Curso de Desarrollo en Lenguaje Python para Inteligencia Artificial (Málaga)

M.374.001.001

22 de marzo 2020 09:30-13:30

Módulo 1 - Tema 6.2

Carmen Bartolomé

Gobierno de España · Ministerio de Industria, Comercio y Turismo · EOI Escuela de organización industrial

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

polo de contenidos digitales

Ayuntamiento de Málaga

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
Escuela de organización industrial

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Módulo 1: Aprendiendo Python. El core de Python

Desarrollo en lenguaje Python

Año de realización: 2021

PROFESORA
Carmen Bartolomé Valentín-Gamazo

Escuela de organización industrial

www.eoi.es

Índice

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Tema 6-2

Funciones Recursivas, Anónimas y Decoradoras

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

Recursión

Funciones recursivas

- Una función puede llamar a otra dentro de su definición
- Pero una función puede también llamarse a sí misma dentro de su definición.
- Ese tipo de funciones se llama “función recursiva”, y al proceso de ejecución, lo denominamos “recursión”.

Funciones recursivas

```
def funcion_recursiva(parametro):
    if condicion_parada:
        …
    else:
        …
        funcion_recursiva(expresion_parametro)
```

1º nivel de ejecución (argumento)
    argumento ≠ condición de parada
    codigo con argumento
    2º nivel de ejecución (argumento2)
        argumento2 ≠ condición de parada
        código con argumento2
        3º nivel de ejecución (argumento3)
            …
            …
                nº nivel de ejecución (argumento_n)
                    argumento_n = condición de parada
                    código de parada
                se cierra el nº nivel de ejecución
            …
        se cierra el 3º nivel de ejecución
    se cierra el 2º nivel de ejecución
se cierra el 1º nivel de ejecución

Diagrama de pilas (Stack Diagram)

```
def mostrar_doble(expresion):
        print(expresion)
        print(expresion)

def repetir(parte1, parte2):
        mensaje = parte1 + parte2
        mostrar_doble(mensaje)

primero = 'supercalifragilistico'
segundo = 'expialidoso'
repetir(primero, segundo)
```

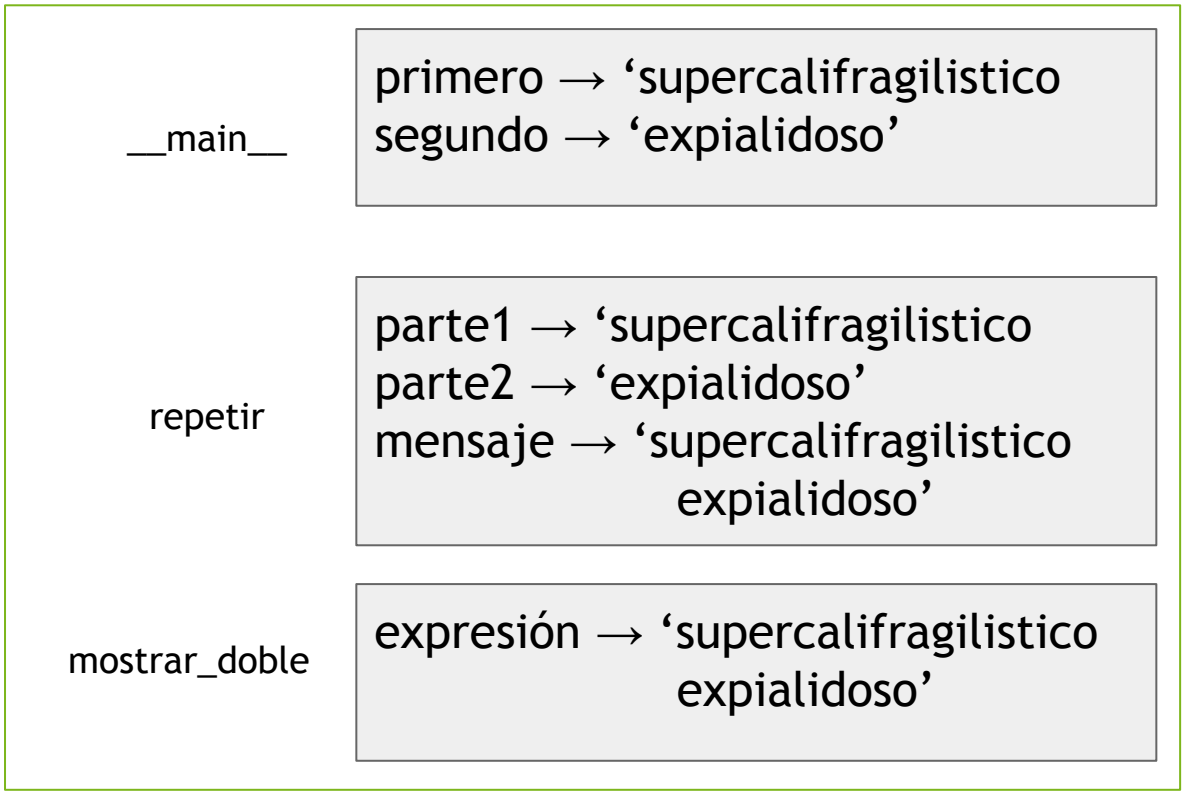
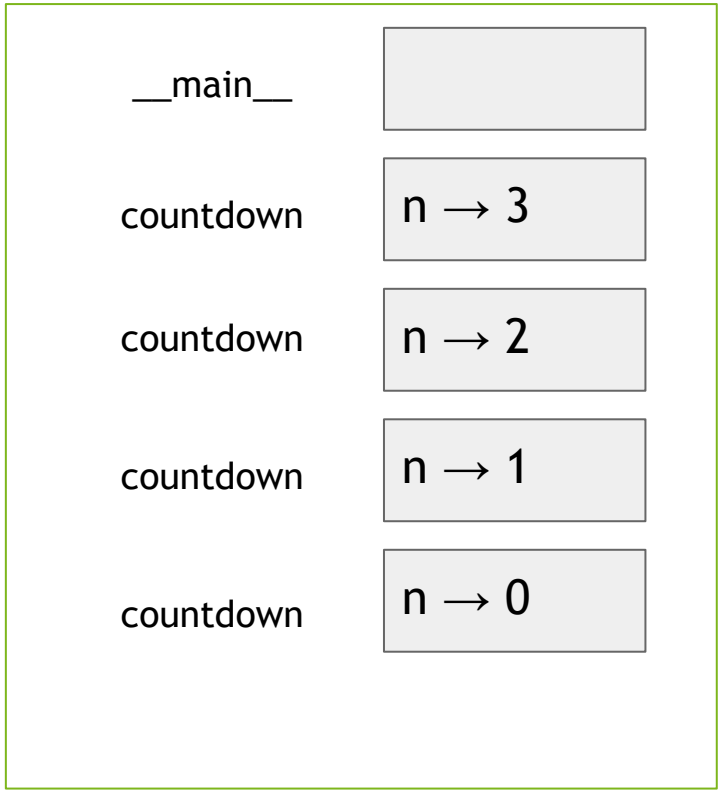


Diagrama de pilas en recursión

Cada vea que llamamos a una función, Python crea un marco para contener las variables locales y parámetros de la función. En el caso de funciones recursivas, habrá más de un marco en la pila al mismo tiempo.

Al principio de la pila está el marco para __main__, vacío en este caso.

El marco para n = 0 se llama el 'caso base' porque ya no hace más llamadas recursivas y, por tanto, ya no hay más marcos



Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

Recursión infinita

- Cuando una recursión nunca alcanza el caso base y continua haciendo llamadas recursivas para siempre, se denomina recursión infinita y no suele ser buena idea…
- En la mayoría de entornos de programación, un programa con una recursión infinita realmente no se queda en una ejecución eterna, porque existe un error que salta cuando se alcanza la profundidad de recursión máxima.
- La clave es tener localizado el caso base y cuándo se alcanza

Funciones “fruitful” y retorno de valores

- En la lección anterior ya comentamos las funciones “fructíferas” como lo contrario a las funciones “void”
- Las primeras, conocidas como funciones fructíferas (fruitful), devuelven valores que, normalmente, asignaremos a variables
- Se caracterizan por llevar “return” seguido de una expresión o variable.
- Las funciones matemáticas que se utilizan para crear modelos de IA son de este tipo.

Respecto al retorno de valores

- Aunque podemos tener una expresión directamente asignada a return, el contar con una variable temporal ayuda en el proceso de depuración del código.
- Es conveniente asegurarse de tener instrucciones return en las distintas ramas de un condicional
- Tras la instrucción return, no se ejecuta nada más. El código que haya detrás de la línea de return se llama “dead code”.
- Importante dentro de estrategias de Desarrollo Incremental (técnica de escribir código muy desarrollado para ir sintetizando tras sucesivas depuraciones)

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

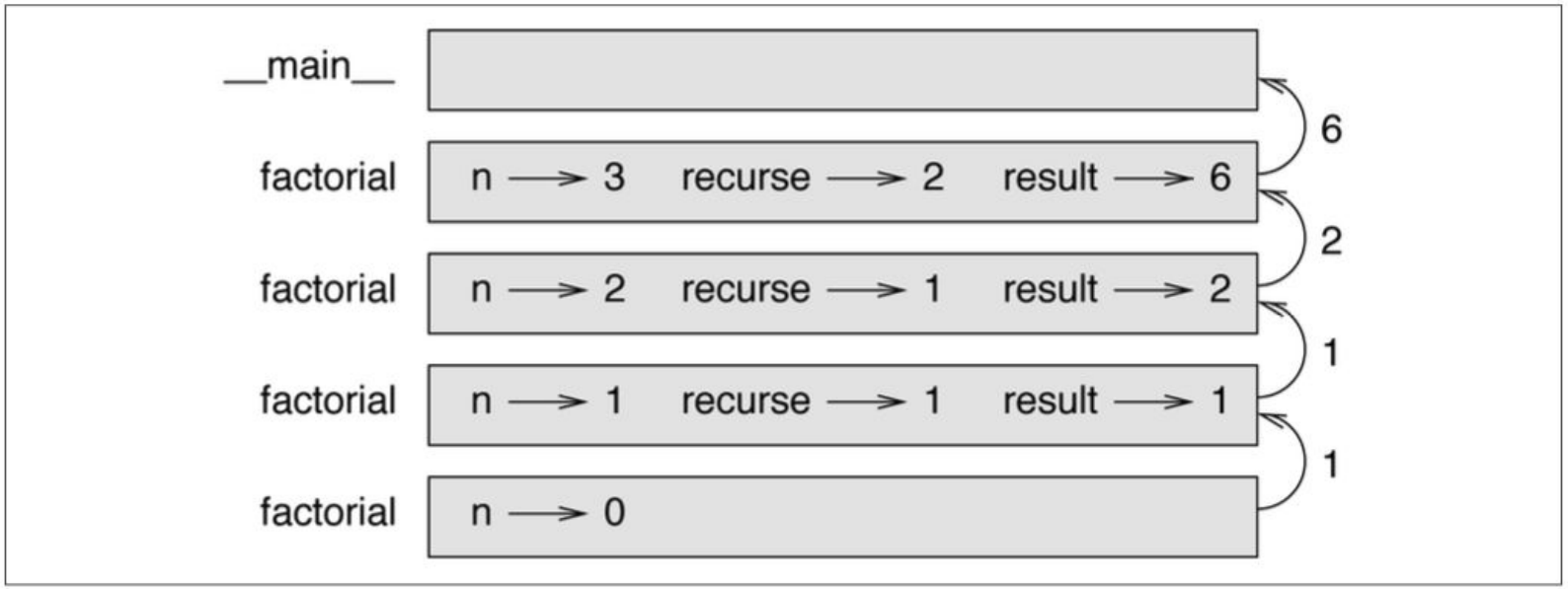
Escuela de organización industrial

Utilizando la recursión

Cualquier proceso real que pueda ser definido con recursión, admite un programa en Python para evaluarlo y hacer cálculos sobre él.

1. Elegir los parámetros necesarios
2. Analizar si hay más de una rama de proceso y definir la o las condiciones para la bifurcación
3. Localizar bien dónde debe ir la llamada recursiva a la función
4. Hacer un test muy sencillo, para un caso del que conozcamos el resultado.

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



Tail Call Optimization

- Las funciones recursivas tienen un coste en memoria que se puede optimizar con una técnica propia de la programación funcional.
- Básicamente, consiste en evitar crear nuevos stacks, reutilizando los que ya han sido creados, haciendo una llamada recursiva de la propia función en sí misma
- Ejemplo Fibonacci

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

Funciones lambda

- Una función lambda es una función anónima definida como una instrucción simple
- Puede sustituir a funciones pequeñas o de poca importancia.
- En general, se va a usar para evitar definir varias pequeñas funciones de poca relevancia
- Cualquier función que pueda ser definida en una línea, es una buena candidata para ser pasada como expresión o función lambda.

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

Funciones lambda

lambda *arg*: expresión

es equivalente a:

```
def nombre_funcion (argumentos):
    return expresión
```

Función lambda como argumento

Funcion(argumentos, lambda *arg*: expresión con *arg*)

"Funcion" debe haber sido definida para admitir una función como argumento y en esa definición debe quedar claro el argumento o argumentos que se le pasan a esa función argumento.

Funciones decoradoras

- Se utilizan cuando se quiere modificar una función sin tener que tocar el código original de su definición.
- Una función decoradora recibe una función como argumento y devuelve otra función.
- Va a ser útil recurrir a los conceptos de:
  - Argumentos indeterminados (*args, **kwargs)
  - Funciones internas
  - Funciones como argumentos

Funciones decoradoras

1. Se define la función decoradora, que debe admitir otra función como argumento:

```
def decoradora(func):
    ....
```

2. Se llama a la función decoradora justo antes de la definición de la función que queremos modificar, con el símbolo @:

```
@decoradora
def función(argumentos):
    ....
```

Plantilla de una decoradora

```
def decoradora(func_a_decorar):
    def envoltorio(*args,**kwargs):
        … # hace algo antes de llamar a la función
        result = func_a_decorar(*args,**kwargs)
        … #hace algo después de llamar a la función
        return result

    envoltorio.__doc__ = func_a_decorar.__doc__
    envoltorio.__name__ = func_a_decorar.__name__
    return envoltorio
```

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Escuela de organización industrial

Plantilla de una decoradora

```
def decoradora(func_a_decorar):

    @functools.wraps(func_a_decorar)

    def envoltorio(*args,**kwargs):
        … # hace algo antes de llamar a la función
        result = func_a_decorar(*args,**kwargs)
        … #hace algo después de llamar a la función
        return result

    return envoltorio
```

Modifica los atributos __doc__, __name__ y __module__

```
>>> def mydecorator(function):
...     def wrapped(*args, **kwargs):
...         # do some stuff before the original
...         # function gets called
...         result = function(*args, **kwargs)
...         # do some stuff after function call and
...         # return the result
...         return result
...     # return wrapper as a decorated function
...     return wrapped
...
>>> def func():
...     """ajksnd"""
...
>>> @mydecorator
... def func():
...     """my docstring"""
...     print("hello")
...
>>> func.__doc__
>>>
>>> from functools import wraps
>>> def mydecorator(function):
...     @wraps(function)  # adding this line
...     # ... rest the same ...
...
>>> @mydecorator
... def func():
...     """should be preserved now"""
...     print("hello")
...
>>> func.__doc__
'should be preserved now'
```

Fuente

Función decoradora parametrizada

A veces necesitamos utilizar parámetros en la “decoración” de una función. Pero las funciones decoradoras no admiten parámetros, ni argumentos al ir decorando una función.

Es necesario crear otra función, que admite parámetros y que dará como resultado una función generadora (fábrica de funciones generadoras)

```
def funcion_parametrizable(parametros):
    def funcion_decoradora(funcion_a_decorar):
        def envoltorio(*args,**kwargs):
            ...
```

carmenbvg@gmail.com

```
1  def gratitude():
2      print("Thank you.")
3
```