Class 6: R Functions

Henry Li (A16354124)

Table of contents

There are at least 3 parts of an R function: 1. Function name 2. Arguments (inputs) 3. Function body (code that does the work)

Our first (silly) function

Write a function to add some numbers

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

Now we can call this function:

```r
add(5)
```

```
[1] 6
```

Now we try a funciton with multiple arguments

```r
add_m <- function(x, y){
  x + y
}
```

Now we can call this function:

```r
add_m(5, 10)
```

```
[1] 15
```

Now we can call a function with a default argument:

```r
add_d <- function(x, y = 2){
  x + y
}
```

Now we can call this function:

```r
add_d(5)
```

```
[1] 7
```

A second function

Write a function to generate random nucleotide sequences of a user specified length:

Helper function `sample()`: takes a sample of the specified size from the elements of x using either with or without replacement. Arguments: x, size, replace, prob.

`x`: a vector of one or more elements from which to choose. `size`: a non-negative integer giving the number of items to choose. `replace`: should sampling be with replacement? `prob`: a vector of probability weights for obtaining the elements of the vector being sampled.

`size`: number of items to choose.

`replace`: should sampling be with replacement?

`prob`: a vector of probability weights for obtaining the elements of the vector being sampled.

Helper function `paste()`: concatenates vectors after converting to character. Arguments: …, sep, collapse. `...`: R objects to be converted to character vectors. `sep`: a character string to separate the terms. `collapse`: an optional character string to separate the results.

`...`: R objects to be converted to character vectors.

`collapse`: an optional character string to separate the results.

```r
# Demonstration of sample()
sample(c("A", "T", "C", "G"), 3, replace = FALSE)
```

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

```r
sample(c("A", "T", "C", "G"), 10, replace = TRUE)
```

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

```r
v <- sample(c("A", "T", "C", "G"), 20, replace = TRUE)
paste(v, collapse = "")
```

```
[1] "ATGCGACGTTACAAGCATTT"
```

```r
# Now we write the function
random_seq <- function(length = 75){
  bases <- c("A", "T", "C", "G")
  seq <- sample(bases, length, replace = TRUE)
  paste(seq, collapse = "")
}
# Calling the function
random_seq(10)
```

```
[1] "CAATGAATCT"
```

```r
random_seq()
```

```
[1] "ACCCGTTGCACGATCTTTGTTATAGATAATCCGCGTTATGCTCAGTCCTGAATGCATACGGCCGTTACCCAATGG"
```

Now let's try the if-else cases:

```r
fasta <- TRUE
if(fasta){
  cat("BYE world!")
}else{
  cat("bye world!")
}
```

```
BYE world!
```

Add the ability to return a multi-element veector or a single element fasta like vector.

```r
fasta_seq <- function(length = 75, fasta = TRUE){
  bases <- c("A", "T", "C", "G")
  seq <- sample(bases, length, replace = TRUE)
  paste(seq, collapse = "")
  if(fasta){
    cat("Cool\n")
    return(paste(paste(">random_sequence_of_Henry_length_", length, sep = ""), paste(seq, co
  }else{
    cat("Very Cool :(\n")
    return(seq)
  }
}

fasta_seq()
```

```
Cool
```

```
[1] ">random_sequence_of_Henry_length_75 ATTCGGGCTCGGGGCTGCCAGTCGTTATGCGGGACCACACGTTCGGATGGT
```

```r
fasta_seq(fasta = FALSE)
```

```
Very Cool :(
```

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

Here we see the fourth feature of a function : `return()`. This is used to return a value from a function. If no return() is used, the function will return the last evaluated expression.

A protein generation function

```r
pro_fasta_seq <- function(size = 9, fasta = TRUE) {
  aas  <- c("A","R","N","D","C","E","Q","G","H","I","L","K","M","F","P","S","T","W","Y","V")
  seqv <- sample(aas, size, replace = TRUE)
  body <- paste(seqv, collapse = "")

  if (fasta) {
    hrd  <- paste0(">", "random_aa_sequence_of_Henry_length_", size)
    out <- paste(hrd, body)   # now body is the full FASTA entry
    return(out)              # or: invisible(out) if you don't want sapply to print
  } else {
    return(seqv)
  }
}

pro_fasta_seq()
```

```
[1] ">random_aa_sequence_of_Henry_length_9 IPRFLFHDN"
```

Use our new function to make random protein sequences of length 6 to 12.

```r
for(i in 6:12){
  cat(pro_fasta_seq(size = i))
  cat("\n")
}
```

```
>random_aa_sequence_of_Henry_length_6 MKWVAC
>random_aa_sequence_of_Henry_length_7 EENINFY
>random_aa_sequence_of_Henry_length_8 LCDDVGNW
>random_aa_sequence_of_Henry_length_9 YWQVWGWLQ
>random_aa_sequence_of_Henry_length_10 ATAMEFVPRP
>random_aa_sequence_of_Henry_length_11 KADRRIFMRDC
>random_aa_sequence_of_Henry_length_12 YTQPYSPWDVSK
```

What is the difference between the argument option 'sep' and 'collapse' in the paste() function? `sep` is used to specify the string that separates individual elements when concatenating multiple vectors, while `collapse` is used to specify a string that separates the final concatenated result into a single string. Examples:

```r
# 1) sep: between arguments, element-wise
paste(c("a","b"), c(1,2), sep = "-")
```

```
[1] "a-1" "b-2"
```

```r
#> "a-1" "b-2"

# 2) collapse: between results, across elements
paste(c("a","b","c"), collapse = ",")
```

```
[1] "a,b,c"
```

```r
#> "a,b,c"

# 3) Both together
paste(letters[1:3], 1:3, sep = "=", collapse = " | ")
```

```
[1] "a=1 | b=2 | c=3"
```

```r
# element-wise with sep: "a=1" "b=2" "c=3"
# then collapsed with " | "
#> "a=1 | b=2 | c=3"

# 3.5)
paste(letters[1:3], 1:3, collapse = " | ", sep = "=")
```

```
[1] "a=1 | b=2 | c=3"
```

```r
# 4) paste0 is paste with sep = ""
paste("file", 1:3, sep = ".txt")
```

```
[1] "file.txt1" "file.txt2" "file.txt3"
```

```r
paste0("file", 1:3, collapse = ".txt")
```

```
[1] "file1.txtfile2.txtfile3"
```

`sapply()`: a user-friendly version of lapply by default returning a vector, matrix or array if appropriate. Arguments: X, FUN, …, simplify, USE.NAMES. `X`: a vector (atomic or list) or an expression object. `FUN`: the function to be applied to each element of X. `...`: optional arguments to FUN. `simplify`: should the result be simplified to a vector, matrix or higher dimensional array if possible? `USE.NAMES`: logical indicating whether to use the names of X as names for the result (if X has names).

```r
sapply(6:12, pro_fasta_seq)
```

```
[1] ">random_aa_sequence_of_Henry_length_6 NMLIDS"
[2] ">random_aa_sequence_of_Henry_length_7 CEDIYFM"
[3] ">random_aa_sequence_of_Henry_length_8 WGWILLMC"
[4] ">random_aa_sequence_of_Henry_length_9 HNNRWEVSC"
[5] ">random_aa_sequence_of_Henry_length_10 PTLMIFYAGI"
[6] ">random_aa_sequence_of_Henry_length_11 YTFAYFYSLVK"
[7] ">random_aa_sequence_of_Henry_length_12 YLSCPWAASDLN"
```