Class 06: R Functions

Renee Zuhars (PID: A17329856)

Table of contents

Section 1. Writing Functions

Let's start writing our first silly function to add some numbers.

Every R function has 3 things:

- name (we get to pick this)
- input arguments (there can be loads of these separated by a comma)
- the body (the R code that does the work)

  Note: before modifications later in this exercise, this function read: add <- function(x,y){x + y}

```
add <- function(x, y=100, z=0){
  x + y + z
}
```

I can just use this function like any other function as long as R knows about it- which means I have to make sure to run the previous code chunk first:

```
add(1, 100)
```

```
[1] 101
```

```
add(x=c(1,2,3,4), y=100)
```

```
[1] 101 102 103 104
```

What if we only put one variable inside the function?

> In order to do this, we need to add a default to the original function. The original function has been modified to set a default of y=100.

```
add(1)
```

```
[1] 101
```

Functions can have "required" input arguments and "optional" input arguments. The optional arguments are defined with an equals default value (`y=100`) in the function definition.

> Here we have added another variable. The original function was modified again to set a default of z=0. Notice how despite this modification, the other code chunks still work!

```
add(x=1, y=100, z=10)
```

```
[1] 111
```

Using the sample() function

> Q. Write a function to return a DNA sequence of a user specified length. Call it `generate_dna()`

The `sample()` function can help here:

```
#generate_dna <- function(size=5){}

students <- c("jeff","jeremy","peter")

sample(students, size=5, replace=TRUE)
```

```
[1] "jeremy" "jeff"   "jeremy" "jeremy" "jeff"
```

Above, the "replace" argument was used to avoid the error of asking for more than the population size.

Section 2. Generate DNA sequence

Now work with **bases** rather than **students**

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

sample(bases, size=10, replace=TRUE)
```

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

Now that I have a working 'snippet' of code, I can use this as the body of my first function version here.

Below, I have changed the 'size' parameter so that it is not limited to only 10 characters. Now I can generate a 100 character long sequence (the size=5 portion of the code is the default, and make the default negligible by adding size=size):

```
generate_dna <- function(size=5) {
  bases <- c("A", "C", "G", "T")
  sample(bases, size=size, replace=TRUE)
}

generate_dna(100)
```

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

Using the `together=` logical

I want the ability to return a sequence like "AGTACCTG" - I want all the characters in one element vector where the bases are all together (not spaced out like above), so it can be pasted into BLAST or some other tool...

```
generate_dna <- function(size=5, together=TRUE) {
  bases <- c("A", "C", "G", "T")
  sequence <- sample(bases, size=size, replace=TRUE)

  if(together) {
    sequence <- paste(sequence, collapse="")
  }
  return(sequence)
}

generate_dna()
```

```
[1] "GAGGA"
```

The sequence above is generated in the format that we wanted. To undo this, we can use the "together" logical:

```
generate_dna(together=FALSE)
```

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

Section 3. Generate Protein Function

> Q. Write a function, `generate_protein()`, to return protein sequences of user determined length.

We can get the set of 20 natural amino acids from the **bio3d** package.

```
aa <- bio3d::aa.table$aa1[1:20]
```

The $aa1 only returns the one-letter code of the 20 natural AAs.

```r
generate_protein <- function(size=20, together=TRUE) {
  ## Get the 20 amino acids as a vector
   amino_acids <- c(aa)
  sequence <- sample(amino_acids, size=size, replace=TRUE)

  ## Optionally return a single element string
  if(together) {
    sequence <- paste(sequence, collapse="")
  }
  return(sequence)
}

generate_protein()
```

```
[1] "MNKPYPHFNHQAHLVDCLDS"
```

Q. Generate random protein sequences of length 6 to 12 amino acids

I found this solution using chatgpt:

```r
num_vars <- sample(6:12, size=1)

generate_protein <- function(size=num_vars, together=TRUE) {
  amino_acids <- c(aa)
  sequence <- sample(amino_acids, size=size, replace=TRUE)

  if(together) {
    sequence <- paste(sequence, collapse="")
  }
  return(sequence)
}

generate_protein()
```

```
[1] "QYQWCTGT"
```

The num_vars function selects a random integer between 6 and 12, then uses that integer as the size, returning one sequence at a time of random length btw 6-12 AAs.

In class, we used this solution (using sapply()):

We can fix the inability to generate multiple sequences by either editing and adding to the function body code (e.g. a for loop) or by using the R **apply** family of utility function.

```
## Using the sapply function, we set the first argument equal to a vector consisting of the

sapply(6:12, generate_protein)
```

```
[1] "CKKSVN"       "ADQLIAE"      "NILAIPWI"     "PWRIPTQHF"    "TFTVDHENNC"
[6] "LCKLMWESNWQ"  "VDAPKEMSVEHF"
```

It would be cool and useful if I could get FASTA format output.

```
ans <- sapply(6:12, generate_protein)
ans
```

```
[1] "HNEAAN"       "HNINRCP"      "HSGKCDFR"     "MCDVYHETE"    "IEIGWWKWWY"
[6] "ANLAFSSMDAW"  "VDQLADRFGATA"
```

```
cat(ans, sep="\n")
```

```
HNEAAN
HNINRCP
HSGKCDFR
MCDVYHETE
IEIGWWKWWY
ANLAFSSMDAW
VDQLADRFGATA
```

I want this to look like FASTA format with an ID line. The functions `paste()` and `cat()` can help us here...

```
id.line <- paste(">ID.", 6:12, sep="")
seq.line <- paste(id.line, ans, sep="\n")
cat(seq.line, sep="\n")
```

```
>ID.6
HNEAAN
>ID.7
HNINRCP
>ID.8
HSGKCDFR
>ID.9
MCDVYHETE
>ID.10
IEIGWWKWWY
>ID.11
ANLAFSSMDAW
>ID.12
VDQLADRFGATA
```

> Q. Determine if these sequences can be found in nature or are they unique? Why or why not?

Simply copy/paste the above results into protein BLAST. (the results used were from an iteration that resulted in the following data):

“>ID.6 RFIDFA” “>ID.7 TKRGLGF” “>ID.8 RSQLENYY” “>ID.9 THMMFWWQQ” “>ID.10 KENIMVNPQE” “>ID.11 CKKGKKMDNDL” “>ID.12 NLQALYNHWHAT”

I BLASTp searched my FASTA format sequences against refseq_protein, and found that lengths 6, 7, and 8 are not unique and can be found in the databases with 100% coverage and 100% identity. Random sequence lengths 9, 10, 11, and 12 are unique, and no matches had both 100% coverage and 100% identity within the database.