class06

Selma Cifric (PID A69042976)

Table of contents

All functions in R have at least 3 things: - A **name**, **arguments**, and **body**

#First silly function

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

now, call function

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

```
[1] 110 110
```

Second function

Write a function to generat erandom nucleotide sequences of a user specific length:

The `sample()` function can be helpful here. Replace within the function = TRUE vs FALSE

```
v <- sample(c("A", "C", "G", "T"), size=50, replace = TRUE)
```

I want a 1 element long character vector that looks like a string and not “a” “c” “t”.. but ACT. We use `paste` function where collapse=“ ” will pull the elements into a string. I could also do collapse=“–SC–” and it would use this as my spacings

```r
paste(v, collapse = "")
```

```
[1] "CTGAGCCTTATAATGAGTATTAACTATGCAACAAACATGGTTCCACTGTT"
```

Generate a function to combine both of codes above

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

```r
generate_dna(60)
```

```
[1] "ACGTAACCTAAGTTTCGCCACGGCAGAGAGGCTAGCGTGTACGCTTGGCATTAATTTGCC"
```

IF and ELSE functions

```r
fasta <- TRUE
if(fasta) {
  cat("HELLO You!")
} else {
  cat("No you dont!")
}
```

```
HELLO You!
```

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

```r
generate_fasta <- function(size=50, fasta=TRUE) {
  v <- sample(c("A", "C", "G", "T"), size=size, replace = TRUE)
  s <- (paste(v, collapse = ""))

  if(fasta) {
    return(s)
  } else {
    return(v)
  }
}
```

```
generate_fasta(100, FALSE)
```

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

A protein generating function

```
generate_protein <- function(size=50, fasta=TRUE) {
  v <- sample(c("A", "R", "N", "D", "C", "Q", "E", "G", "H", "I",
                "L", "K", "M", "F", "P", "S", "T", "W", "Y", "V"),
              size=size, replace=TRUE)
  s <- paste(v, collapse = "")

  if(fasta) {
    return(s)
  } else {
    return(v)
  }
}
```

```
generate_protein(6)
```

```
[1] "ELNSYI"
```

Use our new `generate_protein()` function to make random protein sequences of lenght 6-12 (i.e. one length 6, one length 7, etc up to length 12)

One way is to:

```
generate_protein(6)
```

```
[1] "AREYQE"
```

```
generate_protein(7)
```

```
[1] "MEKEDEW"
```

```
generate_protein(8)
```

```
[1] "DNLREVTN"
```

```
generate_protein(9)
```

```
[1] "LMHTFYIIL"
```

OR use FOR loop `for()` loop:

```
lengths <-  6:12
lengths
```

```
[1]  6  7  8  9 10 11 12
```

```
for(i in lengths) {
  cat(">", i, " ")
  aa <- generate_protein(i)
  cat(aa)
  cat("\n")
}
```

```
> 6  VDQPDK
> 7  HYGDHPR
> 8  ICQKNRTF
> 9  HTGRYCPYG
> 10  FSVLCKPHCY
> 11  LACWWPVHGTI
> 12  IGPPGMRHTNSG
```

```
generate_protein()
```

```
[1] "TMAKTWQRIKFQWYQCVVEQPWYVVRMGHVRGTLRPSKAKIGGSGQDPVW"
```

A third, and better, way to solve this is to use the `apply()` family of functions, specifically the `sapply()` function in this case.

```
help(sapply)

sapply(6:12, generate_protein)
```

```
[1] "FKYQHR"       "RATNDKY"      "TPEFTCND"     "KDGLSMNTC"    "AWMLEKATVL"
[6] "KPPIHTDMPPK"  "LHHLQDWLADIC"
```