

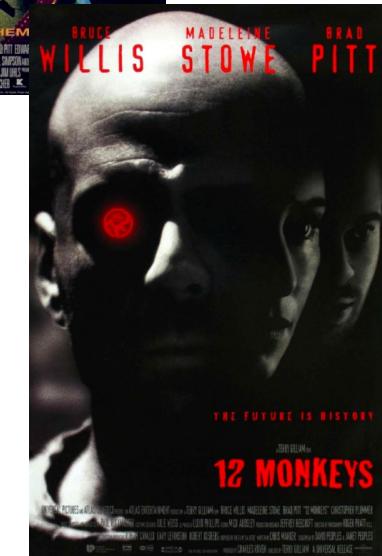
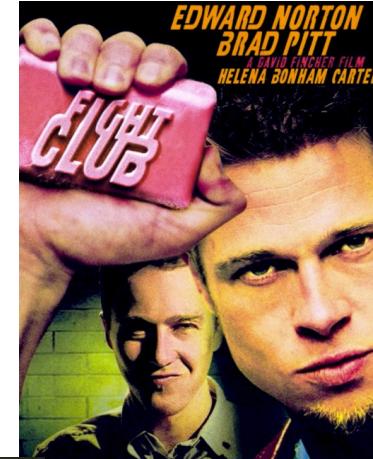
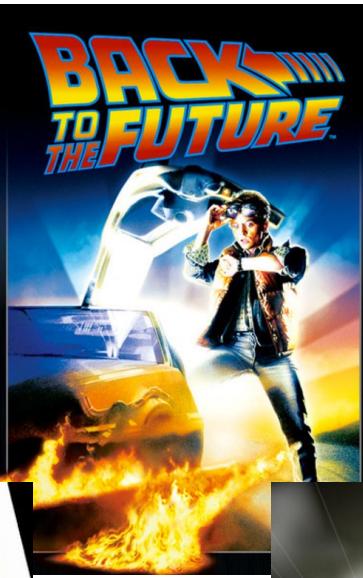
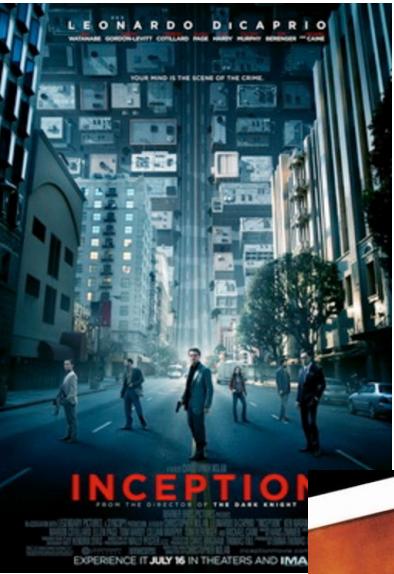
07 – FUNCIONES RECURSIVAS

Jorge Muñoz

IIC1103 – Introducción a la Programación

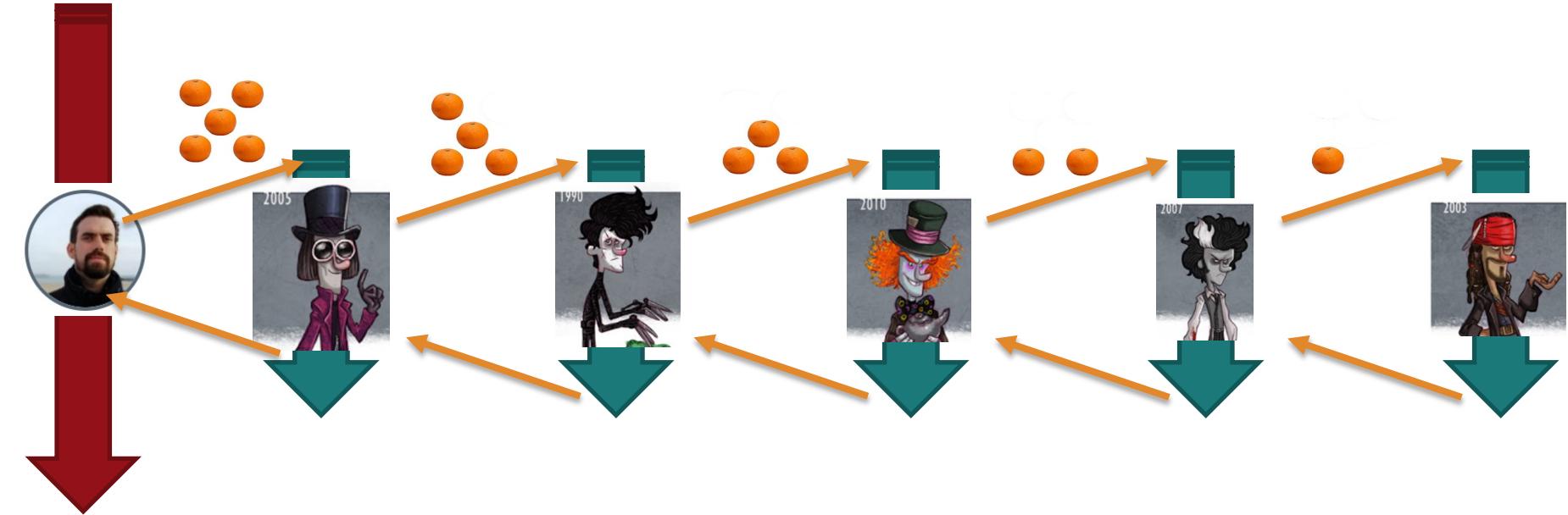
No os voy a mentir. Se nos viene una
clase **intensa**

1. Primera de Python
2. **Funciones recursivas**
3. Primera de Objetos



Pero lo haremos paso a paso ...

NARAJAS DE VALENCIA





#CODIGO
#PRINCIPAL

The diagram illustrates the execution flow of five functions, `f1()` through `f5()`, represented by orange trees. Each tree has a specific number of oranges, which corresponds to the return value of its respective function. The functions are defined as follows:

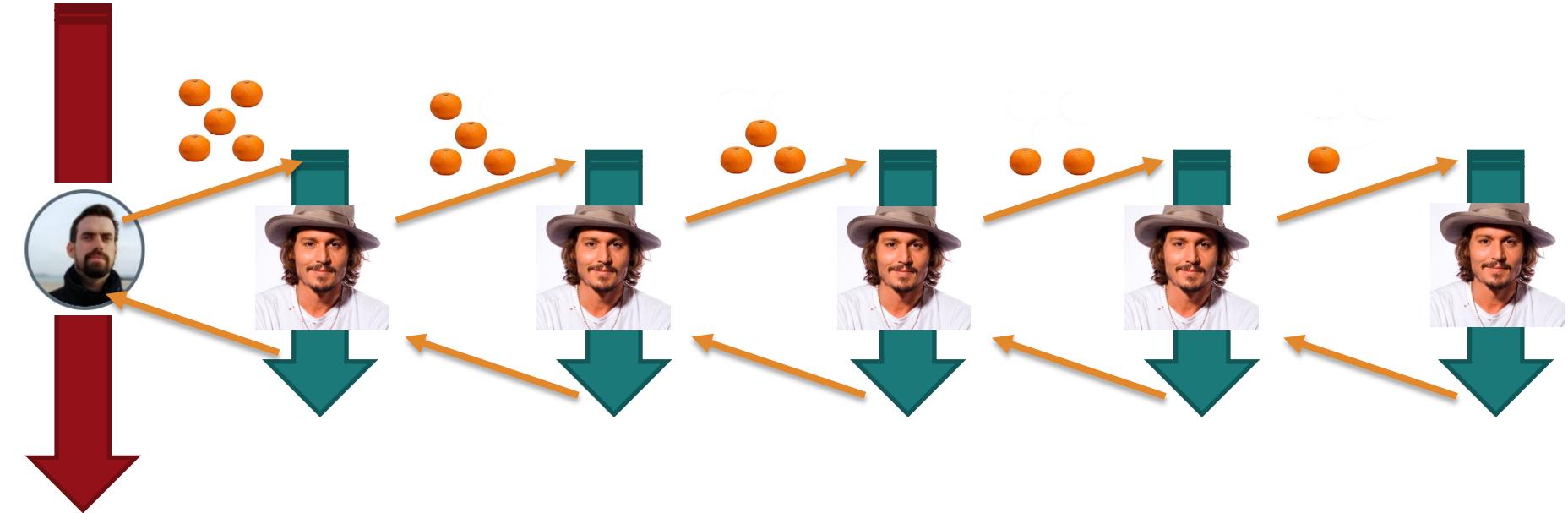
- `f1():` Returns 5 oranges.
- `f2():` Returns 4 oranges.
- `f3():` Returns 3 oranges.
- `f4():` Returns 2 oranges.
- `f5():` Returns 1 orange.

The flow starts with `r = f1()`, followed by `r = f2()`, then `r = f3()`, `r = f4()`, and finally `r = f5()`. The final output is `return ...` for each step.

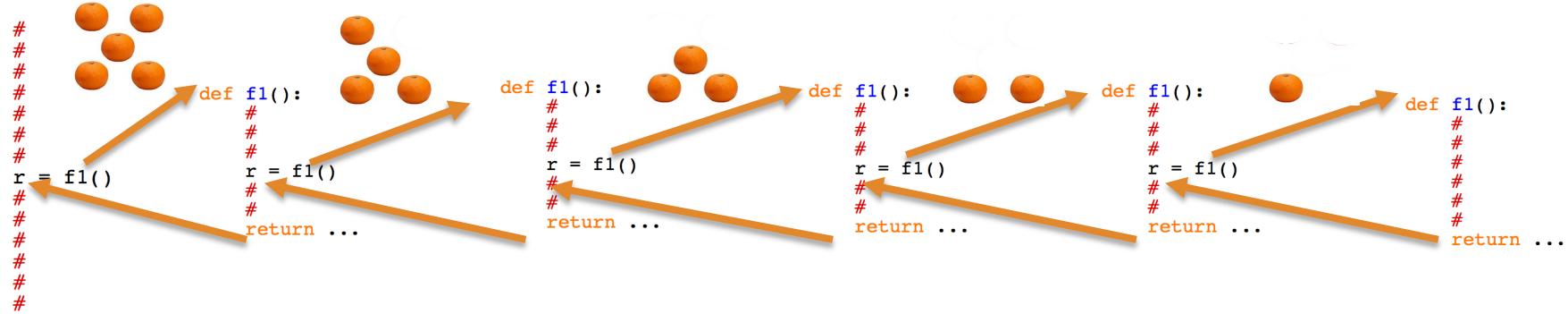
¿Si los 5 hacen exactamente **la misma tarea** (ordenar naranjas), pero cada vez **más pequeño** (menos naranjas) ...

... por que debería tener a **5** colegas y no solo **1**?

Programación: *Por que tengo que escribir **n** funciones pudiendo **escribir solo 1**?*



#CODIGO
#PRINCIPAL



FACTORIAL



$$1! = 1 = 1$$

$$2! = 2 \times 1 = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$



$$1! = 1 = 1$$

$$1! = 1 = 1$$

$$2! = 2 \times 1 = 2$$

$$2! = 2 \times \boxed{1!} = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$3! = 3 \times \boxed{2!} = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$4! = 4 \times \boxed{3!} = 24$$



```
def fact_rec(n):
    if n == 1:
        sol = 1
        return sol
    else:
        sol = n * fact_rec(n-1)
        return sol
```

Creeros que esta es una de esas
funciones “Johnny Depp”

```
120  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                4  
                    4
```

```
24  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                3  
                    6
```

```
3  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                2  
                    2
```

```
2  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                1  
                    1
```

```
1  
def factorial_rec(n):  
    if n == 1:  
        sol = 1  
    return sol
```

```
def fact_rec(n):  
    if n == 1:  
        sol = 1  
    return sol  
else:  
    sol = n * fact_rec(n-1)  
    return sol
```

PARTES DE UNA FUNCIÓN RECURSIVA



#CODIGO PRINCIPAL
ftiempos(5)

```
>>>  
Tiempo 0  
Tiempo 1  
Tiempo 2  
Tiempo 3  
Tiempo 4  
SE ACABO  
>>>
```

#CODIGO PRINCIPAL
ftiempos(10)

```
>>>  
Tiempo 0  
Tiempo 1  
Tiempo 2  
Tiempo 3  
Tiempo 4  
Tiempo 5  
Tiempo 6  
Tiempo 7  
Tiempo 8  
Tiempo 9  
SE ACABO  
>>>
```



```
def ftiempos(n):
    for i in range(0,n):
        print("Tiempo",i)
    print("SE ACABO")
```

#CODIGO PRINCIPAL
ftiempos(10)

```
>>>
Tiempo 0
Tiempo 1
Tiempo 2
Tiempo 3
Tiempo 4
Tiempo 5
Tiempo 6
Tiempo 7
Tiempo 8
Tiempo 9
SE ACABO
>>>
```



... pero sin usar **for in range / while**?





Resolución
Problema

```
def ftiempos(n):  
    print("Tiempo",n)
```

```
#CODIGO PRINCIPAL  
ftiempos(10)
```

```
>>>  
Tiempo 10  
>>>
```





• • •

```
def ftiempos(n):
    print("Tiempo", n)
    ftiempos(n)

#CODIGO PRINCIPAL
ftiempos(10)
```





```
>>>  
Tiempo 10  
Tiempo 11  
Tiempo 12  
Tiempo 13  
Tiempo 14  
Tiempo 15  
Tiempo 16  
Tiempo 17  
Tiempo 18  
Tiempo 19  
Tiempo 20  
Tiempo 21  
Tiempo 22  
Tiempo 23  
Tiempo 24  
Tiempo 25  
Tiempo 26  
Tiempo 27  
Tiempo 28  
Tiempo 29  
Tiempo 30  
Tiempo 31  
Tiempo 32  
Tiempo 33
```

```
def ftiempos(n):  
    print("Tiempo", n)  
    ftiempos(n+1)  
  
#CODIGO PRINCIPAL  
ftiempos(10)
```





```
>>>  
Tiempo 0  
Tiempo 1  
Tiempo 2  
Tiempo 3  
Tiempo 4  
Tiempo 5  
Tiempo 6  
Tiempo 7  
Tiempo 8  
Tiempo 9  
Tiempo 10  
Tiempo 11  
Tiempo 12  
Tiempo 13  
Tiempo 14  
Tiempo 15  
Tiempo 16  
Tiempo 17  
Tiempo 18  
Tiempo 19  
Tiempo 20  
Tiempo 21  
Tiempo 22  
Tiempo 23  
Tiempo 24  
• • •
```

```
def ftiempos(i,n):  
    print("Tiempo",i)  
    ftiempos(i+1,n)  
  
#CODIGO PRINCIPAL  
ftiempos(0,10)
```





```
>>>  
Tiempo 0  
Tiempo 1  
Tiempo 2  
Tiempo 3  
Tiempo 4  
Tiempo 5  
Tiempo 6  
Tiempo 7  
Tiempo 8  
Tiempo 9  
SE ACABO  
>>>
```

```
def ftiempos(i,n):  
    if i == n:  
        print("SE ACABO")  
    else:  
        print("Tiempo",i)  
        ftiempos(i+1,n)  
  
#CODIGO PRINCIPAL  
ftiempos(0,10)
```



Partes de las funciones recursivas

CASO BASE

```
def ftiempos(i,n):
    if i == n:
        print("SE ACABO")
    else:
        print("Tiempo",i)
        ftiempos(i+1,n)
```

El caso más pequeño. Cuando ya has llegado al final

CASO RECURSIVO

```
#CODIGO PRINCIPAL
ftiempos(0,10)
```

Incluye llamada a si mismo, pero cada vez un paso más cerca del final (caso base)

Parámetros **iniciales** para partir

Que pasa si no
ponemos el
caso base?

```
def ftiempos(i,n):
    print("Tiempo",i)
    ftiempos(i+1,n)

#CODIGO PRINCIPAL
ftiempos(0,10)
```

Tiempo 0	Tiempo 16	Tiempo 31	Tiempo 46
Tiempo 1	Tiempo 17	Tiempo 32	Tiempo 47
Tiempo 2	Tiempo 18	Tiempo 33	Tiempo 48
Tiempo 3	Tiempo 19	Tiempo 34	Tiempo 49
Tiempo 4	Tiempo 20	Tiempo 35	Tiempo 50
Tiempo 5	Tiempo 21	Tiempo 36	Tiempo 51
Tiempo 6	Tiempo 22	Tiempo 37	Tiempo 52
Tiempo 7	Tiempo 23	Tiempo 38	Tiempo 53
Tiempo 8	Tiempo 24	Tiempo 39	Tiempo 54
Tiempo 9	Tiempo 25	Tiempo 40	Tiempo 55
Tiempo 10	Tiempo 26	Tiempo 41	Tiempo 56
Tiempo 11	Tiempo 27	Tiempo 42	Tiempo 57
Tiempo 12	Tiempo 28	Tiempo 43	Tiempo 58
Tiempo 13	Tiempo 29	Tiempo 44	Tiempo 59
Tiempo 14	Tiempo 30	Tiempo 45	Tiempo 60
Tiempo 15			



Bibliografía &
Investigación



memesintroalaprogr • Following ...

24w



ingudankmemes

...

Yo: *Olvido poner el caso base
en la función recursiva*

Python:



FACTORIAL (DIY)



$$1! = 1 = 1$$

$$1! = 1 = 1$$

$$2! = 2 \times 1 = 2$$

$$2! = 2 \times \boxed{1!} = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$3! = 3 \times \boxed{2!} = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$4! = 4 \times \boxed{3!} = 24$$



1. Entiendo que tarea tengo que hacer **yo**
2. Pienso en el caso más pequeño, más trivial, más cercano al final => **Caso Base**
3. Me creo que conozco a alguien que hace exactamente **la misma tarea** que yo, pero **más simple**
4. Llama a esa persona para que haga esa tarea pero un poco más simple. **NO te preocupes de cómo lo va a hacer.** No es tu problema
5. Usa el resultado que te ha devuelto para hacer la tarea que te han pedido



1. Entiendo que tarea tengo que hacer **yo**

Calcular el factorial de un número (e.g., 5)

2. Pienso en el caso más pequeño, más trivial, más cercano al final =>

Caso Base

Factorial de 1

3. Me creo que conozco a alguien que hace exactamente **la misma tarea** que yo, pero **más simple**

Alguien que sabe hacer el factorial de un número, pero justo un número anterior al mío (e.g., 4)

4. Llama a esa persona para que haga esa tarea pero un poco más simple. **NO te preocupes de cómo lo va a hacer.** No es tu problema

¡Eh tu, dame el factorial de 4! ¡Búscate la vida!

5. Usa el resultado que te ha devuelto para hacer la tarea que te han pedido

Si tengo el factorial de 4, solo tengo que multiplicarlo por 5 para devolver lo que me han pedido (factorial de 5)



```
def fact_rec(n):
    if n == 1:
        sol = 1
        return sol
    else:
        sol = n * fact_rec(n-1)
    return sol
```

```
120  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                4  
                    4
```

```
24  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                3  
                    6
```

```
3  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                2  
                    2
```

```
2  
def factorial_rec(n):  
    if n == 1:  
  
        else:  
            sol = n * factorial_rec(n-1)  
                1  
                    1
```

QUESTION PAGE

```
def fact_rec(n):  
    if n == 1:  
        sol = 1  
        return sol  
    else:  
        sol = n * fact_rec(n-1)  
        return sol
```

ARQUITECTURA DE VON NEWMAN



¿Has logrado llegar hasta aquí?
¡FELICIDADES!

Creo que te has ganado un pequeño
momento de relax

¿Qué tal unos
videos de
adorables
conejos para
relajarte?



Bibliografía &
Investigación

No compras!
ADOPTA

bunnylovers.cl
(ONG chilena)

WWW.BUNNYLOVERS.CL

**NO
COMPRES!
#ADOPTA**



BUNNY
LOVERS



Bibliografía &
Investigación



¿Y por que “Arquitectura de von Newman”?

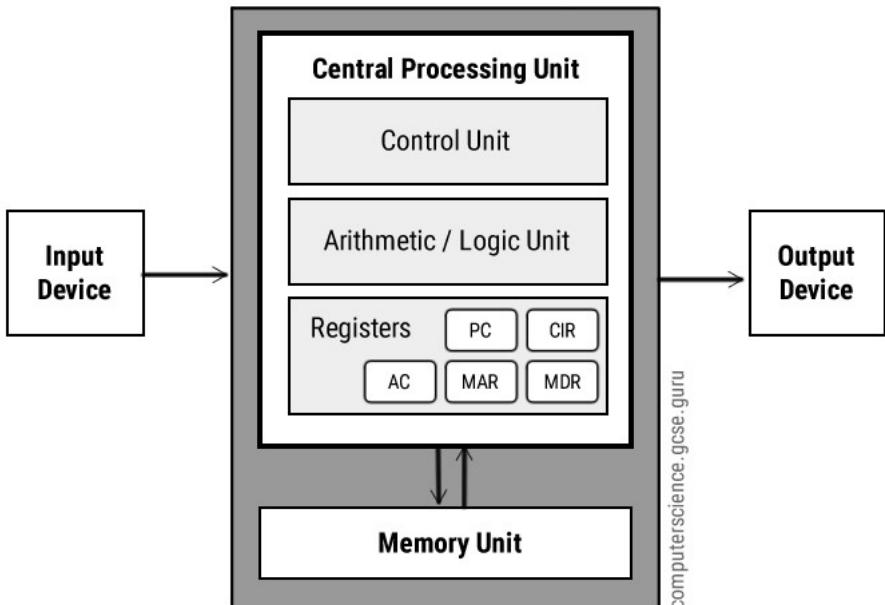
Era para asustar un poco

Pero ya que estamos ...



John von
Newman 1945

Arquitectura de
los computadores





Final feliz de Stu &
Paprika

bunnycarecl



Bunny Lovers Chile
· 2 de octubre de 2019 ·

...

#Repost @bunnycarecl

• • • •

Era un día como hoy, 2 de octubre, y Stu perseguía una pelota en el patio sin saber que su vida estaba a punto de cambiar... Este conejito, de edad desconocida, unos meses antes había sido rescatado y en aquel momento vivía en un hogar de acogida. Ese mismo día una pareja lo adoptó para entregarle todo su amor. No tardó en acostumbrarse a su nueva casa, aunque en un principio un poco austera, con el paso del tiempo llegaron su henera, y su castillo, del que Stu se volvió rey.

Sin embargo, faltaba algo porque un castillo no puede estar completo si no hay una reina. Y esa reina era Paprika, por naturaleza devoradora de zanahorias. Había nacido dos meses antes, junto a sus tres hermanos. Ella es hija de Pepper, una conejita que fue rescatada estando embarazada. Paprika con su curiosidad y sus besitos de buena coneja se ganó el corazón de la pareja. Por cosas del destino Paprika nació en una fecha muy especial... un 2 de octubre.

Como es natural hubieron roces al principio, persecuciones por aquí, gruñidos por allá. Pero en menos de una semana esos roces pasaron a ser muestras de cariño, groomings constantes y flops uno al lado del otro. A ambos les gusta perseguir, dando saltos, a las personas cuando creen que tienen comida.

La adopción de Stu y Paprika fue la decisión más acertada del mundo. En los días más duros y tristes llenan el hogar de amor, calidez, risas y juegos. Para esta pareja a partir de la adopción de sus orejones, los 2 de octubre siempre serán especiales. #bunnycare #adopcioneschile #adopciones #rescate #tenenciasresponsable #conejos #conejoschile



68

5 comentarios



Escribe un comentario...



BunnyLovers

- <https://bunnylovers.cl/>

Arquitectura von Newman

- <https://www.computerscience.gcse.guru/theory/von-neumann-architecture>
- https://en.wikipedia.org/wiki/Von_Neumann_architecture

FIBONACCI

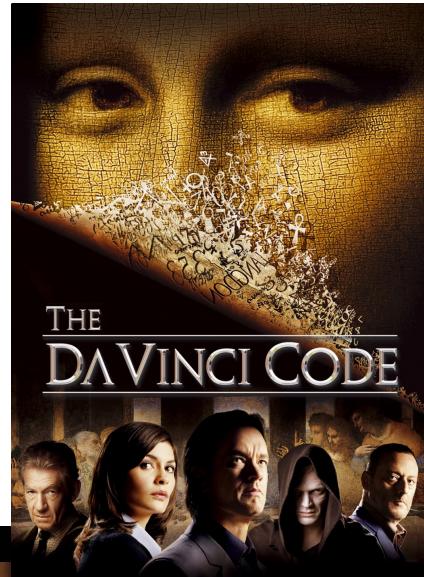


Situación
Real

The Fibonacci Sequence



This sequence can continue on forever...





Situación
Real

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

FIB



```
x = fib_it(3)  
print(x)
```

>>>

2

```
y = fib_it(4)  
print(y)
```

3

5

```
z = fib_it(5)  
print(z)
```

>>>





```
x = fib_it(3)  
print(x)  
  
y = fib_it(4)  
print(y)  
  
z = fib_it(5)  
print(z)
```

```
>>>  
2  
3  
5  
>>>
```

```
def fib_it(n):  
    if n == 1:  
        s = 1  
  
    elif n == 2:  
        s = 1  
  
    else:  
        n1 = 1  
        n2 = 1  
        fib = n1+n2  
        i = 3  
        while i < n:  
            n1 = n2  
            n2 = fib  
            fib = n1 + n2  
            i += 1  
        s = fib  
  
    return s
```



Es de los únicos algoritmos que la versión recursiva tiene 2 casos bases



```
x = fib_rec(3)
print(x)

y = fib_rec(4)
print(y)

z = fib_rec(5)
print(z)
```

```
>>>
2
3
5
>>>
```



```
def fib_rec(n):

    #CASO BASE 1
    [REDACTED]

    #CASO BASE 2:
    [REDACTED]

    #CASO RECURSIVO
    [REDACTED]
```



1. Entiendo que tarea tengo que hacer **yo**

Calcular el i -esimo numero de FIB (e.g., 5º)

2. Pienso en el caso más pequeño, más trivial, más cercano al final =>
Caso Base

1º FIB es 1, 2º FIB es 1 (dos casos más pequeños)

3. Me creo que conozco a alguien que hace exactamente **la misma tarea** que yo, pero **más simple**

Alguien que sabe hacer el i -esimo FIB, pero solo números más pequeños que el mío (e.g., 4º y el 3º)

4. Llama a esa persona para que haga esa tarea pero un poco más simple. **NO te preocupes de cómo lo va a hacer.** No es tu problema

¡Eh tu, dame el 4º FIB! ¡Y tu, el 3º! ¡Buscaros la vida!

5. Usa el resultado que te ha devuelto para hacer la tarea que te han pedido

Si tengo el 4º FIB y el 3º FIB, solo tengo que sumarlos para devolver lo que me han pedido (el 5º FIB)



```
x = fib_rec(3)
print(x)

y = fib_rec(4)
print(y)

z = fib_rec(5)
print(z)
```

```
>>>
2
3
5
>>>
```



```
def fib_rec(n):

    #CASO BASE 1
    if n == 1:
        s = 1
    #CASO BASE 2:
    elif n == 2:
        s = 1
    #CASO RECURSIVO
    else:
        s = fib_rec(n-1)+fib_rec(n-2)

    return s
```



- Fibonacci
 - https://en.wikipedia.org/wiki/Fibonacci_number
- FIB-UPC / UC (Convenio)
 - <https://www.ing.uc.cl/noticias/ingenieria-uc-materializa-acuerdo-de-intercambio-de-estudiantes-con-facultat-dinformatica-de-barcelona/>
- FIB-UPC Ramos
 - <https://www.fib.upc.edu/es/estudios/grados/grado-en-ingenieria-informatica/plan-de-estudios/asignaturas>



INGENIERÍA UC MATERIALIZA ACUERDO DE
INTERCAMBIO DE ESTUDIANTES CON FACULTAT
D'INFORMÀTICA DE BARCELONA



Barcelona acogerá el superordenador europeo más veloz

Europa destina 100 millones al Centro Nacional de Computación, su mayor aportación a una infraestructura de investigación española