

**CONQUER**  
**BLOCKS**

**PYTHON**

FUNCIONES: ESTILO,  
RECURSIVIDAD Y MÉMOIZACIÓN

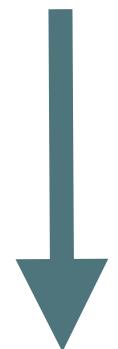
---



# GUIA DE ESTILO

**Funciones y módulos deben tener nombres descriptivos  
(en minúscula y con “\_” separando las palabras)**

**Las funciones deben incluir un comentario conciso que explique su función en formato docstring “”” ”””**



**Sabiendo el nombre de la función, los argumentos necesarios y los valores de retorno cualquier programador debe poder integrar la función en sus programas**



# GUIA DE ESTILO

```
def nombre_funcion(parametro_0, parametro_1='valor por defecto')
```

```
llamada_funcion(parametro_0, parametro_1="valor")
```

**No debe haber espacios  
alrededor del “=”**



# GUIA DE ESTILO

PEP 8: <https://peps.python.org/pep-0008/>

**Recomienda limitar las líneas de código a 79 caracteres**

```
def nombre_funcion(  
    parameter_0, parameter_1, parameter_2,  
    parameter_3, parameter_4, parameter_5):  
    cuerpo de la funcion
```



# GUIA DE ESTILO

PEP 8: <https://peps.python.org/pep-0008/>

**La separación entre la definición de dos funciones es de dos líneas en blanco**

```
def nombre_funcion(  
    parameter_0, parameter_1, parameter_2,  
    parameter_3, parameter_4, parameter_5):  
    cuerpo de la función
```



# GUIA DE ESTILO

PEP 8: <https://peps.python.org/pep-0008/>

**Los import deben escribirse al comienzo del script después de los comentarios que describen el programa al completo.**

```
#-----  
#Name: Union Tool Sample Script  
#Purpose: Runs the Union geoprocessing tool from ArcGIS Pro  
#Author: Esri & Tripp Corbin  
#  
#Created: 09/15/2015  
#Updated: 07/05/2020  
#  
#Usage: Union two feature classes together  
#Software Version: ArcGIS Pro 2.5 Basic  
#-----  
#import system modules  
import arcpy  
From arcpy import env  
  
# Sets the current workspace to avoid having to specify the full path  
# to the feature classes each time  
arcpy.env.workspace = c:\\student\\IntroArcPro\\Databases\\Trippville.gdb"  
#Runs Union geoprocessing tool on 2 feature classes  
arcpy.Union_analysis(["Parcels", "Floodplains"], "Parcels_Floodplain_Union", "NO_FID",  
0.0003)
```

Información general del script

Importación de módulos

Comentarios en las secciones del script



# RECURSIVIDAD



# RECUSIVIDAD

Concepto basico en computer science: Se basa en dividir el problema en sub-problemas faciles de resolver uno a uno



# RECUSIVIDAD

Se trata de llamar a una función dentro de si misma.

```
def recursividad(i):
    if i ==1:
        return i
    else:
        return recursividad(i-1)

# caso de uso
recursividad(100)

✓ 0.0s
```



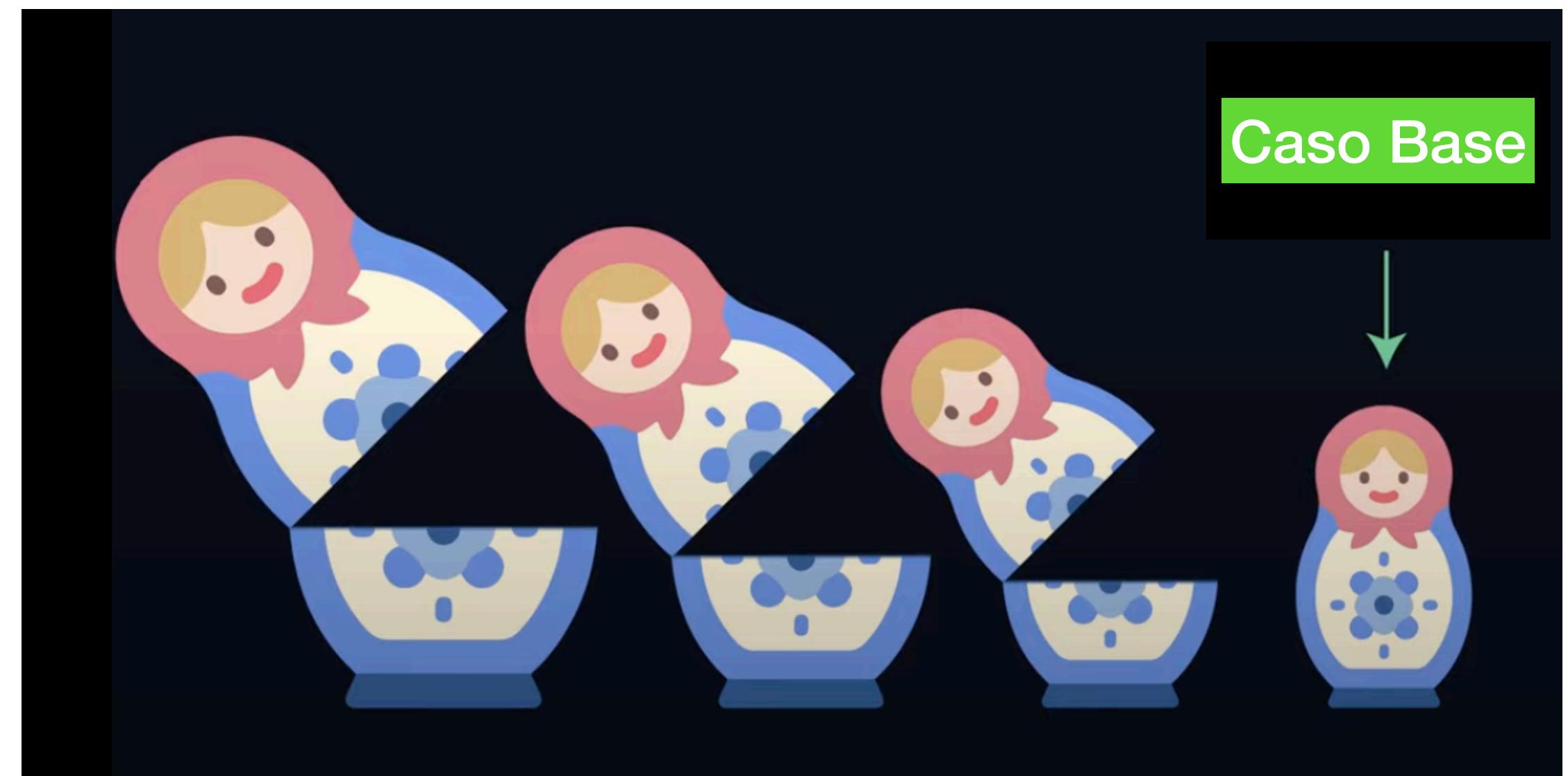
# RECUSIVIDAD

Se trata de llamar a una función dentro de si misma.

```
def recursividad(i):
    if i ==1:
        return i
    else:
        return recursividad(i-1)

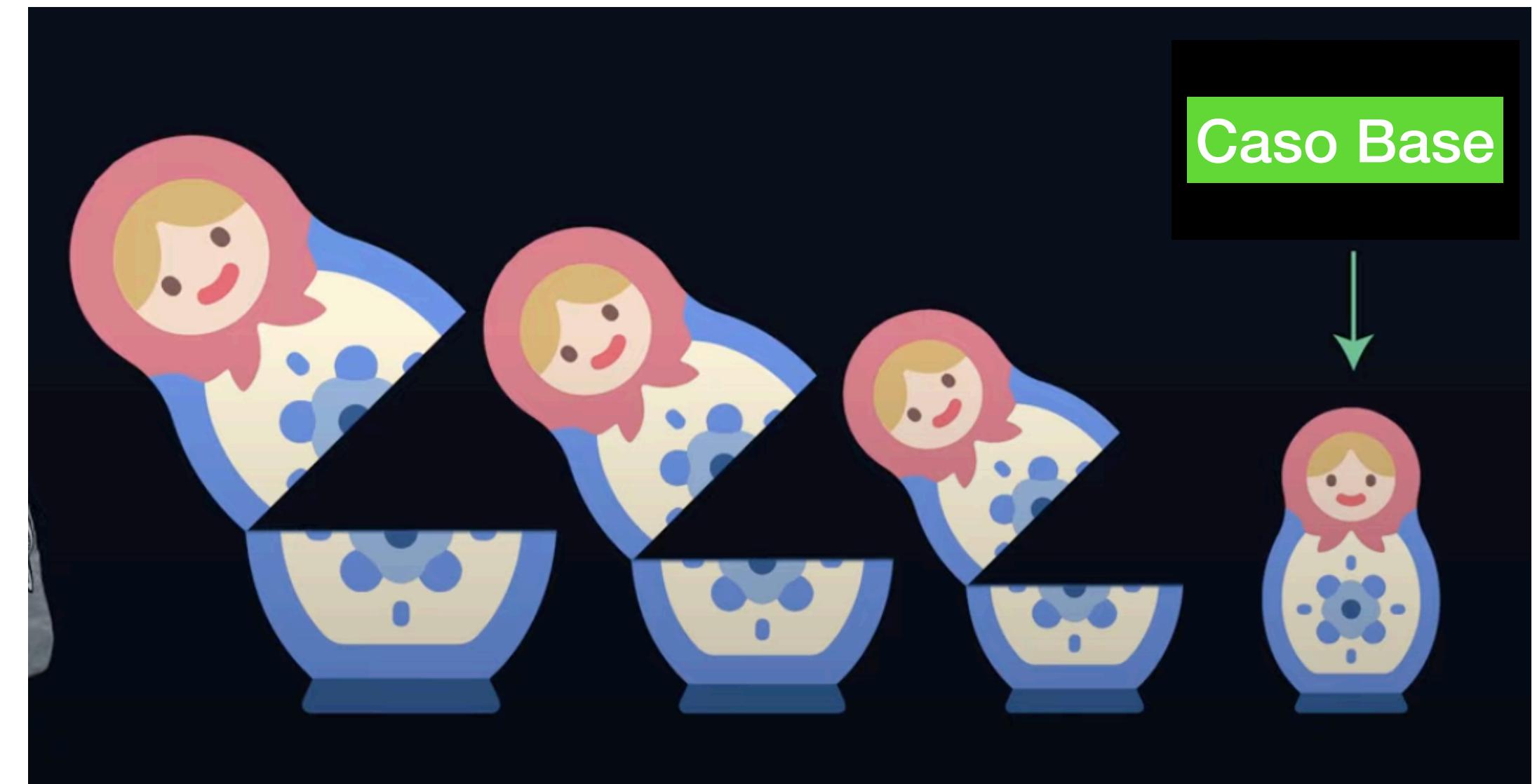
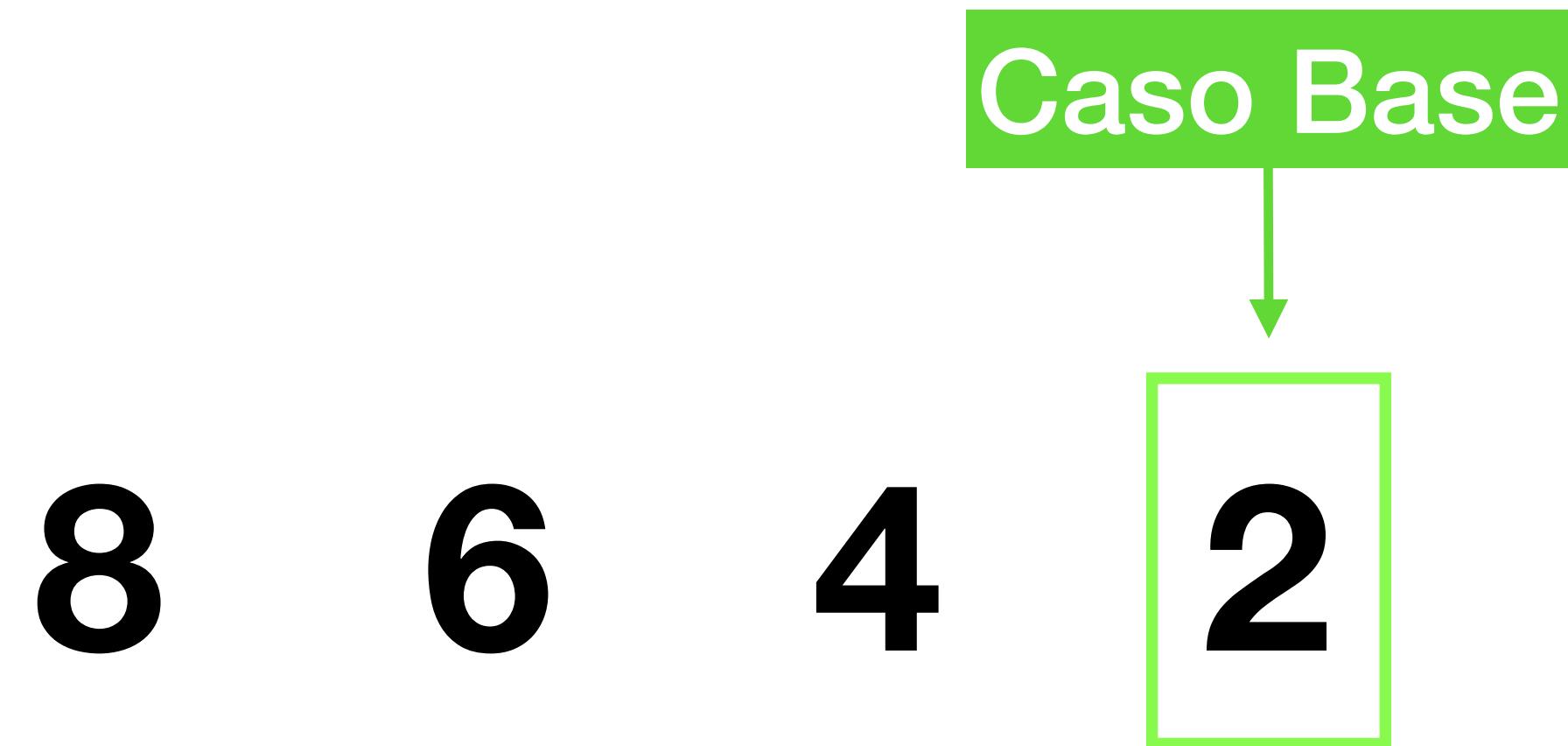
# caso de uso
recursividad(100)

✓ 0.0s
```



# RECUSIVIDAD

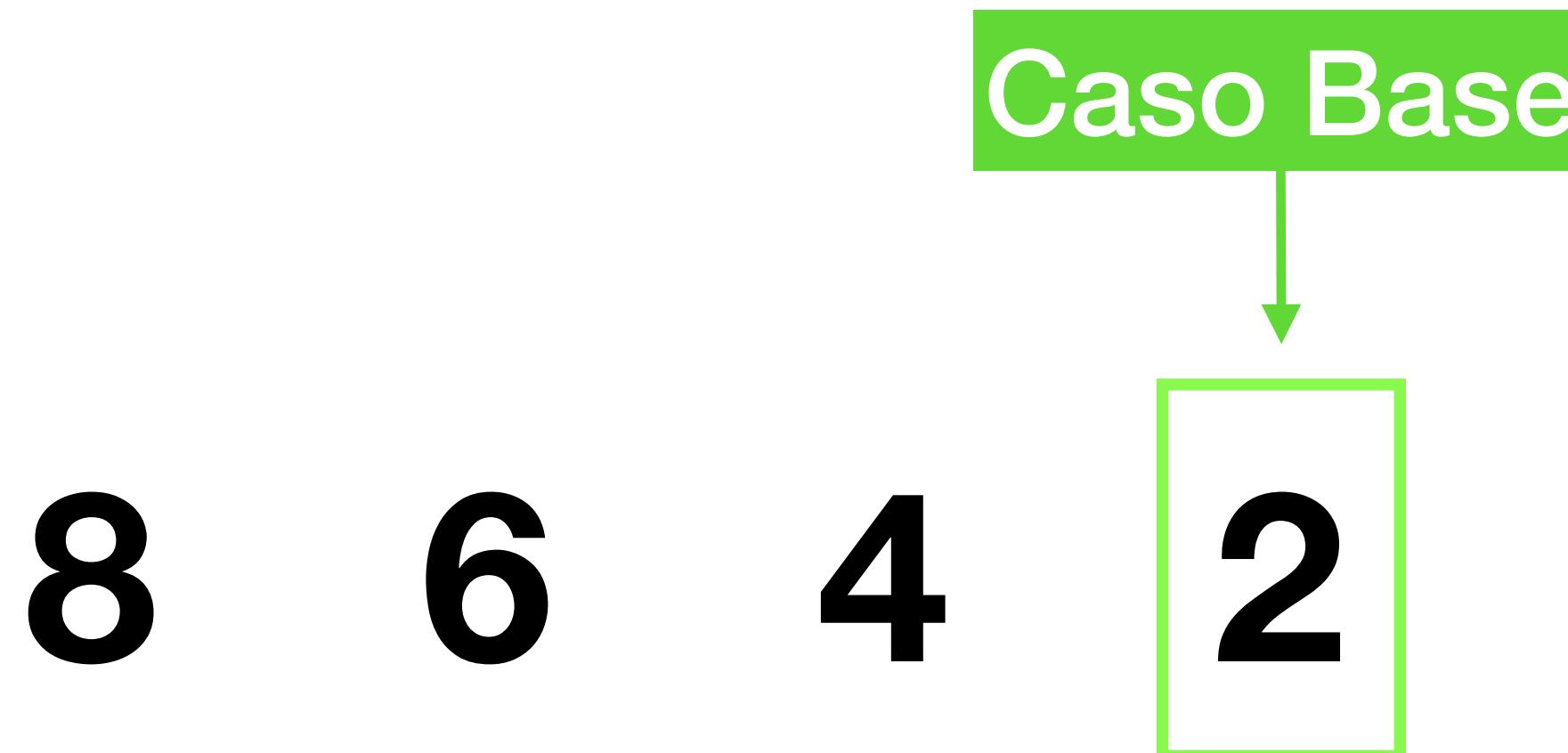
EJEMPLO: Escribir todos los números pares positivos menores que 8





# RECUSIVIDAD

EJEMPLO: Escribir todos los números pares positivos menores que 8



```
def numeros_pares(num):  
    print(num)  
    if num == 2:  
        return num  
    else:  
        return numeros_pares(num-2)  
  
numeros_pares(8)  
✓ 0.0s
```

8  
6  
4  
2



# RECUSIVIDAD

EJEMPLO: Escribir todos los números pares positivos menores que 8

```
def numeros_pares(num):
    for i in range(num, 0, -2):
        print(i)

numeros_pares(8)
```

✓ 0.0s

8  
6  
4  
2

```
def numeros_pares(num):
    print(num)
    if num == 2:
        return num
    else:
        return numeros_pares(num-2)

numeros_pares(8)
```

✓ 0.0s

8  
6  
4  
2

Caso Base



# RECUSIVIDAD

**EJEMPLO:** Serie de Fibonacci

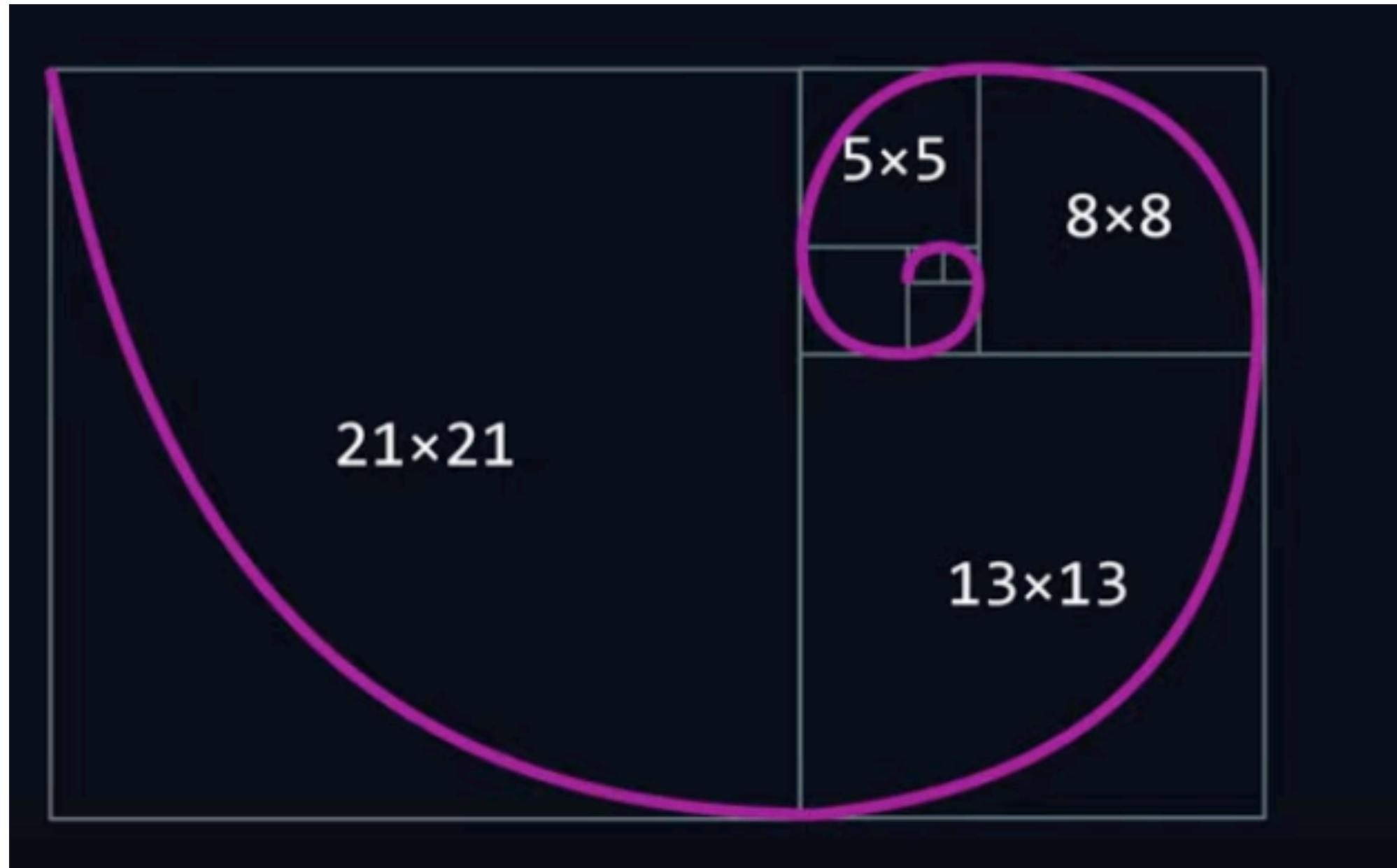
**Es una secuencia donde cada numero es la suma de los dos anteriores**

0 1 1 2 3 5 8 13 21 ...



# RECURSIVIDAD

EJEMPLO: Serie de Fibonacci





# RECUSIVIDAD

EJEMPLO: Serie de Fibonacci

Es una secuencia donde cada numero es la suma de los dos anteriores

índice	0	1	2	3	4	5	6	7	8	...
--------	---	---	---	---	---	---	---	---	---	-----

número	0	1	1	2	3	5	8	13	21	...
--------	---	---	---	---	---	---	---	----	----	-----



# RECUSIVIDAD

## EJEMPLO: Serie de Fibonacci

```
def fibonacci(indice):
    if indice <=1:
        return indice
    else:
        return fibonacci(indice-1)+fibonacci(indice-2)
```



# RECUSIVIDAD

## EJEMPLO: Serie de Fibonacci

```
def fibonacci(indice):
    if indice <=1:
        |   return indice
    else:
        |   return fibonacci(indice-1)+fibonacci(indice-2)

✓for i in range(0,11):
    |   print(fibonacci(i))
```

```
0
1
1
2
3
5
8
13
21
34
55
```



# RECUSIVIDAD

```
def fibonacci(indice):
    if indice <=1:
        return indice
    else:
        return fibonacci(indice-1)+fibonacci(indice-2)
```

**fibonacci(3) → fibonacci(2) + fibonacci(1)**



# RECUSIVIDAD

```
def fibonacci(indice):
    if indice <=1:
        return indice
    else:
        return fibonacci(indice-1)+fibonacci(indice-2)
```

**fibonacci(3) → fibonacci(2) + fibonacci(1)**

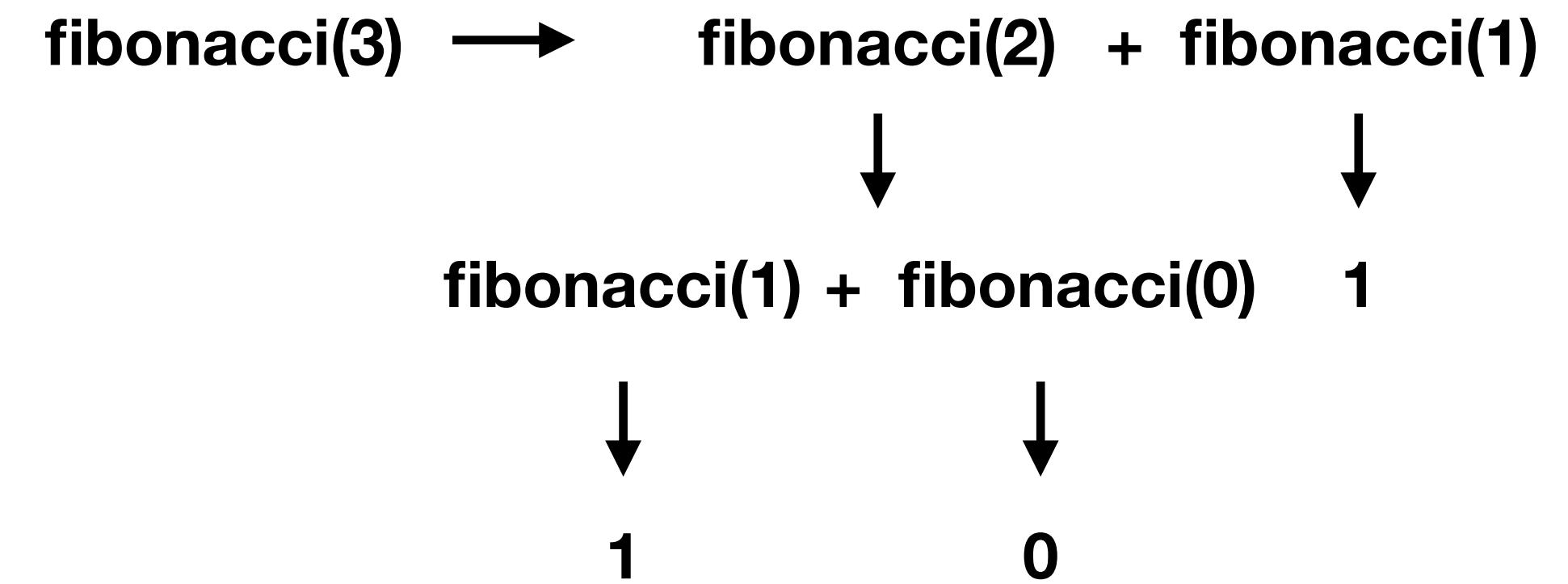
↓                    ↓

**fibonacci(1) + fibonacci(0) 1**



# RECUSIVIDAD

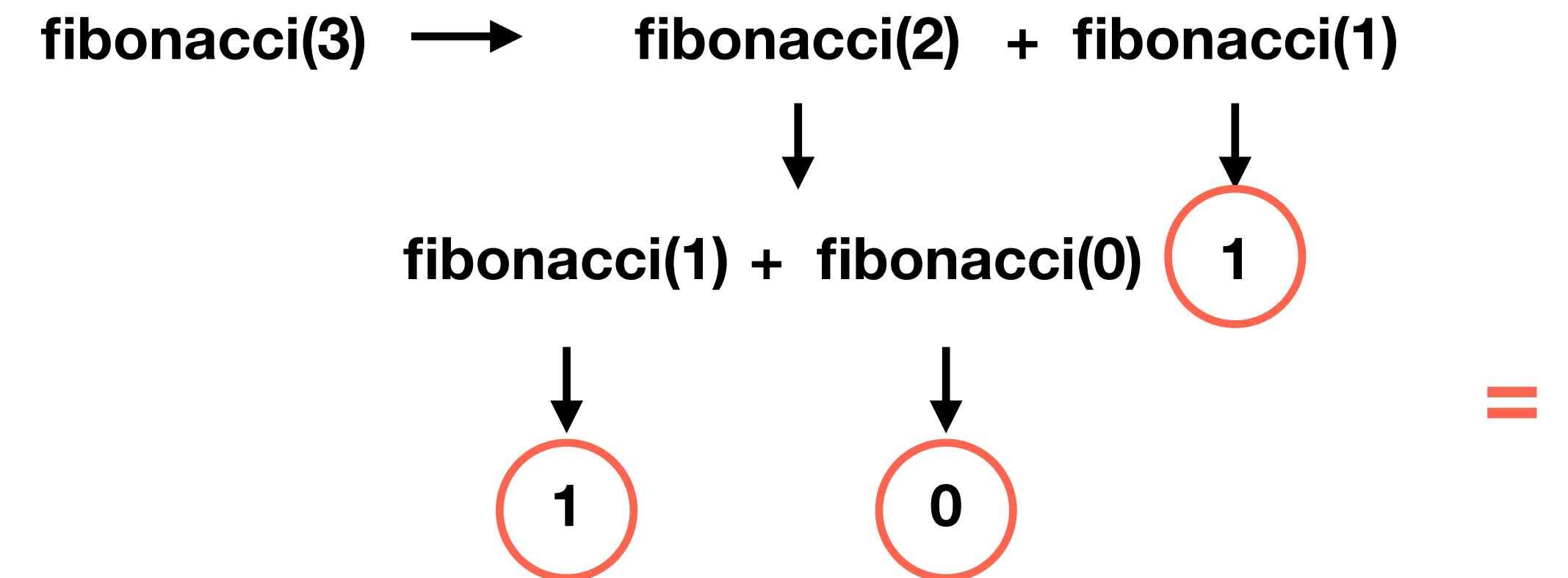
```
def fibonacci(indice):
    if indice <=1:
        return indice
    else:
        return fibonacci(indice-1)+fibonacci(indice-2)
```





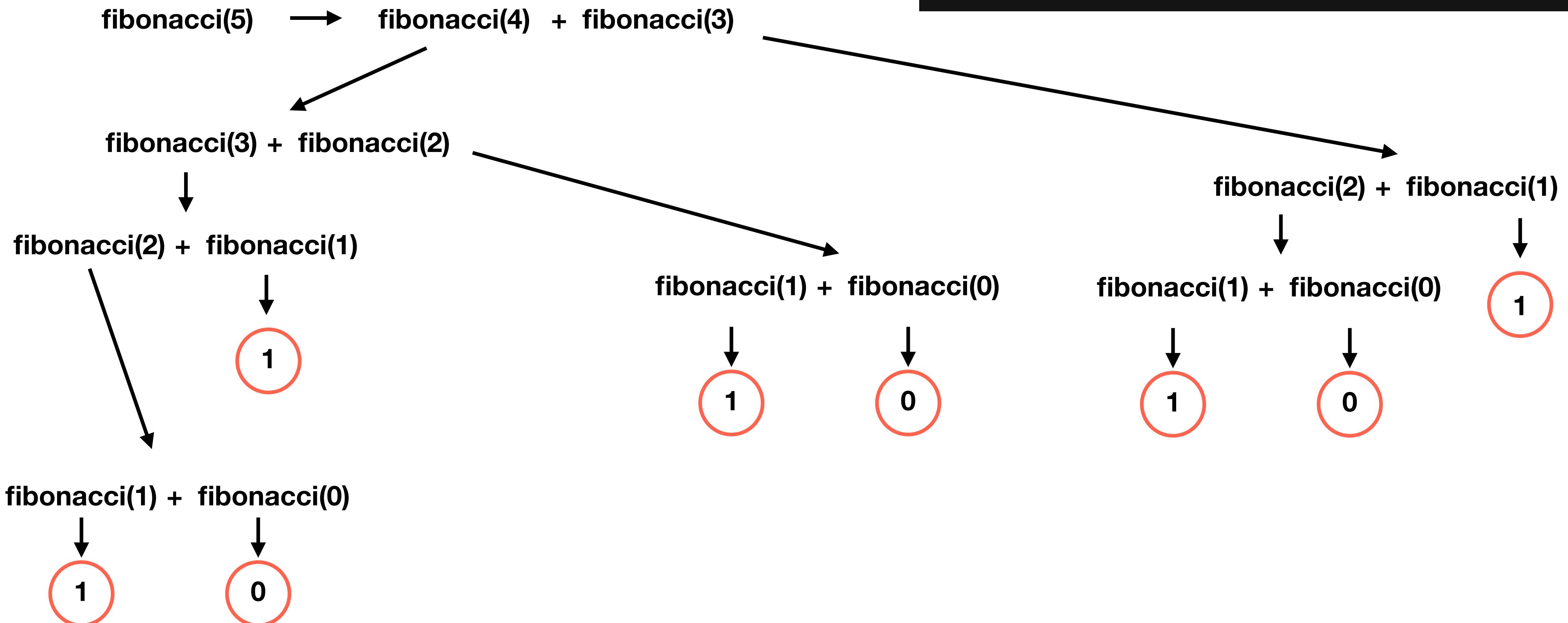
# RECUSIVIDAD

```
def fibonacci(indice):
    if indice <=1:
        return indice
    else:
        return fibonacci(indice-1)+fibonacci(indice-2)
```





# RECUSIVIDAD



```
def fibonacci(indice):  
    if indice <=1:  
        return indice  
    else:  
        return fibonacci(indice-1)+fibonacci(indice-2)
```



# RECUSIVIDAD

## EJEMPLO: Serie de Fibonacci

```
def fibonacci(indice):
    if indice <=1:
        |   return indice
    else:
        |   return fibonacci(indice-1)+fibonacci(indice-2)

✓for i in range(0,11):
    |   print(fibonacci(i))
```

```
0
1
1
2
3
5
8
13
21
34
55
```



# RECUSIVIDAD

## EJEMPLO: Serie de Fibonacci

```
def fibonacci_iter(indice):
    secuencia = [0,1]
    for i in range(indice):
        secuencia.append(secuencia[-1] + secuencia[-2])

    return secuencia[-2]

for i in range(0,11):
    print(fibonacci_iter(i))
```

0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55



# RECUSIVIDAD

## EJEMPLO: Serie de Fibonacci

```
start_recursive = time.time()
fibonacci(8)
end_recursive = time.time()
print("total time recursive...", end_recursive - start_recursive)

start_iter = time.time()
fibonacci_iter(8)
end_iter = time.time()
print("total time iterative...", end_iter - start_iter)
```

```
total time recursive... 5.245208740234375e-06
total time iterative... 2.1457672119140625e-06
```



# RECUSIVIDAD

## EJEMPLO: Serie de Fibonacci

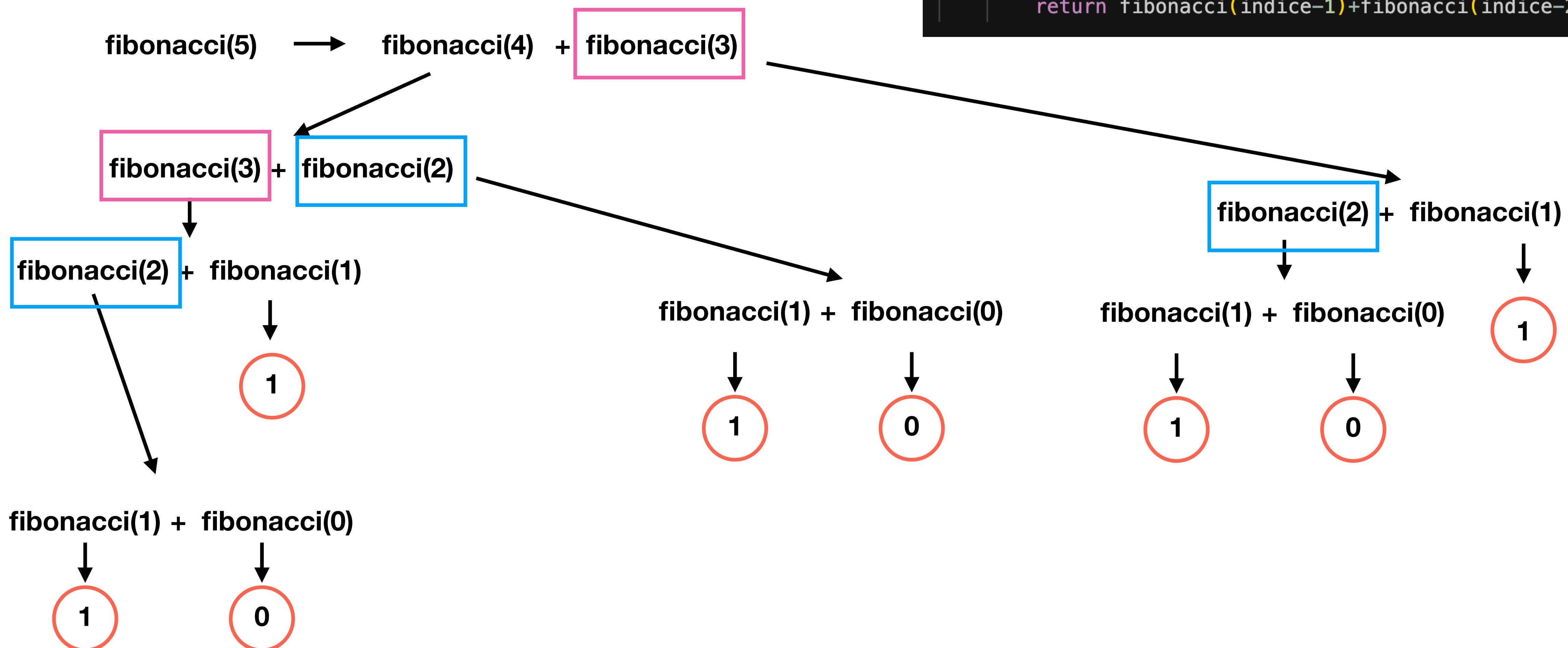
```
start_recursive = time.time()
fibonacci(8)
end_recursive = time.time()
print("total time recursive...", end_recursive - start_recursive)

start_iter = time.time()
fibonacci_iter(8)
end_iter = time.time()
print("total time iterative...", end_iter - start_iter)
```

```
total time recursive... 5.245208740234375e-06
total time iterative... 2.1457672119140625e-06
```



# MEMOIZACIÓN





# MEMOIZACIÓN

Técnica de optimización en la programación en la cual se almacenan en memoria los resultados de una función para evitar recalcularlos en llamadas futuras con los mismos parámetros.

La memoización puede mejorar significativamente el rendimiento de las funciones que realizan cálculos costosos o repetitivos.



# MEMOIZACIÓN - IMPLEMENTACION EXPLICITA

```
fibonacci_cache = {}

def fibonacci_ca(indice):
    #Si tenemos el valor en cache lo devolvemos
    if indice in fibonacci_cache:
        return(fibonacci_cache[indice])

    #Calcular el termino de orden n
    if indice <=1:
        valor = indice
    else:
        valor = fibonacci_ca(indice-1)+fibonacci_ca(indice-2)

    # Guardar el valor en cache y devolverlo
    fibonacci_cache[indice] = valor
    return(valor)
```



# MEMOIZACIÓN - IMPLEMENTACION EXPLICITA

```
fibonacci_cache = {}

def fibonacci_ca(indice):
    #Si tenemos el valor en cache lo devolvemos
    if indice in fibonacci_cache:
        return(fibonacci_cache[indice])

    #Calcular el termino de orden n
    if indice <=1:
        valor = indice
    else:
        valor = fibonacci_ca(indice-1)+fibonacci_ca(indice-2)

    # Guardar el valor en cache y devolverlo
    fibonacci_cache[indice] = valor
    return(valor)
```

```
total time recursive... 5.7220458984375e-06
total time iterative... 3.0994415283203125e-06
total time cache... 2.86102294921875e-06
```



# MEMOIZACIÓN - IMPLEMENTACION IMPLICITA

```
from functools import lru_cache

@lru_cache(maxsize = 20)
def fibonacci_cache_implicito(indice):
    #Calcular el termino de orden n
    if indice <=1:
        return indice
    else:
        return fibonacci(indice-1)+fibonacci(indice-2)

def fibonacci_iter(indice):
    secuencia = [0,1]
    for i in range(indice):
        secuencia.append(secuencia[-1] + secuencia[-2])

    return secuencia[-2]
```

```
total time recursive... 2.47955322265625e-05
total time iterative... 8.344650268554688e-06
total time cache explicito... 6.198883056640625e-06
total time cache implicito... 5.245208740234375e-06
```

CONQUER  
**BLOCKS**