IIC1103 - Sección 4

Clase 11: Funciones recursivas

Prof. Francisca Cattan

Universidad Católica de Chile

13 de septiembre, 2022

En la clase anterior

- Aprendimos a usar módulos de Python e importar funciones como `math - sqrt, exp, log, pi, degrees; random - randint`.
- Aprendimos cómo crear nuestras propias funciones y llamarlas.

En la clase anterior: Funciones de Python

```
1 from random import random, randint, randrange
2
3 print(random())                # float entre 0 y 1
4 print(randint(1,2))           # random entre [1,2]
5 print(randrange(4))           # random entre [0,1,2,3]
6 print(randrange(1,4))         # random entre [1,2,3]
7 print(randrange(1,4,2))       # random entre [1,3]
8 print("texto prueba")
```

```
1 from math import exp,cos,pi,log
2
3 print(exp(5.5))
4 print(cos(pi/2))
5 print(log(exp(4)))
```

Funciones propias

La instrucción **`def`** permiten definir un nombre para un trozo de código. Permite recibir parámetros y retorna un valor.

Sintaxis: definir una función

```
1 def nombre_funcion(param1, param2):     # Encabezado
2     cuerpo_funcion                      # Este bloque se ejecuta
3     .                                   # en cada llamada de la funcion
4     .
5     .
6     return ret                          # Valor de retorno
```

Funciones propias

La instrucción **`def`** permiten definir un nombre para un trozo de código. Permite recibir parámetros y retorna un valor.

Sintaxis: definir una función

```
1 def nombre_funcion(param1, param2):     # Encabezado
2     cuerpo_funcion                      # Este bloque se ejecuta
3     .                                   # en cada llamada de la funcion
4     .
5     .
6     return ret                          # Valor de retorno
```

Sintaxis: llamar a la función creada

```
1 out = nombre_funcion(input1, input2)     # Usar la funcion
```

Funciones propias

La instrucción **`def`** permiten definir un nombre para un trozo de código. Permite recibir parámetros y retorna un valor.

Sintaxis: definir una función

```
1 def nombre_funcion(param1, param2):     # Encabezado
2     cuerpo_funcion                      # Este bloque se ejecuta
3     .                                   # en cada llamada de la funcion
4     .
5     .
6     return ret                          # Valor de retorno
```

Sintaxis: llamar a la función creada

```
1 out = nombre_funcion(input1, input2)     # Usar la funcion
```

Obs: En esta sintaxis, el retorno de la función será asignado a la variable `out`.

En la clase anterior: Funciones propias

Parámetros:

- Lista de variables que recibe la función para trabajar.
- Podría no tener parámetros.

Código:

- El código de una función no tiene ninguna restricción (puede ser un programa completo).

Retorno:

- Sirve para entregar un resultado a quien llame a la función
- Una funció podría no retornar nada.
- Al retornar la función se acaba.

Uso recursivo de funciones

En programación existen dos formas de enfrentar un problema:

- En forma iterativa.
- En forma recursiva.

Ambos estilos son buenos, pero hay muchos problemas que pueden ser abordados manera muy simple mediante una solución recursiva.

Uso recursivo de funciones

En programación existen dos formas de enfrentar un problema:

- En forma iterativa: Una función iterativa repite una parte del código mediante ciclos.
- En forma recursiva: Una función es recursiva si se llama a si misma para repetir el código.

Uso recursivo de funciones

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

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

Uso recursivo de funciones

Pregunta: ¿Qué pasa en este caso y cómo debemos intervenir el código?

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

Uso recursivo de funciones

Pregunta: ¿Qué pasa en este caso y cómo debemos intervenir el código?

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

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

Uso recursivo de funciones

Pregunta: ¿Qué pasa en este caso y cómo debemos intervenir el código?

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

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

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

Uso recursivo de funciones

Pregunta: ¿Cual 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)
```

Uso recursivo de funciones

Pregunta: ¿Cual 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)
```

5

4

3

2

1

Uso recursivo de funciones

Pregunta: Y ahora... ¿cual es la salida de este programa?

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

Uso recursivo de funciones

Pregunta: Y ahora... ¿cual es la salida de este programa?

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

1

2

3

4

5

Uso recursivo de funciones

Pensemos juntos: Implemente una función que calcule $n!$

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

Uso recursivo de funciones

Pensemos juntos: Implemente una función que calcule $n!$

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

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

Pregunta: ¿Funciona?

Uso recursivo de funciones

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

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

$$1! = 1$$

$$0! = 1$$

Cada función recursiva debe tener una condición base que detenga la recursividad o, de lo contrario, **la función se llama a sí misma infinitamente**.

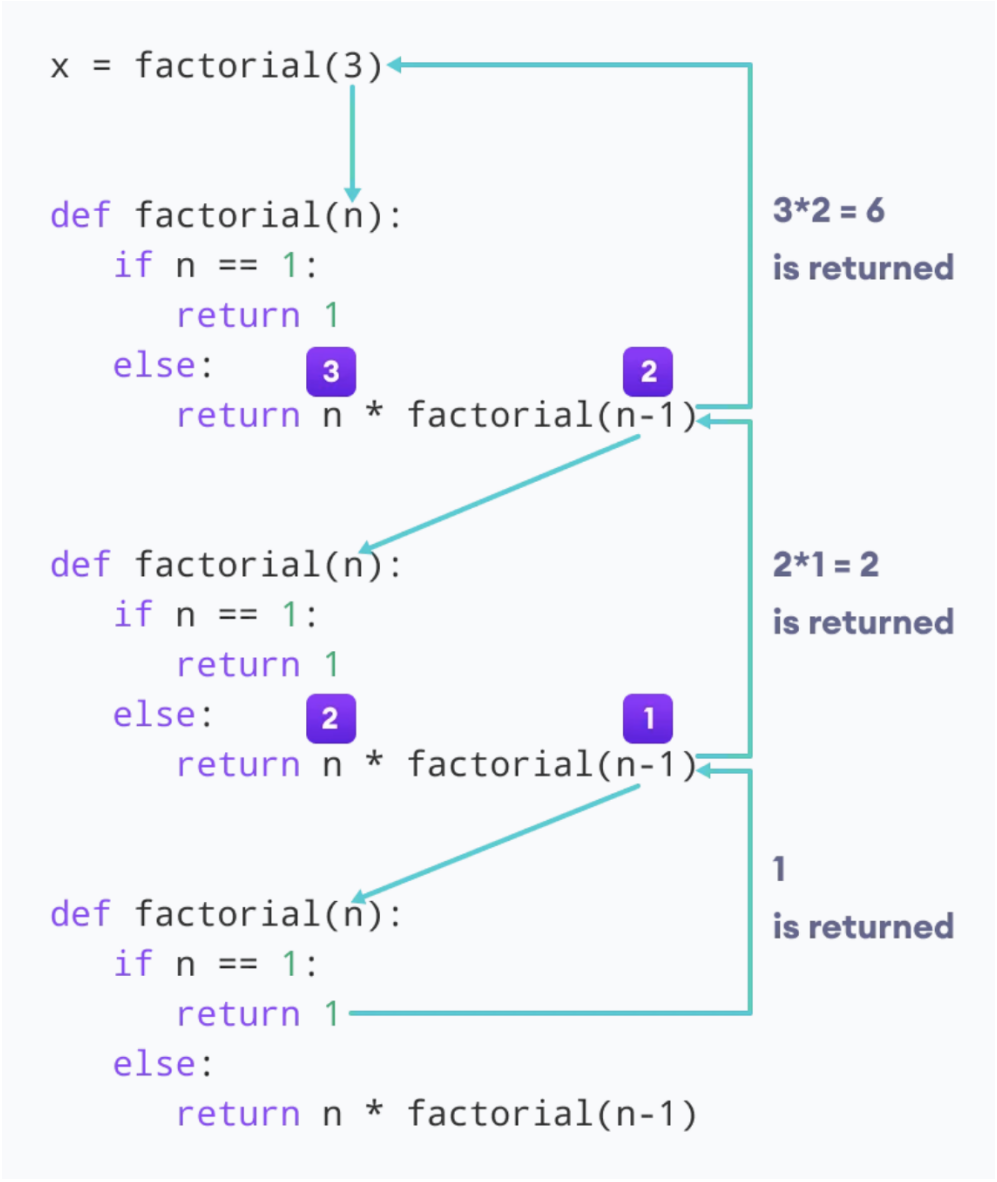
Uso recursivo de funciones

Pensemos juntos: Implemente una función que calcule $n!$

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

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

Uso recursivo de funciones



Uso recursivo de funciones

Ventajas de la recursividad

1. Las funciones recursivas hacen que el código se vea limpio y elegante.
2. Una tarea compleja se puede dividir en subproblemas más simples mediante la recursividad.
3. La generación de secuencias es más fácil con la recursividad que con alguna iteración anidada.

Uso recursivo de funciones

Ventajas de la recursividad

1. Las funciones recursivas hacen que el código se vea limpio y elegante.
2. Una tarea compleja se puede dividir en subproblemas más simples mediante la recursividad.
3. La generación de secuencias es más fácil con la recursividad que con alguna iteración anidada.

Desventajas de la recursividad

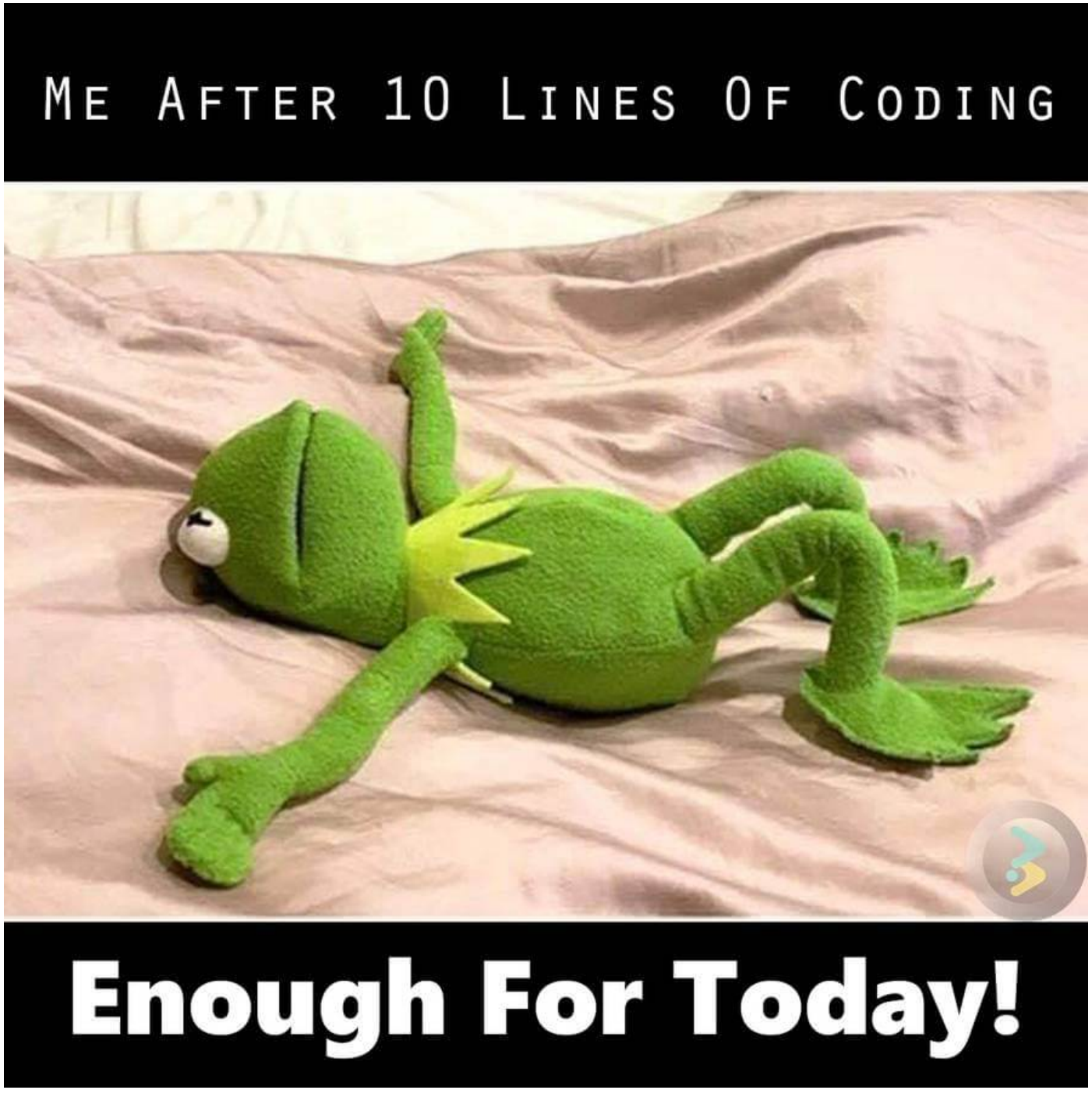
1. A veces, la lógica detrás de la recursividad es difícil de seguir.
2. Las llamadas recursivas son caras (ineficientes) ya que ocupan mucha memoria y tiempo.
3. Las funciones recursivas son difíciles de depurar


CHALLENGE ACCEPTED
LET'S CODE IT
memegenerator.net

Uso recursivo de funciones

Ejercicios:

1. Cree codigo recursivo que retorne el n-esimo fibonacci (1, 1, 2, 3, 5, 8, 13, 21, etc...).
2. Programe un algoritmo recursivo que sume los dıgitos de un numero. Ejemplo: 12345=1+2+3+4+5=15
3. Escriba una función que reste de 3 unidades a un entero positivo y llegue a 0.
4. Escriba un programa que calcule la suma armónica hasta $n1$.
5. Sea $pow(x, y) = x^y$. Escriba un programa que calcule la suma geométrica hasta $n - 1$ de cierto número x.
6. Escriba una función que encuentre el máximo común divisor entre dos números usando recursión.


Me After 10 Lines Of Coding
Enough For Today!