

# ***INSTITUTO TECNOLÓGICO DE NUEVO LEÓN***

## ***Actividades en RStudio***

**Materia.** Algoritmos y lenguajes de Programación

**Catedrático.** Juan Pablo Baldazo

**Alumnas.** Fátima Sorayda Tavarez Alejandro.

Bianca Guadalupe Hernández Hernández.

**Número de control.** 18480370

**Carrera.** Ingeniería Industrial

**2° Semestre**

**PERIODO AGOSTO – DICIEMBRE 2018**

**Horario.** 11:00am-12:00pm

**Guadalupe, Nuevo León, México.**

**Actividad #1.** Crear una función que reciba 4 parámetros (nombre, a, b, c), e imprima su nombre y la operación  $a + b * c$ , los nombres de las variables pueden variar.

```
Untitled4* x  Untitled2* x  Untitled3* x  Untitled5 x
Source on Save  Run  Source
1 #Actividad 1
2 prueba <-function(a=1,b=2,d=3)
3 {j='Fatima'
4   y= a+b*d
5   c= paste(j,y)
6   print(c)
7 }
8 prueba()
9
```

**Actividad #2.** Crear una función la cual tiene un ciclo que repite 5 veces la llamada a su función creada en la actividad anterior, la cual en cada iteración del ciclo incrementa los valores de las variables a, b y c en 1.

```
10 #Actividad 2
11 ciclo <- function(){a1 <- 1
12 b1 <- 1
13 c1 <- 1
14 for (h in 1:5) {
15   prueba(a1,b1,c1)
16   a1=a1+1
17   b1=b1+1
18   c1=c1+1
19 }
20 }
21 ciclo()
```

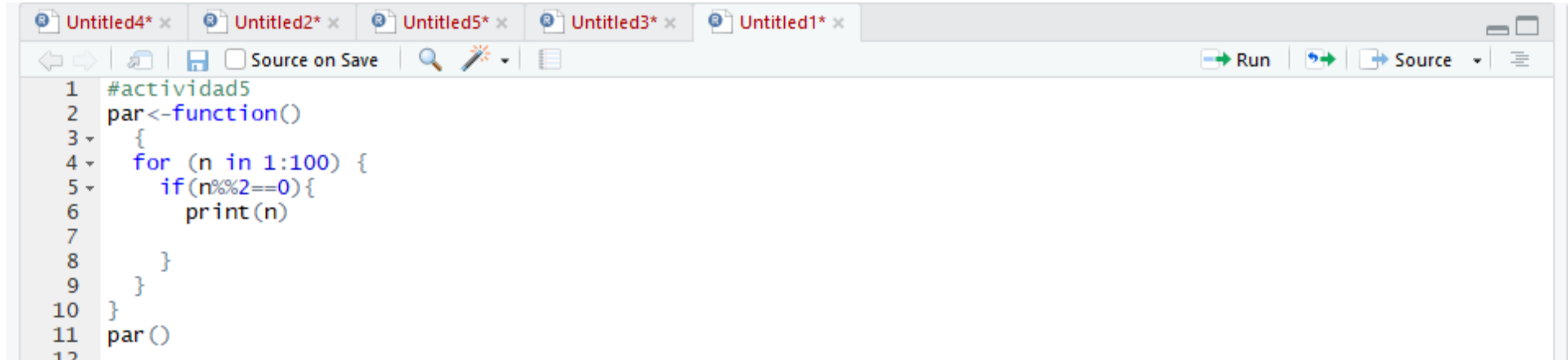
**Actividad #3.** Crear una función que reciba 3 parámetros (a, b y c) y resuelva la formula general. Traten de que los valores con los que prueben su función no provoquen raíces negativas porque marcara un error ya que caen en números imaginarios.

```
Untitled4* x  Untitled2* x  Untitled5* x  Untitled3* x  Untitled1* x
← →  Source on Save  Run  Source
1 #Actividad 3
2 formula_general <- function(a=2,b=4,c=1){
3   x=-b+(sqrt((b*b)-4*a*c))/2*a
4   print(x)
5   y=-b-(sqrt((b*b)-4*a*c))/2*a
6   print(y)
7 }
8 formula_general()
9 |
```

**Actividad #4.** Crear un vector que contenga 9 números enteros, los cuales serán los valores de a, b y c que le pasen a la función que crearon en la actividad anterior mediante un ciclo.

```
10 #Actividad 4
11 Hector<-c(21,62,20,23,8,27,12,02,42)
12 {
13   KYC<-function()
14     for (i in 1:9)
15       {
16         if(i%3==0)
17           {
18             i1<-i-1
19             i2<-i-2
20             formula_general(Hector[i],Hector[i1],Hector[i2])
21           }
22       }
23 }
24 KYC()
```

**Actividad #5.** Crear una función que no reciba parámetros que imprima los números pares entre el 1 y el 100.

The screenshot shows the RStudio interface with five untitled files open. The active file, 'Untitled4\*', contains the following R code:

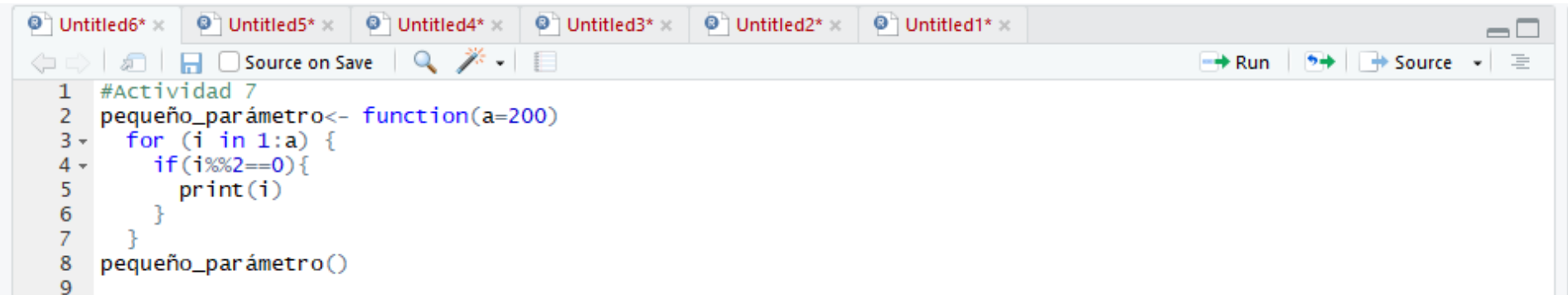
```
1 #actividad5
2 par<-function()
3 {
4   for (n in 1:100) {
5     if(n%%2==0){
6       print(n)
7     }
8   }
9 }
10 }
11 par()
12
```

**Actividad #6.** Crear una función que no reciba parámetros que imprima los números impares entre el 1 y el 100.

The screenshot shows the RStudio interface with the same five untitled files. The active file, 'Untitled4\*', contains the following R code:

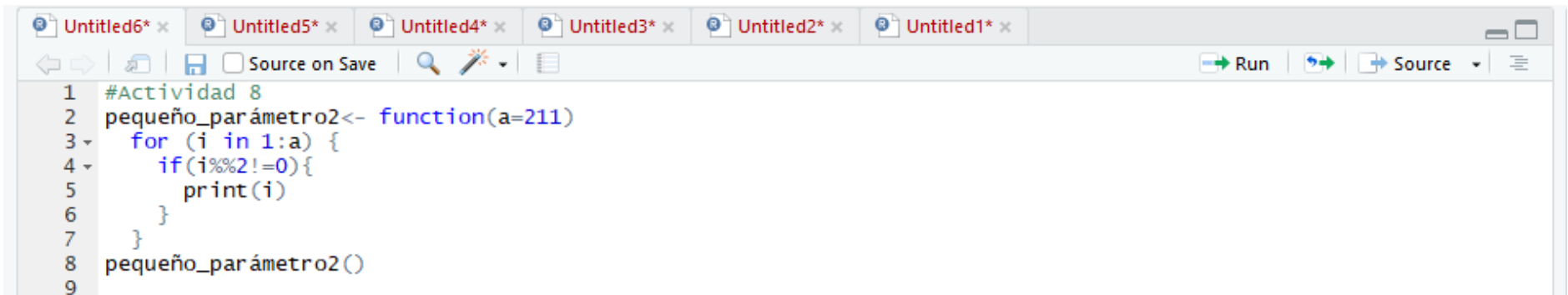
```
12
13 #actividad 6
14 impar<-function()
15 {
16   for (j in 1:100) {
17     if(j%%2!=0){
18       print(j)
19     }
20   }
21 }
22 impar()
```

**Actividad #7.** Crear una función que reciba un parámetro que imprima los números pares entre el 1 y el número que ingreses como parámetro.



```
1 #Actividad 7
2 pequeño_parámetro<- function(a=200)
3   for (i in 1:a) {
4     if(i%%2==0){
5       print(i)
6     }
7   }
8 pequeño_parámetro()
9
```

**Actividad #8.** Crear una función que reciba un parámetro que imprima los números impares entre el 1 y el número que ingreses como parámetro.



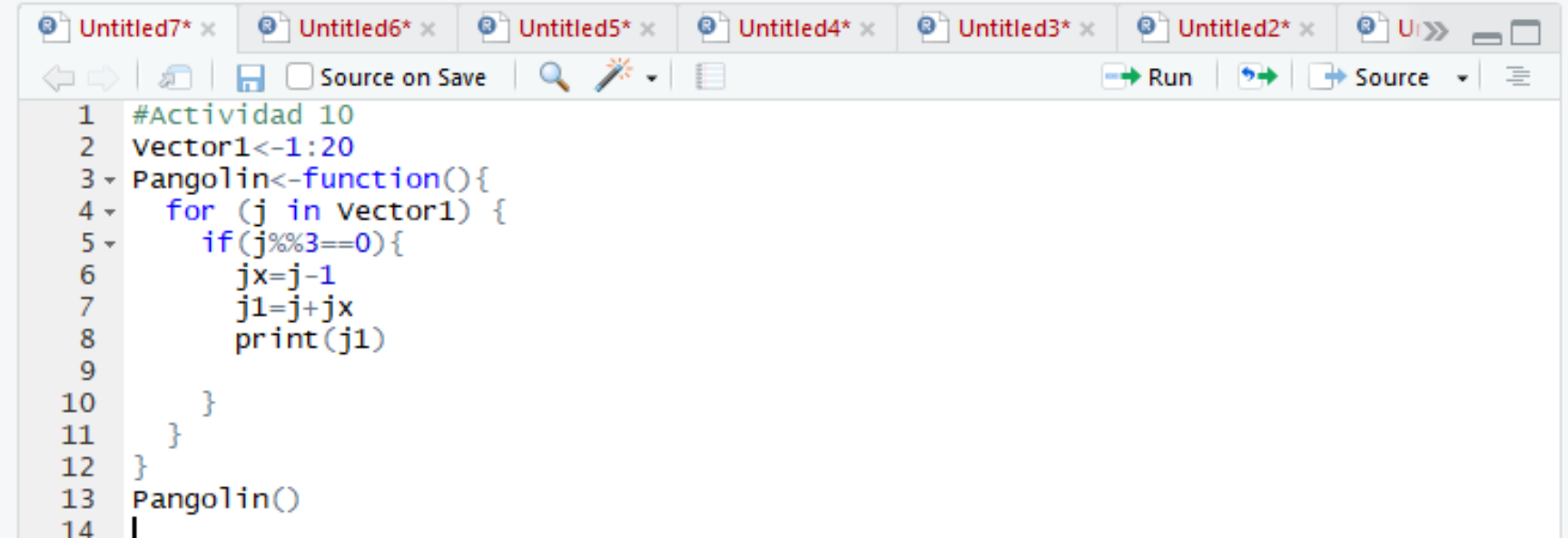
```
1 #Actividad 8
2 pequeño_parámetro2<- function(a=211)
3   for (i in 1:a) {
4     if(i%%2!=0){
5       print(i)
6     }
7   }
8 pequeño_parámetro2()
9
```

**Actividad #9.** Crear función que reciba como parámetro un número y realice el algoritmo  $3n+1$ .

```
10 |
11 #Actividad 9
12 variable<- 2
13 operación<- function(){
14     n=variable
15     c=3*n+1
16     print(c)
17 }
18 operación()
```

**Actividad #10.** Crear función que no reciba parámetros y cada tres posiciones realice la suma del valor del vector en la posición anterior a su posición actual. El vector es:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20.



```
1 #Actividad 10
2 Vector1<-1:20
3 Pangolin<-function(){
4     for (j in Vector1) {
5         if(j%%3==0){
6             jx=j-1
7             j1=j+jx
8             print(j1)
9         }
10    }
11 }
12 }
13 Pangolin()
14 |
```

**Actividad #11.** Crear una función que no reciba parámetros, pero que regrese la suma de los valores del arreglo bidimensional cuando los índices sean iguales, es decir [1,1],[2,2], etc. El arreglo es:

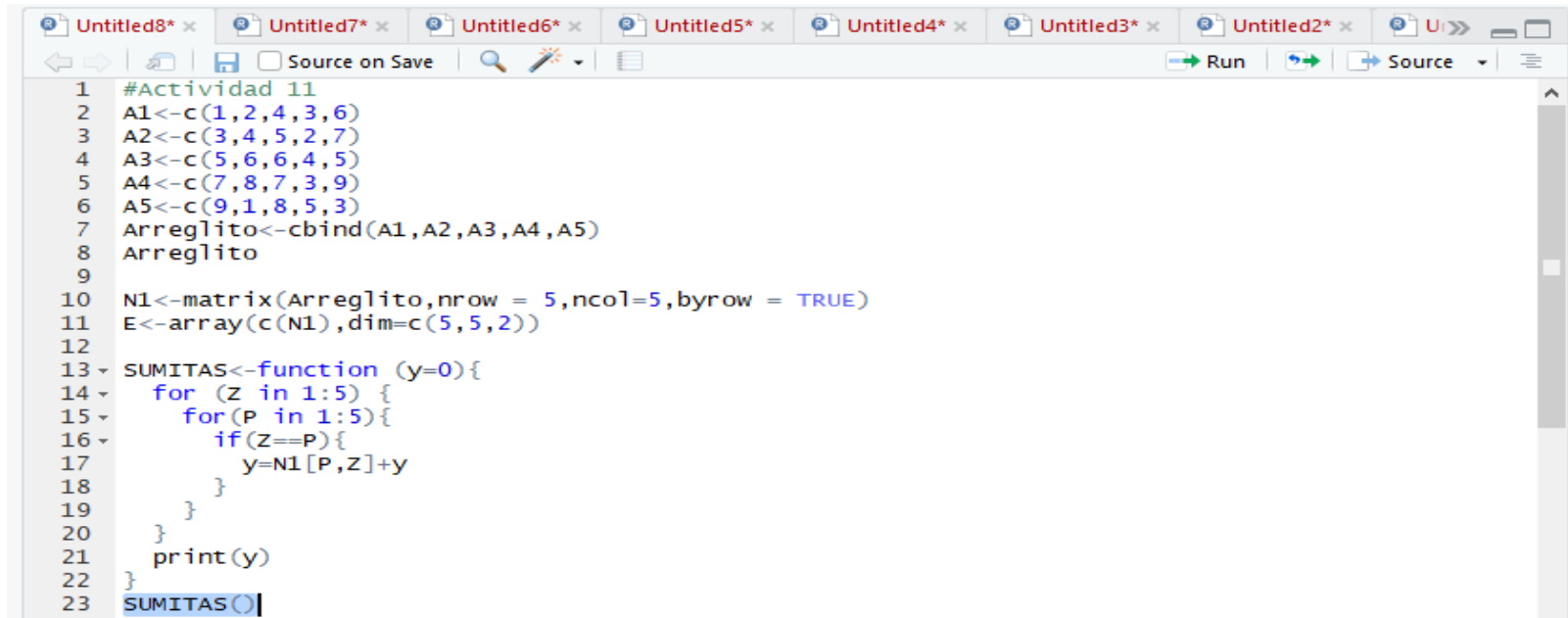
1 3 5 7 9

2 4 6 8 1

4 5 6 7 8

3 2 4 3 5

6 7 5 9 3

A screenshot of an R script editor window with multiple tabs labeled 'Untitled8\*' through 'Untitled2\*' and 'U1'. The script contains the following R code:

```
1 #Actividad 11
2 A1<-c(1,2,4,3,6)
3 A2<-c(3,4,5,2,7)
4 A3<-c(5,6,6,4,5)
5 A4<-c(7,8,7,3,9)
6 A5<-c(9,1,8,5,3)
7 Arreglito<-cbind(A1,A2,A3,A4,A5)
8 Arreglito
9
10 N1<-matrix(Arreglito,nrow = 5,ncol=5,byrow = TRUE)
11 E<-array(c(N1),dim=c(5,5,2))
12
13 SUMITAS<-function (y=0){
14   for (Z in 1:5) {
15     for(P in 1:5){
16       if(Z==P){
17         y=N1[P,Z]+y
18       }
19     }
20   }
21   print(y)
22 }
23 SUMITAS()
```

**Actividad #12.** Crear una función que reciba un parámetro que indica cuantos números primos requiere y devuelva un vector con los números o imprima esa cantidad de números primos.

```
Untitled11* x  Untitled10* x  Untitled8* x  Untitled7* x  Untitled6* x  Untitled5* x  Untitled4* x
Source on Save  Run  Source
1 #Actividad 12
2 mi_primo<- function(b) {
3   y<-0
4   x<-2
5   while(y<b){
6     x<-x+1
7   }
8
9   {
10    if(x%%2!=0){
11      if(x%%3!=0){
12        if(x%%4!=0){
13          if(x%%5!=0){
14            if(x%%6!=0){
15              if(x%%7!=0){
16                if(x%%8!=0){
17                  if(x%%9!=0){
18                    print(x)
19                    y<-y+1
20                  }
21                }
22              }
23            }
24          }
25        }
26      }
27    }
28  }
29 }
30 }
31 mi_primo(5)
32
```

**Actividad #13.** Crear una función que reciba un parámetro que indica cuantos números de la sucesión de Fibonacci requiere y devuelva un vector con los números o imprima esa cantidad de números de la sucesión.

```
13
14 #Actividad 13
15 Sucesión<-function(Y){
16   x<- 1
17   Y<-as.integer(Y)
18   if(Y<1){
19     }
20   B<- -1
21   B1<- 1
22   while (x<=Y) {
23     Bx<- B+B1
24     x<- x+1
25     print(Bx)
26     B<- B1
27     B1<- Bx
28   }
29 }
30 Sucesión(10)
31
```



**Actividad #14.** Crear una función que reciba un número y debe devolver verdadero si el número es primo y falso si no lo es.

```
14
15 #Actividad 14
16 Número<-function(a)
17   for (i in a) {
18     if(i%%2==0){
19       dime<- TRUE
20       print(dime)
21     }else{
22       dime<- FALSE
23       print(dime)
24     }
25   }
26   print(Número(20))
```

**Actividad #15.** Crear una función que reciba un número y debe devolver verdadero si el número pertenece a la sucesión de fibonacci y falso si no lo es.

```
Untitled11* x  Untitled10* x  Untitled8* x  Untitled7* x  Untitled6* x  Untitled5* x  Untitled4* x
Source on Save  Run  Source

1 #Actividad 15
2 Función_principal<-function(a){
3   Sucesión<-function(Y){
4     x<- 1
5     Y<-as.integer(Y)
6     if(Y<1){
7     }
8     B<- -1
9     B1<- 1
10    while (x<=Y) {
11      Bx<- B+B1
12      x<- x+1
13      print(Bx)
14      B<- B1
15      B1<- Bx
16    }
17    AVISO<-function(Sucesión){
18      if(i==Sucesión){
19        dime<-TRUE
20        print(dime)
21      }
22      else{
23        dime<-FALSE
24        print(dime)
25      }
26    }
27  }
28 }
29 Función_principal(a(1))
30 |
```