class06

Joshua Martin (PID: A18545389)

All functions in R have at least 3 things: - A **name**, we pick this and use it to call the function. - Input **arguments**, there can be multiple comma separated inputs to the function. - The **body**, lines of R code that do the work of the function.

Our first wee function:

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

Let's test our function

```
add(c(1,2,3), y=10)
```

```
[1] 11 12 13
```

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

```
[1] 110
```

A second function

Let's try something more interesting. Make a sequence generation tool.

The 'sample()' function could be useful here.

```
sample(1:10, size = 3)
```

```
[1] 2 4 6
```

Change this to work with the nucleotides A C G and T and return 3 of them

```
n <- c("A","C","G","T")
sample(n, size=15, replace = TRUE)
```

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

Turn this snippet into a function that returns a user specified length dna sequence. Let's call it 'generate_dna()'...

```
generate_dna <- function(len=10) {
  n <- c("A","C","G","T")
  v <- sample(n, size=len, replace = TRUE)
  cat("Well done you!")
  return(v)
}
```

```
generate_dna(5)
```

```
Well done you!
```

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

```
s <- generate_dna(15)
```

```
Well done you!
```

```
s
```

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

I want the option to return a single element character vector with my sequence all together like this: "GGAGTAC"

```r
lookatme <- function(len = 10, as_string = FALSE) {
  n <- c("A", "C", "G", "T")
  v <- sample(n, size = len, replace = TRUE)
  cat("Well done you!\n")

  if (as_string) {
    # Return a single character string, e.g., "GGAGTAC"
    paste(v, collapse = "")
  } else {
    # Return a character vector of individual bases
    v
  }
}
```

```r
lookatme(5)
```

```
Well done you!
```

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

```r
lookatme(7, as_string = TRUE)
```

```
Well done you!
```

```
[1] "TAGAATG"
```

Make a third function that generates protein sequence of a user specified length and format.

```r
generate_protein <- function(len = 15, fasta = TRUE) {
  # 20 standard amino-acid single-letter codes
  aa <- c(
    "A","R","N","D","C","E","Q","G","H","I",
    "L","K","M","F","P","S","T","W","Y","V"
  )

  # Randomly sample with replacement
  seq_vec <- sample(aa, size = len, replace = TRUE)
  cat("Protein sequence generated!\n")
```

```
  if (fasta) {
    # Return a single character string, e.g., "MKTFLV..."
    paste(seq_vec, collapse = "")
  } else {
    # Return a character vector of individual residues
    seq_vec
  }
}
```

```
generate_protein(10)
```

```
Protein sequence generated!

[1] "IFNWEVCTAW"
```

Q. Generate random protein sequences between length 5 and 12 amino-acids.

```
generate_protein(5)
```

```
Protein sequence generated!

[1] "HMDKC"
```

```
generate_protein(6)
```

```
Protein sequence generated!

[1] "GKRTII"
```

Once approach is to do this by brute force calling our function for each length 5 to 12.

Another approach is to write a 'for()' loop to itterate over the input valued 5 to 12

A very useful third R specific approach is to use the 'sapply()' function

```
seq_lengths <- 6:12
for (i in seq_lengths) {
  cat(i, "\n")
  cat( generate_protein(i))
  cat("\n")
}
```

```
6
Protein sequence generated!
VVRDLD
7
Protein sequence generated!
AWSFRTQ
8
Protein sequence generated!
NEMREVED
9
Protein sequence generated!
VGMIATQMR
10
Protein sequence generated!
CGPFPRQTLR
11
Protein sequence generated!
FYLQQAEHDTT
12
Protein sequence generated!
RFFEWASNDVTN
```

```
protein_sapply <- sapply(5:12, generate_protein)
```

```
Protein sequence generated!
Protein sequence generated!
Protein sequence generated!
Protein sequence generated!
Protein sequence generated!
Protein sequence generated!
Protein sequence generated!
Protein sequence generated!
```

```
protein_sapply
```

```
[1] "AWWQN"        "HSSYMR"       "LYRKCNS"      "TTAMDCAF"     "GVRIPFFFA"
[6] "DQMQVSNRTF"   "KPQGWHKSIYE"  "DMGHYNECMKNP"
```

> **Key-Point**: Writing functions in R is doable but not the easiest thing in the world. Starting with a working snippet of code and then using LLM tools to improve and generalize your function code is a productive approach.