Recursión – I

Clase #23

IIC1103 – Introducción a la Programación

Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

- Concepto de recursión
- Comparación con métodos iterativos
- Ejemplos

¿Qué es recursión?

- Es una función que se define en términos de sí misma.
  - Ejemplo:
    El valor de la función para el argumento **n** puede definirse en términos del argumento **n-1** (o alguno anterior).

- Siempre debe existir un **caso base**.

- Ejemplo:

$$n! = \begin{cases} 1, & \text{si } n = 0 \\ n \cdot (n-1)!, & \text{si } n > 0 \end{cases}$$

Caso Base

Caso Recursivo

Factorial recursivo en Python

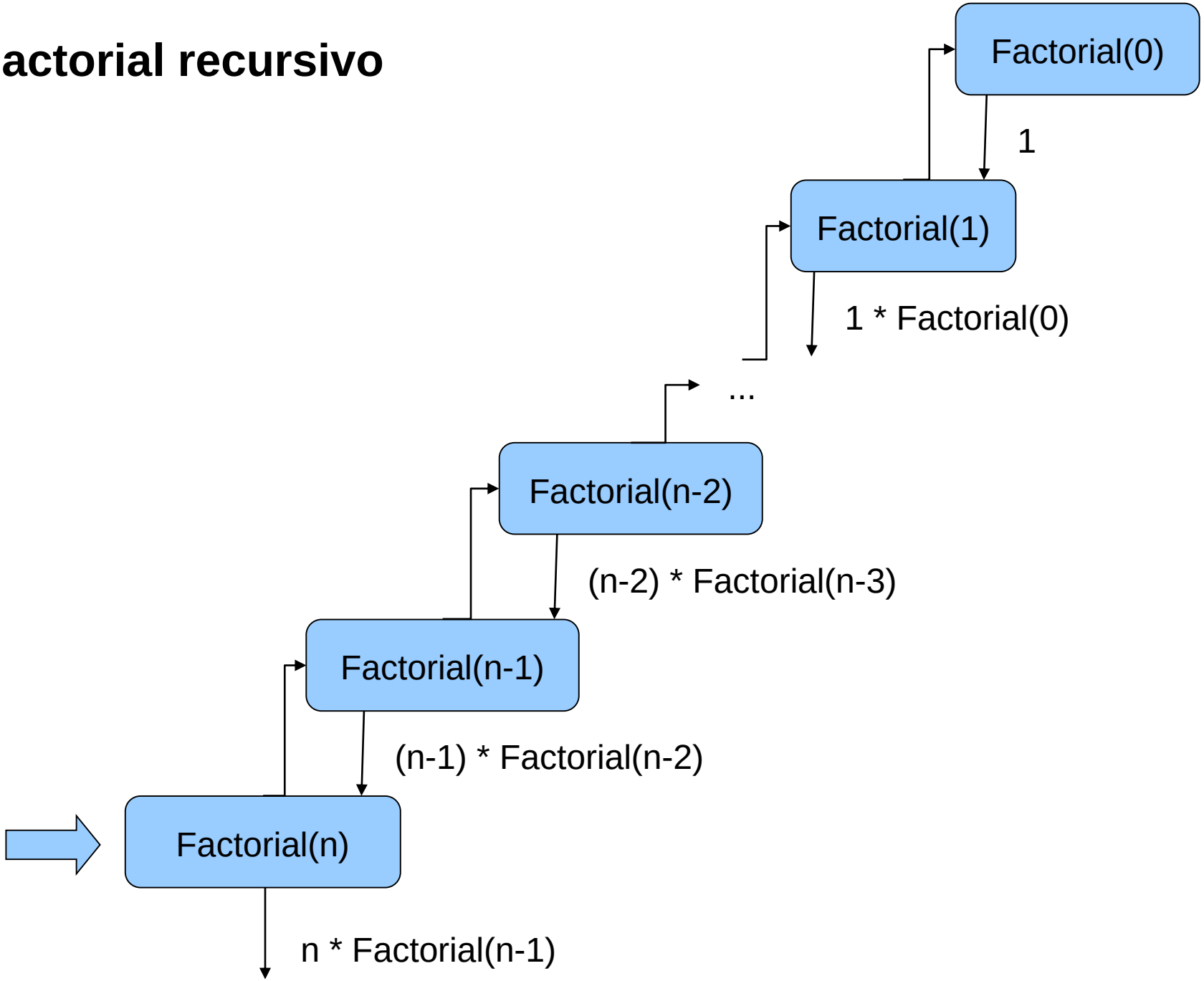
```
def Factorial(n):
    if (n == 0):
        return 1
    else:
        return n * Factorial(n - 1)

print("5! =", Factorial(5))
```

Caso Base

Caso Recursivo

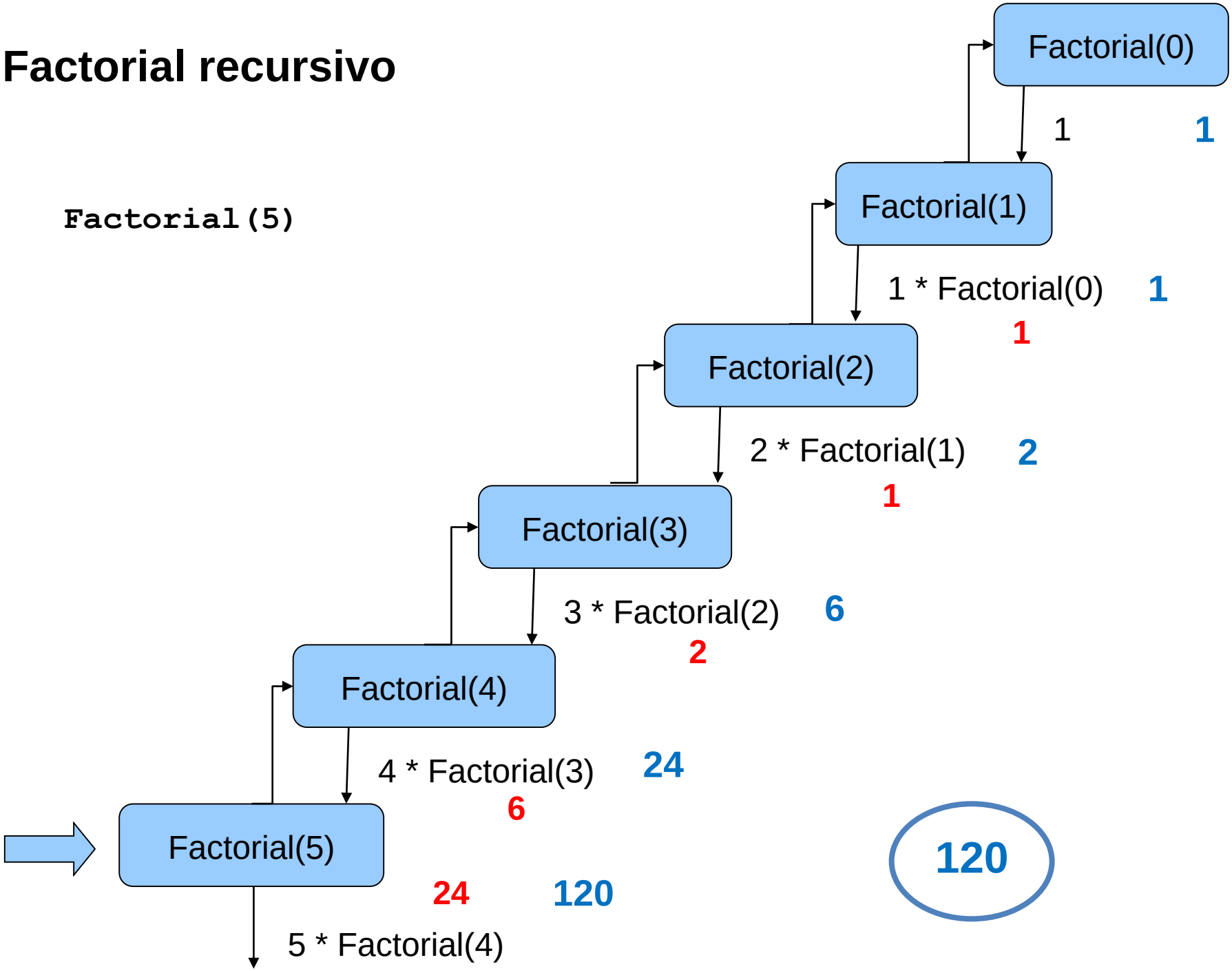
```
>>>
5! = 120
```

Factorial recursivo



Factorial recursivo

`Factorial(5)`



Es posible eliminar la recursión, y resolver de manera iterativa

```
def Factorial(n):
    fact = 1
    for i in range(2,n+1):
        fact = fact * i
    return fact

print("5! =", Factorial(5))
```

```
>>>
5! = 120
```

- Eliminar la recursión, normalmente tiene dos efectos contrapuestos:
  - Aumenta la eficiencia (menos tiempo de ejecución)
  - Dificulta la programación (más tiempo de programación)

Fibonacci con y sin recursión

- F(0)=0
- F(1)=1
- F(n)=F(n-1)+F(n-2)

Recursivo

```
def Fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return Fibonacci(n-1) + \
               Fibonacci(n-2)
```

Iterativo

```
def Fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        fib_n2 = 0
        fib_n1 = 1
        for i in range(2,n+1):
            fib_i  = fib_n1 + fib_n2
            fib_n2 = fib_n1
            fib_n1 = fib_i
        return fib_i
```

```
>>> for i in range(0,11):
      print(Fibonacci(i), end=',')
    print("")

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

Fibonacci en la naturaleza

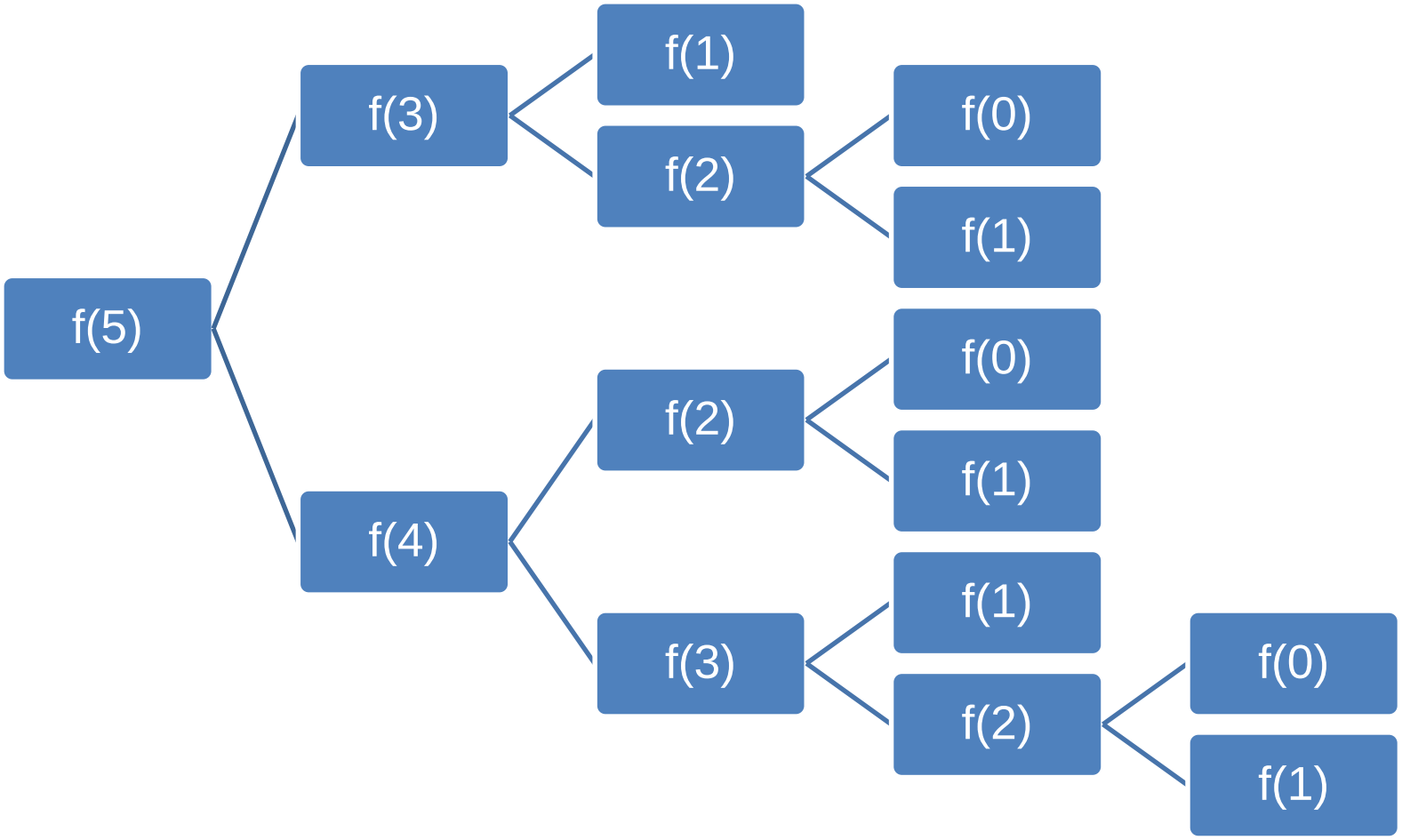
Los números de Fibonacci en el mundo real

Los números de Fibonacci son bastante curiosos, pues aparecen espontáneamente en la naturaleza. Te presentamos algunos ejemplos:

- Las abejas comunes viven en colonias. En cada colonia hay una sola reina (hembra), muchas trabajadoras (hembras estériles), y algunos zánganos (machos). Los machos nacen de huevos no fertilizados, por lo que tienen madre, pero no padre. Las hembras nacen de huevos fertilizados y, por tanto, tienen padre y madre. Estudiemos el árbol genealógico de **1** zángano: tiene **1** madre, **2** abuelos (su madre tiene padre y madre), **3** bisabuelos, **5** tatarabuelos, **8** tatara-tatarabuelos, **13** tatara-tatara-tatarabuelos... Fíjate en la secuencia: 1, 1, 2, 3, 5, 8, 13... A partir del tercero, cada número se obtiene sumando los dos anteriores. Esta secuencia es la serie de Fibonacci.
- Muchas plantas tienen un número de pétalos que coincide con esa secuencia de números: la flor del iris tiene 3 pétalos, la de la rosa silvestre, 5 pétalos, la del dephinium, 8, la de la cineraria, 13, la de la chicoria, 21... Y así sucesivamente (las hay con 34, 55 y 89 pétalos).
- El número de espirales cercanas al centro de un girasol que van hacia a la izquierda y las que van hacia la derecha son, ambos, números de la secuencia de Fibonacci.
- También el número de espirales que en ambos sentidos presenta la piel de las piñas coincide con sendos números de Fibonacci.

Fuente: “Introducción a la programación con Python y C”, Andrés Marzal Varó, Isabel Gracia Luengo, 2002

Problema de Fibonacci recursivo

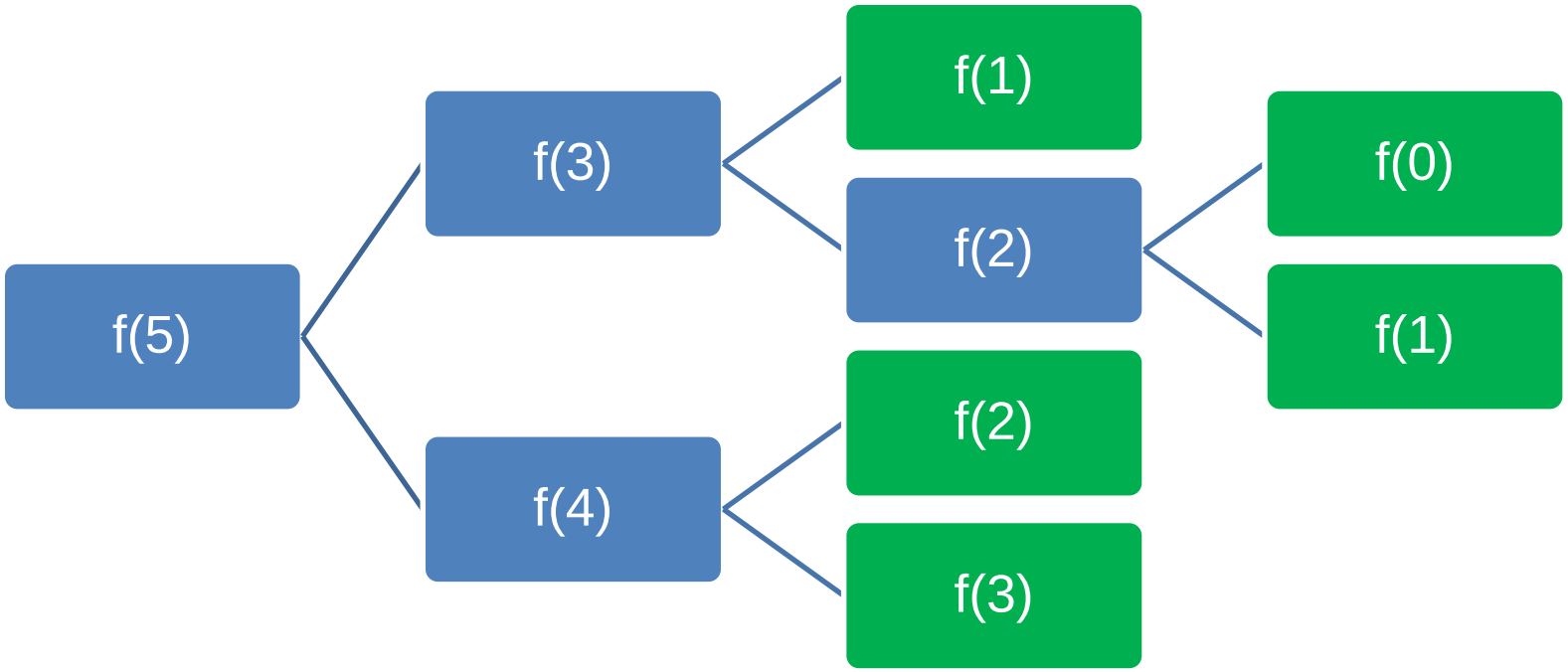


Solución – Fibonacci recursivo con memoria

```
# lista que almacena número de Fibonacci de acuerdo a su índice
memoria = [0, 1]

def Fibonacci_M(n):
    global memoria # no es estrictamente necesario usar 'global'
    if n >= len(memoria):
        memoria.append(Fibonacci_M(n-1) + Fibonacci_M(n-2))
    return memoria[n]
```

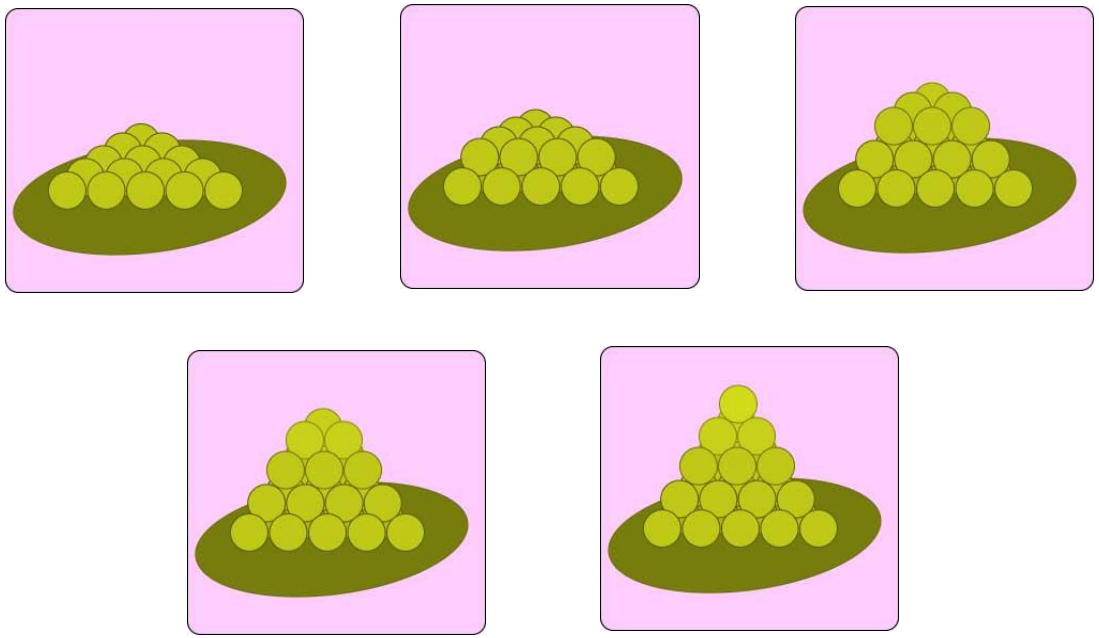
```
>>> for i in range(0,11):
      print(Fibonacci_M(i), end=',')
    print("")

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

Fibonacci con memoria



Ejercicio – mini control n°4 – 2015.2

- El ayudante del curso ha recurrido en forma desesperada a ti porque le falta completar la última clase de su programa para apoyar Torneos de Tenis. La clase se llama **`ContarPelotas`** y se necesita para una singular tarea, contar las pelotas utilizadas durante cada torneo. Al final de cada partido, los peloteros van acumulando las pelotas en una pirámide, tal como se muestra en el siguiente esquema:



Ejercicio – mini control n°4 – 2015.2

- Suponiendo que al final de cada torneo la pirámide está completa, se te pide que agregues a la clase **`ContarPelotas`** dos métodos, llamados **`Triangulo`** y **`Piramide`**. El primero cuenta la cantidad de pelotas que hay en un triángulo de una cierta base, y el segundo la cantidad de pelotas que hay en una pirámide de una cierta cantidad de pisos. Por ejemplo, en el caso anterior, la pirámide tiene 5 pisos y tiene:
  - 15 pelotas en el triángulo inferior (de base 5) +
  - 10 pelotas en el segundo triángulo (de base 4) +
  - 6 pelotas en el tercer triángulo (de base 3) +
  - 3 pelotas en el cuarto triángulo (de base 2) +
  - 1 pelota en el triángulo superior (de base 1)
  - En total, 35 pelotas.

- La única restricción es que ambos métodos **DEBEN SER RECURSIVOS**.

Ejercicio – mini control n°4 – 2015.2

- Para ilustrar el comportamiento deseado, ya se ha escrito parte del código que utiliza esta clase:

```
pisos = int(input("¿Cantidad de pisos de la pirámide? "))

cp = ContarPelotas()
resultado = cp.Piramide(pisos)
print("Una pirámide de pelotas de", pisos, "pisos,", \
      "tiene", resultado, "pelotas.")
```

- El resultado sería:

```
>>>
¿Cantidad de pisos de la pirámide? 5
Una pirámide de pelotas de 5 pisos, tiene 35 pelotas.
```

Recursión – en resumen …

- La recursión consiste en funciones que se llaman a sí mismas

- **Ventajas**
  - Sirve para plantear de una forma sucinta y elegante problemas que de otra forma serían complejos de plantear.
  - La ventaja de dividir el problema principal en otros más pequeños es que la solución es, por lo general, más fácil de pensar, y el código para ésta es generalmente mucho más corto que la solución iterativa.

- **Desventajas**
  - No son eficientes en el uso de memoria ni en la velocidad de procesamiento