Class6

Hanhee Jo

Today we are going to get more exposure to funtions in R.

Let's start with a silly simple function to add some numbers:

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

}
```

Can we use this function?

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

```
[1] 2
```

```
add(c(100, 200), 1)
```

```
[1] 101 201
```

```
log(10, base=10)
```

```
[1] 1
```

```
add(100, 1, 200)
```

```
[1] 301
```

A more interesting example

Let's have a look at the `sample()` function.

> Q. What does it do?

the `sample()` function is used to randomly select elements from a vector or generate random permutations. It has several key uses:

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

```
[1] 10
```

What if I want 11 things taken from my vector 1 to 10?

```
sample(1:10, size=11, replace=T)
```

```
 [1]  8  6  6 10  9  6  6  3  5  7  7
```

Key parameters

-x: Vector to sample from - size: Number of items to sample - replace: Whether sampling can repeat elements - prob: Optional vector of sampling probabilities

Generate DNA sequences

> Q. Write a function to generate a random nucleotide sequence of a user specified size/length.

```
x <- c("A", "G", "T", "C")
sample(x, size=9, replace=T)
```

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

All functions in R have at least 3 things:

- a **name** (we pick this "generate_dna")
- input **arguments**("length" of the output sequence)
- the **body** (where the work gets done, line by line)

```
generate_dna <- function(length=10) {
  x <- c ("A", "C", "T", "G")
  ans <- sample(x, size=length, replace=T)
  return(ans)
}
```

It's a good practice to use `return()` function.

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

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

```
generate_dna(400)
```

```
  [1] "A" "C" "T" "T" "G" "A" "G" "A" "C" "G" "G" "G" "C" "T" "G" "A" "G" "T"
 [19] "T" "G" "A" "C" "A" "G" "C" "A" "C" "G" "T" "C" "G" "A" "T" "G" "A" "C"
 [37] "C" "A" "G" "G" "G" "G" "A" "A" "C" "T" "A" "T" "A" "A" "C" "C" "T" "T"
 [55] "C" "A" "T" "C" "T" "C" "A" "A" "T" "T" "T" "A" "C" "T" "G" "G" "A" "A"
 [73] "A" "G" "C" "T" "C" "A" "T" "C" "T" "C" "C" "C" "A" "G" "A" "T" "A" "T"
 [91] "G" "T" "A" "T" "A" "C" "A" "C" "A" "G" "A" "A" "G" "G" "G" "G" "C" "G"
[109] "C" "T" "T" "A" "T" "T" "T" "C" "C" "T" "C" "C" "G" "T" "T" "T" "T" "G"
[127] "T" "T" "A" "A" "T" "C" "G" "G" "A" "A" "C" "A" "A" "A" "G" "A" "T" "G"
[145] "G" "T" "G" "T" "G" "T" "C" "T" "A" "A" "C" "C" "C" "G" "A" "T" "T" "G"
[163] "C" "C" "T" "T" "C" "G" "C" "A" "T" "A" "C" "C" "A" "A" "T" "C" "C" "C"
[181] "G" "G" "A" "C" "T" "A" "G" "G" "C" "T" "A" "G" "T" "G" "A" "G" "T" "T"
[199] "C" "T" "G" "A" "C" "A" "A" "C" "C" "A" "T" "A" "C" "T" "C" "A" "A" "C"
[217] "A" "A" "T" "A" "C" "G" "A" "A" "G" "G" "T" "A" "G" "G" "T" "G" "A" "A"
[235] "C" "G" "A" "A" "G" "C" "A" "T" "A" "C" "G" "G" "C" "T" "T" "G" "G" "C"
[253] "T" "A" "A" "G" "C" "A" "C" "G" "A" "C" "A" "A" "A" "A" "G" "A" "A" "T"
[271] "C" "G" "A" "G" "A" "C" "T" "C" "G" "G" "T" "A" "T" "T" "A" "T" "C" "T"
[289] "C" "C" "A" "T" "G" "A" "C" "T" "T" "G" "T" "G" "T" "T" "G" "T" "G" "T"
[307] "A" "T" "G" "T" "A" "T" "A" "A" "C" "G" "T" "C" "A" "C" "A" "A" "A" "A"
[325] "C" "G" "G" "T" "G" "T" "G" "G" "G" "G" "A" "G" "G" "G" "C" "A" "A" "G"
[343] "A" "C" "A" "G" "G" "G" "G" "A" "A" "G" "C" "G" "G" "C" "A" "A" "G" "G"
[361] "T" "G" "C" "C" "T" "C" "C" "T" "C" "A" "C" "T" "T" "T" "A" "G" "G" "A"
[379] "T" "T" "A" "G" "A" "A" "G" "C" "C" "C" "A" "A" "C" "T" "A" "A" "C" "C"
[397] "C" "T" "C" "G"
```

I would like my function to print out a single element vector "GATGATCT". To help with this I can maybe use the `paste()` function.

```
  paste(s, collapse ="")
```

```
[1] "GCTTGTAGAG"
```

```
  generate_dna <- function(length=10) {
    # The nucleotides to draw/sample from
    x <- c ("A", "C", "T", "G")
    # Draw n=length nucleotides to make our sequence
    ans <- sample(x, size=length, replace=T)
    # Concatenate/join/past drwurnvr into one word
    ans <- paste(ans, collapse ="")
    return(ans)
  }
```

```
  generate_dna()
```

```
[1] "TCTGCGACTC"
```

I want the ability to switch between these two output formats. I can do this with an extra input argument to my function that controls this with TRUE/FALSE.

```
  generate_dna <- function(length=10, collapse = FALSE) {
    # The nucleotides to draw/sample from
    x <- c ("A", "C", "T", "G")
    # Draw n=length nucleotides to make our sequence
    ans <- sample(x, size=length, replace=T)

    # Concatenate/join/past drwurnvr into one word
    if(collapse) {
      ans <- paste(ans, collapse ="")
    }
    return(ans)
  }
```

```
  generate_dna(length = 5, collapse = FALSE)
```

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

> Q. Add the ability to print a wee message if the user is sad. Control this with a new input prarmeter called `mood`.

```
generate_dna <- function(length=10, collapse = FALSE, mood=FALSE) {
  # The nucleotides to draw/sample from
  x <- c ("A", "C", "T", "G")
  # Draw n=length nucleotides to make our sequence
  ans <- sample(x, size=length, replace=T)

  # Concatenate/join/past drwurnvr into one word
  if(collapse) {
    ans <- paste(ans, collapse ="")
  }

  if(mood) {
    cat("Hello")
  }
  return(ans)
}
```

```
generate_dna(4, mood=TRUE)
```

```
Hello
```

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

Q. Write a protein sequence generating function with the ability to output random amino acid sequences of a user defined length.

```
aa <- c("A", "R", "N", "D", "C", "E", "Q", "G", "H", "I", "L", "K", "M", "F", "P", "S", "T

length(aa)
```

```
[1] 20
```

```
generate_protein <- function(length=10, collapse=FALSE) {
  aa <- c("A", "R", "N", "D", "C", "E", "Q", "G", "H", "I", "L", "K", "M", "F", "P", "S",
  ans <- sample(aa, size=length, replace=T)
  if(collapse) {
    ans <- paste(ans, collapse ="")
  }
  return(ans)
```

```
}
```

```
generate_protein()
```

```
 [1] "F" "N" "E" "I" "W" "V" "R" "L" "P" "F"
```

Q. Generate protein sequences from length 6 to 12 amino acids long.

```
>generate_protein(6:12, TRUE)
```

This does not work because my functionis not vectorized (in other words, setup to work on each element of the first input argument `length`).

In particular, we can use `sapply()`.

The `sapply()` function applies a function to each element of a vector/list and simplifies the output.

```
protein_sequences <- sapply(6:12, generate_protein)
print(protein_sequences)
```

```
[[1]]
[1] "V" "C" "V" "P" "G" "P"

[[2]]
[1] "E" "A" "H" "P" "Q" "V" "V"

[[3]]
[1] "N" "T" "F" "L" "K" "P" "Q" "W"

[[4]]
[1] "T" "M" "E" "Q" "E" "F" "K" "F" "N"

[[5]]
 [1] "Q" "K" "M" "L" "A" "R" "H" "A" "I" "M"

[[6]]
 [1] "N" "C" "W" "V" "L" "C" "I" "A" "R" "F" "P"

[[7]]
 [1] "D" "Y" "H" "W" "G" "Y" "W" "W" "I" "I" "G" "F"
```

Q. Are any of these suquences unique in the sense that they have never been found in nature?

To make this accessible, let's get our sequences in FASTA format.

FASTA format looks like...

id.1 PCSFIM id.2 NKLPAMG

```
myseqs <- sapply(6:12, generate_protein, collapse=T)
myseqs
```

```
[1] "LWEQEC"       "QNEQRWK"      "KPKWPFHD"     "RLGQYSIPY"    "WMCDSALRKR"
[6] "VIDAAFNTYTY"  "ISCVVSKYRSEH"
```

The functions `paste()` cna `cat()` will help here

```
cat( paste(">id.", 6:12,"\n", myseqs, "\n", sep = ""), sep = "" )
```

```
>id.6
LWEQEC
>id.7
QNEQRWK
>id.8
KPKWPFHD
>id.9
RLGQYSIPY
>id.10
WMCDSALRKR
>id.11
VIDAAFNTYTY
>id.12
ISCVVSKYRSEH
```