Class 06: R Functions

Matthew Chan (PID: A18130675)

Background

Functions are at the heart of using R. Everything we do involves calling an dusing functions (from data input, analysis to result output)

All functions in R have at least 3 things:

1. A **name** the thing we use to call then function
2. One or more input **arguments**
3. The **body**, lines of code between curly brackets { } that does the work of the function.

A first function

Let's write a silly function to add numbers

```r
add <- function (x) {
  x + 1

}
```

Lets try it out

```r
add(100)
```

```
[1] 101
```

Will this work

```r
add (c(100,200,300))
```

```
[1] 101 201 301
```

Modify to be more useful and add more than just 1

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

}
```

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

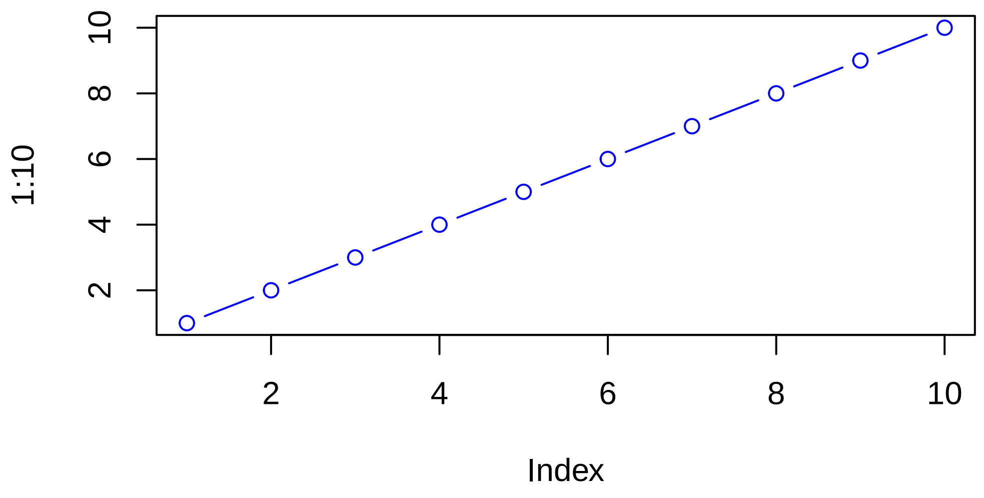
```
[1] 110
```

Will this work?

```
add(100,)
```

```
[1] 101
```

```
plot (1:10, col="blue", typ="b")
```



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

```
[1] 1
```

> **Note** Input arguments can either be **required** or **optional**, optional is specified with an equal sign as a default option

```r
#add(x=100, y=200, z=300)
```

A second function

All functions in R look like this

```
name <- function(argument) {
  body
}
```

The `sample()` function in R…

```r
sample(1:10, size=4)
```

```
[1] 3 2 4 6
```

> Q. Return 12 numbers picked randomly from the input 1:10

```r
sample(1:10, size=12, replace=TRUE)
```

```
 [1] 10  8  9  1  1  9  1  3  8  9  8  3
```

> Q. Write the code to generate a 12 nucleotide long DNA sequence

```r
bases <- c("A","G","C","T")
DNAseq <- sample(bases, size=12, replace=T)
  paste(DNAseq, collapse="")
```

```
[1] "GTTCGTACGCGC"
```

> Q. Write a first version function called `generate_dna()` that generates a user specified length 'n' random DNA sequence?

```r
generate_dna <- function(x=100) {
  bases <- c("A","G","C","T")
  DNAseq <- sample(bases, size=x, replace=T)
  ans <- paste(DNAseq, collapse="")
  return(ans)
}
```

```r
generate_dna(200)
```

```
[1] "CCGGGGAAGCCCGGTCGGCCTCCGGTGCTATTTCCTAGTAATATCTCTCGTTATTGCGCTTCGACTATGATGTAGTATGGGAGAGTGC
```

```r
x<- c("H", "E", "L", "L", "O")
paste(x, collapse="hello")
```

```
[1] "HhelloEhelloLhelloLhelloO"
```

> Q. Give the user an option to return FASTA format output sequence or standard multi-element vector format

```r
generate_dna <- function(x=100, fasta=T) {
  bases <- c("A","G","C","T")
  DNAseq <- sample(bases, size=x, replace=T)

  if(fasta) {
    DNAseq <- paste(DNAseq, collapse="")
      cat("Bye")
  } else {
      cat("Hello")
    }
return(DNAseq)
}
```

```r
generate_dna(5, fasta=FALSE)
```

```
Hello
```

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

A new cool function

> Q. Write a function called `generate_protein` that generates a user specified length protein sequence in FASTA like format?

```
generate_protein <- function(x=6) {
  aminoacids <- c("A", "R", "N", "D", "C", "Q", "E", "G", "H", "I", "L", "K", "M", "F", "P",


  protein <- sample(aminoacids, size = x, replace = T)
  protein <- paste(protein, collapse="")
  return(protein)
}
```

> Q. Use your new `generate_protein` function to generate all sequences between length 6 and 12 amino-acids in length and check if any of these are unique in nature (ie found in the NR database at NCBI)?

This is the wrong code but randomly generates protein sequences between the lengths of 6 and 12 amino acids

```
generate_protein <- function(x=6) {
  aminoacids <- c("A", "R", "N", "D", "C", "Q", "E", "G", "H", "I", "L", "K", "M", "F", "P",
  length <- sample(6:12,size = 1, replace=T)

  protein <- sample(aminoacids, size = length, replace = T)
  protein <- paste(protein, collapse="")

  return(protein)
}
```

```
generate_protein()
```

```
[1] "WNELWNW"
```

VEGWCAEC matches with Accession Number NJD86855.1

```
generate_protein(6)
```

```
[1] "WGCYELCTWVQM"
```

```r
generate_protein(7)
```

```
[1] "WWFCMIGCWYVQ"
```

```r
generate_protein(8)
```

```
[1] "RCFFCYK"
```

```r
generate_protein(9)
```

```
[1] "FWLVWGA"
```

```r
generate_protein(10)
```

```
[1] "QKTTDCKQKFMM"
```

```r
generate_protein(11)
```

```
[1] "YYRWFK"
```

```r
generate_protein(12)
```

```
[1] "MGCSHDCV"
```

Or we could fo a `for()` loop:

```r
for (i in 6:12) {
  cat(">", sep="", i, "\n")

  cat(generate_protein(i), "\n" )
}
```

```
>6
AHTDNDYYLI
>7
PHGRFKSDS
>8
HPINKF
>9
SYLCKYDDNRYA
>10
IYVPNKWSGLRP
>11
LNWHMIW
>12
RFEPCFQGH
```