

Implementación de funciones con Threads

Cristian José Rodríguez Rojas

October 2022

1. Trabajo Práctico - N° 001

Implementar según el ejemplo de clase una función **numPar** que reciba un parámetro **n** y calcule el número pares entre 1 y **n**, así mismo realizar el cálculo de tiempo para la evaluación de la implementación, para hacer el llamado a la función se deberá hacer a través de uno o más threads

- Las pruebas unitarias deberán ser con los siguientes parámetros a la función **numPar**:

- **n** = 10
- **n** = 30
- **n** = 200
- **n** = 5030

2. Código Fuente en Python

```
\lstinputlisting[language=Python]{0_code.py}
```

```
#Implementacion de funciones con Threads
#Alumno: Cristian Jose Rodriguez Rojas
#Codigo: 191427
#FINESI
*****
import math
import time #pip install time
from threading import Thread
import threading

#Forma 1 numeros pares entre 1 y n
def numPar1(r):
    time_ini = time.time()
    c = 0
    while(c < r):
        print(c)
        c = c + 2
    print("Total:", math.floor(r/2))
    time_end = time.time()
    time_total = time_end - time_ini
    print("Time_elapsed:~", time_total)

#Utilizando Threads
t = Thread(target=numPar1(10), args=())
t.start()
```

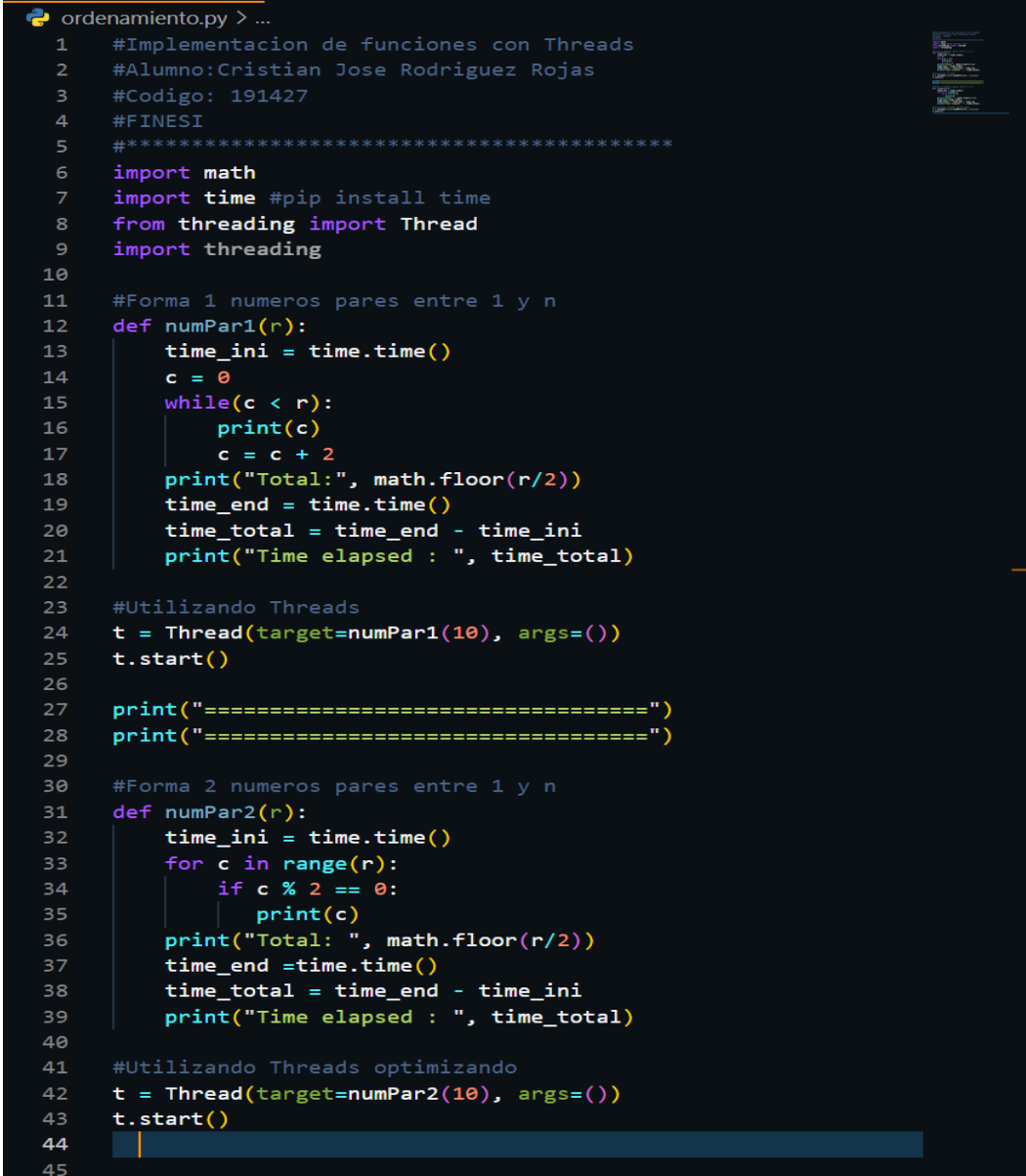
```
print("=====")
print("=====")
```

#Forma 2 numeros pares entre 1 y n

```
def numPar2(r):
    time_ini = time.time()
    for c in range(r):
        if c % 2 == 0:
            print(c)
    print("Total: ", math.floor(r/2))
    time_end = time.time()
    time_total = time_end - time_ini
    print("Time elapsed: ", time_total)
```

#Utilizando Threads optimizando

```
t = Thread(target=numPar2(10), args=())
t.start()
```



```
ordenamiento.py > ...
1  #Implementacion de funciones con Threads
2  #Alumno:Cristian Jose Rodriguez Rojas
3  #Codigo: 191427
4  #FINESI
5  #*****
6  import math
7  import time #pip install time
8  from threading import Thread
9  import threading
10
11 #Forma 1 numeros pares entre 1 y n
12 def numPar1(r):
13     time_ini = time.time()
14     c = 0
15     while(c < r):
16         print(c)
17         c = c + 2
18     print("Total:", math.floor(r/2))
19     time_end = time.time()
20     time_total = time_end - time_ini
21     print("Time elapsed : ", time_total)
22
23 #Utilizando Threads
24 t = Thread(target=numPar1(10), args=())
25 t.start()
26
27 print("=====")
28 print("=====")
29
30 #Forma 2 numeros pares entre 1 y n
31 def numPar2(r):
32     time_ini = time.time()
33     for c in range(r):
34         if c % 2 == 0:
35             print(c)
36     print("Total: ", math.floor(r/2))
37     time_end = time.time()
38     time_total = time_end - time_ini
39     print("Time elapsed : ", time_total)
40
41 #Utilizando Threads optimizando
42 t = Thread(target=numPar2(10), args=())
43 t.start()
44
45
```

CODIGO EN CAPTURA POR SEGURIDAD

3. Capturas de los resultados -Modo Consola

Captura de $n = 10$ -> Ambos metodos utilizando threads numeros pares

```
PS E:\VIII SEMESTRE_2022_II_FINESI\COMPUTACION PARALELA\TRABAJOS\python> & C:/Python310/python.exe "e:/VIII SEMESTRE_2022_II_FINESI/COMPUTACION PARALELA/TRABAJOS/ordenamiento.py"

Metodo 1 :

0
2
4
6
8
Total: 5
Time elapsed : 0.008552789688110352
=====
=====
Metodo 2 :

0
2
4
6
8
Total: 5
Time elapsed : 0.0
PS E:\VIII SEMESTRE_2022_II_FINESI\COMPUTACION PARALELA\TRABAJOS\python> █
```

Captura de $n = 30$ -> Ambos metodos utilizando threads numeros pares

```
Time elapsed : 0.00800013542175293
PS E:\VIII SEMESTRE_2022_II_FINESI\COMPUTACION PARALELA\TRABAJOS\python> & C:/Python310/p
ython.exe "e:/VIII SEMESTRE_2022_II_FINESI/COMPUTACION PARALELA/TRABAJOS/ordenamiento.py"

Metodo 1 :

0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
Total: 15
Time elapsed : 0.011904001235961914
=====
=====
Metodo 2 :

0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
Total: 15
Time elapsed : 0.0
PS E:\VIII SEMESTRE_2022_II_FINESI\COMPUTACION PARALELA\TRABAJOS\python> █
```

Captura de n = 200 ->Primer metodo utilizando threads numeros pares

```
ordenamiento.py X
ordenamiento.py > ...
1  #Implementacion de funciones con Threads
2  #Alumno:Cristian Jose Rodriguez Rojas
3  #Codigo: 191427
4  #FINESI
5  #*****
6  import math
7  import time #pip install time
8  from threading import Thread
9  import threading
10
11 #Forma 1 numeros pares entre 1 y n
12 def numPar1(r):
13     time_ini = time.time()
14     c = 0
15     while(c < r):
16         print(c)
17         c = c + 2
18     print("Total:", math.floor(r/2))
19     time_end = time.time()
20     time_total = time_end - time_ini
21     print("Time elapsed : ", time_total)
22
23 #Utilizando Threads
24 t = Thread(target=numPar1(200), args=())
25 t.start()
26
27 print("=====")
28 print("=====")
29
30 #Forma 2 numeros pares entre 1 y n
31 def numPar2(r):
32     time_ini = time.time()
33     for c in range(r):
34         if c % 2 == 0:
35             print(c)
36     print("Total: ", math.floor(r/2))
37     time_end =time.time()
38     time_total = time_end - time_ini
39     print("Time elapsed : ", time_total)
40
41 #Utilizando Threads optimizando
42 t = Thread(target=numPar2(200), args=())
43 t.start()
```

118
120
122
124
126
128
130
132
134
136
138
140
142
144
146
148
150
152
154
156
158
160
162
164
166
168
170
172
174
176
178
180
182
184
186
188
190
192
194
196
198
Total: 100
Time elapsed : 0.0009999275207519531

Captura de n = 200 -> Segundo metodo utilizando threads numeros pares

```

11     c = 0
12     while(c < r):
13         print(c)
14         c = c + 2
15     print("Total:", math.floor(
16         time_end = time.time()
17         time_total = time_end - tim
18     print("Time elapsed : ", ti
19
20     #Utilizando Threads
21     t = Thread(target=numPar1(200),
22     t.start()
23
24     print("=====")
25     print("=====")
26
27     #Forma 2 numeros pares entre 1
28     def numPar2(r):
29         print("Metodo 2 :\n")
30         time_ini = time.time()
31         for c in range(r):
32             if c % 2 == 0:
33                 print(c)
34         print("Total: ", math.floor
35         time_end =time.time()
36         time_total = time_end - tim
37         print("Time elapsed : ", ti
38
39     #Utilizando Threads optimizando
40     t = Thread(target=numPar2(200),
41     t.start()
42
43
44

```

Total: 100
 Time elapsed : 0.016001224517822266
 PS E:\VIII SEMESTRE_2022_II_FINESI\COMPUTACION PARALELA\TRABAJOS\python>

Captura de n = 5030 -> Primer metodo utilizando threads numeros pares

```
ordenamiento.py > ...
1  #Implementacion de funciones con Threads
2  #Alumno:Cristian Jose Rodriguez Rojas
3  #Codigo: 191427
4  #FINESI
5  #*****
6  import math
7  import time #pip install time
8  from threading import Thread
9  import threading
10
11 #Forma 1 numeros pares entre 1 y n
12 def numPar1(r):
13     time_ini = time.time()
14     c = 0
15     while(c < r):
16         print(c)
17         c = c + 2
18     print("Total:", math.floor(r/2))
19     time_end = time.time()
20     time_total = time_end - time_ini
21     print("Time elapsed : ", time_total)
22
23 #Utilizando Threads
24 t = Thread(target=numPar1(5030), args=())
25 t.start()
26
27 print("=====")
28 print("=====")
29
30 #Forma 2 numeros pares entre 1 y n
31 def numPar2(r):
32     time_ini = time.time()
33     for c in range(r):
34         if c % 2 == 0:
35             print(c)
36     print("Total: ", math.floor(r/2))
37     time_end = time.time()
38     time_total = time_end - time_ini
39     print("Time elapsed : ", time_total)
40
41 #Utilizando Threads optimizando
42 t = Thread(target=numPar2(5030), args=())
43 t.start()
44
45
```

4948
4950
4952
4954
4956
4958
4960
4962
4964
4966
4968
4970
4972
4974
4976
4978
4980
4982
4984
4986
4988
4990
4992
4994
4996
4998
5000
5002
5004
5006
5008
5010
5012
5014
5016
5018
5020
5022
5024
5026
5028
Total: 2515
Time elapsed : 0.008005142211914062
=====

Captura de n = 5030 -> Segundo metodo utilizando threads numeros pares

```
ordenamiento.py X
ordenamiento.py > ...
1 #Implementacion de funciones con Threads
2 #Alumno:Cristian Jose Rodriguez Rojas
3 #Codigo: 191427
4 #FINESI
5 #*****
6 import math
7 import time #pip install time
8 from threading import Thread
9 import threading
10
11 #Forma 1 numeros pares entre 1 y n
12 def numPar1(r):
13     time_ini = time.time()
14     c = 0
15     while(c < r):
16         print(c)
17         c = c + 2
18     print("Total:", math.floor(r/2))
19     time_end = time.time()
20     time_total = time_end - time_ini
21     print("Time elapsed : ", time_total)
22
23 #Utilizando Threads
24 t = Thread(target=numPar1(5030), args=())
25 t.start()
26
27 print("=====")
28 print("=====")
29
30 #Forma 2 numeros pares entre 1 y n
31 def numPar2(r):
32     time_ini = time.time()
33     for c in range(r):
34         if c % 2 == 0:
35             print(c)
36     print("Total: ", math.floor(r/2))
37     time_end = time.time()
38     time_total = time_end - time_ini
39     print("Time elapsed : ", time_total)
40
41 #Utilizando Threads optimizando
42 t = Thread(target=numPar2(5030), args=())
43 t.start()
44
45
```

4940
4950
4952
4954
4956
4958
4960
4962
4964
4966
4968
4970
4972
4974
4976
4978
4980
4982
4984
4986
4988
4990
4992
4994
4996
4998
5000
5002
5004
5006
5008
5010
5012
5014
5016
5018
5020
5022
5024
5026
5028
Total: 2515
Time elapsed : 0.01200246810913086
[Done] exited with code=0 in 0.145 seconds

4. Interpretación de resultados

1. Como se puede observar cuando nos pide el dato $n = 10$ y por el metodo de ordenamiento de numeros pares en el primer metodo se medoro un poquito mas que el segundo metodo que fue mas rapido y se puede constatar en la captura de pantalla utilizando threads
2. Como se puede observar cuando nos pide el dato $n = 30$ y por el metodo de ordenamiento de numeros pares en el primer metodo se medoro un poquito mas que el segundo metodo que fue mas rapido y se puede constatar en la captura de pantalla utilizando threads
3. Como se puede observar cuando nos pide el dato $n = 100$ y por el metodo de ordenamiento de numeros pares en el primer metodo se medoro 0.01600 que el segundo metodo que fue 0.0009 mas rapido se puede constatar en la captura de pantalla utilizando threads
4. Como se puede observar cuando nos pide el dato $n = 100$ y por el metodo de ordenamiento de numeros pares en el primer metodo se medoro 0.0120 que el segundo metodo que fue 0.00800 mas rapido se puede constatar en la captura de pantalla utilizando threads
 - a) En conclusion general se puede decir que el metodo que se construyo en python fue realizo para optimizar al momento de imprimir pero esta ves implementando threads que hace que un codigo imprimira a la misma ves

Este tranbajo se realizo en L^AT_EX