Class06: R Functions

Barry (PID: 911)

Table of contents

Background 1
A first function 1
A second function 3
A new cool function 6

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 use to call the function.
2. One or more input **arguments** that a 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)
```

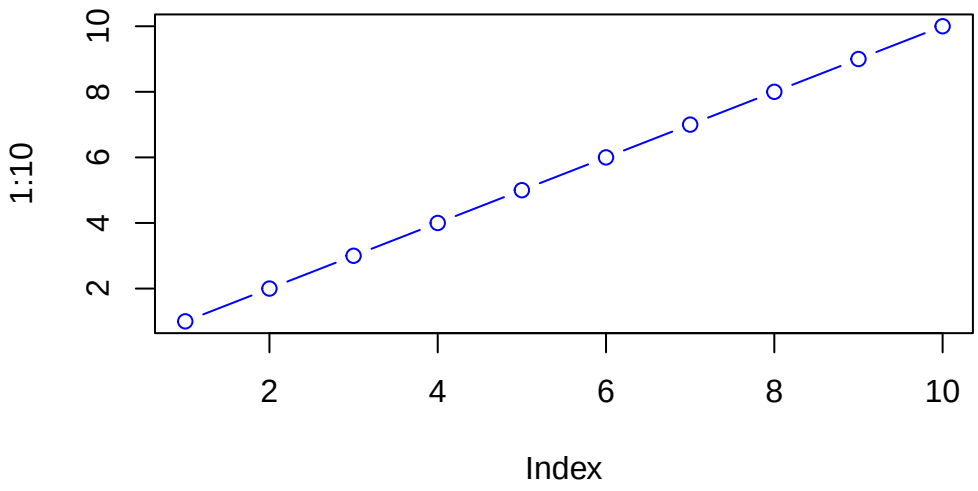
```
[1] 110
```

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 later have a fall-back default that is specifed in the function code with an equals sign.

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

A second function

All functions in R look like this

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

The `sample()` function in R ...

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

```
[1] 1 2 5 6
```

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

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

```
 [1]  8 10  1  7  2  1 10  2  2  7  5  5
```

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

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

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

Q. Wite 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","C","G","T")
  sample(bases, size=n, replace=TRUE)
}
```

```
generate_dna(100)
```

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

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","C","G","T")
  ans <- sample(bases, size=n, replace=TRUE)
  ans <- paste(ans, collapse = "")
  return(ans)
  x <-"poopoopants"
  x
}
```

```
generate_dna(10)
```

```
[1] "TAGTGGACTG"
```

An example

```
# Example pattern (not using your bases)
x <- c("H", "E", "L", "L", "O")

paste(x, collapse = "****")
```

```
[1] "H****E****L****L****O"
```

```
# returns "HELLO"
```

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

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

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

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

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

A new cool function

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

```
generate_protein <- function(n) {

  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 = T)
  ans <- paste(ans, collapse = "")
  return( ans )
}
```

```
generate_protein(10)
```

```
[1] "PEGYGSHRSF"
```

> 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 (i.e. found in the NR database at NCBI)?

```
generate_protein(6)
```

```
[1] "LDSMMS"
```

```
generate_protein(7)
```

```
[1] "YTVKVET"
```

```
generate_protein(8)
```

```
[1] "EQNRCGSC"
```

```
generate_protein(9)
```

```
[1] "EVYCAYMAS"
```

```
generate_protein(10)
```

```
[1] "KLKWEFIFLH"
```

```
generate_protein(11)
```

```
[1] "SENPTMTANNW"
```

```
generate_protein(12)
```

```
[1] "LLWIVNTETTRL"
```

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

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

```
>6
TAIGQQ
>7
WRWRNTL
>8
AACHNQCR
>9
SMPCYGYFP
>10
ISYTQLKVEW
>11
MICNQASASIN
>12
MTFSNYNMTQIC
```

id AGKRTST next GKRTTST