Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 20: Ejercicios Recursión

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

01 de Junio, 2015

Recursión

Recursión

¿Qué pasa si una función se llama dentro de sí misma?

```
1 def funcion():
2   funcion()
3
4 funcion()
```

Recursión

¿Qué pasa si una función se llama dentro de sí misma?

```
1 def funcion():
2   funcion()
3
4 funcion()
```

... es una especie de loop infinito

Recursión

¿Qué pasa si una función se llama dentro de sí misma?

```
1 def funcion():
2   funcion()
3
4 funcion()
```

... es una especie de loop infinito

¿Qué hacíamos para cortar el loop?

Recursión

¿Cuál es la salida de este programa?

```
1 def funcion(contador):
2   if(contador == 0):
3     return
4   print(contador)
5   funcion(contador-1)
6
7 funcion(5)
```

Recursión

... y de este programa?

```
1 def funcion(contador):
2   if(contador == 0):
3     return
4   funcion(contador-1)
5   print(contador)
6
7 funcion(5)
```

Recursión

Definición

Recursión es una estrategia para solucionar problemas llamando a una función dentro de si misma.

Recursión

Definición

Recursión es una estrategia para solucionar problemas llamando a una función dentro de si misma.

Ventajas:

- Códigos más cortos.
- Códigos más legibles.

Recursión

¿Cuándo usar recursión?

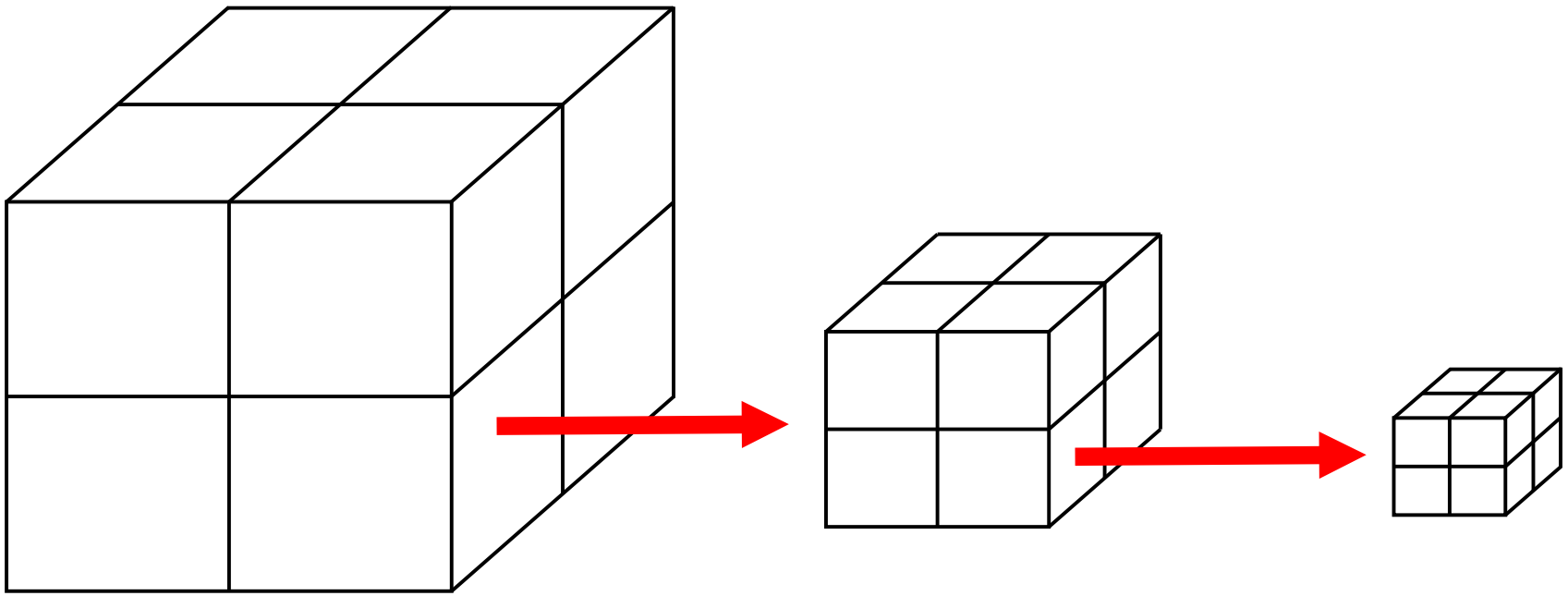
Recursión

¿Cuándo usar recursión?

1) Cuando un problema se puede dividir en subproblemas idénticos, pero más pequeños.

2) Para explorar el espacio en un problema de búsqueda.

Recursión



Dividir para conquistar

Recursión

Ejemplo: Implemente una función que calcule $n!$

$$n! = n * (n-1)! \quad 1! = 1 \quad 0! = 1$$

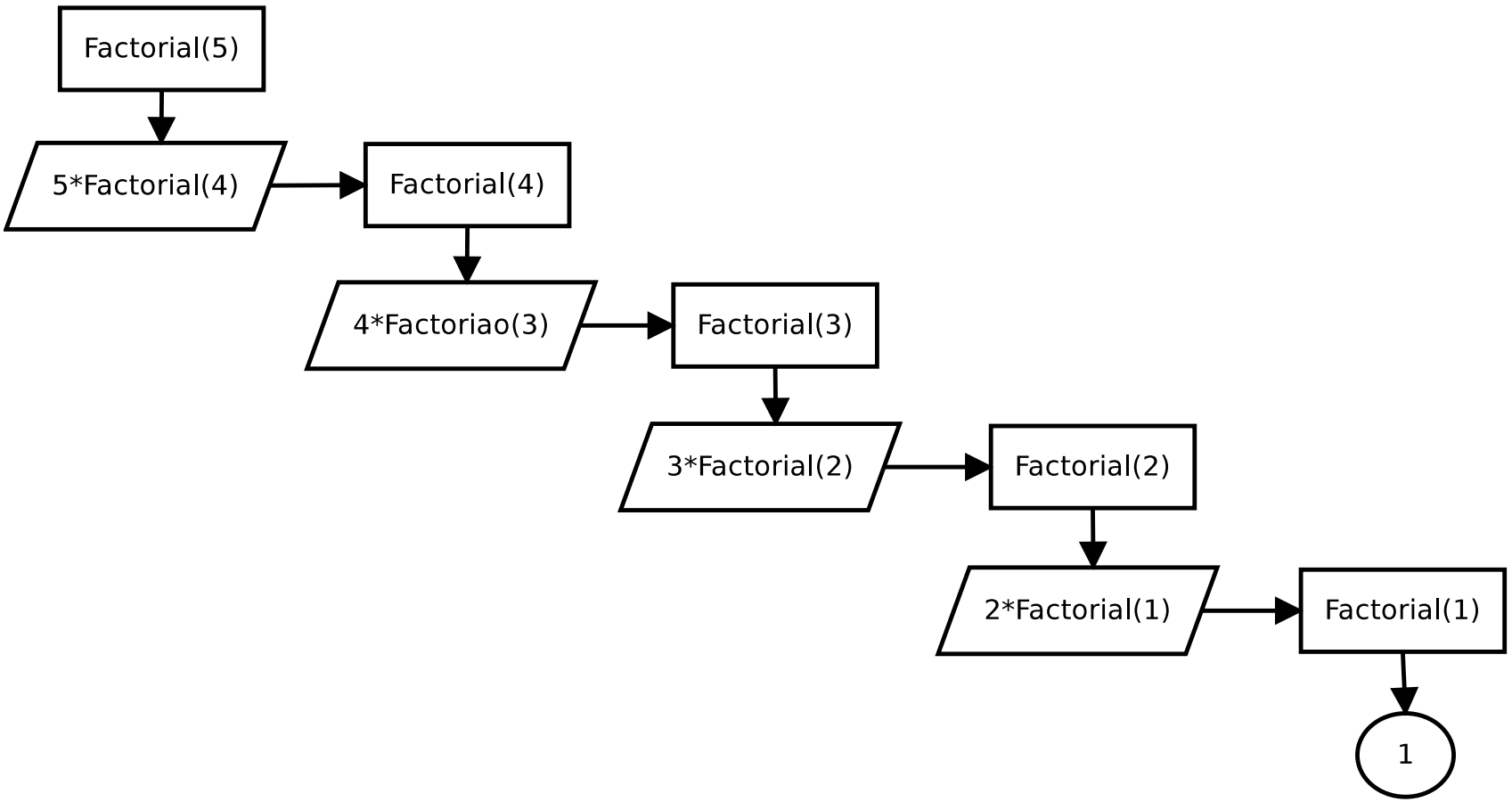
Recursión

Ejemplo: Implemente una función que calcule $n!$

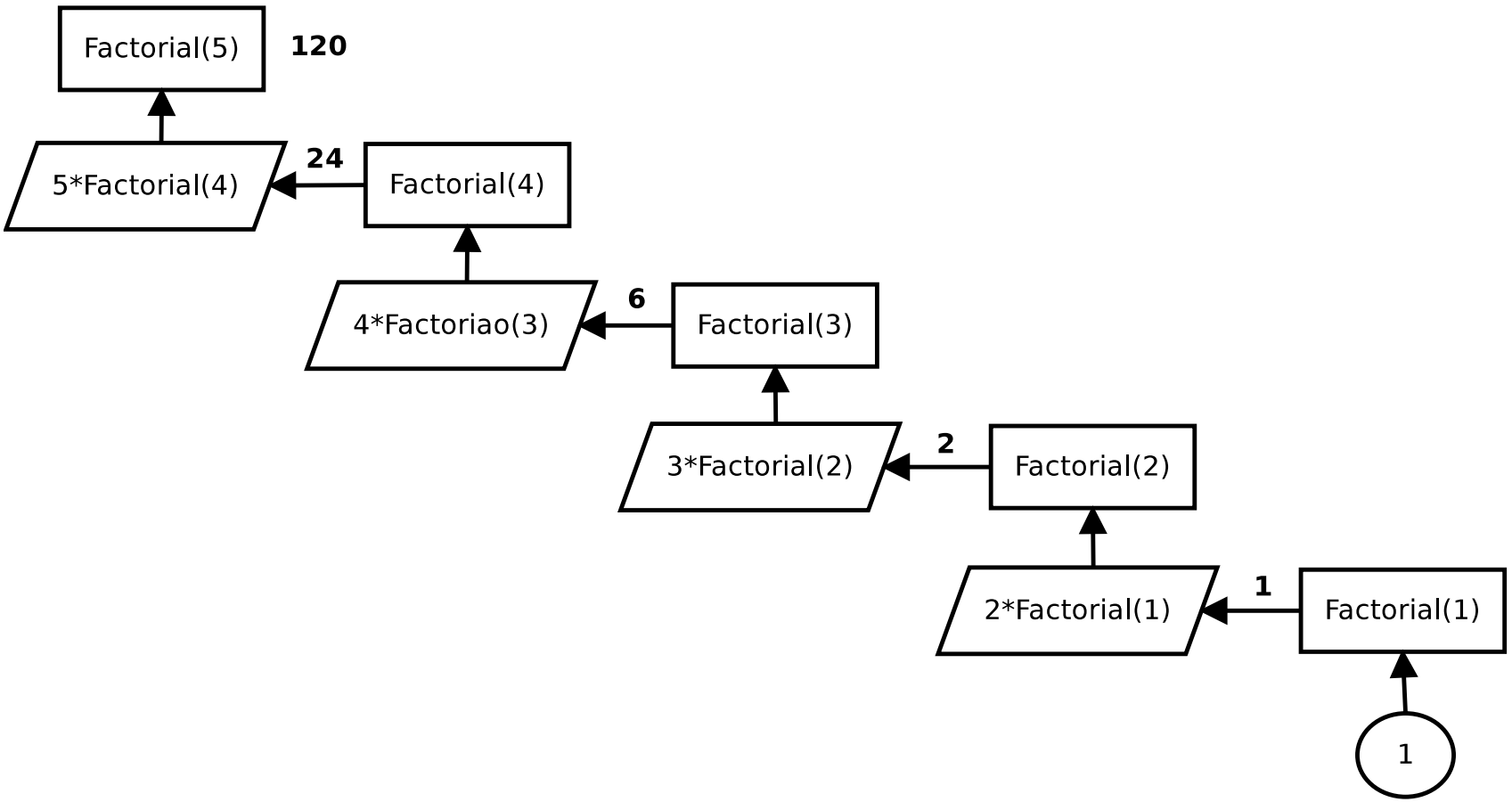
$$n! = n * (n-1)! \quad 1! = 1 \quad 0! = 1$$

```
1 def factorial(n):
2   if(n <= 1):
3     return 1
4   return n*factorial(n-1)
5
6 print(factorial(5))
```

Recursión


Factorial(5)
5*Factorial(4)
Factorial(4)
4*Factoriao(3)
Factorial(3)
3*Factorial(2)
Factorial(2)
2*Factorial(1)
Factorial(1)
1

Recursión


Factorial(5)
120
5*Factorial(4)
24
Factorial(4)
4*Factoriao(3)
6
Factorial(3)
3*Factorial(2)
2
Factorial(2)
2*Factorial(1)
1
Factorial(1)
1

Recursión

```
1 def factorial(n):
2   if(n <= 1):
3     return 1
4   return n*factorial(n-1)
```

Estructura solución recursiva:

- Firma: Nombre y parámetros.
- Casos bases.
- Llamados recursivos.
- Formar solución a partir de subsoluciones.

Ejemplos

Importante: Estudien los ejemplos de la clase 08.

Ejemplos

¿Qué hace el siguiente código?

```
1 def mystery(a, b):
2   if (b == 0): return 0
3   if (b % 2 == 0): return mystery(a+a, b//2)
4   return mystery(a+a, b//2) + a
```

Ejemplos

Programe una función recursiva para verificar si una palabra es un palíndromo.

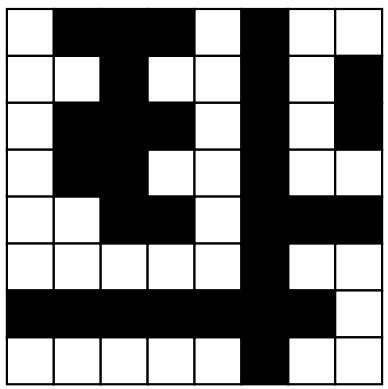
Ejemplos

Programe una función recursiva para verificar si una palabra es un palíndromo.

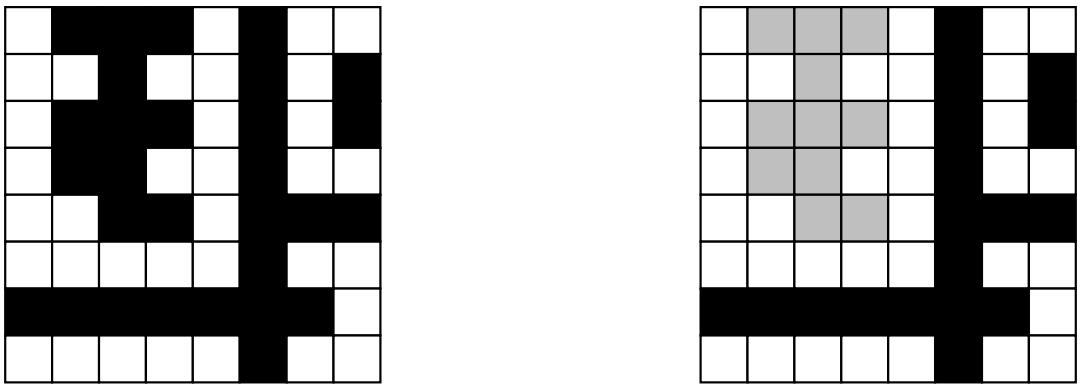
```
1 def palindromo(s):
2   if(len(s) <= 1):
3     return True
4   return (s[0]==s[-1]) and palindromo(s[1:len(s)-1])
```

Ejemplos

[**Ex 2014-2**] Tienes un tablero de $n \times n$ celdas blancas y negras, distribuidas de cualquier manera. Una región blanca es un conjunto contiguo máximo de celdas blancas: dos celdas blancas son contiguas si tienen un borde común (si solo tienen un vértice común, no son contiguas). Una región negra se define similarmente.

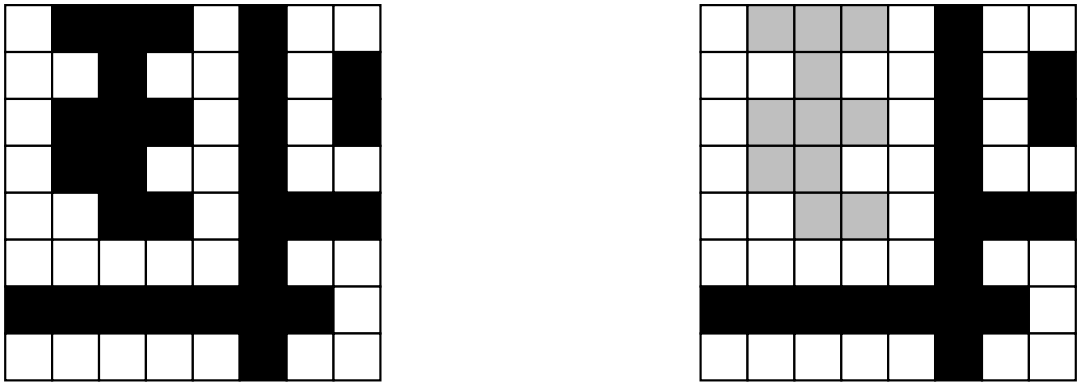


Ejemplos



El problema consiste en que dada una celda y un color, se pide pintar de ese color toda la región a la que pertenece la celda. Por ejemplo, si para el tablero de la figura anterior se especifica la celda (3, 2) y el color gris, el resultado debe ser el tablerdo de la derecha.

Ejemplos



El problema consiste en que dada una celda y un color, se pide pintar de ese color toda la región a la que pertenece la celda. Por ejemplo, si para el tablero de la figura anterior se especifica la celda (3, 2) y el color gris, el resultado debe ser el tablerdo de la derecha.

Para ello, implementa la función `pintar_region(...)`, que reciba los parámetros apropiados.

Ejemplos

```
1  def pintar_region(t,p):
2    c = t[p[0]][p[1]]
3    t[p[0]][p[1]] = 2
4    for d in [[1,0],[-1,0],[0,1],[0,-1]]:
5      di = p[0]+d[0]
6      dj = p[1]+d[1]
7      if(0 <= di < len(t) and
8         0 <= dj < len(t[0]) and
9         t[di][dj] == c):
10       pintar_region(t,[di,dj])
```

Ejemplos

Sea L una lista de números y n un número natural. ¿Se puede formar n a partir de sumar elementos de L?

`L=[5, 3, 9]` y `n=31` es `True`: `31 = 2×9 + 2×5 + 1×3`.

Ejemplos

Sea L una lista de números y n un número natural. ¿Se puede formar n a partir de sumar elementos de L?

`L=[5, 3, 9]` y `n=31` es `True`: `31 = 2×9 + 2×5 + 1×3`.

```
1  def verificar_suma(L,n):
2    if(len(L) == 0):
3      return n == 0
4    for i in range(n//L[0] + 1):
5      if(verificar_suma(L[1:],n-i*L[0])):
6        return True
7    return False
8
9  print(verificar_suma([5, 3, 9],7))  # >>> False
10 print(verificar_suma([5, 3, 9],31)) # >>> True
```

Ejemplos

¿Cómo retorno los factores que hacen posible la multiplicación?

Ejemplo: Para `L=[5, 3, 9]` y `n=31` retornar `[2, 1, 2]`.

Ejemplos

¿Cómo retorno los factores que hacen posible la multiplicación?

Ejemplo: Para `L=[5, 3, 9]` y `n=31` retornar `[2, 1, 2]`.

```
1  def verificar_suma(L,n,f=[]):
2    if(len(L) == 0):
3      if(n == 0):
4        return []
5      return None
6    for i in range(n//L[0] + 1):
7      r = verificar_suma(L[1:],n-i*L[0])
8      if(not r is None):
9        return [i] + r
10   return None
11
12 print(verificar_suma([5, 3, 9],7))  # >>> None
13 print(verificar_suma([5, 3, 9],31)) # >>> [2,1,2]
```

Ejemplos

Paréntesis balanceados:

Genere un programa recursivo que indique si un string posee paréntesis balanceados

Ejemplos

Paréntesis balanceados:

Genere un programa recursivo que indique si un string posee paréntesis balanceados

✓	✗
()	((
(())()	)(
((())())	((())((()))

Ejemplos

Reglas recursivas de construcción:

- S $\rightarrow$ ""
- S $\rightarrow$ (S)
- S $\rightarrow$ SS

Ejemplos

Reglas recursivas de construcción:

- `S` $\rightarrow$ `""`
- `S` $\rightarrow$ `(S)`
- `S` $\rightarrow$ `SS`

¿Cómo lo resolverían?

Ejemplos

Caso base: $(\texttt{S} \rightarrow \texttt{""})$

- `s == ""`

Ejemplos

Caso base: (`S` → `""`)

- `s == ""`

Casos recursivos: (`S` → `(S)`)

- `s[0] == "("` and `s[len(s)-1] == ")"` and `par(s[1:len(s)-1])`

Ejemplos

Problema:

- ¿Cómo hacemos $\mathtt{S} \to \mathtt{SS}$?
- No sabemos dónde cortar $\mathtt{S}$...

Ejemplos

Problema:

- ¿Cómo hacemos $\texttt{S} \to \texttt{SS}$?
- No sabemos dónde cortar $\texttt{S}$...

Solución: Probemos todas las opciones de corte.

Ejemplos

Problema:

- ¿Cómo hacemos `S` $\rightarrow$ `SS`?
- No sabemos dónde cortar `S`...

Solución: Probemos todas las opciones de corte.

Casos recursivos: (`S` $\rightarrow$ `SS`)

- `par(s[:i]) and par(s[i:])` con `i = 1...len(s)-1`

Ejemplos

```
1  def par(s):
2    if(s == ""):
3      return True
4    f = len(s)-1
5    if(s[0] == "(" and s[f] == ")" and par(s[1:f])):
6      return True
7    for i in range(1,f):
8      if(par(s[:i]) and par(s[i:])):
9        return True
10   return False
```

Ejemplos

Calculadora convencional:

Genere un programa recursivo que reciba un string con operación aritmética (con paréntesis) y retorne su resultado.

Ejemplos

Calculadora convencional:

Genere un programa recursivo que reciba un string con operación aritmética (con paréntesis) y retorne su resultado.

Input	Output
`"3"`	3
`"(3+2)"`	5
`"((3+2)*5)"`	25
`"((3+2)*(5/2))"`	12.5

Ejemplos

Reglas recursivas de construcción:

- `S` $\rightarrow$ `d`.
- `S` $\rightarrow$ `(S1 [op] S2)`.

... donde `d` es un número entero, `S1` y `S2` son operaciones aritméticas bien formadas; y `[op]` es `+`, `-`, `*` o `/`.

Ejemplos

Reglas recursivas de construcción:

- `S` $\rightarrow$ `d`.
- `S` $\rightarrow$ `(S1 [op] S2)`.

... donde `d` es un número entero, `S1` y `S2` son operaciones aritméticas bien formadas; y `[op]` es `+`, `-`, `*` o `/`.

¿Cómo lo resolverían?

Ejemplos

Caso base: `S` $\rightarrow$ `d` (entero positivo)

- `s.isdigit()`

Ejemplos

Caso base: `S` $\rightarrow$ `d` (entero positivo)

- `s.isdigit()`

Casos recursivos: `S` $\rightarrow$ `(S1 [op] S2)`

- Necesitamos conocer la posición del operador.

Ejemplos

Idea: Se debe cumplir con: `S` $\rightarrow$ `(S1 [op] S2)`.

Ejemplos

Idea: Se debe cumplir con: `S` $\rightarrow$ `(S1 [op] S2)`.

Luego en `S1` los paréntesis deben estar balanceados (se abren tantos paréntesis como los que se cierran).

Ejemplos

Idea: Se debe cumplir con: `S` $\rightarrow$ `(S1 [op] S2)`.

Luego en `S1` los paréntesis deben estar balanceados (se abren tantos paréntesis como los que se cierran).

Algoritmo:

- Recorrer `S` de izquierda a derecha (sin considerar primer paréntesis).
- Si veo un `"("` sumo 1 a un contador.
- Si veo un `")"` resto 1 a un contador.
- Si veo un operador y el contador es `0`, entonces esa es la posición buscada.

Ejemplos

Una vez conocida la posición del operador, basta con:

- Llamar recursivamente para obtener resultado del operando `S1`.
- Llamar recursivamente para obtener resultado del operando `S2`.
- Retornar `S1 [op] S2`

Ejemplos

```
1  def calcular(s):
2    if(s.isdigit()):
3      return int(s)
4    # Quito parétesis externos
5    s = s[1:len(s)-1]
6    # Encontramos posición operador
7    c = 0; pos = -1;
8    for i in range(len(s)):
9      if(s[i] == "("): c += 1
10     if(s[i] == ")"): c -= 1
11     if(s[i] in ['+','-','*','/'] and c==0):
12       pos = i; break
13   # Retornamos el cálculo
14   s1 = calcular(s[:i])
15   s2 = calcular(s[i+1:])
16   if(s[i] == "+"): return s1+s2
17   if(s[i] == "-"): return s1-s2
18   if(s[i] == "*"): return s1*s2
19   if(s[i] == "/"): return s1/s2
```

Ejercicios Propuestos

1) Sin volver a ver las soluciones, intente resolver los ejemplos puestos en la clase.

2) Implementa la función `vueltos_posibles(L,n)` que retorna el número de formas posibles en que podemos formar `$n` de vuelto a partir de las monedas presentes en `L`. Por ejemplo `vueltos_posibles([100],1000)` debería retornar `1`, mientras que `vueltos_posibles([1, 10, 50,100],1000)` debería retornar `4.246`.

Ejercicios Propuestos

3) Crea un programa que verifique si un String corresponde a una suma buen formada. Es decir, si cumple con:

- `S` → entero positivo (i.e. `s.isdigit() == True`).
- `S` → `(S1+S2)`, donde `S1` y `S2` son sumas bien formadas.

Por ejemplo, las expresiones `(1+7)`, `125`, `(23+(5+1))` y `((4+12)+(5+(6+7)))` son sumas bien formadas, mientras que `(1+(2+4)`, `(a+2)`, `4+` y `3+6` no lo son (notar que `3+6` no es una suma bien formada por la ausencia de paréntesis).

Hint: Si el string `S` *no es un número, para verificar si es de la forma* `(S1+S2)` *puedes revisar cada uno de los signos* `+` *que aparecen en* `S`*.*