Class 06: R Functions

Kimberly Navarro

Table of contents

Background

Functions are at the heart of using R. Everything we do involves calling and using fractions (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 are a comma separared
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:

```r
add <- function(x) {
  x + 1
}
```

Let's try it out

```r
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)
```

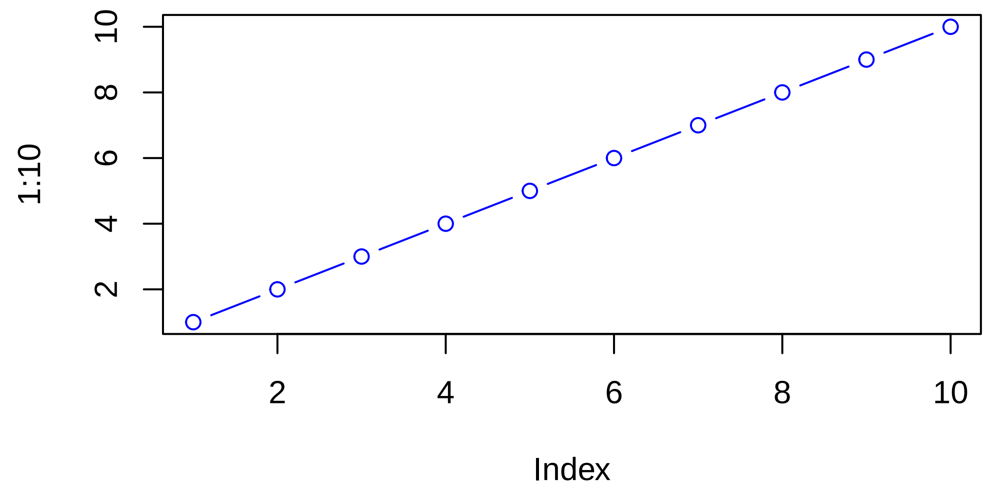
```
[1] 110
```

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 specified in the function code with an equal 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...?

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

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

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

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

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

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

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

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

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

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

```r
generate_dna(100)
```

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

> Q. Modify your function to return a FASTA like sequence so rather than [1] " "A" "T" "C" "C" "G" we want "ATCCG"

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

```r
generate_dna(10)
```

```
[1] "GGATTGGGGG"
```

```r
x<- 100
x<- 10
x<- 300
x
```

```
[1] 300
```

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

```r
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)
}
```

```r
generate_dna(10)
```

```
Hello...
```

```
[1] "GTAGATGTAA"
```

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

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

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

A new cool function

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

```r
generate_protein <- function(n){

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

```r
generate_protein(10)
```

```
[1] "HPYGLCYCNK"
```

> 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)

```r
generate_protein(6)
```

```
[1] "EIGSFH"
```

```r
generate_protein(7)
```

```
[1] "KVKYDFI"
```

```r
generate_protein(8)
```

```
[1] "ACYWRHEY"
```

```r
generate_protein(9)
```

```
[1] "LGMESNGHY"
```

```r
generate_protein(10)
```

```
[1] "NCDLWVNMPI"
```

```r
generate_protein(11)
```

```
[1] "KTPEPEAIIIR"
```

```r
generate_protein(12)
```

```
[1] "SCWFSLHCIQKI"
```

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

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

```
>6
EQTAGP
>7
LEMFTAQ
>8
VLEGYVGV
>9
HGSDSHLKQ
>10
RGHDFSGMIT
```

```
>11
CMWCKMRDPCV
>12
GNYKKKAEQMGM
```

Results: Only 6-8 were found within the NCBI database.