Recursividad
Fundamentos de la programación

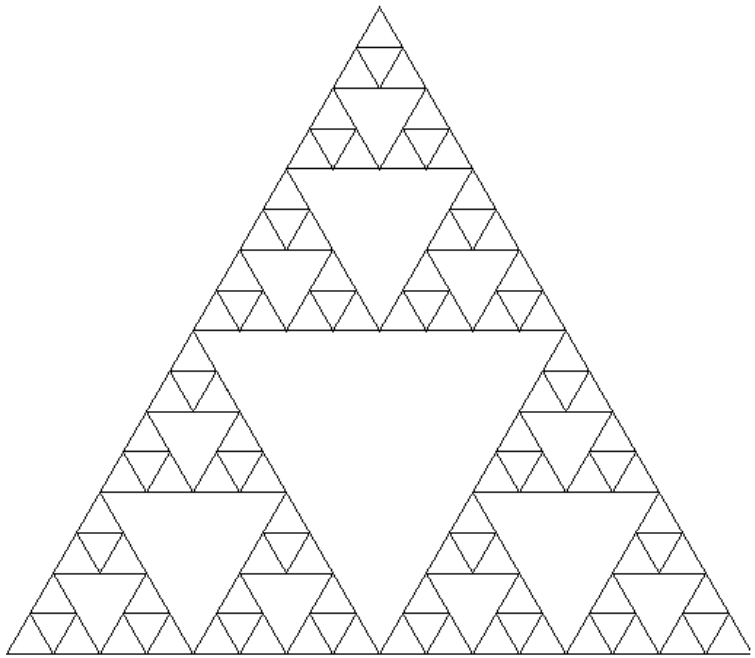
Elena G. Barriocanal y Salvador Sánchez

Universidad de Alcalá

Noviembre de 2015

Concepto de recursividad

- Definir un objeto o proceso en función de sí mismo.
- Es útil para resolver problemas por descomposición en subproblemas.
  - Los subproblemas tienen idéntica estructura al problema original, pero el caso que resuelven está reducido o es más pequeño.



Definiciones

Definición

Un **objeto** recursivo es aquel que forma parte de sí mismo o se define en función de sí mismo

Definición

Una **función** recursiva es aquella que contiene llamadas a ella misma.

```
n! = n * (n-1)!
```

Análisis conceptual de una función recursiva

Planteamiento:

- Se basa en la descomposición de un problema en subproblemas de más fácil solución.
- Cada subproblema se vuelve a descomponer a su vez, hasta que se alcanza el problema más básico, de solución trivial.
- Cuando se alcanza el caso más básico, se resuelve.

Análisis conceptual de una función recursiva

Planteamiento:

- Se basa en la descomposición de un problema en subproblemas de más fácil solución.
- Cada subproblema se vuelve a descomponer a su vez, hasta que se alcanza el problema más básico, de solución trivial.
- Cuando se alcanza el caso más básico, se resuelve.

Ejecución:

- Cada vez que se invoca una resolución de un subproblema, el problema actual queda suspendido y pendiente de resolución (apilado)
- Cuando se termina de resolver un subproblema, se retorna el resultado al problema que lo invocó (desapilado)

Análisis conceptual de una función recursiva

Planteamiento:

- Se basa en la descomposición de un problema en subproblemas de más fácil solución.
- Cada subproblema se vuelve a descomponer a su vez, hasta que se alcanza el problema más básico, de solución trivial.
- Cuando se alcanza el caso más básico, se resuelve.

Ejecución:

- Cada vez que se invoca una resolución de un subproblema, el problema actual queda suspendido y pendiente de resolución (apilado)
- Cuando se termina de resolver un subproblema, se retorna el resultado al problema que lo invocó (desapilado)

Importante:

- El subproblema y el problema deben tener la misma estructura.
- La descomposición debe permitir alcanzar el caso básico en un número finito de pasos.

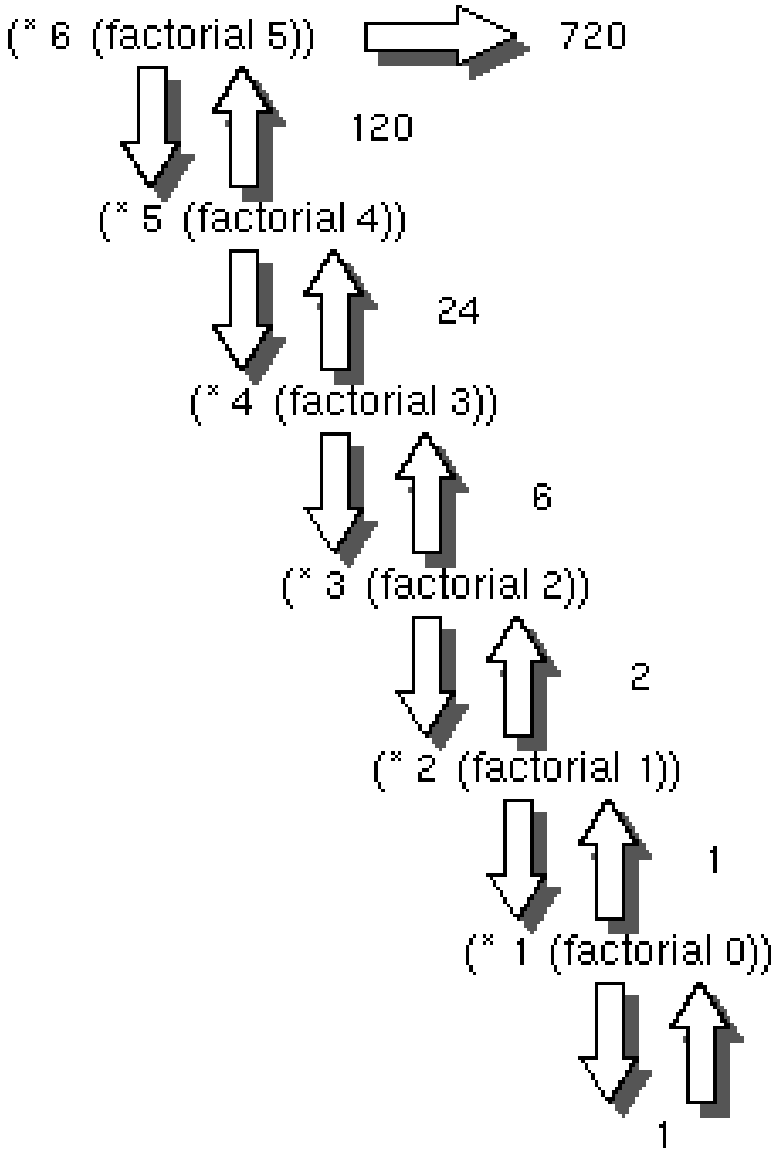
Esquema básico de una función recursiva

```
funcion R (problema)

  si el problema es suficientemente simple # caso basico
    resolverlo directamente

  si_no # descomponer el problema en subproblemas

    # llamada recursiva
    por cada subproblema s, invocar R(s)
```

Sugerencias para diseñar algoritmos recursivos

- Identificar el problema a resolver: problema, ejemplar o caso.
- Identificar el/los problemas que se resuelven directamente, de manera sencilla: caso o problema básico.
- Identificar cómo se puede descomponer un caso cualquiera no básico para reducir el problema a uno o varios subproblemas más pequeños que tiendan al caso básico.
    - Si no tienden al caso básico ¡se desbordará la pila de invocaciones en tiempo de ejecución!

Ejemplo: Factorial

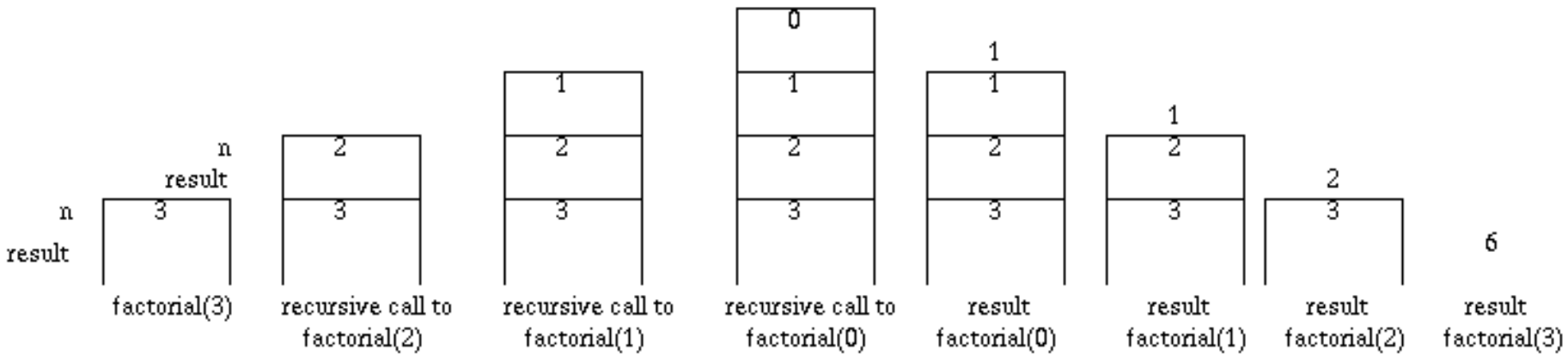


Ejemplo: Factorial

```
def factorial (n):
    if (n==1):
        resultado = 1
    else:
        resultado = n * factorial (n-1)
    return resultado
```

Ejemplo: Factorial

```
def factorial (n):
    if (n==1):
        resultado = 1
    else:
        resultado = n * factorial (n-1)
    return resultado
```



Análisis de un problema recursivo

Problema: Multiplicar dos números por sumas sucesivas.

Análisis de un problema recursivo

Problema: Multiplicar dos números por sumas sucesivas.

- Cualquier número multiplicado por 1 es ese número $\rightarrow$ caso básico

Análisis de un problema recursivo

Problema: Multiplicar dos números por sumas sucesivas.

- Cualquier número multiplicado por 1 es ese número $\rightarrow$ caso básico
- Para multiplicar 6 por 4 podemos hacer la siguiente descomposición: «Sumamos 6 a la multiplicación de 6 por 3»

Análisis de un problema recursivo

Problema: Multiplicar dos números por sumas sucesivas.

- Cualquier número multiplicado por 1 es ese número $\rightarrow$ caso básico
- Para multiplicar 6 por 4 podemos hacer la siguiente descomposición: «Sumamos 6 a la multiplicación de 6 por 3»
- El problema original es 6*4 y lo hemos reducido a una operación de suma y a la resolución del subproblema 6*3

Análisis de un problema recursivo

Problema: Multiplicar dos números por sumas sucesivas.

- Cualquier número multiplicado por 1 es ese número $\rightarrow$ caso básico
- Para multiplicar 6 por 4 podemos hacer la siguiente descomposición: «Sumamos 6 a la multiplicación de 6 por 3»
- El problema original es 6*4 y lo hemos reducido a una operación de suma y a la resolución del subproblema 6*3
- Es posible también descomponer 6*3 en 6 + (6*2) y 6*2 en 6+(6*1)

Ejemplo: Código en Python

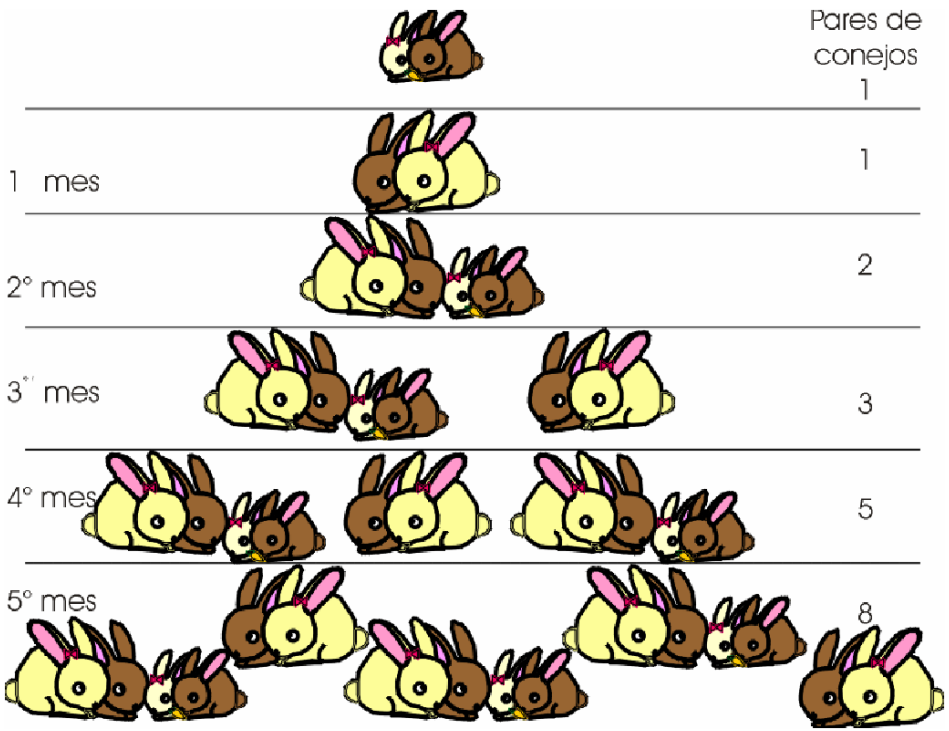
```
def multiplicar (n, m):
  if (m==1):
   resultado = n
  else:
    resultado = n + multiplicar(n, m-1)
  return resultado
```

Ejemplo: Serie de Fibonacci

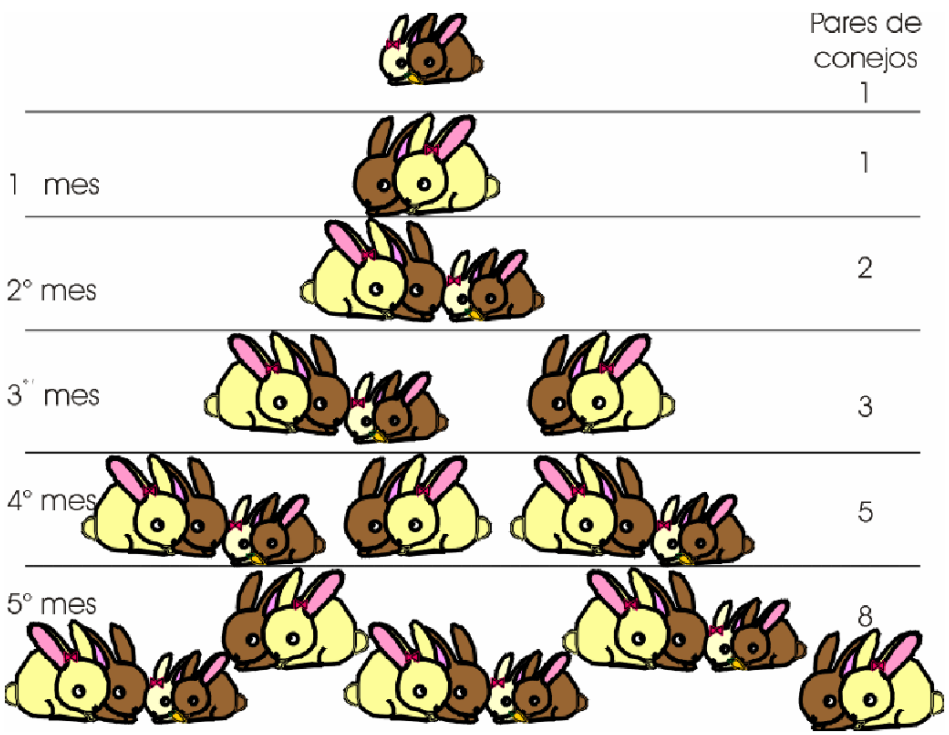
Fibonacci se preguntaba cuántos conejos habría en una granja tras un cierto número de meses, si se parte de una sola pareja y asumiendo que...

- Los conejos alcanzan la madurez sexual a la edad de un mes.
- En cuanto alcanzan la madurez sexual los conejos se aparean.
- El periodo de gestación de los conejos es de un mes.
- Los conejos nunca mueren.
- Los conejos adultos se aparean regularmente cada mes.
- La hembra siempre pare una pareja de conejos de sexos opuestos.
- Los conejos tienen una moral y un instinto de variedad genética muy relajados y se aparean entre parientes.

Ejemplo: Serie de Fibonacci



Ejemplo: Serie de Fibonacci




```
def fibonacci(mes):
  if (mes == 1 or mes == 0):
    resultado = 1
  else:
    resultado = fibonacci(mes-2) + fibonacci(mes-1)
  return resultado
```

Ejemplo: Qué hace el siguiente código?

```
def misterio (n):
  if (n==1):
    i=int(input('Introduce un numero: '))
    print(i)
  else:
    i=int(input('Introduce un numero: '))
    misterio(n-1)
    print(i)
  return
```

Ejemplo: Qué hace el siguiente código?

```
def misterio (n):
  if (n==1):
    i=int(input('Introduce un numero: '))
    print(i)
  else:
    i=int(input('Introduce un numero: '))
    misterio(n-1)
    print(i)
  return
```

```
Entrada: 1 2 3 4 5
Salida: 5 4 3 2 1
```

Ejemplo: Palíndromo

Ejemplo: Palíndromo


```
def es_palindromo (pal, posIni, posFin):
    """ str, int, int --> bool
        OBJ: Calcula si un texto entre 2 indices se lee igual
             al derecho y al reves (palindromo) """
  if (posIni>=posFin):
    res= True
  else:
    if (pal[posIni]!=pal[posFin]):
      res=False
    else:
      res=es_palindromo(pal, posIni+1, posFin-1)
  return res
```

Ejemplo: Palíndromo (con *slicing*)

```
def es_palindromo (frase):
    """ str --> bool
        OBJ: Calcula si una frase puede leerse igual al
             derecho o al reves (palindromo) """
    if (len(frase)<=1):
        respuesta = True
    elif (frase[0] != frase[len(frase)-1]):
        respuesta = False
    else:
        respuesta = es_palindromo (frase[1:len(frase)-1])
    return respuesta
```

Ejemplo: Suma contenido vector

Ejemplo: Suma contenido vector

```
def suma (v, ini, fin):
  if (fin==ini):
    aux=v[ini]
  else:
    aux=v[ini] + suma(v, ini+1, fin)
  return aux
```

Ejemplo: Suma contenido vector

```
def suma (v, ini, fin):
  if (fin==ini):
    aux=v[ini]
  else:
    aux=v[ini] + suma(v, ini+1, fin)
  return aux
```

Usando *slicing*:

```
def suma (vector):
  """ lista -> long
  OBJ: suma todos los elementos de un array
  PRE: el array tiene al menos un elemento   """
  if (len(vector) == 1):
      resul = vector[0]
  else:
      resul = vector[0] + suma (vector[1:len(vector)])
  return resul
```

Ejemplo: Mostrar los elementos de un vector

Ejemplo: Mostrar los elementos de un vector

```
def imprimir(v, ini, fin):
  if (fin==ini):
    print(v[ini])
  else:
    print (v[ini])
    imprimir(v,ini+1,fin)
  return
```

Ejemplo: Mostrar los elementos de un vector

```
def imprimir(v, ini, fin):
  if (fin==ini):
    print(v[ini])
  else:
    print (v[ini])
    imprimir(v,ini+1,fin)
  return
```

- y si queremos que imprima el contenido al revés?

Ejemplo: Mostrar los elementos de un vector

```
def imprimir(v, ini, fin):
  if (fin==ini):
    print(v[ini])
  else:
    print (v[ini])
    imprimir(v,ini+1,fin)
  return
```

- y si queremos que imprima el contenido al revés?

```
def imprimir(v, ini, fin):
  if (fin==ini):
    print(v[ini])
  else:
    imprimir(v,ini+1,fin)
    print (v[ini])
  return
```

Ejercicios propuestos

- Codificar un módulo recursivo que calcule la división entera entre dos números A y B sin utilizar multiplicaciones ni divisiones.
- Codificar un módulo recursivo que calcule x^n, con x real y n entero.
- Codificar un programa recursivo que toma una cadena como parámetro y devuelve otra cadena que es la original pero con sus caracteres invertidos.
- Los detectives de una agencia se envían todos los mensajes cifrados por motivos de seguridad. El algoritmo que están utilizando en la actualidad consiste en intercambiar cada vocal por la letra que la precede (si existe). Por ejemplo: El resultado de codificar el mensaje:

  ```
  esta en Leon
  esate n eoLn
  ```

Recap

- La recursividad es una técnica de programación que facilita la codificación de programas descomponiendo en problema a resolver en problemas más pequeños.
- Un objeto recursivo está formado por sí mismo.
- Una función recursiva contiene llamadas a ella misma.
- Las funciones recursivas deben tener una condición de parada cuando se llegue al caso básico y una o más llamadas recursivas con parámetros que tiendas al caso base.
- Es importante comprender cómo se apilan las llamadas para comprender el resultado de la recursividad.