INTRODUCCIÓN A LA GEOCOMPUTACIÓN CON R
EJERCICIOS NO. 7 (WHAT FOR()?)

INDER TECUAPETLA-GÓMEZ

1. Escribe un ciclo *for* que imprima en consola el cubo de los números $1, 2, \ldots, 10$. **Hint:** `?cat`

2. Escribe una función que solicite al usuario ingresar un número natural positivo, y luego imprima en consola el cubo de cada número desde el 1 hasta el número ingresado. **Hint1:** considera el ejercicio anterior así como también el 3 de los Ejercicios No. 6. **Hint2:** posible solución

```
> getCubesTill()
---Calcula el cubo del 1 hasta el numero ingresado---
---El numero ingresado debe ser positivo---
Ingresa un numero: 5
El cubo de  1 es:  1
El cubo de  2 es:  8
El cubo de  3 es:  27
El cubo de  4 es:  64
El cubo de  5 es:  125
```

3. Sin codificarla, ¿puedes deducir cuál es el *output* de la siguiente función?

```
agregar_a_vector <- function(n) {
  output <- c()
  for (i in seq_len(n)) {
    output <- c(output, i)
  }
  output
}
```

4. Sin codificarla, ¿puedes deducir cuál es el *output* de la siguiente función?

```
agregar_a_vector_2 <- function(n) {
  output <- vector("integer", n)
  for (i in seq_len(n)) {
    output[[i]] <- i
  }
  output
}
```

5. Después de cargar a tu sesión de trabajo las funciones `agregar_a_vector()` y `agregar_a_vector2()` ejecuta el siguiente codigo:

Octubre 12, 2023

```
> install.packages(microbenchmark)
> library(microbenchmark)
> timings <- microbenchmark(agregar_a_vector(10000), agregar_a_vector_2(10000), times = 10)
>
> timings
```

El resultado puede variar dependiendo de tu sistema, pero básicamente `agregar_a_vector_2()` es 350 veces más rápido que `agregar_a_vector()`. Nota que la gran diferencia entre estas funciones es la *asignación* de memoria que se realiza en `agregar_a_vector_2()`. Concretamente, la instrucción `output <- vector("integer", n)` asigna/crea/guarda/reserva espacio en memoria para el objeto `output`, un vector de clase `integer` de tamaño `n`; nota que esto es hecho una sola vez. Por el contrario, en `agregar_a_vector()` optamos por `output <- c()` para definir el objeto `output`. Esta forma *dinámica* de reservar espacio en memoria para este objeto posee el inconveniente de ser invocada repetidas veces dentro del ciclo `for` lo que puede explicar la sub optimalidad de `agregar_a_vector_2()`. Puedes leer la documentación de la función `vector()` para conocer su correcto uso. También se recomienda leer la documentación de `microbenchmark()`.

6. Probablemente el siguiente ejercicio puede resolverse con alguna combinación de funciones del `tidyverse()`, sin embargo, de momento, escribe un ciclo `for` para calcular el número de valores únicos (sin repetir) en cada columna del *dataset* `iris`. **Hint1:** `?unique`. **Hint2:** posible salida

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##           35           23           43           22            3
```

7. Probablemente el siguiente ejercicio puede resolverse con alguna combinación de funciones del `tidyverse()`, sin embargo, de momento, escribe un ciclo `for` para determinar el tipo de objeto en cada columna del *dataset* `nycflights13::flights`; `nycflights13` es un paquete. **Hint:** posible salida

```
## $year
## [1] "integer"
##
## $month
## [1] "integer"
##
## $day
## [1] "integer"
##
## $dep_time
## [1] "integer"
##
## $sched_dep_time
## [1] "integer"
##
## $dep_delay
## [1] "numeric"
##
## $arr_time
## [1] "integer"
##
## $sched_arr_time
```

```
## [1] "integer"
##
## $arr_delay
## [1] "numeric"
##
## $carrier
## [1] "character"
##
## $flight
## [1] "integer"
##
## $tailnum
## [1] "character"
##
## $origin
## [1] "character"
##
## $dest
## [1] "character"
##
## $air_time
## [1] "numeric"
##
## $distance
## [1] "numeric"
##
## $hour
## [1] "numeric"
##
## $minute
## [1] "numeric"
##
## $time_hour
## [1] "POSIXct" "POSIXt"
```

8. Probablemente el siguiente ejercicio puede resolverse con alguna combinación de funciones del `tidyverse()`, sin embargo, de momento, escribe un ciclo `for` para calcular la media de cada columna del *dataset* `mtcars`. **Hint:** solución

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

9. Escribe código para simular la tirada de 50 volados, almacenando el resultado (1 = águila, 0 = sol) en un vector. **Hint1:** `?rbinom`. **Hint2:** posible solución

```
##  [1] 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1
## [39] 1 0 0 0 1 0 0 0 1 1 0 0
```

10. Escribe una función para estudiar el comportamiento a largo plazo del experimento *tirar volados* siguiendo las siguientes especificaciones:

(a) En consola, solicita al usuario la cantidad de volados a generar

(b) En consola, despliega el resultado de los volados

(c) Calcula el promedio aritmético (también conocido como *media*) de los volados generados. Calcula también la desviación estándar de los volados generados. En consola, despliega el promedio y la desviación estándar. **Hint:** ejemplos

```
> getVolados()
--- Genera volados ---
Cuantos volados quieres generar? 5
Estos son tus volados:  1 0 0 1 0
El promedio de tus volados es:  0.4
La desviacion estandar de tus volados es:  0.5477226
> getVolados()
--- Genera volados ---
Cuantos volados quieres generar? 10
Estos son tus volados:  0 0 1 0 1 0 1 0 0 0
El promedio de tus volados es:  0.3
La desviacion estandar de tus volados es:  0.4830459
> getVolados()
--- Genera volados ---
Cuantos volados quieres generar? 20
Estos son tus volados:  1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 0
El promedio de tus volados es:  0.4
La desviacion estandar de tus volados es:  0.5026247
```

11. Sin codificarlo, ¿puedes deducir cuál es la salida del siguiente código?

```
for (i in 1:2) {
    for (j in 1:3) {
        print(paste("i =", i, "j =", j))
    }
}
```