Class006: R Functions

Ricardo D. Carias PID:A18573289

Table of contents

Background

Functions are at the core of coding within R. **Everything** involves calling and using functions. This is true for data input, analysis, and design.

ALL FUNCTIONS IN R REQUIRE:

1. A **NAME**, the thing used 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.

First Function:

Function— High Level Math:

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

Let's run it and find out.

```
add(100)
```

```
[1] 101
```

Now, lets try something else.

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

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

Ok, now we are going to modify it to be more useful.

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

Now, if you don't specify a value for y, the code will automatically use the value 1.

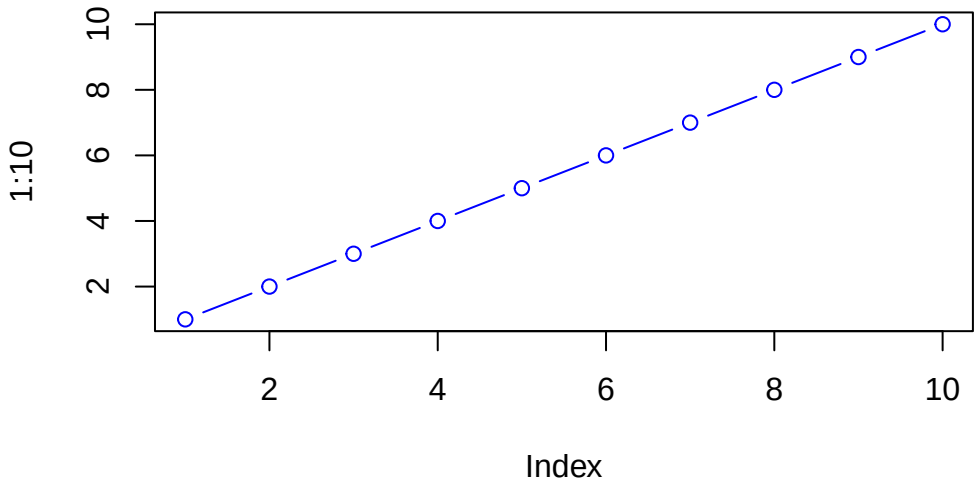
```
add(100,10)
```

```
[1] 110
```

```
add(100,0)
```

```
[1] 100
```

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




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

```
[1] 1
```

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

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

Second Function:

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

The `sample()` function in R, takes a sample of the specified size from the elements of the vector you indicate. With or without replacement, depending on your code. In other words, random shuffle of the elements within the specified vector.

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

```
[1] 7 1 5 9
```

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

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

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

Use your Stats brain. If we set replace equal to false, then we are 'taking items out of the bag without replacementand it is impossible to ask for 12 returns if all we have is 1-10. We literally run out of elements to pull'.

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

```
nucleotide <- c( "A", "T", "C", "G")

sample(nucleotide, size= 12, replace= TRUE)
```

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

Q. Write a first version function called `generate_dna()` that generates a use specefied length `n`?

First Attempt:

```
n <- nucleotide[1:100]
generate_dna <- sample(nucleotide, replace= TRUE){

}
```

Second Attempt:

```
generate_dna <- function(n=6){
  nucleotide <- c( "A", "T", "C", "G")

sample(nucleotide, size= n, replace= TRUE)

  }
```

```
generate_dna(25)
```

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

Q. Modify your function to return a FASTA file like sequence.

```
generate_dna <- function(n=6){
  nucleotide <- c( "A", "T", "C", "G")
mod1<- sample(nucleotide, size= n, replace= TRUE)
paste(mod1, collapse = "")
}

generate_dna()
```

```
[1] "GCCAAC"
```

Q. What if we want our output to be in FASTA format? **AKA** sequence or standard multi-element vector?

```
generate_dna <- function(n = 6, fasta = TRUE) {
  nucleotide <- c("A", "T", "C", "G")
  mod1 <- sample(nucleotide, size = n, replace = TRUE)

  list(mod1, paste(mod1, collapse = ""))

  if (fasta) {
    mod1 <- paste(mod1, collapse = "")
  }

  mod1
}

generate_dna(fasta = FALSE)
```

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

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

generate_dna(fasta = TRUE)
```

```
[1] "AACTTT"
```

```
# [1] "ATCGAT"
```

```
generate_dna( 25, fasta= T)
```

```
[1] "AAGGCTTGGGTAATGGAATTCGTCA"
```

Q. Write a function alled `generate_protein()` that generates a user specified length protein sequence in FASTA?

```
generate_protein <- function(n=10, fasta =T){
aa <- c(
  "A","R","N","D","C","E","Q","G","H","I",
  "L","K","M","F","P","S","T","W","Y","V")

  mod2 <- sample(aa, size= n, replace = T)
list(mod2, paste(mod2, collapse = ""))

if(fasta){
  mod2<- paste(mod2, collapse = "")
}

mod2
}
```

Q. Use your new `generate_protein()` function to generate seqeunces between 6-12 aa and check if any of these are unique, real, within NCBI.(Seq1=6, Seq2=7, Seq3=8, etc.)

```
generate_protein(6)
```

```
[1] "VPAKHE"
```

```
generate_protein(7)
```

```
[1] "CICPTMN"
```

```
generate_protein(8)
```

```
[1] "YTGWCRDK"
```

```
generate_protein(9)
```

```
[1] "SQFVCGGMD"
```

```
generate_protein(10)
```

```
[1] "MPTNAIHFER"
```

```
generate_protein(11)
```

```
[1] "NLTMNNFPYTK"
```

```
generate_protein(12)
```

```
[1] "CSRSHDMCVEGQ"
```

We were able to find unique sequences. I was even able to find one for a 10 aa sequence.

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

```
>6
FKVPWI
>7
GMGLWPL
>8
```

```
CQMNQNTD
>9
WRFPHEVFA
>10
EVRYDGTQIS
>11
YQHKPPVTNQI
>12
FHEMFESHNWQR
```