Class06: R Functions

Nicole Lin (PID: A17826971)

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 3 things:

1. A **name** the thing we 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 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=1) {
  x + y
}
```

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

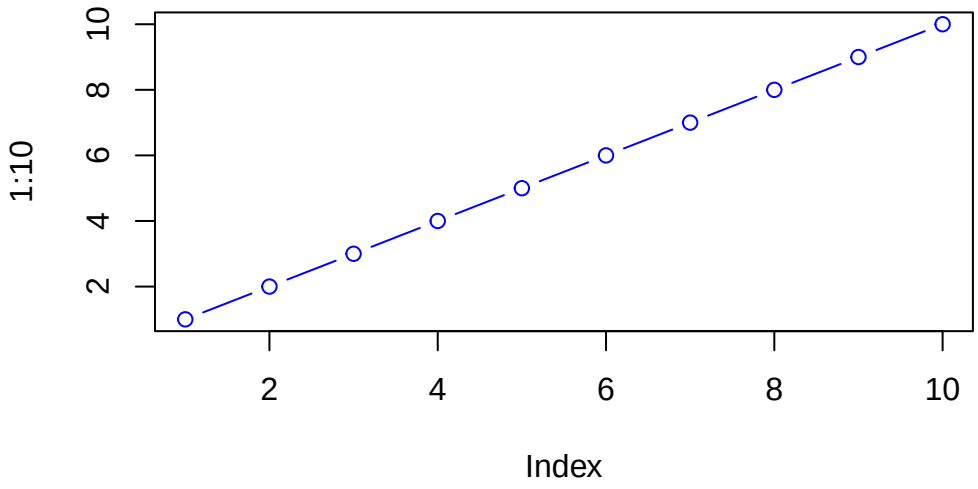
```
[1] 110
```

Will this work?

```
add(100)
```

```
[1] 101
```

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




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

```
[1] 1
```

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

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

A second function

All functions in R look like this

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

The `sample()` function in R randomly selects a specified amount of items from a vector with or without replacement. With 4 arguments: x, size, replace, and prob (x and size are required). Default for replace is FALSE (without replacement)

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

```
[1]  9  3 10  6
```

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

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

```
 [1] 10  4 10  7  8  2  4  3  7  1  8  7
```

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

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

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

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

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

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

```
generate_dna(100)
```

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

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

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

```
generate_dna(10)
```

```
[1] "ATTACAGTAC"
```

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) {
  bases <- c("A", "T", "G", "C")
  ans <- sample(bases, 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] "ACAATCGAAT"
```

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

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

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

A new cool function

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

```
generate_protein(30)
```

```
[1] "GCMYRRMVEIPIVKLTFGTWHWHCFAPLTF"
```

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

We could de a `for()` loop:

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

```
>6
PVKEVH
>7
ILILDEG
>8
SCQGNWPI
>9
YICQQDCDI
>10
NGPSMQVQNM
>11
SMRFNLGRRIY
```

>12
YSKWIYQGYKDW