
**Programación Unidad 3:** Programación orientada a objetos y librerías

Ph.D. Santiago Echeverri-Arteaga

# Índice

# Índice

“Para entender la recursividad, primero hay que entender la recursividad”


X 06
X 067 EH 9


TOP DEFINITION
recursion
See recursion.
by Anonymous December 05, 2002
1007
43

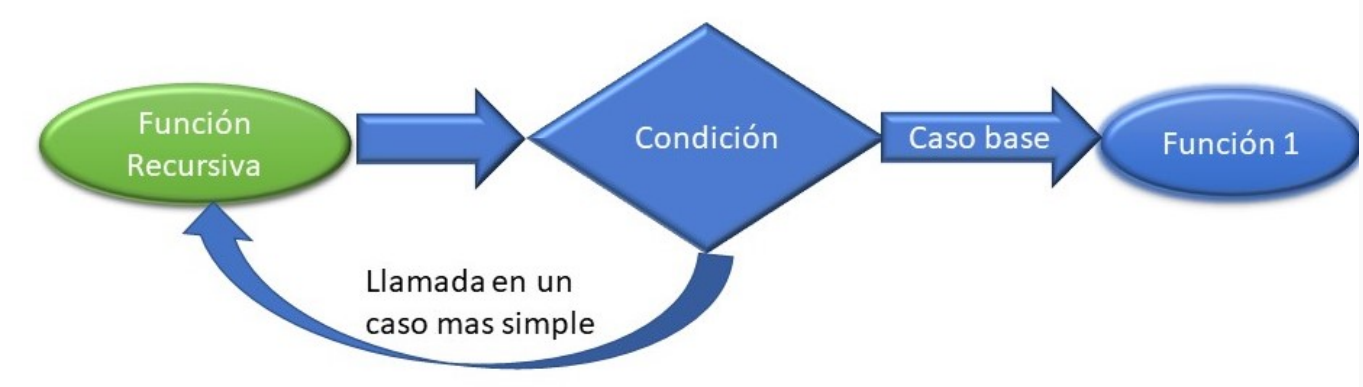
# Recursividad

# Definición

La recursividades un método matemático de abordar problemas sustituyéndolos por otros problemas de la misma categoría, pero más simples.
Debe existir un caso base (el más simple) que se resuelva de forma convencional.

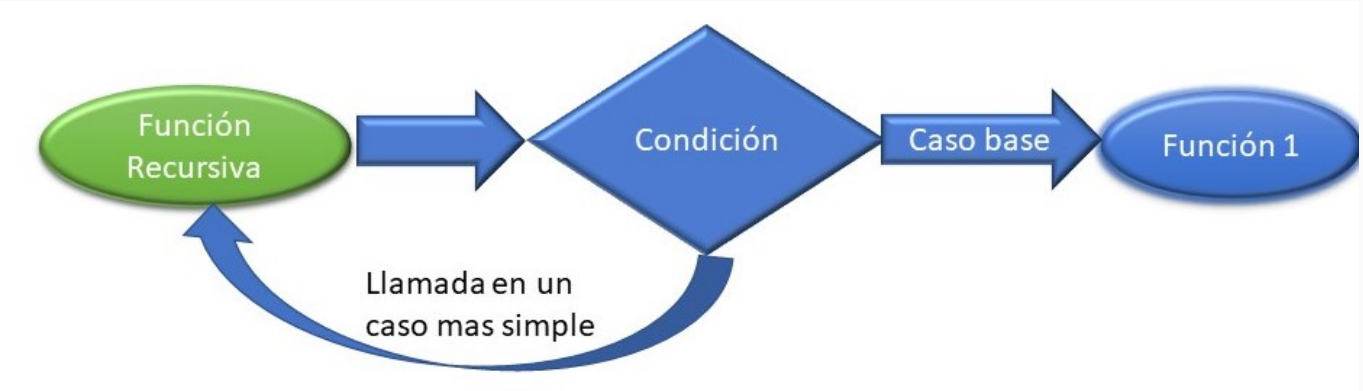
Optamos por una solución recursiva cuando sabemos cómo resolver de mane-ra directa un problema para un cierto conjunto de datos, y para el resto de los datos somos capaces de resolverlo utilizando la solución al mismo problema con unos datos “más simples”.
Implementar una recursión es hacer una función que se llame a si misma en casos cada vez más simples

# Esquema de una recursión




1. **Recursión simple:**

```
def recursion_simple:
    VARIABLES LOCALES
if CASO BASE:
    FUNCION BASE
else
    recursion_simple en caso mas sencillo
```




1. **Recursión múltiple:**

```
def recursion_multiple:
    VARIABLES LOCALES
if CASO BASE:
    FUNCION BASE
else
    recursion_multiple en caso mas sencillo
    recursion_multiple en otro caso mas sencillo
    ...
    Combinacion recursiones simples
```

# Factorial

# Factorial

- factorial(n) = n * factorial(n-1)
  factorial(1) = 1
-
-

# Factorial

- factorial(n) = n * factorial(n-1)
  factorial(1) = 1

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

-
  ```

# Factorial

- factorial(n) = n * factorial(n-1)
  factorial(1) = 1

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

- factorial = **lambda** n : 1 if n <= 1 else n*factorial(n-1)
  ```

# Método de Euclides

# Máximo común divisor

Realizar un programa que calcule el MCD, para ello use el algoritmo de Euclides

1. Calcule el residuo de dividir el número más grande por el número más pequeño
2. Reemplace el número más grande con el número más pequeño y el número más pequeño con el residuo.
3. Repita este proceso hasta que el número menor sea 0. El MCD es el último residuo diferente de cero

# Máximo común divisor

Realizar un programa que calcule el MCD, para ello use el algoritmo de Euclides

1. Calcule el residuo de dividir el número más grande por el número más pequeño
2. Reemplace el número más grande con el número más pequeño y el número más pequeño con el residuo.
3. Repita este proceso hasta que el número menor sea 0. El MCD es el último residuo diferente de cero

- **Caso base:** $b = 0$ El MCD es el valor de $a$

# Fibonacci

# Factores primos de un número

# map, filter y reduce

# map filter y reduce

- La forma vieja de hacer comprensión de listas: map y filter:
- L = list(map(len, ['this', 'is', 'a', 'test']))
- L = [len(word) for word in ['this', 'is', 'a', 'test'] ]
- L = list(filter(lambda x: len(x)>2, ['this', 'is', 'a', 'test']))
- L = [word for word in ['this', 'is', 'a', 'test'] if len(word)>2]
- **reduce**: Ha sido movida a la librería functools.
  Toma una función y un iterable y aplica la función a los elementos de izquierda a derecha, acumulando el resultado.
  Ejemplo sin reduce:
- total = 0
  for i in range(1,101):
      total = total + i
- Con **reduce**:
  total = reduce(lambda x,y: x+y, range(1,101))

# Reduce

Se suele usar reduce con el módulo operator:
https://docs.python.org/3/library/operator.html

- from operator import add
  total = reduce(add, range(1,101))

# *args y **kwargs

# ¿Cómo funciona print?

A la función print se le pueden pasar N argumentos separados por , y todos ellos los va a imprimir en pantalla.

¿Cómo hago una función de ese tipo? Es decir, una función a la que le pueda pasar 1 o N argumentos y entienda que son del “mismo tipo”.

Además, cuando llamo a la función print, puedo poner un argumento llamado sep y otro end y los identifica como variables adicionales (que no se deben imprimir)

¿Cómo hago eso?

# *args

Argumento precedido de un * al momento de definir la función
Argumento sin palabra clave (tupla que guarda todos los argumentos que se pasen a la función separados por coma)
Deben ir después de los argumentos normales (posicionales).
Importa el orden y no se puede usar igualación para los argumentos posicionales.

- def myFun(*args):
  print(“args: ”, args)
- def product(*nums):
  prod = 1
  for i in nums:
  prod*=i
  return prod

Se le puede pasar de argumento a la función una variable (TUPLA) precedida de un *

# **kwargs

Argumento precedido de dos ** al momento de definir la función
Un kwarg es un argumento **con palabra clave**, en particular un diccionario que guarda todos los argumentos que se pasen a la función separados por coma e igualados a una llave.
Deben ir después de los *args
Importa el orden y no se puede usar igualación para los argumentos posicionales

- def myFun(*args,**kwargs):
  print(“args: ”, args)
  print(“kwargs: ”, kwargs)
- def product(**keyargs):
  for i in keyargs:
  print(’{0}^2= {1}’.format(i,keyargs[i]**2))

# Argumentos posicionales, *args y **kwargs simultaneamente

- def func(a, b, c=5, *d, **e):
- def func(a, b, *c, d=5, **e):
- Se pueden llamar a las funciones que tengan *args o kwargs como funcion(*arg) o funcion(**kwarg) en donde arg es una tupla y kwarg un diccionario

# ¿Cómo puedo contar cuantas veces se ha llamado una función?

# ¿Cómo puedo contar cuantas veces se ha llamado una función?

- ```
  def f():
      f.count = f.count+1
      print(f.count)
  f.count=0
  print(f.count)
  ```
  ```

# Ejercicio

# Ejercicio

Realice un programa que le solicite al usuario el número de discos de la torre de Hannoi, el tiempo que el usuario se demora en mover cada disco, el poste de origen y el de destino. El programa debe dar el paso a paso de como resolver la torre de Hannoi, decir cuantos movimientos debe realizar y el tiempo que el usuario se demoraría resolviendola.