or more input **arguments** that are comma separated
3. the **body**, lines of code between curly brackets {} that does the work of the function.

A first function

Let's write a silly wee fucntion 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
```

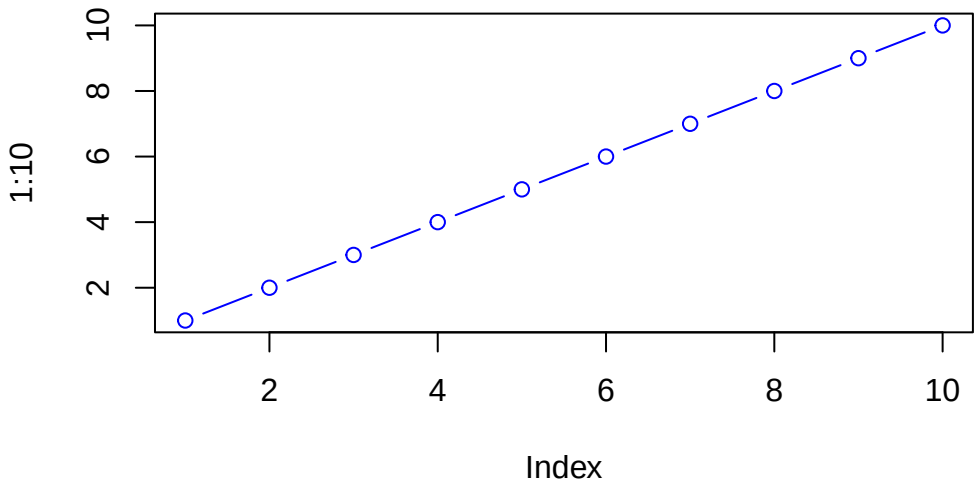
Modify to be more useful and add more than just 1

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

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

```
[1] 110
```

```
plot(1:10, col="blue", typ= "b")
```



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

```
[1] 1
```

Given y a default, so if we don't give a value for y, it will automatically think 1.

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

```
add(100)
```

```
[1] 101
```

> **N.B** Input arguments can either be **required** or *optional*. The latter have a fall-back default that is specified in the function code with an equal sign.

```
#add(x=100, y=200, z=300)
```

A Second Function

All funcitons in R look like this

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

The `sample()` function in R

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

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

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

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

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

you have to replace to TRUE, when wanting to generate more numbers than the size you have.

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

```
DNA <- c("A", "C", "T", "G")
sample(DNA, size=12, replace=TRUE)
```

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

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

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

```
generate_dna()
```

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

```
generate_dna(100)
```

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

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

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

```
hello <- c("H", "E", "L", "L", "O")
paste(hello, collapse="")
```

```
[1] "HELLO"
```

THIS is how you get the quotations marks out of the way.

```
paste(generate_dna(20), collapse="")
```

```
[1] "CTGGATCCGTTACCTGTTCT"
```

or

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

```
generate_dna(10)
```

```
[1] "CAGAAGAGCA"
```

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

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

if(fasta) {
  ans <- paste(ans, collapse="")
}
return(ans)
}
```

```
generate_dna(10)
```

```
[1] "ATTTTTTCCC"
```

```
generate_dna(10, fasta=FALSE)
```

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

just to show how it works:

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

if(fasta) {
  ans <- paste(ans, collapse="")
  cat("Hello...")
}
return(ans)
}
```

```
generate_dna(10)
```

```
Hello...
```

```
[1] "GTACCGTGAC"
```

```
generate_dna(10, fasta=FALSE)
```

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

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

if(fasta) {
  ans <- paste(ans, collapse="")
  cat("Hello...")
```

```
} else {
  cat("...is it me you are looking for")
}
return(ans)
}
```

```
generate_dna(10)
```

```
Hello...
```

```
[1] "ACGGTTGGGG"
```

```
generate_dna(10, fasta=FALSE)
```

```
...is it me you are looking for
```

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

A new cool function

> Q. Write a function called `generate_protein()` that generates a use specified 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
ans <- sample(aa, size=n, replace=TRUE)
ans <- paste(ans, collapse="")
return(ans)
}
```

```
generate_protein(10)
```

```
[1] "NKTKIFNEMA"
```

> Q. Use your new `generate_protein()` function to generate all sequences between length 6 and 12 amino acids in length and check that any of these are unique in nature (ie found in the NR database at NBI)

```
generate_protein(6)
```

```
[1] "DDDMHY"
```

this is found in NBI

```
generate_protein(12)
```

```
[1] "LADYKGQYRPMD"
```

there is no 100% cover and identity

```
generate_protein(10)
```

```
[1] "RIQWRQCGSI"
```

to not keep repeating yourself, using the for() loop

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

```
6
QNEEPN
7
LFETDVQ
8
WDDTPSYG
9
QCHVTHVYG
10
ALCVFCPTIH
11
WKPDISCPCTT
12
QWYYGWWVFKST
```

what a FASTA format looks like: >id AGKRTST >next GKSTR

sep is for getting rid of the space between the > and the numbers

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

```
>6
RKNRAF
>7
LTLWLWQ
>8
SSNWPVLR
>9
PTDGFMYIG
>10
TNNQVWNDNY
>11
QNNNILLFFAV
>12
QDGGYKELIKQS
```

number 9, 10, 11, 12 does not have a match