Class6

Table of contents

Background

Functions are at the heart of using R. Evrything we do involves calling and sing 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 are comma separated
3. The **body**, lines of code between curly bracketa { }, that does the work of the function

A First Function

Let's write a function to add some numbers:

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

Let's try it out

```r
add(100)
```

```
[1] 101
```

Will this work:

```r
add(c(100,200,300))
```

```
[1] 101 201 301
```

We want to modify the code, for it to be more useful and add more than just 1:

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

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

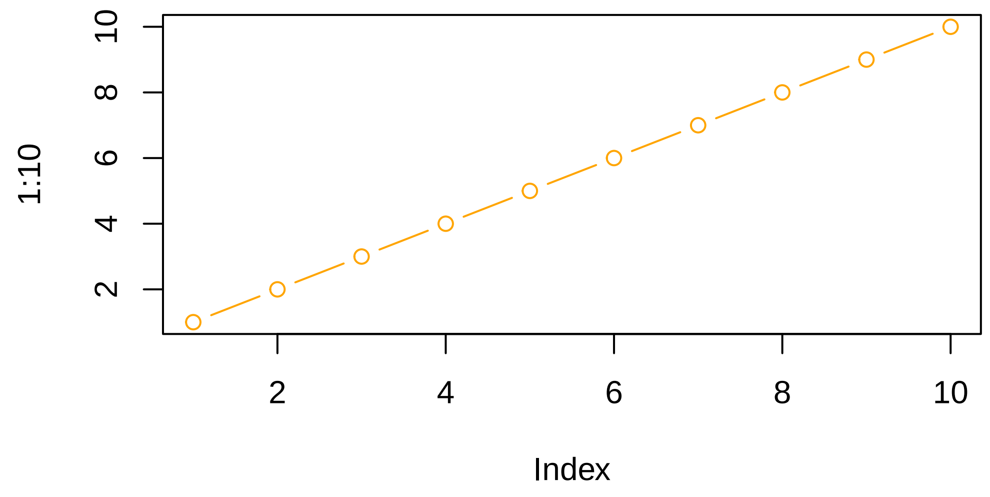
```
[1] 110
```

```r
add(100)
```

```
[1] 101
```

We cannot run the code as we need y!

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



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

```
[1] 1
```

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

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

A Second Function

All R functions in R look like this:

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

The `sample()` function in R …

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

```
[1]  5  3  8 10
```

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

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

```
 [1]  3  8  1  7  9  4  3  3 10  2 10  5
```

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

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

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

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

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

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

```r
generate_dna(25)
```

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

Q. Modify our function to return a FASTA-like sequence. So, rather than "A" "A" "C" "T", we want to get "AACT". (Use `collapse`)

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

```r
generate_dna (100)
```

```
[1] "AGGGTCTGCCATTGGTGTAAACCGCTTACACGAAGAGCACAACAGTGTGTAAATTAGGACGATACGAACCTATGATAGAAGAGAACGC
```

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

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

if(FASTA) {
 ans<-paste(ans, collapse = "")
 cat("Hello>>>")
} else {
  cat("LALALALLA")
}
 return(ans)
}
```

```r
generate_dna(10)
```

```
Hello>>>

[1] "CACCTGCATT"
```

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

```
LALALALLA

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

A third, 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, FASTA=T) {
  amino_acids <- c ("A","R","N","D","C","E","Q","G","H","I",
                    "L","K","M","F","P","S","T","W","Y","V")
  ans <- sample(amino_acids, size=n, replace = T)
 ans<-paste(ans, collapse = "")
 return(ans)
}
```

```r
generate_protein (7)
```

```
[1] "RLVEQHN"
```

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

```r
generate_protein(6)
```

```
[1] "GGEFHI"
```

```r
generate_protein(7)
```

```
[1] "MKEMWTQ"
```

```r
generate_protein(8)
```

```
[1] "SGGPRPHL"
```

```r
generate_protein(9)
```

```
[1] "GCKNCVRCF"
```

```r
generate_protein(10)
```

```
[1] "YPGYIVRITK"
```

```r
generate_protein(11)
```

```
[1] "IVFRGVKPCSK"
```

```r
generate_protein(12)
```

```
[1] "THRRFNAFLLKF"
```

Or.. we could fo a `for()` loop:

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

```
>6
YRWWQK
>7
GNEMHFG
>8
LVDYEQWQ
>9
NATWIKWDH
>10
HRYQHLHKQE
>11
KNTNRCVKVLI
>12
STDARNKMHERR
```