DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

RECURSIÓN

10145 - FUNDAMENTOS DE PROGRAMACIÓN PARA INGENIERÍA
10110 – FUNDAMENTOS DE COMPUTACIÓN Y PROGRAMACIÓN

RESUMEN DE CONTENIDOS

RECURSIÓN

- La recursión ocurre cuando una cosa se define en términos de su propia definición

Etiqueta recursiva de *Royal Baking Powder*
Fuente: usachvirtual.cl

RECURSIÓN

- La recursión permite realizar ciclos a través de **condicionar la salida de una función** al cumplimiento de una **condición**

- En caso de no cumplirse la condición, se ejecuta nuevamente la **función completa**

- Esto genera una **nueva instancia de ejecución** de la función

RECURSIÓN

- Para construir una función recursiva se requiere por lo menos:
  - Un caso “pequeño”, “conocido” o “base” que **pueda ser resuelto directamente** o cuya **solución sea conocida**
  - Una forma de **descomponer** el problema en:
    - **Uno o más problemas** más pequeños
    - Que puedan **reducirse de la misma forma** que el original
  - Una estrategia para **combinar las soluciones parciales** en la solución general
- En general las soluciones recursivas son **más elegantes**, **pero menos eficientes**

FUNCIONES RECURSIVAS

- Para que una función sea recursiva debe poseer:

Regla de recursión:
<Llamado de la función a sí misma>

Condición de borde:
<Instrucción que detiene la recursión>

- Se debe ser extremadamente cuidadoso para poner una condición de borde que siempre sea **alcanzable**
  - De lo contrario la función se llamará a si misma **hasta que la memoria se agote**

CONSIDERACIONES PARA LA RECUSIÓN

- Las funciones recursivas también en algunos casos pueden ser más lentas que su alternativa iterativa, porque podrían resolver más de una vez el mismo caso, por ejemplo, la sucesión de Fibonacci:

$$Fibonacci(n) = \begin{cases} 0 & , si\ n = 0 \\ 1 & , si\ n = 1 \\ Fibonacci(n-1) + Fibonacci(n-2) & , si\ n > 1 \end{cases}$$

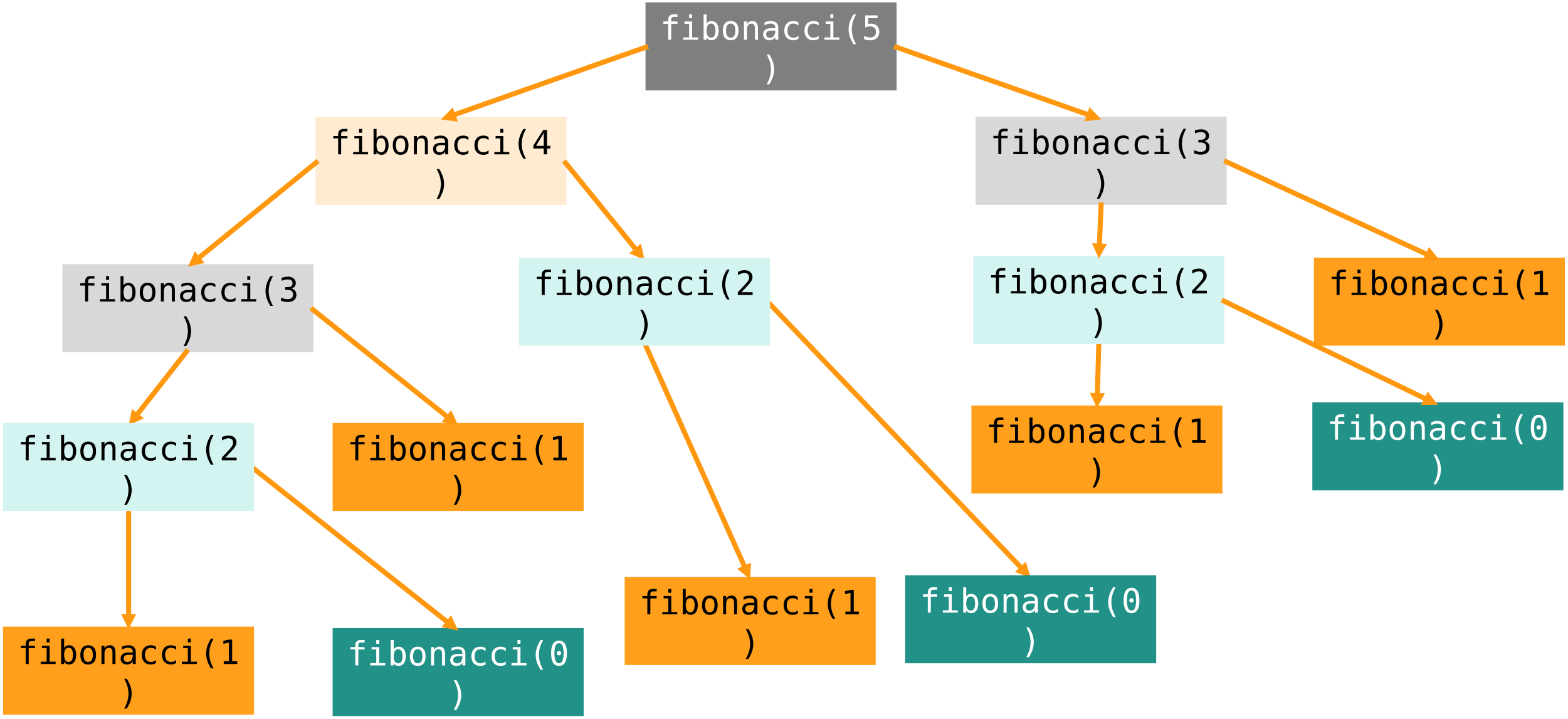
- Podría fácilmente definirse de forma recursiva, sin embargo, revisemos que pasa con *Fibonacci(5)*

TRAZA DE FIBONACCI

```
def calcular_fibonacci(n):
    if n == 0 :
        return 0
    elif n == 1 :
        return 1
    return calcular_fibonacci(n - 1) + calcular_fibonacci(n
- 2)
```

- Consideremos que cada vez que no caemos en una condición de borde, invocaremos **dos instancias nuevas** de la función `calcular_fibonacci()`

TRAZA DE FIBONACCI



CONCLUSIÓN

- En general, las soluciones recursivas, a diferencia de la iteración con while o for-in, pueden generalizarse como:
  - Si el caso que estoy revisando es **suficientemente simple**:
    - Resolverlo directamente
  - Sino:
    - Dividirlo en **problemas más pequeños**
    - Resolver **cada problema** recursivamente
    - Ordenar los resultados parciales para que **combinados entreguen la solución al problema** completo

EJERCICIOS

EJERCICIO 1

- Construya una función en Python que calcule recursivamente la reducción según la Conjetura de Collatz. Recordemos que las reglas de esta eran:

$$collatz(n) = \begin{cases} 1 \; si \; n = 1 \\ \frac{n}{2}, si \; n \; es \; par \\ 3n + 1, si \; n \; es \; impar \end{cases}$$

- Esto considerando que $\frac{n}{2}$ corresponde a division entera y que las entradas siempre serán enteros positivos

EJERCICIO 2

Juanito es un lingüista que está investigando las redundancias en el lenguaje, para ello está intentando reducir las palabras a su mínima expresión, para ello ha consultado en sus referencias y un algoritmo que podría servirle es el de reducción de strings, el cuál es un algoritmo sencillo que funciona sólo con un par de reglas:

- Se pueden eliminar cualquier par de letras adyacentes, siempre y cuando estas sean iguales.
- Mientras existan dos letras iguales adyacentes, se deben seguir eliminando los pares hasta que no quede ningún par de letras iguales adyacentes.
  - Por ejemplo:
    - Si la entrada fuese “murcielago”, el resultado sería “murcielago”, pues no hay letras iguales adyacentes para eliminar.
    - Si la entrada fuese “aaabccddd”, el resultado sería “abd” pues:
      - Se eliminan las “aa” del inicio, quedando “abccddd”
      - Se eliminan las “cc” del medio, quedando “abddd”
      - Se elimina un par de “d”s, quedando finalmente “abd” que no puede reducirse más
    - Si la entrada fuese “aabccbaa”, el resultado sería un string vacío pues:
      - Se eliminan las “aa” del inicio, quedando “bccbaa”
      - Se eliminan las “cc” del medio, quedando “bbaa”
      - Se eliminan las “bb” del inicio, quedando “aa”
      - Se eliminan las “aa” restantes, quedando un string vacío

- Construya una función recursiva que implemente este algoritmo

TAREAS PARA TRABAJO AUTÓNOMO

- Cuándo buscamos un elemento en un conjunto previamente ordenado, no tiene sentido comenzar desde el inicio y buscar secuencialmente hasta el final un elemento, sino que la estrategia más eficiente es buscar desde el medio y en base al elemento del medio descartar una mitad y seguir con la mitad en la que podría estar el elemento. A esta estrategia se le conoce como **búsqueda binaria**

- Implementa una función que realice la **búsqueda binaria** de un número entero en una lista ordenada de números utilizando **recursión**, considera que el número podría no estar en la lista y que el programa debería indicarlo

- Para tu desarrollo considera que en Python existe el operador de corte (*slicing*) con sintaxis `lista[inicio:fin:salto]`

¿CONSULTAS?