

# Recursividad

## Programación recursiva

Jonatan Gómez Perdomo, Ph.D.

[jgomezpe@unal.edu.co](mailto:jgomezpe@unal.edu.co)

Arles Rodríguez, Ph.D.

[aerodriguezp@unal.edu.co](mailto:aerodriguezp@unal.edu.co)

Camilo Cubides, Ph.D.(c)

[eccubidesg@unal.edu.co](mailto:eccubidesg@unal.edu.co)

Grupo de investigación en vida artificial – Research Group on Artificial Life – (Alife)

Departamento de Ingeniería de Sistemas e Industrial

Facultad de Ingeniería

Universidad Nacional de Colombia



# Agenda

- 1 **Introducción**
- 2 Programación recursiva
- 3 Ejemplos de funciones recursivas
- 4 Algunos problemas clásicos de programación recursiva
- 5 Teorema fundamental de la programación recursiva



# Introducción I

Aunque a muchos el término *recursión* les parece extraño, la recursión es un concepto muy natural, tanto como para definir los números naturales.

Pensemos cuando tenemos un problema en nuestro diario vivir y un familiar muy respetado por nosotros dice “sea recursivo”, ¿Qué nos quieren decir?

Que nos miremos a nosotros mismos, que sólo contemos con nuestros propios recursos para solucionar nuestros problema, que no se necesita algo adicional. Que pensemos que lo podemos solucionar para que lo solucionemos.



# Agenda

- 1 Introducción
- 2 Programación recursiva**
- 3 Ejemplos de funciones recursivas
- 4 Algunos problemas clásicos de programación recursiva
- 5 Teorema fundamental de la programación recursiva



# Recursividad I

Dado que la mayoría de los lenguajes de programación disponen de un mecanismo para definir funciones (son estructurados), y dentro de la definición del cuerpo de las funciones se puede invocar el llamado a funciones (lo que se llama **composición de funciones** en matemáticas), una pregunta natural que uno se puede hacerse es: ¿se puede invocar dentro de una función a la misma función para definirse?, en algunos lenguajes de programación esto si se puede hacer, a estos se les denominan lenguajes de programación con recursividad. Por ejemplo: C++, Java, Matlab, Python, Lisp, y la mayoría de lenguajes de programación.



# Recursividad II

## El mecanismo de la recursión

Si un problema de cierto tamaño  $T$  puede ser solucionado usando instancias del mismo problema pero de menor tamaño  $t$  ( $t < T$ ), y además se conoce la solución de algunas instancias de menor tamaño ( $t_0$ ) que no dependan del problema, entonces se puede aplicar un mecanismo recursivo para implementar la solución del problema usando un lenguaje de programación.



# Recursividad III

## Definición (Definición débil de función recursiva)

Una función  $f : A \rightarrow B$  se dice **recursiva** si y sólo si  $f$  está definida por casos (mediante un predicado sobre los argumentos), en donde al menos uno de los casos se define usando la misma función  $f$  y los argumentos, y al menos uno de los otros casos se define usando solamente los argumentos sin involucrar la función  $f$ .

Aquellos casos en los cuales se invoca a la misma función se llaman **casos recursivos**, y aquellos casos en los cuales no interviene la función se denominan **casos base**.

## Observación

En teoría una función no necesita tener casos base, pero cualquier función recursiva escrita sin casos base generará un bucle infinito (ciclo sin fin).

# Recursividad IV

## Metodología para resolver problemas recursivamente

Solucionar problemas, y en particular programar, de manera “recursiva” se puede alcanzar siguiendo las tres máximas del profesor Jonatan Gómez.

- 1 Piense que está bien para que esté bien.
- 2 Siempre se es el primero y el último.
- 3 Un problema es simple o se puede dividir en problemas que se pueden solucionar siguiendo las tres máximas del profesor Jonatan Gómez.





# Introducción

## Problema 1.1

### Problema

Calcular la suma de los números naturales desde el 0 hasta  $n$ .

### Solución

**Máximas dos y tres:** El problema se puede solucionar si se pueden sumar los números naturales desde el 0 hasta en  $n - 1$  pues solo sería sumar ese número con el número  $n$ .

**Máxima uno:** Se supone que la función que suma los números naturales desde el 0 hasta el  $n$  está bien por lo tanto se puede usar para sumar los números naturales desde el 0 hasta el  $n - 1$ .

**Máximas dos y tres:** Sumar los números naturales desde el 0 hasta el 0 es fácil, se debe retornar 0 pues es el primer número natural. De esta manera será el último número en ser considerado en la suma.

# Introducción

## Problema 1.2

### Solución (continuación)

$sumar : \mathbb{N} \rightarrow \mathbb{N}$

$$(n) \mapsto \begin{cases} 0, & \text{si } n = 0; \\ n + sumar(n - 1), & \text{si } n > 0. \end{cases}$$



# Introducción

## Problema 2.1

### Problema

Calcular la suma de los números almacenados en una lista.

### Solución

**Máximas dos y tres:** Supongamos que la lista tiene  $n$  números. El problema se puede solucionar si se pueden sumar los primeros  $n - 1$  números en la lista pues sólo sería sumar ese número con el número que está en la posición  $n - 1$ .

**Máxima uno:** Se supone que la función que suma de los primeros  $n$  números de una lista está bien por lo tanto se puede usar para sumar los primeros  $n - 1$  números en la lista.

**Máximas dos y tres:** Sumar los primeros 1 números de la lista es fácil, se debe retornar el número en la posición 0 pues es el primer número. De esta manera será el último número en ser considerado en la suma.

# Introducción

## Problema 2.2

### Solución (continuación)

$$\text{sumar\_parcial} : \mathbb{N}^* \times \mathbb{N}^+ \rightarrow \mathbb{N}$$

$$(L, n) \mapsto \begin{cases} L[0], & \text{si } n = 1; \\ L[n-1] + \text{suma\_parcial}(L, n-1), & \text{si } n > 1. \end{cases}$$

$$\text{sumar\_lista} : \mathbb{N}^* \rightarrow \mathbb{N}$$

$$(L) \mapsto \text{sumar\_parcial}(L, \text{len}(L))$$



# Introducción

## Problema 3.1

### Problema

Determinar si un carácter está en una cadena.

### Solución

**Máximas dos y tres:** Supongamos que la cadena tiene  $n$  caracteres. El problema se puede solucionar si se puede determinar si un carácter está entre los primeros  $n - 1$  caracteres pues sólo sería mirar si es el último o está entre los primeros  $n - 1$  caracteres.

**Máxima uno:** Se supone que la función que determina si un carácter está entre los primeros  $n$  está bien por lo tanto se puede usar para determinar si está entre los primeros  $n - 1$  caracteres.



# Introducción

## Problema 3.2

### Solución (continuación)

**Máximas dos y tres:** Determinar si está entre los primeros 1 caracteres de la cadena es fácil, se debe retornar verdadero si el carácter es el que está en la posición 0 y falso en otro caso. De esta manera será el último carácter en ser considerado en la búsqueda.

$$\text{buscar} : \text{ASCII}^* \times \text{ASCII} \times \mathbb{N}^+ \rightarrow \mathbb{B}$$

$$(str, ch, n) \mapsto \begin{cases} str[0] == ch, & \text{si } n = 1; \\ (str[n-1] == ch) \vee \text{buscar}(str, ch, n-1), & \text{si } n > 1. \end{cases}$$



# Agenda

- 1 Introducción
- 2 Programación recursiva
- 3 Ejemplos de funciones recursivas**
- 4 Algunos problemas clásicos de programación recursiva
- 5 Teorema fundamental de la programación recursiva



## Ejemplo

Sea  $f$  la función definida por:

$$f : \mathbb{N} \rightarrow \mathbb{B}$$

$$(n) \mapsto \begin{cases} V, & \text{si } n = 0; \\ F, & \text{si } n = 1; \\ f(n-2), & \text{en otro caso.} \end{cases}$$

La traducción a Python es

```
def f(n):
    if n == 0:
        return True
    elif n == 1:
        return False
    else:
        return f(n-2)
```

¿Cuál es el resultado de evaluar la función  $f$  con los valores  $n = 0, 1, 2, 3, 4, 5, 6, 7, 8$ ? ¿qué función matemática es  $f$ ?



## Ejemplo

Sea  $g$  la función definida por:

$$g : \mathbb{N} \rightarrow \mathbb{N}$$

$$(n) \mapsto \begin{cases} 0, & \text{si } n = 0; \\ 1, & \text{si } n = 1; \\ 2, & \text{si } n = 2; \\ g(n-3), & \text{en otro caso.} \end{cases}$$

¿Cuál es el resultado de evaluar la función  $g$  con los valores  $n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ ?

¿qué función matemática es  $g$ ?

¿podría construir la función  $g$  que haga una sola comparación?

La traducción a Python es

```
def g(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 2
    else:
        return g(n-3)
```

## Ejemplo

Sea  $h$  la función definida por:

$$h : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$(n, m) \mapsto \begin{cases} n, & \text{si } m = 0; \\ h(n+1, m-1), & \text{en otro caso.} \end{cases}$$

La traducción a Python es

```
def h(n,m):
    if m == 0:
        return n
    else:
        return h(n+1,m-1)
```

¿Cuál es el resultado de evaluar la función  $h$  con los valores  $(n, m) = (2, 3), (8, 5), (6, 6)$ ? ¿qué función matemática es  $h$ ?

# Problemas I

## Problema

Diseñe un modelo matemático que permita calcular la función **módulo**, es decir, que reciba dos (2) números naturales y retorne el residuo de dividir el primero (dividendo) entre el segundo (divisor), cuando el segundo es distinto de cero (0).

No se puede utilizar ni división (ni entera ni real) ni el mismo operador módulo.

Codifique el modelo matemático utilizando el lenguaje Python.



# Problemas II

## Problema

Diseñe un modelo matemático que permita calcular la función **producto**, es decir, que reciba dos (2) números naturales y retorne la multiplicación del primer número por el segundo.

No se pueden utilizar los operadores de multiplicación ni de división (ni entera ni real).

Codifique el modelo matemático utilizando el lenguaje Python.



# Agenda

- 1 Introducción
- 2 Programación recursiva
- 3 Ejemplos de funciones recursivas
- 4 Algunos problemas clásicos de programación recursiva**
- 5 Teorema fundamental de la programación recursiva



# Potencia de un número I

## Ejemplo

En este ejemplo se definirá una función recursiva que permita hallar un número real elevado a un número natural. Para expresar una función que calcule esta operación, en primera instancia se construye la expresión *potencia* :  $\mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$  que define la función que tiene como entrada un número real que representa la base y un número natural que indica el exponente, y como salida se obtendrá un número real que será la potencia. Por facilidad, aquí se asumirá que  $0^0 = 1$ .



# Potencia de un número II

## Ejemplo (continuación)

Ahora observese que en general si se tiene una base  $b$  y un exponente  $n$ , entonces por definición

$$b^n = \underbrace{b * b * b * \dots * b * b}_{n\text{-veces}}$$

si se usa la propiedad asociativa del producto de números reales, se tiene que

$$b^n = \underbrace{b * b * b * \dots * b * b}_{n\text{-veces}} = \underbrace{(b * b * \dots * b * b)}_{n-1\text{-veces}} * b$$

lo que es equivalente a

$$b^n = \underbrace{b * b * b * \dots * b * b}_{n\text{-veces}} = \underbrace{(b * b * \dots * b * b)}_{n-1\text{-veces}} * b = b^{n-1} * b$$

# Potencia de un número III

A partir de esta observación se puede dar una definición recursiva usando funciones. La declaración de esta función junto con su cuerpo se hará de la siguiente manera

$$\text{potencia}(b, n) = p$$

Si se establecen las variables:

$b :=$  Base

$n :=$  Exponente

$p :=$  Potencia  $b^n$

entonces

$$\text{potencia} : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$$

$$(b, n) \mapsto \begin{cases} 1, & \text{si } n = 0; \\ \text{potencia}(b, n - 1) * b, & \text{en otro caso.} \end{cases}$$





# Potencia de un número III

La codificación en Python de esta función es

```
def potencia(b, n):  
    if n == 0:  
        return 1  
    else:  
        return potencia(b, n-1) * b
```

Una solicitud por consola de la base y el exponente puede ser

```
base = float(input('Por favor digite la base: '))  
exp = int(input('Por favor digite el exponente: '))  
print(base, '^', exp, '=', potencia(base, exp))
```



# Principio del producto para conteo de listas I

## Ejemplo

Suponga que se seleccionan cuatro cartas distintas de una baraja de poker, que se van a representar por los símbolos



si con estas cartas se conforma el conjunto  $\text{cartas} = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$ . ¿De cuántas formas distintas se pueden organizar las cartas?

## Solución

Como se van a listar todas las formas posibles en que se pueden organizar las cartas, el orden si importa.



# Principio del producto para conteo de listas II

## Solución (continuación)

Una estrategia para encontrar el número de listas puede ser el siguiente:

- 1 Se selecciona una carta del conjunto *cartas* de forma arbitraria pero fija, por ejemplo la carta  $\diamond$ .
- 2 Ya fijada la carta  $\diamond$ , el resto del trabajo consiste en hallar el número de formas distintas de organizar las cartas restantes, es decir, el conjunto  $cartas \setminus \{\diamond\} = \{\clubsuit, \heartsuit, \spadesuit\}$ .
- 3 Ahora por ejemplo se selecciona la carta  $\spadesuit$  de forma arbitraria pero fija.
- 4 A continuación, el trabajo se reduce a hallar el número de formas distintas de organizar las cartas restantes, es decir, el conjunto  $cartas \setminus \{\diamond, \spadesuit\} = \{\clubsuit, \heartsuit\}$ .



# Principio del producto para conteo de listas III

## Solución (continuación)

- 5 Posteriormente, por ejemplo se puede seleccionar de forma arbitraria pero fija la carta ♥.
- 6 Para finalizar, el trabajo se reduce a hallar el número de formas distintas de organizar las cartas restantes, es decir el conjunto  $\text{cartas} \setminus \{\diamond, \spadesuit, \heartsuit\} = \{\clubsuit\}$ . Como para este conjunto sólo se tiene una opción, entonces el número de formas distintas de organizar un conjunto de una carta es 1.

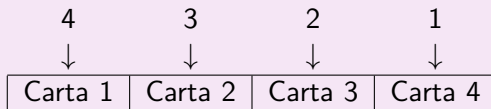
Siguiendo los pasos anteriores, se obtuvo la lista



# Principio del producto para conteo de listas IV

## Solución (continuación)

De lo anterior se concluye que el número de formas de listar los elementos de un conjunto con cuatro elementos se puede representar de la siguiente manera



de lo anterior y del principio del producto se puede concluir que el número de todas la posibles listas que se forman con las cartas ♣, ♦, ♥, ♠ es

$$\prod_{i=1}^4 i = 1 \cdot 2 \cdot 3 \cdot 4 = 24 = 4!$$



# Función factorial I

En general, para un conjunto  $A$  con cardinal  $|A| = n$ , se tiene que el número de formas de listar todas lo formas en que se pueden organizar los elementos de  $A$  es

$$\prod_{i=1}^n i = n * (n - 1) * (n - 2) * \cdots * 3 * 2 * 1$$

este valor que depende solamente de  $n$ , es una función, se denota por el símbolo  $n!$  y se llama el factorial del número  $n$ . Para el caso del conjunto  $\emptyset$ , se puede demostrar que  $0! = 1$ . A partir de la asociatividad de la multiplicación de los naturales se tiene que

$$n! = n * \underbrace{(n - 1) * (n - 2) * \cdots * 3 * 2 * 1}_{n-1\text{-veces}} = n * (n - 1)!$$



# Función factorial II

A partir de lo anterior se puede obtener una función recursiva para el factorial  $n!$ , que nombraremos

$$fact(n) = f$$

Si se establecen las variables:

$n$  := Número al cual se le va a calcular el factorial

$f$  := Factorial de  $n$

entonces



# Función factorial III

$$fact : \mathbb{N} \rightarrow \mathbb{N}$$

$$(n) \mapsto \begin{cases} 1, & \text{si } n = 0; \\ n * fact(n - 1), & \text{en otro caso.} \end{cases}$$

La codificación en Python de esta función es

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





# Pago del interés compuesto mes vencido I

## Ejemplo

Suponga que usted solicita un préstamo de \$1'000.000 durante un año, el prestamista cobra un interés del 5% mensual con la modalidad de interés compuesto mes vencido. ¿Cuál es el total del dinero que debe pagar cuando ha transcurrido el año por el cual solicitó el préstamo, resolviendo el problema recursivamente?.

## Solución

Para calcular el valor solicitado hay que observar que para cero (0) meses hay que pagar

$$\text{pago}(0) = \$1'000.000$$



# Pago del interés compuesto mes vencido II

## Solución (continuación)

Para calcular la cantidad de dinero que hay que pagar al cabo de los 12 meses, es suficiente con tener en cuenta que en el mes 12 se debe pagar lo que se debe en el mes 11 más los intereses que se producen en ese mes, por ejemplo si  $pago(11)$  es el valor del pago en el mes 11, entonces la cantidad de dinero que hay que pagar en el mes 12 es igual a

$$\begin{aligned} pago(12) &= pago(11) + pago(11) * 0.05 \\ &= pago(11)(1 + 0.05) \\ &= pago(11) * 1.05 \end{aligned}$$



# Pago del interés compuesto mes vencido III

## Solución (continuación)

ahora para calcular el valor a pagar en el mes 11 se puede hacer un razonamiento similar, y para todos los meses anteriores hasta el mes cero (0) se hace de manera similar, lo que genera un esquema recursivo como el que se presenta a continuación

$$\text{pago}(12) = \text{pago}(11) * 1.05$$

$$\text{pago}(11) = \text{pago}(10) * 1.05$$

$$\text{pago}(10) = \text{pago}(9) * 1.05$$

$$\text{pago}(9) = \text{pago}(8) * 1.05$$

$$\text{pago}(8) = \text{pago}(7) * 1.05$$

$$\text{pago}(7) = \text{pago}(6) * 1.05$$

$$\text{pago}(6) = \text{pago}(5) * 1.05$$

$$\text{pago}(5) = \text{pago}(4) * 1.05$$

$$\text{pago}(4) = \text{pago}(3) * 1.05$$

$$\text{pago}(3) = \text{pago}(2) * 1.05$$

$$\text{pago}(2) = \text{pago}(1) * 1.05$$

$$\text{pago}(1) = \text{pago}(0) * 1.05$$

$$\text{pago}(0) = \$ 1'000.000$$

# Pago del interés compuesto mes vencido IV

## Solución (continuación)

Así, si  $m = \$1'000.000$ ,  $i = 0.05$  y  $n = 12$ , se tiene que el pago al cabo de los 12 meses será de  $\text{pago}(12) = \$1'795.856,326$ .

A partir de las observaciones anteriores ya se puede construir la regla recursiva con la que se puede calcular el interés compuesto mes vencido, y si se quiere una regla general que se pueda utilizar con cualquier monto  $m$ , con un interés  $i$ , y al cabo de  $n$  periodos se tiene la siguiente función recursiva de tres (3) parámetros



# Pago del interés compuesto mes vencido V

## Solución (continuación)

$$\text{pago}(m, i, n) = \text{valor}$$

donde se tienen las variables

$m :=$  Cantidad de dinero solicitado como prestamo

$i :=$  Interes

$n :=$  Número de meses por el cual se solicita el pretamo

$\text{valor} :=$  Valor total a pagar por el prestamo de la cantidad  $m$   
por  $n$  meses con un interés  $i$  utilizando el método de  
interés compuesto mes vencido

entonces



# Pago del interés compuesto mes vencido VI

## Solución (continuación)

$$\text{pago} : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{N} \rightarrow \mathbb{R}^+$$

$$(m, i, n) \mapsto \begin{cases} m, & n = 0; \\ \text{pago}(m, i, n-1) * (1+i), & \text{en otro caso.} \end{cases}$$

La codificación en Python de esta función es

```
def pago(m, i, n):
    if n == 0:
        return m
    else:
        return pago(m, i, n-1)*(1+i)

print(pago(1E6, 0.05, 12))
```

# Los Simpsons en el parque de diversiones I

## Ejemplo

Los Simpsons van a un parque de diversiones y quieren subir a la montaña rusa, por lo que sólo pueden subir Homero, Marge, Bart y Lisa, y en dicha montaña rusa cada vagón sólo dispone de dos puestos. ¿De cuantas formas se pueden formar parejas de la familia Simpson para que suban al vagón de la montaña rusa?.

## Solución

Como se van a formar parejas, el orden no importa. Una estrategia para encontrar el número de parejas puede ser el siguiente:

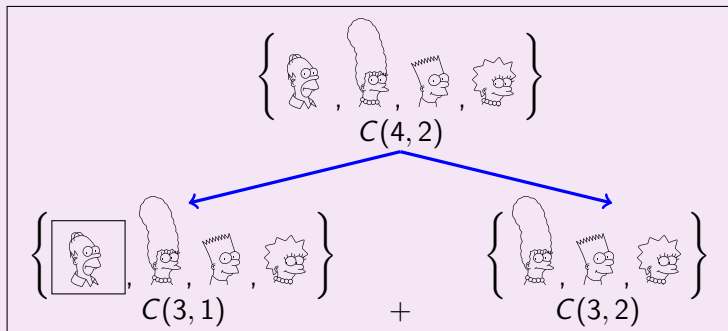
- 1 Se listan los elementos del conjunto, en este caso podría ser

$$\text{Simpsons} = \left\{ \text{Homero}, \text{Marge}, \text{Bart}, \text{Lisa} \right\}$$

# Los Simpsons en el parque de diversiones II

## Solución

- 2 Dado el conjunto *Simpsons*, para la pareja que se va seleccionar se puede escoger o no a Homero, como se observa a continuación

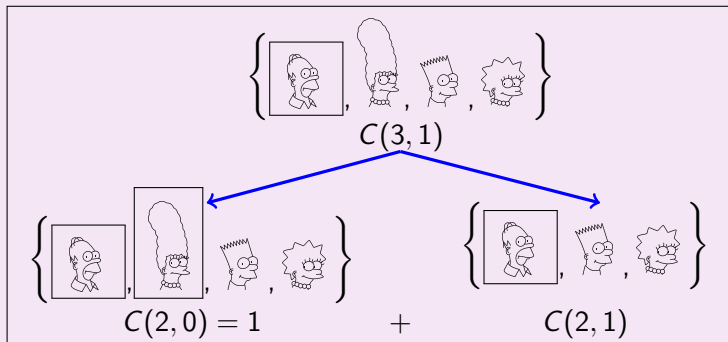




# Los Simpsons en el parque de diversiones III

## Solución

- 3 Si en el numeral 2 se seleccionó a Homero, entonces ahora se puede escoger o no a Marge, como se observa a continuación

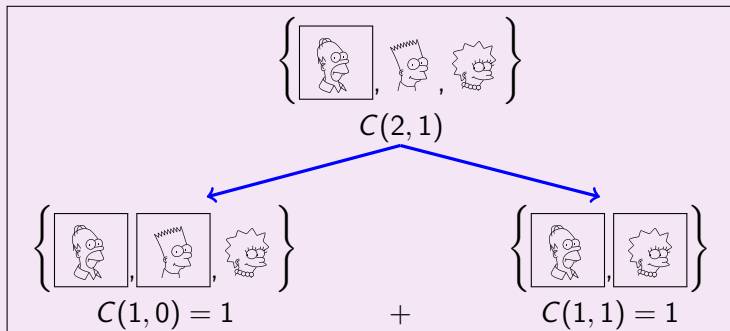


si se escoge a Marge, entonces ya se tiene una pareja.

# Los Simpsons en el parque de diversiones IV

## Solución

- 4 Si en el numeral 2 se seleccionó a Homero y en el 3 no se seleccionó a Marge, entonces ahora se puede escoger o no a Bart, como se observa a continuación

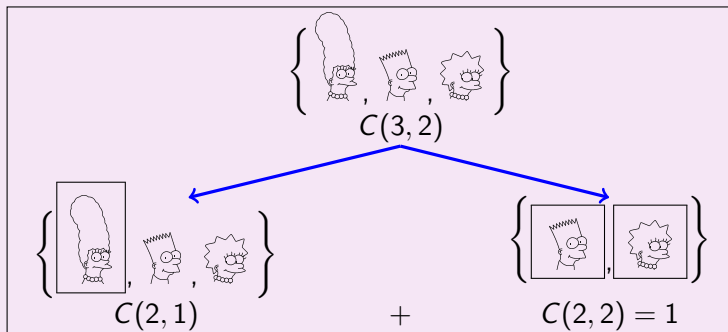


si se escoge a Bart, entonces ya se tiene otra pareja, si no entonces Lisa debe hacer parte de la siguiente pareja.

# Los Simpsons en el parque de diversiones V

## Solución

- 5 Si en el numeral 2 no se seleccionó a Homero, entonces ahora se puede escoger o no a Marge, como se observa a continuación

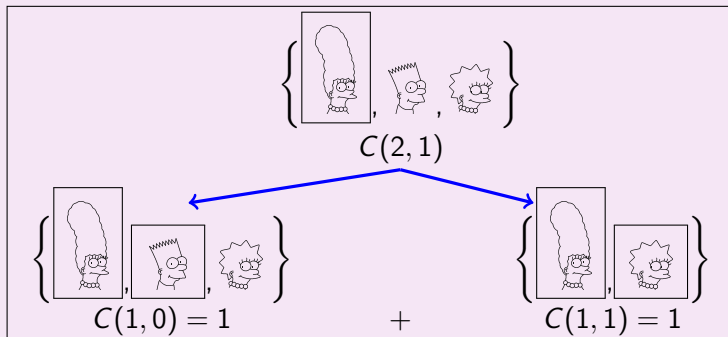


si se no escoge a Marge, entonces ya se tiene una nueva pareja.

# Los Simpsons en el parque de diversiones VI

## Solución

- 6 Si en el numeral 5 se seleccionó a Marge, entonces ahora se puede escoger o no a Bart, como se observa a continuación

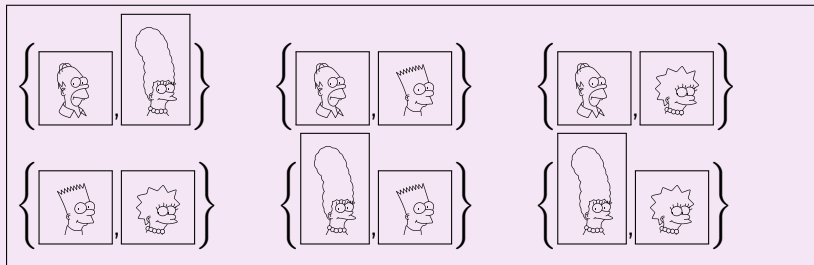


si se escoge a Bart, entonces ya se tiene otra pareja, si no entonces Lisa debe hacer parte de la última pareja.

# Los Simpsons en el parque de diversiones VII

## Solución

- 7 Después de hacer el conteo exhaustivo de la parejas que se pueden formar, se observa que existen 6 parejas que contienen dos miembros de la familia de los *Simpsons*, estas son:



# Conteo de subconjuntos I

A partir del ejemplo anterior, si la función

$$C(n, k) = c$$

representa el número de subconjuntos de  $k$  elementos de un conjunto de  $n$  elementos, ahora se puede construir una función recursiva que permita hacer el conteo del número de estos subconjuntos, de esta forma

Si se establecen las variables:

$n$  := Número de elementos del conjunto

$k$  := Número de elementos de los subconjuntos

$c$  := Número de subconjuntos de  $k$  elementos de un conjunto de  $n$  elementos

entonces



# Conteo de subconjuntos II

$$C : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$(n, k) \mapsto \begin{cases} 0, & \text{si } k > n; \\ 1, & \text{si } (k = 0) \vee (n = k); \\ C(n-1, k-1) + C(n-1, k), & \text{en otro caso.} \end{cases}$$

La codificación en Python de esta función es

```
def C(n, k):
    if k > n:
        return 0
    elif k == 0 or n == k:
        return 1
    else:
        return C(n-1, k-1) + C(n-1, k)
```



# Agenda

- 1 Introducción
- 2 Programación recursiva
- 3 Ejemplos de funciones recursivas
- 4 Algunos problemas clásicos de programación recursiva
- 5 Teorema fundamental de la programación recursiva**





# Teorema fundamental de la programación recursiva

## Teorema (Teorema fundamental de la programación recursiva)

*Un lenguaje de programación es completo en Turing si tiene valores enteros no negativos, funciones aritméticas elementales sobre dichos valores, así como un mecanismo para definir nuevas funciones utilizando las funciones ya existentes (composición), la selección (if) y la recursión.*



# Problemas III

## Problemas

- 1 Modele mediante una función matemática y diseñe un programa recursivo que calcule la suma de los primeros  $n$  números naturales ( $\sum_{i=0}^n i$ ).
- 2 Modele mediante una función matemática y diseñe un programa sin cadenas, tuplas o listas que retorne el último dígito de un número natural  $n$  (leído de izquierda a derecha). Por ejemplo,  $ultimo(13579) = 9$ .
- 3 Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que dado un número natural  $n$  elimine el último dígito del número. Por ejemplo,  $elimina\_ult(654321) = 65432$ .



# Problemas IV

## Problemas

- 4 Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que determine la cantidad de dígitos que componen un número natural  $n$ . Por ejemplo,  $longitud(1230321) = 7$ .
- 5 Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que calcule la suma de los dígitos que componen un número natural  $n$ . Por ejemplo,  $suma\_digitos(123456) = 21$ .



# Problemas V

## Problemas

- ⑥ Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que retorne el primer dígito de un número natural  $n$  (leído de izquierda a derecha). Por ejemplo,  $\text{primero}(86420) = 8$ .
- ⑦ Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que dado un número natural  $n$  elimine el primer dígito del número. Por ejemplo,  $\text{elimina\_pri}(654321) = 54321$ .
- ⑧ Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que dado un número natural  $n$  inserte un dígito al comienzo del número. Por ejemplo,  $\text{inserta}(7, 654321) = 7654321$ .



# Problemas VI

## Problemas

- 9 Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que invierta la cifras de un número  $n$  dado. Por ejemplo,  $inversa(654321) = 123456$ .
- 10 Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que determine si un número es capicua. Un número se dice palíndromo si al leerlo de izquierda a derecha es lo mismo que leerlo de derecha a izquierda. Por ejemplo,  $capicua(1) = V$ ,  $capicua(1234321) = V$ ,  $capicua(123421) = F$ .

