Act#2

Cyrce

April 16, 2023

Bruno Vázquez Espinoza - A00838521

Cyrce Danae Salinas Rojas - A01666121

#1. Escribe una función que genere una secuencia aleatoria de DNA de tamaño "n". Probar con n = 50 y guardarla en una variable.

```
sequenceDNA <-function(size){
  x<- c("A", "T", "G", "C")
  sample(x, size, replace = TRUE )
}
DNA <-paste0(sequenceDNA(90))
DNA_89<-paste0(sequenceDNA(89))
c("DNA:",  DNA, "DNA of 89:" , DNA_89)
```

```
##   [1] "DNA:"       "C"          "C"          "G"          "C"
##   [6] "T"          "A"          "C"          "A"          "A"
##  [11] "A"          "G"          "T"          "A"          "C"
##  [16] "A"          "G"          "G"          "A"          "A"
##  [21] "G"          "A"          "C"          "A"          "C"
##  [26] "A"          "G"          "C"          "G"          "G"
##  [31] "T"          "C"          "A"          "C"          "T"
##  [36] "G"          "G"          "C"          "G"          "G"
##  [41] "T"          "A"          "A"          "G"          "C"
##  [46] "C"          "A"          "C"          "A"          "C"
##  [51] "T"          "G"          "G"          "A"          "A"
##  [56] "C"          "T"          "A"          "G"          "C"
##  [61] "T"          "T"          "C"          "T"          "A"
##  [66] "T"          "T"          "A"          "T"          "G"
##  [71] "A"          "T"          "G"          "C"          "T"
##  [76] "G"          "A"          "A"          "G"          "T"
##  [81] "G"          "T"          "T"          "C"          "T"
##  [86] "C"          "C"          "T"          "T"          "A"
##  [91] "T"          "DNA of 89:" "T"          "C"          "A"
##  [96] "T"          "G"          "T"          "T"          "T"
## [101] "T"          "T"          "C"          "T"          "A"
## [106] "C"          "T"          "G"          "T"          "G"
## [111] "C"          "T"          "A"          "G"          "C"
## [116] "A"          "T"          "T"          "G"          "A"
## [121] "G"          "C"          "A"          "T"          "G"
## [126] "T"          "C"          "A"          "G"          "G"
## [131] "C"          "G"          "A"          "C"          "G"
```

```
## [136] "T"          "C"          "T"          "G"          "T"
## [141] "T"          "T"          "C"          "G"          "A"
## [146] "T"          "A"          "G"          "G"          "A"
## [151] "G"          "T"          "G"          "G"          "G"
## [156] "A"          "G"          "G"          "T"          "A"
## [161] "T"          "G"          "C"          "A"          "A"
## [166] "T"          "T"          "T"          "G"          "A"
## [171] "A"          "T"          "G"          "G"          "C"
## [176] "C"          "A"          "A"          "T"          "C"
## [181] "T"
```

2.Codifica una función que calcule el tamaño de una secuencia de DNA. Probar la función con la variable del ejercicio 1

```
sizeDNA <- function(sequence) {
  length(sequence)
}
sizeA <- sizeDNA(DNA)

sizeA
```

```
## [1] 90
```

3.Crea una función que reciba una secuencia de DNA e imprima el porcentaje de cada base en la secuencia. Genera una nueva variable del ejercicio 1 con n = 90 para probar la función.

```
percentDNA<- function(sequence){
countA <- 0
countT <- 0
countG <- 0
countC <- 0
  for (i in sequence){
    if (i == "A"){
      countA <- countA +1
    }
    if( i == "T"){
      countT <- countT +1
    }
    if( i == "G"){
      countG <- countG +1
    }
    if (i =="C"){
      countC <- countC +1
    }
  }
  cat("A:", countA, "T:", countT, "G:", countG, "C:", countC)
}
percents <- percentDNA(DNA)
```

```
## A: 25 T: 22 G: 21 C: 22
```

4. Programa una función que transcriba de DNA a RNA (sólo cambiar las Timinas por Uracilos). Usa la variable del ejercicio 3 para probar la función

```
DNAtoRNA<-function(sequence){
  RNA <- gsub("T", "U", sequence)
  return(RNA)
}

RNA <- DNAtoRNA(DNA)
RNA
```

```
##  [1] "C" "C" "G" "C" "U" "A" "C" "A" "A" "A" "G" "U" "A" "C" "A" "G" "G" "A" "A"
## [20] "G" "A" "C" "A" "C" "A" "G" "C" "G" "G" "U" "C" "A" "C" "U" "G" "G" "C" "G"
## [39] "G" "U" "A" "A" "G" "C" "C" "A" "C" "A" "C" "U" "G" "G" "A" "A" "C" "U" "A"
## [58] "G" "C" "U" "U" "C" "U" "A" "U" "U" "A" "U" "G" "A" "U" "G" "C" "U" "G" "A"
## [77] "A" "G" "U" "G" "U" "U" "C" "U" "C" "C" "U" "U" "A" "U"
```

5. Crea una función que traduzca una secuencia de RNA a una secuencia de proteínas usando la siguiente tabla. Prueba el ejercicio con dos variables: la variable del ejercicio 4 y una nueva variable del ejercicio 1 con n = 89 para probar una salida que no sea múltiplo de 3. 6. [10 pts] Para representar una molécula de ADN de doble hebra basta con escribir la secuencia de una de sus hebras. Consideremos, por ejemplo, la secuencia TGCGATAC. Como no se indica lo contrario, se asume que la secuencia está escrita en sentido 5'→3' y, por lo tanto, se trata de la hebra directa (forward strand):

```
translateRNAtoProtein <- function(sequence) {
  codon_table <- c(
    "UUU" = "Phe", "UUC" = "Phe", "UUA" = "Leu", "UUG" = "Leu",
    "CUU" = "Leu", "CUC" = "Leu", "CUA" = "Leu", "CUG" = "Leu",
    "AUU" = "Ile", "AUC" = "Ile", "AUA" = "Ile", "AUG" = "Met",
    "GUU" = "Val", "GUC" = "Val", "GUA" = "Val", "GUG" = "Val",
    "UCU" = "Ser", "UCC" = "Ser", "UCA" = "Ser", "UCG" = "Ser",
    "CCU" = "Pro", "CCC" = "Pro", "CCA" = "Pro", "CCG" = "Pro",
    "ACU" = "Thr", "ACC" = "Thr", "ACA" = "Thr", "ACG" = "Thr",
    "GCU" = "Ala", "GCC" = "Ala", "GCA" = "Ala", "GCG" = "Ala",
    "UAU" = "Tyr", "UAC" = "Tyr", "UAA" = "Stop", "UAG" = "Stop",
    "CAU" = "His", "CAC" = "His", "CAA" = "Gln", "CAG" = "Gln",
    "AAU" = "Asn", "AAC" = "Asn", "AAA" = "Lys", "AAG" = "Lys",
    "GAU" = "Asp", "GAC" = "Asp", "GAA" = "Glu", "GAG" = "Glu",
    "UGU" = "Cys", "UGC" = "Cys", "UGA" = "Stop", "UGG" = "Trp",
    "CGU" = "Arg", "CGC" = "Arg", "CGA" = "Arg", "CGG" = "Arg",
    "AGU" = "Ser", "AGC" = "Ser", "AGA" = "Arg", "AGG" = "Arg",
    "GGU" = "Gly", "GGC" = "Gly", "GGA" = "Gly", "GGG" = "Gly"
  )
```

```
  protein <- character()
  for (i in seq(1, length(sequence), by = 3)) {
    codon <- paste(sequence[i:(i + 2)], collapse = "")
    if (codon %in% names(codon_table)) {
      protein <- c(protein, codon_table[codon])
    } else {
      protein <- c(protein, "Invalid codon")
    }
  }

  return(protein)
}

Prot<-translateRNAtoProtein(RNA)
Prot89<-translateRNAtoProtein(DNA_89)
c("caso con RNA: ", Prot, "caso con DNA de 89:", Prot89)
```

```
##                                       CCG                   CUA
##      "caso con RNA: "                "Pro"                 "Leu"
##                   CAA                   AGU                   ACA
##                 "Gln"                 "Ser"                 "Thr"
##                   GGA                   AGA                   CAC
##                 "Gly"                 "Arg"                 "His"
##                   AGC                   GGU                   CAC
##                 "Ser"                 "Gly"                 "His"
##                   UGG                   CGG                   UAA
##                 "Trp"                 "Arg"                "Stop"
##                   GCC                   ACA                   CUG
##                 "Ala"                 "Thr"                 "Leu"
##                   GAA                   CUA                   GCU
##                 "Glu"                 "Leu"                 "Ala"
##                   UCU                   AUU                   AUG
##                 "Ser"                 "Ile"                 "Met"
##                   AUG                   CUG                   AAG
##                 "Met"                 "Leu"                 "Lys"
##                   UGU                   UCU                   CCU
##                 "Cys"                 "Ser"                 "Pro"
##                   UAU
##                 "Tyr" "caso con DNA de 89:"       "Invalid codon"
##
##       "Invalid codon"       "Invalid codon"       "Invalid codon"
##
##       "Invalid codon"       "Invalid codon"       "Invalid codon"
##                   GCA                                         AGC
##                 "Ala"       "Invalid codon"                 "Ser"
##                                                             GGC
##       "Invalid codon"       "Invalid codon"                 "Gly"
##                   GAC
##                 "Asp"       "Invalid codon"       "Invalid codon"
##                                                             AGG
##       "Invalid codon"       "Invalid codon"                 "Arg"
##                                       GGG                   AGG
##       "Invalid codon"                 "Gly"                 "Arg"
##                                       GCA
```

```
##       "Invalid codon"                 "Ala"       "Invalid codon"
##                                                               GCC
##       "Invalid codon"       "Invalid codon"                 "Ala"
##
##       "Invalid codon"       "Invalid codon"
```

6. [10 pts] Para representar una molécula de ADN de doble hebra basta con escribir la secuencia de una de sus hebras. Consideremos, por ejemplo, la secuencia TGCGATAC. Como no se indica lo contrario, se asume que la secuencia está escrita en sentido 5'→3' y, por lo tanto, se trata de la hebra directa (forward strand): Hebra directa: 5'-TGCGATAC-3' Análogamente, si decido escribir esta misma secuencia empezando por el extremo 3' se obtiene la hebra inversa (reverse strand): Hebra inversa: 3'-CATAGCGT-5' Escribe una función que reciba una hebra directa y regrese la hebra inversa. Utiliza cualquiera de las 3 variables generadas anteriormente. Se deberán imprimir ambas hebras; no hay necesidad de especificar en el ejercicio los extremos 5' y 3', pero sí es necesario especificar cuál es la hebra directa y cuál la hebra inversa

```
reverse_strand <- function(direct_strand) {
    reverse_strand <- strsplit(direct_strand, "")[[1]]
    reverse_strand <- paste(reverse_strand[length(reverse_strand):1], collapse = "")
    return(reverse_strand)
}

direct_strand <- "TGCGATAC"
reverse_strand <- reverse_strand(direct_strand)

c("Direct strand:", direct_strand, "Reverse strand:", reverse_strand)
```

```
## [1] "Direct strand:"  "TGCGATAC"        "Reverse strand:" "CATAGCGT"
```

7. [10 pts] Normalmente se representa la molécula escribiendo primero la hebra directa y debajo la hebra complementaria (complementary strand). La hebra complementaria se escribe en sentido 3'→5' para que las bases de ambas hebras queden emparejadas: Hebra directa: 5'-TGCGATAC-3' Hebra complementaria: 3'-ACGCTATG-5'. Escribe una función qué reciba una hebra directa y obtenga la hebra complementaria. Utiliza cualquiera de las 3 variables generadas anteriormente. Se deberán imprimir ambas hebras; no hay necesidad de especificar en el ejercicio los extremos 5' y 3', pero sí es necesario especificar cuál es la hebra directa y cuál la hebra complementaria.

```
complementary_strand <- function(direct_strand) {
    complementary_strand <- gsub("T", "a", direct_strand)
    complementary_strand <- gsub("G", "c", complementary_strand)
    complementary_strand <- gsub("C", "g", complementary_strand)
    complementary_strand <- gsub("A", "t", complementary_strand)
    return(complementary_strand)
}

direct_strand <- "TGCGATAC"
complementary_strand <- complementary_strand(direct_strand)
complementary_strand <- toupper(complementary_strand)

print(paste("Direct strand:", direct_strand))
```

```
## [1] "Direct strand: TGCGATAC"
```

```
print(paste("Complementary strand:", complementary_strand))
```

```
## [1] "Complementary strand: ACGCTATG"
```

8. [10 pts] Si escribimos la secuencia de la hebra complementaria en sentido inverso (5'→3') se obtiene la complementaria inversa (reverse-complement): Hebra directa: 5'-TGCGATAC-3' Hebra complementaria inversa: 5'-GTATCGCA-3' Escribe la función en R para obtener la hebra complementaria inversa, desde una hebra directa. Utiliza cualquiera de las 3 variables generadas anteriormente. Se deberán imprimir ambas hebras; no hay necesidad de especificar en el ejercicio los extremos 5' y 3', pero sí es necesario especificar cuál es la hebra directa y cuál la hebra complementaria inversa.

```
reverse_complement_strand <- function(direct_strand) {
    reverse_strand <- strsplit(direct_strand, "")[[1]]
```

```
    reverse_strand <- paste(reverse_strand[length(reverse_strand):1], collapse = "")
    complementary_strand <- gsub("T", "a", reverse_strand)
    complementary_strand <- gsub("G", "c", complementary_strand)
    complementary_strand <- gsub("C", "g", complementary_strand)
    complementary_strand <- gsub("A", "t", complementary_strand)
    reverse_complement_strand <- toupper(complementary_strand)
    return(reverse_complement_strand)
}

direct_strand <- "TGCGATAC"
reverse_complement_strand <- reverse_complement_strand(direct_strand)

print(paste("Direct strand:", direct_strand))
```

```
## [1] "Direct strand: TGCGATAC"
```

```
print(paste("Reverse complement strand:", reverse_complement_strand))
```

```
## [1] "Reverse complement strand: GTATCGCA"
```