Class 6: R functions

Courtney Anderson

Table of contents

All functions in R have at least 3 things -A **name** we pick this an use it to call our function -Input **arguments** (there can be multiple) -The **body** lines of R code that do the work

Our first (silly) function

Write a function to add some numbers

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

Now we can call this funcion

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

```
[1] 110 110
```

A second function

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

The `sample` function can be helpful here

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

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

I want a 1 element long character vector that looks like this "CACGC" not "C" "A" "C" "A" "G" "C"

```
generate_dna <- function(size=50) {
v <- sample(c("A", "C","G","T"), size= size, replace= TRUE)
paste(v,collapse="")
}
```

Test it:

```
generate_dna(60)
```

```
[1] "CGTGAGAGCGTGCGTCCCACGAGTACTAGGCCTCAAGAAGGCACCTTAATGCTATTGCCC"
```

=

```
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.

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

```
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(60)
```

```
[1] "TCGAGTCGCGTCCGATGAGGGTACATGTCATGTTGTGAAGGCTATGTCCCGCCCACCCCT"
```

```
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(fasta=FALSE)
```

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

```
generate_fasta(fasta=TRUE)
```

```
[1] "CCTTTGAAGTATGGACCCGGGACGGATTACGAGCGATAACAGTGAGAAGC"
```

A protein generating function

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

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

```
generate_protein(6)
```

```
[1] "RWKTTG"
```

Use our new`generate protein` function to make random protein seqeunces of length 6 to 12 (i.e. one length 6, on length 7, etc. up to length 12).

One way to do this is brute force

```
generate_protein(6)
```

```
[1] "IRDWRG"
```

```
generate_protein(7)
```

```
[1] "KVNAYYP"
```

```
generate_protein(8)
```

```
[1] "PCDQQHFP"
```

```
generate_protein(9)
```

```
[1] "TELMTRSMG"
```

A second way is to use `for()` loop

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

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

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

```
>6
GTPCQS
>7
WHEFGQQ
>8
HWCSNTHT
>9
QCDESYKCT
>10
WDSHRIKQRH
>11
INKACCLIPSY
>12
GNIQPWANEESI
```

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

```
sapply(6:12, generate_protein)
```

```
[1] "GYMNNK"       "EFSHKYY"      "RKELCMRP"     "LENCDFSYE"    "MCEMDQYPPA"
[6] "HFRNFLPTFCL"  "FTFAGKWNAFAH"
```