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

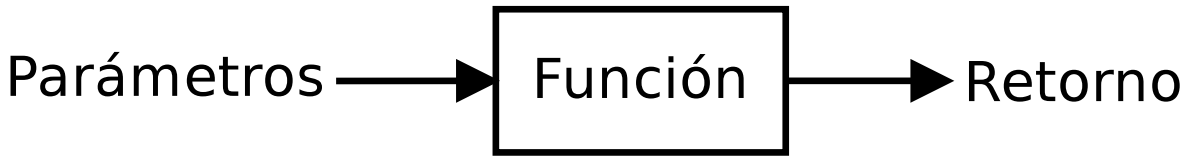
Clase 08: Recursión

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

08 de Abril, 2015

¿Qué aprendimos la clase pasada?



Dos pasos:

- Definir la función.
- Llamar a la función.

¿Qué aprendimos la clase pasada?

1. Definir la función: Aquí definimos el comportamiento de la función (su código).

Sintaxis: definir función

```
def nombre_función(param_1, param_2, ...):
    inst_1
    ...
    inst_n
    return ret
```

¿Qué aprendimos la clase pasada?

Ejemplo:

```
"""
Esta función retorna la suma de los dígitos
del número 'n'
"""
def sumar_digitos(n): # <- Función recibe 1 parámetro
  # Código de la función
  suma = 0
  while(n!=0):
    suma+=n%10
    n//=10
  # Retornamos la suma de los dígitos
  return suma
```

Obs: Notar elementos de la función (parámetros, código y retorno).

¿Qué aprendimos la clase pasada?

2. Llamar función:

- Desde tu código puedes *llamar* a funciones ya definidas.
- Al llamarla debes dar valor a **todos** sus parámetros.
- La función devolverá su valor de retorno.

Sintaxis: llamar a una función

nombre_función(in_1, in_2, ...)

¿Qué aprendimos la clase pasada?

Ejemplo:

```
15 a = int(input("Ingrese un número: "))
16 b = int(input("Ingrese otro número: "))
17 s_a = sumar_digitos(a)
18 s_b = sumar_digitos(b)
19 print("La multiplicación es:",s_a*s_b)
```

Comentarios: Estructura código

Sus programas serán cada vez más largos y complejos.

La única forma de no volverse loco es ser ordenado.

Comentarios: Estructura código

Sus programas serán cada vez más largos y complejos.

La única forma de no volverse loco es ser ordenado.

Consejos:

- Usen variables y funciones con nombre intuitivos.
- Separen secciones de código complejas en funciones independientes.
- Sigan esta estructura:
  - Primero todos sus imports.
  - Luego todas sus funciones.
  - Al final su código principal.

Comentarios: Estructura código

```python
# Primero los imports
import random

# Luego tus funciones
def tirar_dado_cargado():
  d = random.randint(1,7)
  if(d == 7):
    return 1
  return d

# Al final tu código principal
print("Pepito paga! sale par usted gana!")
while(True):
  apuesta = int(input("Ingrese apuesta: "))
  dado = tirar_dado_cargado()
  print("salió:",dado)
  if(dado%2==1): print("Que mala suerte, perdió")
  else: print("Muy bien! ganaste")
```

Comentarios: Main()

Comentarios: Main()

```
1  import random
2
3  def tirar_dado_cargado():
4    d = random.randint(1,7)
5    if(d == 7):
6      return 1
7    return d
8
9  def main():
10   print("Pepito paga! sale par usted gana!")
11   while(True):
12     apuesta = int(input("Ingrese apuesta: "))
13     dado = tirar_dado_cargado()
14     print("salió:",dado)
15     if(dado%2==1): print("Que mala suerte, perdió")
16     else: print("Muy bien! ganaste")
17
18 main()
```

Comentarios: Números capicúa

"Crea una función que retorne True ssi un número `n` *es capicúa. Los números capicúa son aquellos que se escriben igual en ambos sentidos."*

Comentarios: Números capicúa

"Crea una función que retorne True ssi un número `n` *es capicúa. Los números capicúa son aquellos que se escriben igual en ambos sentidos."*

Idea: Invertir el número y ver si es igual a su versión original.

Comentarios

Solución:

```
1  def capicua(n):
2    n_rev = 0
3    m = n
4    while(m != 0):
5      unidad = m%10 # obtengo unidad
6      m = m //10    # quito unidad
7      # agrego unidad a n_rev
8      n_rev = n_rev*10 + unidad
9    # retorno True ssi n == n_rev
10   return n == n_rev
```

Hoy veremos...

Hoy veremos...

Un poco de background

En programación existen dos formas de enfrentar un problema:

- En forma iterativa.
- En forma recursiva.

Un poco de background

En programación existen dos formas de enfrentar un problema:

- En forma iterativa.
- En forma recursiva.

Ambos estilos son buenos.

Sin embargo, hay muchos problemas para los que su solución recursiva es tan simple (y elegante) que la solución iterativa (de existir) no tiene sentido.

Un poco de background

¿Es difícil recursión?

Un poco de background

¿Es difícil recursión?

¿Por qué a la gente le cuesta tanto aprender recursión? (incluso a los cracks)

Un poco de background

¿Es difícil recursión?

¿Por qué a la gente le cuesta tanto aprender recursión? (incluso a los cracks)

¿Por qué veremos ahora la materia, y no al final del semestre?

Un poco de background

¿Es difícil recursión?

¿Por qué a la gente le cuesta tanto aprender recursión? (incluso a los cracks)

¿Por qué veremos ahora la materia, y no al final del semestre?

- Más tiempo para digerir la materia.
- Podré poner ejemplos recursivos en las diapos :)
- Su cerebro aún no es del todo iterativo.

Idea

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

Idea

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

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

Idea

¿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

Idea

¿Qué debería pasar en este caso?

```
1 def funcion(contador):
2   print(contador)
3   funcion(contador-1)
4
5 funcion(10)
```

Idea

¿Qué debería pasar en este caso?

```
1 def funcion(contador):
2   print(contador)
3   funcion(contador-1)
4
5 funcion(10)
```

Problema: Necesitamos una condición que nos permita salir del loop infinito.

Idea

¿Qué debería pasar en este caso?

```
1 def funcion(contador):
2   print(contador)
3   funcion(contador-1)
4
5 funcion(10)
```

Problema: Necesitamos una condición que nos permita salir del loop infinito.

```
2   if(contador == 0):
3     return
```

Idea

¿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)
```

Idea

... 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)
```

Idea

¿Qué ocurre en este caso?

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

Idea

Problema práctico: Implemente una función que calcule $n!$

Idea

Problema práctico: Implemente una función que calcule $n!$

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

Idea

Problema práctico: Implemente una función que calcule $n!$

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

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

Idea

Problema práctico: Implemente una función que calcule $n!$

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

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

¿Funciona?

Idea

Nos falta considerar los casos bases que rompen el loop infinito.

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

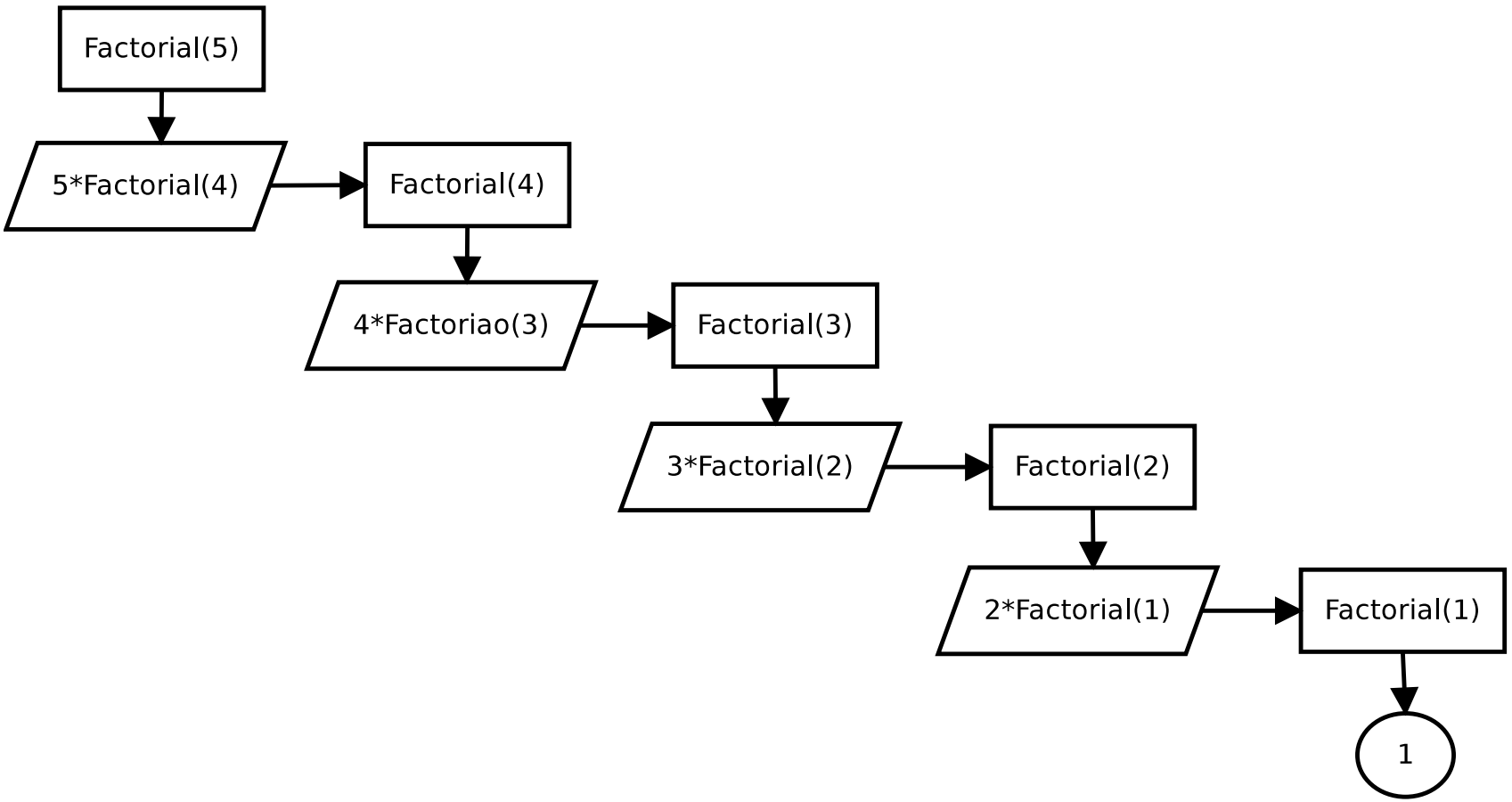
Idea

Nos falta considerar los casos bases que rompen el loop infinito.

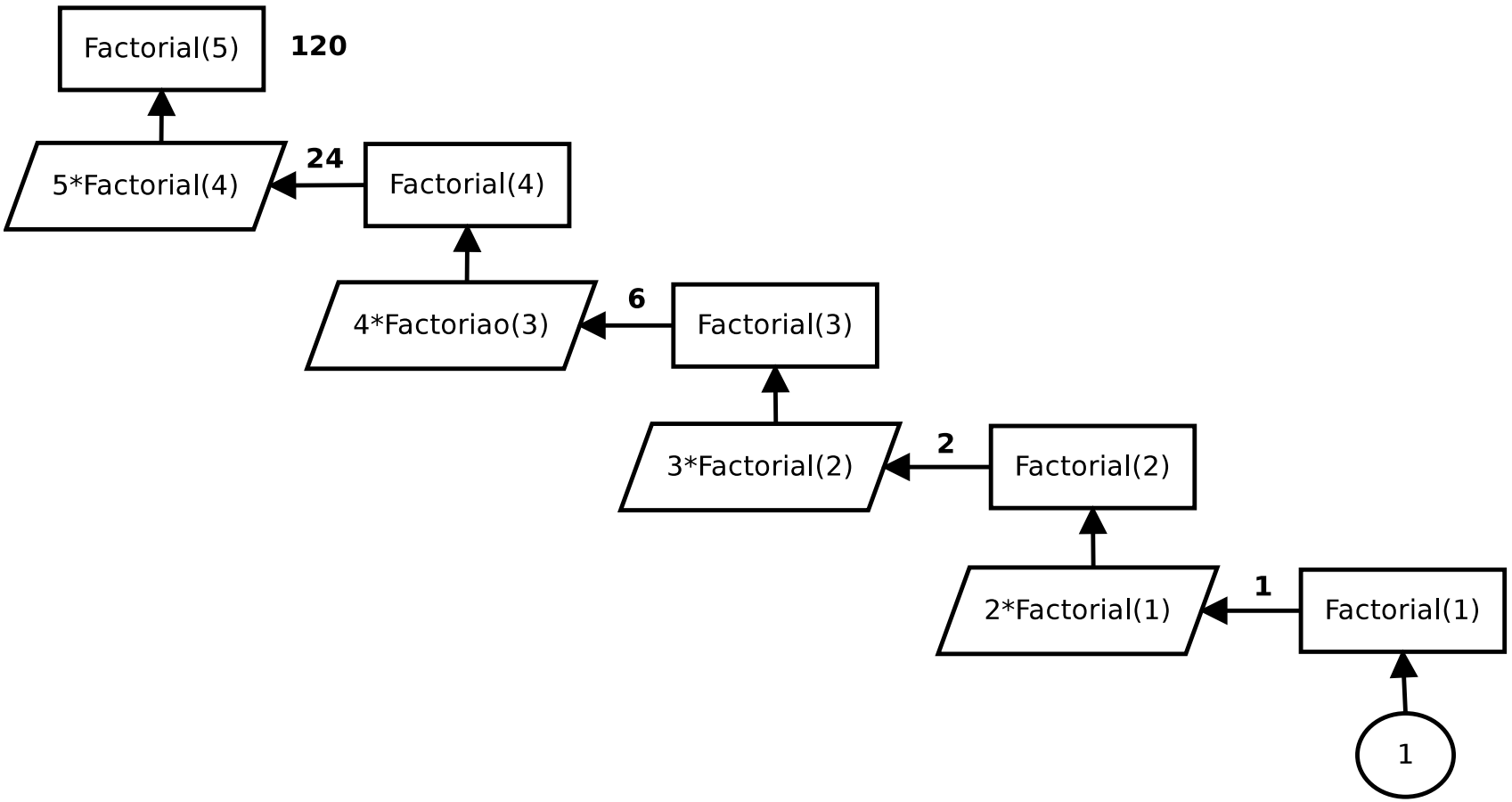
$$n! = n * (n-1)! \quad 1! = 1 \quad 0! = 1$$

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

Idea


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

Idea


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

Idea

¿Qué pasa si el caso base está bajo la llamada recursiva?

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

Idea

Desafío: Cree código recursivo que retorne el n-ésimo fibonacci (1, 1, 2, 3, 5, 8, 13, 21, etc...).

Idea

Desafío: Cree código recursivo que retorne el n-ésimo fibonacci (1, 1, 2, 3, 5, 8, 13, 21, etc...).

$$fib(n) = fib(n-2) + fib(n-1) \quad fib(1) = 1 \quad fib(2) = 1$$

Idea

Desafío: Cree código recursivo que retorne el n-ésimo fibonacci (1, 1, 2, 3, 5, 8, 13, 21, etc...).

$$fib(n) = fib(n-2) + fib(n-1) \quad fib(1) = 1 \quad fib(2) = 1$$

```
1 def fibonacci(n):
2   if(n <= 2):
3     return 1
4   return fibonacci(n-1) + fibonacci(n-2)
5
6 print(fibonacci(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?

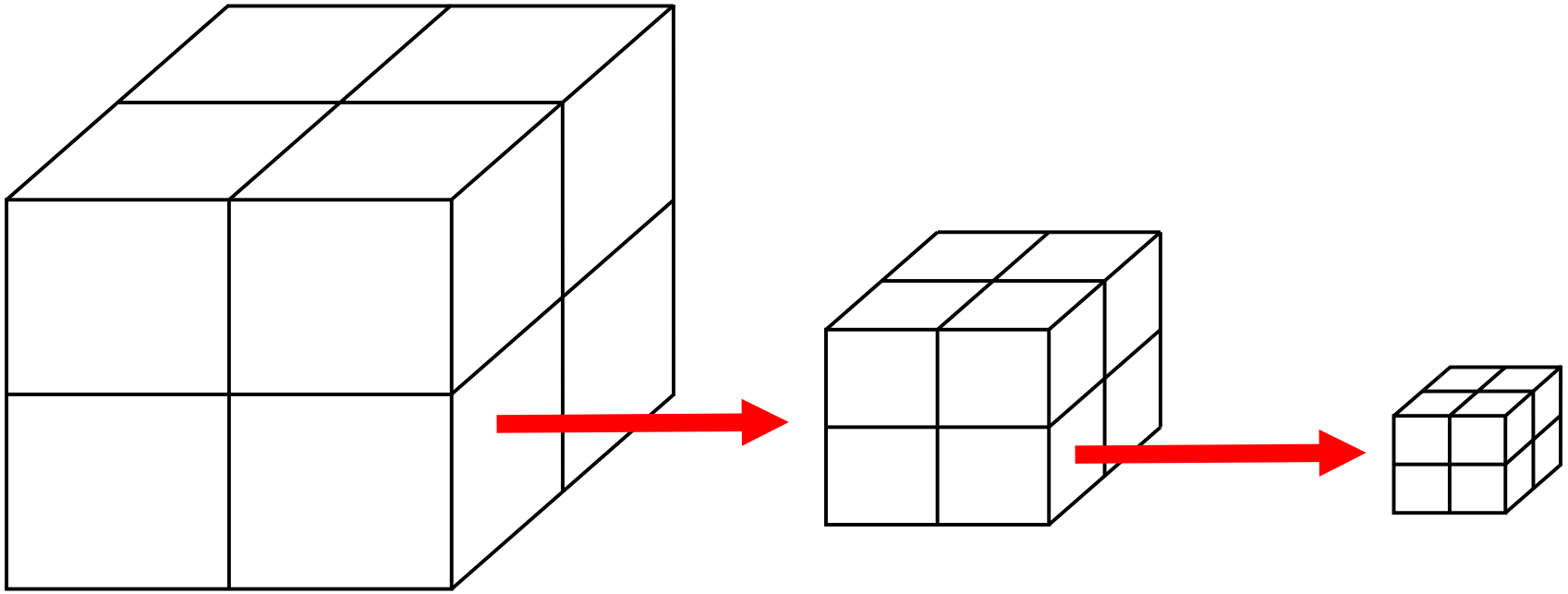
Recursión

¿Cuándo usar recursión?

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

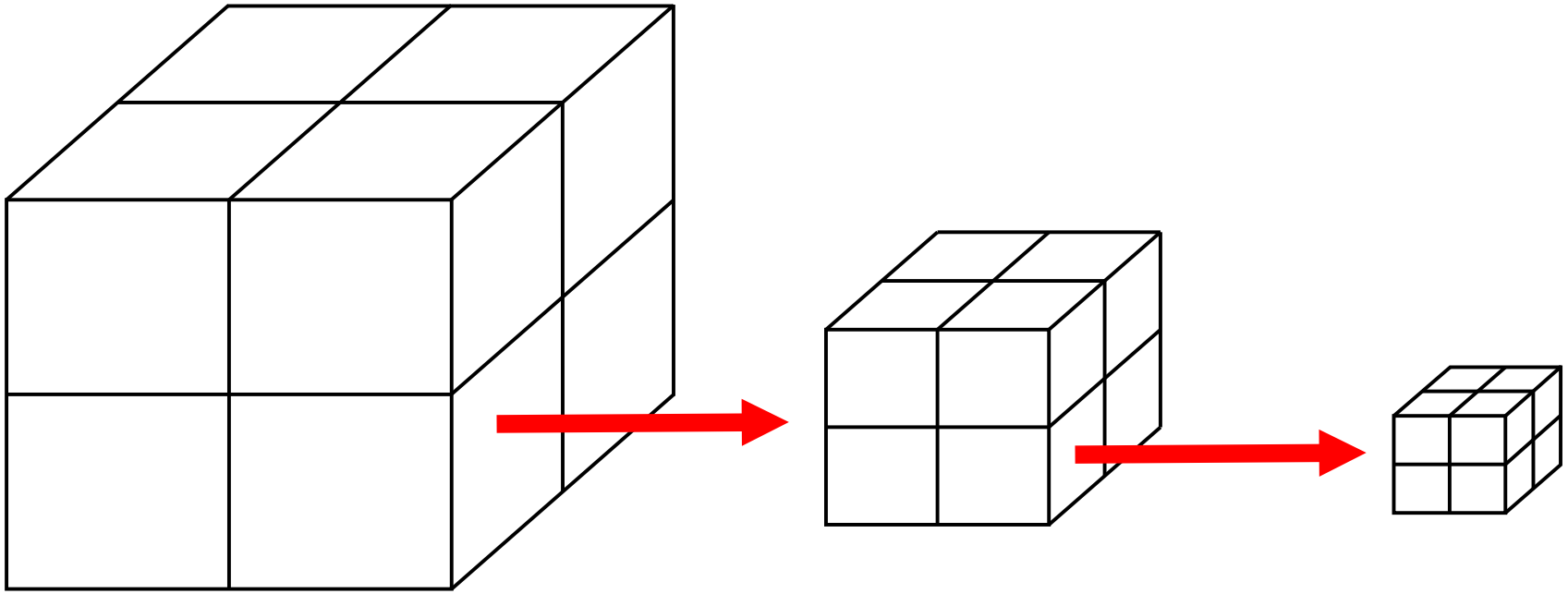
2) Para explorar el espacio en un problema de búsqueda.

Recursión



Dividir para conquistar

Recursión



Dividir para conquistar

- Dividimos el problema en subproblemas iguales.
- Solucionamos cada subproblema.
- Usamos subsoluciones para formar solución final.

Recursión

Ejemplos:

$$\text{Fib(n)} = \text{Fib(n-1)} + \text{Fib(n-2)}$$

Recursión

Ejemplos:

Fib(n) = Fib(n-1) + Fib(n-2)

Factorial(n) = n * Factorial(n-1)

Recursión

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

Recursión

```
1 def factorial(n):
```

Firma:

- **Parámetros**: Deben contener todas las variables necesarias para resolver el problema.
- **Nombre**: Función debe retornar la solución para el problema descrito en su nombre.

Recursión

```
2   if(n <= 1):
3     return 1
```

Casos bases: Subproblema menor con solución directa. Se retorna el resultado sin ningún llamado recursivo.

Recursión

```
2   if(n <= 1):
3     return 1
```

Casos bases: Subproblema menor con solución directa. Se retorna el resultado sin ningún llamado recursivo.

```
4   return n*factorial(n-1)
```

- **Llamado recursivo**: Si solución no es directa, se obtiene la solución de subproblemas usando llamadas recursivas.
- **Solución final**: A partir de subsoluciones formo solución final y la retorno.

¿Cómo ideo una solución recursiva?

¿Cómo ideo una solución recursiva?

Pasos:

- Entiendan el problema.
- Definan cuál será el input de su función.

¿Cómo ideo una solución recursiva?

Pasos:

- Entiendan el problema.
- Definan cuál será el input de su función.
- Encuentren su caso base (un caso tan simple que la solución sea trivial).

¿Cómo ideo una solución recursiva?

Pasos:

- Entiendan el problema.
- Definan cuál será el input de su función.
- Encuentren su caso base (un caso tan simple que la solución sea trivial).
- Asuman que un llamando recursivo a la función retornará el valor correcto para un input menor que el actual.

¿Cómo ideo una solución recursiva?

Pasos:

- Entiendan el problema.
- Definan cuál será el input de su función.
- Encuentren su caso base (un caso tan simple que la solución sea trivial).
- Asuman que un llamando recursivo a la función retornará el valor correcto para un input menor que el actual.
- Piensen cómo usar ese retorno para solucionar el problema para el input actual.

Ejemplos

Programe un algoritmo recursivo que sume los dígitos de un número.

Ejemplo: $12345 = 1 + 2 + 3 + 4 + 5 = 15$

Ejemplos

Programe un algoritmo recursivo que sume los dígitos de un número.

Ejemplo: $12345 = 1 + 2 + 3 + 4 + 5 = 15$

- ¿Parámetros de la función?
- ¿Cuáles son los casos base?
- ¿Cómo lo divido en subproblemas?
- ¿Cómo uno subsoluciones?

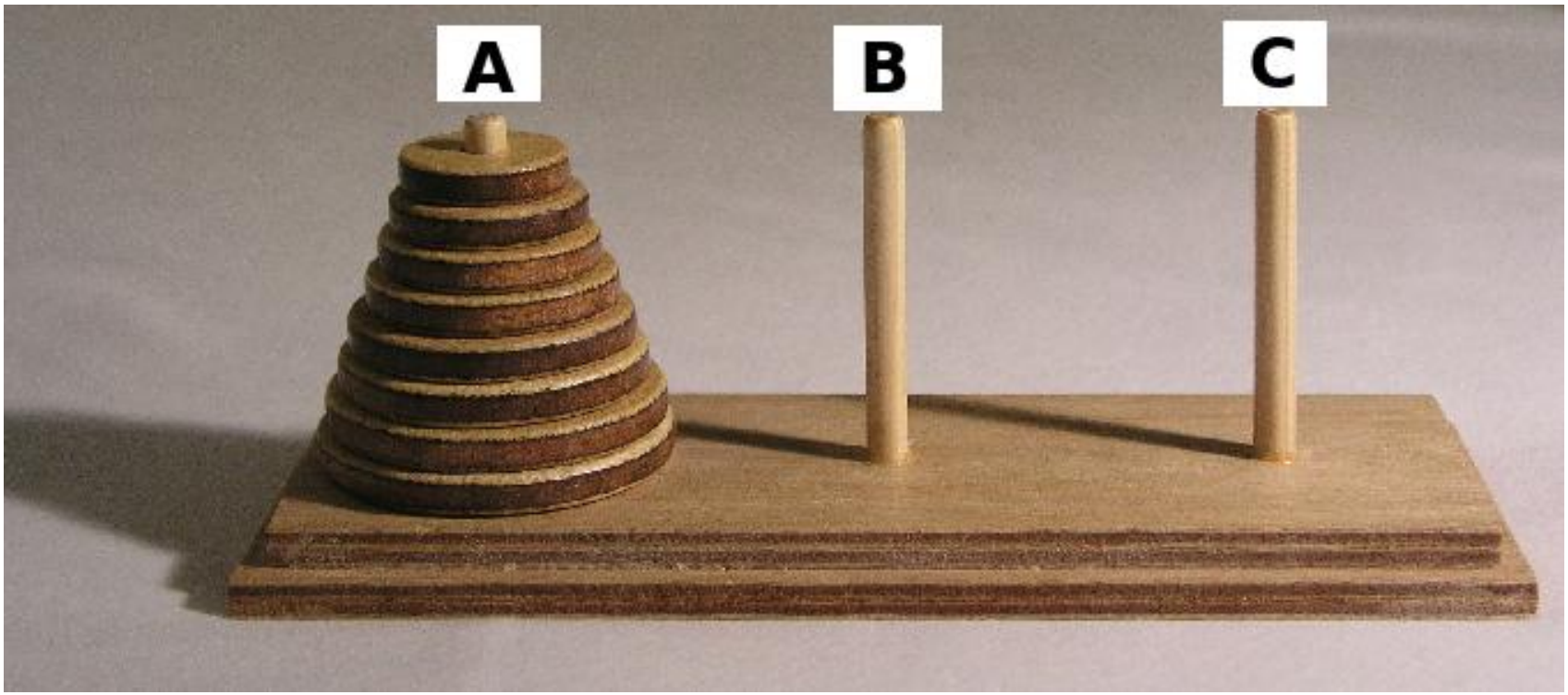
Ejemplos

Solución

```
1 def sumar_digitos(n):
2   if(n//10 == 0):
3     return n
4   return n%10 + sumar_digitos(n//10)
5
6 print(sumar_digitos(12345))
```

Ejemplos

Ejemplo: Resuelva una torre de hanoi de n discos



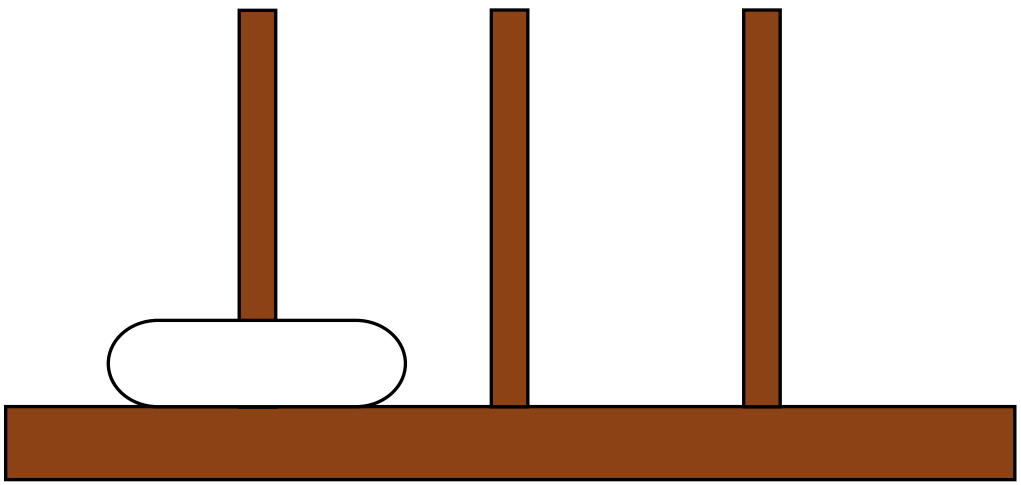
Ejemplos

Reglas:

- Se deben mover los discos desde **A** hasta **C**.
- Para esto, debe utilizar **B** como pilar auxiliar.
- En cada turno, se puede mover un disco de un pilar a otro.
- No se puede colocar un disco grande sobre uno pequeño.

Ejemplos

Caso base:



Ejemplos

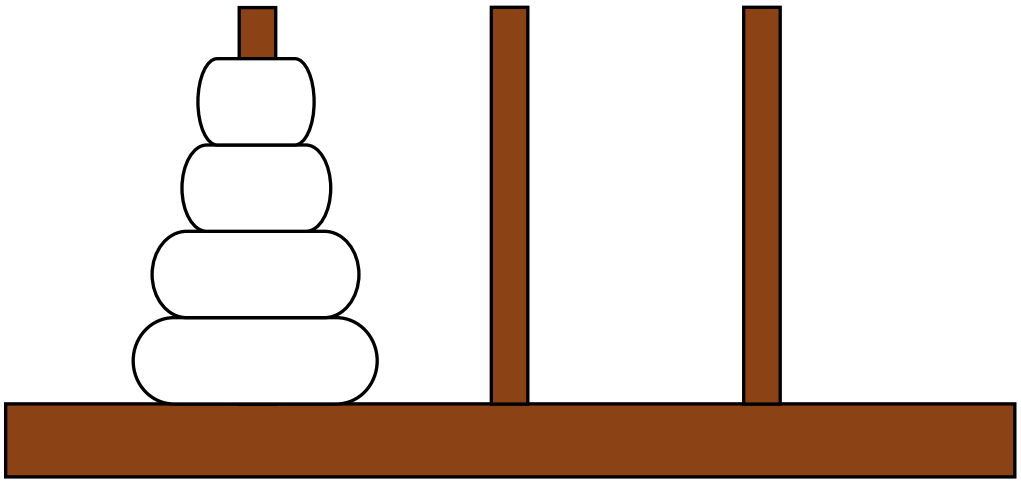
Paso inductivo: Supongamos que somos capaces de mover `n-1` discos de una columna a otra (mediante un llamado recursivo).

Ejemplos

Paso inductivo: Supongamos que somos capaces de mover `n-1` discos de una columna a otra (mediante un llamado recursivo).

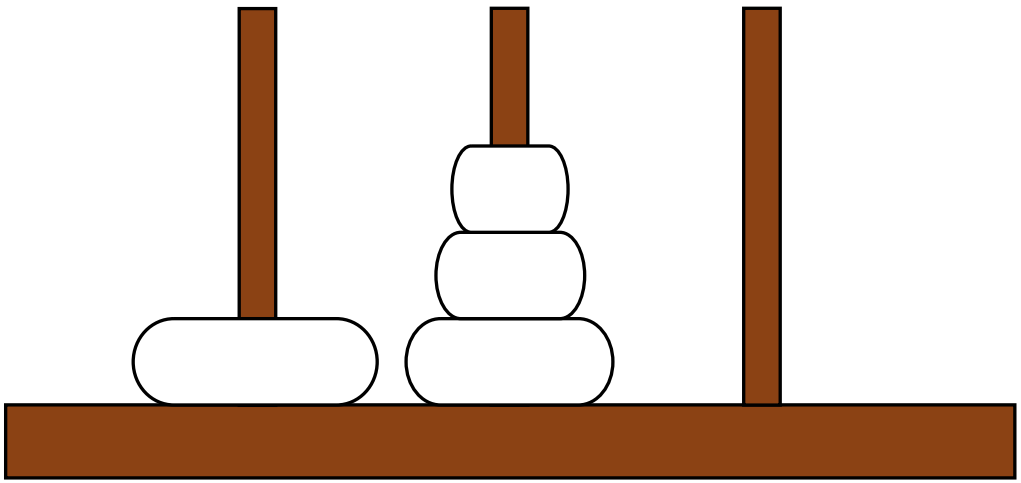
Desafío: ¿Cómo movemos una torre de `n` discos de A hasta C?

Ejemplos



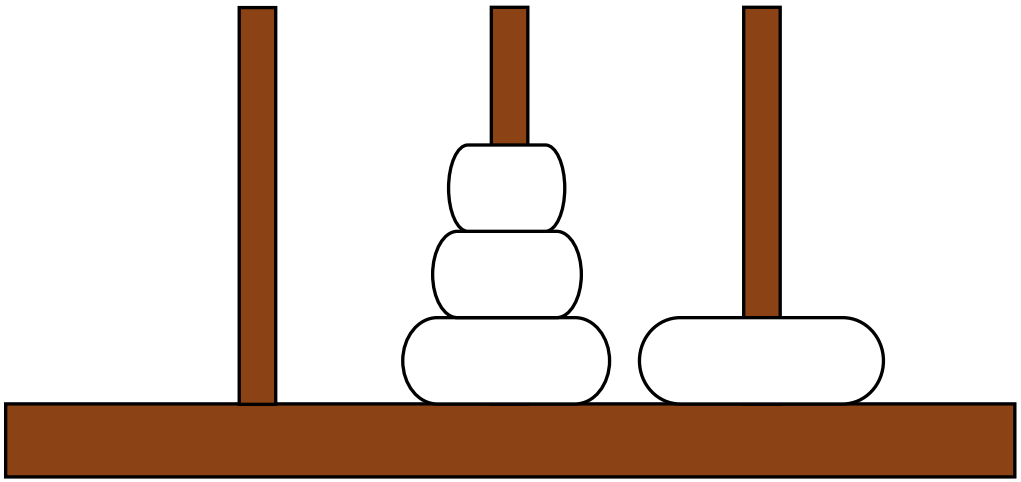
- Movemos los primeros $n-1$ discos de **A** a **B** (mediante un llamado recursivo).

Ejemplos



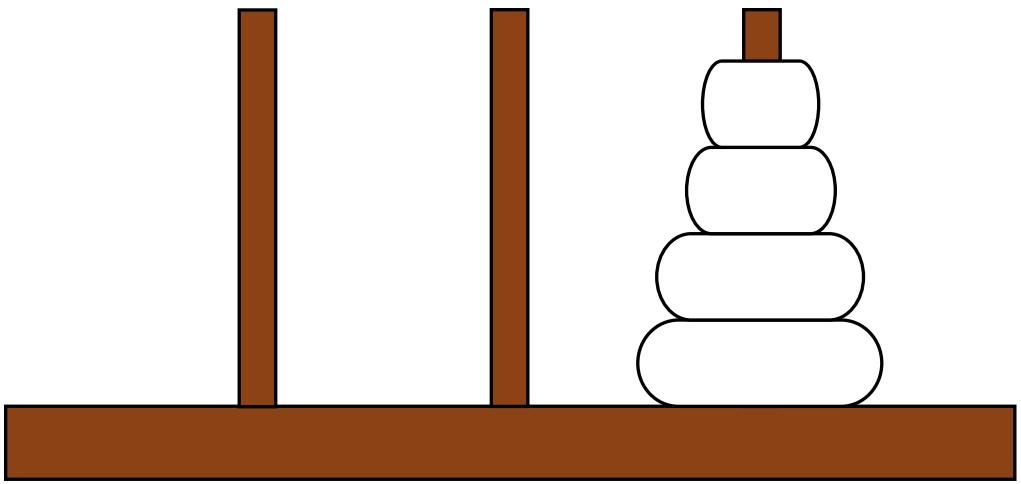
- Movemos el disco que queda en A hasta C (sin romper ninguna regla).

Ejemplos



- Finalmente podemos mover los n-1 discos de B a C (mediante un llamado recursivo).

Ejemplos



Resultado final :)

Ejemplos

Parámetros método recursivo:

```
1 def mover(n,inicio,fin,temporal):
```

Caso base:

```
2   if(n==1):
3     print("mover",inicio,fin)
```

Llamado recursivo:

```
4   else:
5     mover(n-1,inicio,temporal,fin)
6     mover(1,inicio,fin,temporal)
7     mover(n-1,temporal,fin,inicio)
```

Ejemplos

Código completo:

```
1 def mover(n,inicio,fin,temporal):
2   if(n==1):
3     print("mover",inicio,fin)
4   else:
5     mover(n-1,inicio,temporal,fin)
6     mover(1,inicio,fin,temporal)
7     mover(n-1,temporal,fin,inicio)
8
9 mover(3,"A","C","B")
```

Ejemplos

Programe una calculadora polaca.

Ejemplos:

- `+ 4 5` $\rightarrow$ `4+5`
- `+ * 4 5 6` $\rightarrow$ `(4*5)+6`
- `+ * 4 5 - 6 3` $\rightarrow$ `(4*5) + (6-3)`
- `+ * 4 / 7 5 - 6 3` $\rightarrow$ `((4*(7/5))+(6-3))`

Ver demo.

Ejemplos

La calculadora cumple con las siguientes reglas:

- num $\rightarrow$ num.
- + OP1 OP2 $\rightarrow$ OP1 + OP2.
- - OP1 OP2 $\rightarrow$ OP1 - OP2.
- * OP1 OP2 $\rightarrow$ OP1 * OP2.
- / OP1 OP2 $\rightarrow$ OP1 / OP2.

Ejemplos

Solución:

```
1  def calcular():
2    n = input()
3    # Caso base
4    if(n!="+" and n!="-" and n!="*" and n!="/"):
5      return int(n)
6    # Retorno resultado según operación
7    if(n == "+"):
8      return calcular() + calcular()
9    if(n == "-"):
10     return calcular() - calcular()
11   if(n == "*"):
12     return calcular() * calcular()
13   if(n == "/"):
14     return calcular() / calcular()
```

Control 1: Cuadrado Mágico

		Columna			
		0	1	2	3
F	0	**16**	**3**	**2**	**13**
i	1	**5**	**10**	**11**	**8**
l	2	**9**	**6**	**7**	**12**
a	3	**4**	**15**	**14**	**1**

Problemas:

- Encontrar constante mágica.
- Verificar filas sumen constante mágica.
- Verificar que números sean únicos.

Control 1: Cuadrado Mágico

```
"""
Reemplaza este texto por tu nombre
Reemplaza este texto por tu número de alumno
Reemplaza este texto por tu correo UC
"""
import cuadradomagico

def validar():
  cumpleRegla1=True
  cumpleRegla2=True
  constante_magica=0

  #Comienza aquí la respuesta de tu control

  #Termina aquí la respuesta de tu control

  return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Funciones:

- `cuadradomagico.dimensionTablero()`: Retorna la dimensión del tablero actualmente cargado.
- `cuadradomagico.obtener(i,j)`: Retorna el valor de la casilla ubicada en la posición (i,j).

Control 1: Cuadrado Mágico

Pregunta 1: Encontrar constante mágica.

Control 1: Cuadrado Mágico

Pregunta 1: Encontrar constante mágica.

```
8  def validar():
9    cumpleRegla1=True
10   cumpleRegla2=True
11   constante_magica=0
12   #Comienza aquí la respuesta de tu control
13
14   # Obtengo dimensión
15   n = cuadradomagico.dimensionTablero()
16   # Pregunta 1
17   j = 0
18   while(j < n):
19     constante_magica+=cuadradomagico.obtener(0,j)
20     j+=1
21
22   #Termina aquí la respuesta de tu control
23   return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Error 1: No respetar tab de la función.

Control 1: Cuadrado Mágico

Error 1: No respetar tab de la función.

```
8  def validar():
9    cumpleRegla1=True
10   cumpleRegla2=True
11   constante_magica=0
12   #Comienza aquí la respuesta de tu control
13
14 # Obtengo dimensión
15 n = cuadradomagico.dimensionTablero()
16 # Pregunta 1
17 j = 0
18 while(j < n):
19   constante_magica+=cuadradomagico.obtener(0,j)
20   j+=1
21
22   #Termina aquí la respuesta de tu control
23   return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Error 2: Retornar antes de tiempo.

Control 1: Cuadrado Mágico

Error 2: Retornar antes de tiempo.

```
8  def validar():
9    cumpleRegla1=True
10   cumpleRegla2=True
11   constante_magica=0
12   #Comienza aquí la respuesta de tu control
13
14   # Obtengo dimensión
15   n = cuadradomagico.dimensionTablero()
16   # Pregunta 1
17   j = 0
18   while(j < n):
19     constante_magica+=cuadradomagico.obtener(0,j)
20     j+=1
21   return constante_magica
22
23   #Termina aquí la respuesta de tu control
24   return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Error 3: Definir funciones dentro de `validar()`.

Control 1: Cuadrado Mágico

Error 3: Definir funciones dentro de `validar()`.

```
8  def validar():
9    cumpleRegla1=True; cumpleRegla2=True
10   constante_magica=0
11   #Comienza aquí la respuesta de tu control
12
13   # Obtengo dimensión
14   n = cuadradomagico.dimensionTablero()
15   # Pregunta 1
16   def obtener_constante_magica():
17     j = 0; c = 0
18     while(j < n):
19       c+=cuadradomagico.obtener(0,j)
20       j+=1
21     return c
22
23   #Termina aquí la respuesta de tu control
24   return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Error 4: Parámetros funciones.

Control 1: Cuadrado Mágico

Error 4: Parámetros funciones.

```
8  def obtener_constante_magica():
9    j = 0; c = 0
10   while(j < n):
11     c+=cuadradomagico.obtener(0,j)
12     j+=1
13   return c
14
15 def validar():
16   cumpleRegla1=True; cumpleRegla2=True
17   constante_magica=0
18   #Comienza aquí la respuesta de tu control
19   # Pregunta 1
20   n = cuadradomagico.dimensionTablero()
21   constante_magica = obtener_constante_magica()
22   #Termina aquí la respuesta de tu control
23   return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Python permite ver valores de variables en scope superior.

Control 1: Cuadrado Mágico

Python permite ver valores de variables en scope superior.

```
1 def mostrar():
2   i = 1
3   while(i < n):
4     print(i)
5     i+=1
6
7 n = 5
8 mostrar()
```

Control 1: Cuadrado Mágico

... pero no de scopes hermanos.

Control 1: Cuadrado Mágico

... pero no de scopes hermanos.

```
1  def mostrar():
2    i = 1
3    while(i < n): # -> ERROR!
4      print(i)
5      i+=1
6
7  def main():
8    n = 5
9    mostrar()
10
11 main()
```

Control 1: Cuadrado Mágico

Consejo: Toda variable externa que usen en una función agréguenla a sus parámetros.

Control 1: Cuadrado Mágico

Consejo: Toda variable externa que usen en una función agréguenla a sus parámetros.

```
1  def mostrar(dim):
2    i = 1
3    while(i < dim): # -> Funciona!
4      print(i)
5      i+=1
6
7  def main():
8    n = 5
9    mostrar(n)
10
11 main()
```

Control 1: Cuadrado Mágico

Error 5: Llamar a una función igual que una variable.

Control 1: Cuadrado Mágico

Error 5: Llamar a una función igual que una variable.

```
8  def constante_magica(n):
9    j = 0; c = 0
10   while(j < n):
11     c+=cuadradomagico.obtener(0,j)
12     j+=1
13   return c
14
15 def validar():
16   cumpleRegla1=True; cumpleRegla2=True
17   constante_magica=0
18   #Comienza aquí la respuesta de tu control
19   # Pregunta 1
20   n = cuadradomagico.dimensionTablero()
21   constante_magica = constante_magica(n) # ERROR!
22   #Termina aquí la respuesta de tu control
23   return cumpleRegla1,cumpleRegla2,constante_magica
```

Control 1: Cuadrado Mágico

Solución al control:

- Sin funciones.
- Con funciones.

Control 1: Solución sin funciones

Control 1: Solución sin funciones

Pregunta 1: Encontrar constante mágica.

```
15  # Obtengo dimensión
16  n = cuadradomagico.dimensionTablero()
17
18  # Pregunta 1
19  j = 0
20  while(j < n):
21      constante_magica+=cuadradomagico.obtener(0,j)
22      j+=1
```

Control 1: Solución sin funciones

Pregunta 2: Verificar filas sumen constante mágica.

Control 1: Solución sin funciones

Pregunta 2: Verificar filas sumen constante mágica.

```
24  # Pregunta 2
25  i = 0
26  while(i < n):
27    j = 0; suma_fila = 0
28    tiene_cero = False
29    while(j < n):
30      suma_fila+=cuadradomagico.obtener(i,j)
31      if(cuadradomagico.obtener(i,j) == 0):
32        tiene_cero = True
33      j+=1
34    if(not tiene_cero and
35        suma_fila != constante_magica):
36      cumpleRegla2 = False
37    i+=1
```

Control 1: Solución sin funciones

Pregunta 3: Verificar que números sean únicos.

Control 1: Solución sin funciones

Pregunta 3: Verificar que números sean únicos.

```
39  # Pregunta 3
40  v = 1
41  while(v <= n**2):
42      contador = 0
43      i = 0
44      while(i < n):
45          j = 0
46          while(j < n):
47              if(cuadradomagico.obtener(i,j) == v):
48                  contador += 1
49              j+=1
50          i+=1
51      if(contador > 1):
52          cumpleRegla1 = False
53      v+=1
```

Control 1: Solución con funciones

Control 1: Solución con funciones

Pregunta 1: Encontrar constante mágica.

```
8  def obtener_constante_magica(n):
9    constante = 0
10   j = 0
11   while(j < n):
12     constante+=cuadradomagico.obtener(0,j)
13     j+=1
14   return constante
```

Control 1: Solución con funciones

Pregunta 2: Verificar filas sumen constante mágica.

```
16 def validar_regla2(n):
17   i = 0
18   while(i < n):
19     j = 0; suma_fila = 0
20     tiene_cero = False
21     while(j < n):
22       suma_fila+=cuadradomagico.obtener(i,j)
23       if(cuadradomagico.obtener(i,j) == 0):
24         tiene_cero = True
25       j+=1
26     if(not tiene_cero and suma_fila !=
       constante_magica):
27       return False
28     i+=1
29   return True
```

Control 1: Solución con funciones

Pregunta 3: Verificar que números sean únicos.

```
31 def validar_regla1(n):
32   v = 1
33   while(v <= n**2):
34     contador = 0
35     i = 0
36     while(i < n):
37       j = 0
38       while(j < n):
39         if(cuadradomagico.obtener(i,j) == v):
40           contador += 1
41         j+=1
42       i+=1
43     if(contador > 1):
44       return False
45     v+=1
46   return True
```

Control 1: Solución con funciones

Finalmente, desde `validar()` llamamos a las funciones.

```
48 def validar():
49   cumpleRegla1=True
50   cumpleRegla2=True
51   constante_magica=0
52   #Comienza aquí la respuesta de tu control
53
54   n = cuadradomagico.dimensionTablero()
55   constante_magica = obtener_constante_magica(n)
56   cumpleRegla2 = validar_regla2(n)
57   cumpleRegla1 = validar_regla1(n)
58
59   #Termina aquí la respuesta de tu control
60   return cumpleRegla1,cumpleRegla2,constante_magica
```

Ejercicios Propuestos

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

2) ¿Qué hace la siguiente función?

```
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
```