Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 1 de febrero de 2014

Funciones

Una **función** es como una caja negra: una vez creada no debemos preocuparnos por lo que tiene en su interior, simplemente, tenemos que recordar su nombre y los datos que necesita para resolver un proceso. Generalmente, devuelven un resultado.

La principal virtud de una función está en la reutilización del código, es decir, una vez creada puede ser llamada cada vez que se necesite. Para mejor aprovechamiento debemos procurar que las funciones ofrezcan soluciones a necesidades muy concretas.

Funciones con un número fijo de parámetros

La siguiente función calcula el área de un triángulo. Una vez definida se utiliza para calcular el área de dos triángulos de distintas dimensiones.

Para definir la función escribiremos **def** seguido del nombre de la función y entre paréntesis los dos parámetros que son necesarios para calcular el área del triángulo: base y altura. Con **return** la función devolverá el resultado de la fórmula matemática expresada. Los dos parámetros son obligatorios. Si alguno falta habrá una excepción.

```
def area_triangulo(base, altura):  # define función con dos parámetros
    ''' Calcular el área de un triangulo'''  # cadena de documentación
    return base * altura / 2  # devuelve el resultado de la expresión

print(area_triangulo(6, 4))  # la función retornará el valor 12
print(area_triangulo(3.5, 2.4))  # la función retornará el valor 4.2
```

Funciones con un número variable de parámetros

La siguiente función suma la distancia de un número variable de tramos. Si se utiliza sin aportar ningún valor devolverá 0. También, como cabría pensar es posible pasar variables.

```
def distancia(*tramos): # define función con nº variable de parámetros
    ''' Suma distancia de tramos '''  # cadena de documentación
    total = 0  # inicializa variable numérica
    for distancia in tramos:  # recorre, uno a uno, los tramos...
        total = total + distancia  # … y acumula la distancia
    return total  # devuelve la suma de todos los parámetros

tramo1 = 10
print(distancia(tramo1, 20, 30, 40))  # la función devuelve 100
print(distancia())  # la función retornará el valor 0
```

Funciones con parámetros con valores por defecto

La función **pagar** tiene el parámetro **dto_aplicado** con el valor 5 asignado por omisión. Dicho valor se utilizará en la solución en el caso de omitirse este dato cuando sea llamada la función.

```
def pagar(importe, dto_aplicado = 5):
    ''' La función aplica descuentos '''
    return importe - (importe * dto_aplicado / 100)

print(pagar(1000))  # 950
print(pagar(1000, 10))  # 900
```

Función con todos los parámetros con valores por defecto

Todos los parámetros tienen un valor por defecto. Cuando se utiliza la función si se especifican los nombres de los parámetros éstos pueden estar en distinto orden.

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

```
def repite_caracter(caracter="-", repite=3):
    return caracter * repite

print(repite_caracter())  # Se utilizan valores por omisión
print(repite_caracter('.',30))  # Muestra línea con 30 puntos
print(repite_caracter(repite=10, caracter='*'))  # Muestra: **********
```

Funciones con parámetros que contienen diccionarios

La función **porc_aprobados** tiene el parámetro ****aulas** que es un diccionario que contendrá las aulas de una escuela con el número alumnos de cada una. Cuando es llamada la función se pasa también el número de alumnos que aprobaron el curso. La función suma los alumnos de todas las aulas y calcula el porcentaje de aprobados.

```
def porc_aprobados(aprobados, **aulas):
    ''' Calcula el % de aprobados '''

    total=0
    for alumnos in aulas.values():
        total += alumnos

    return aprobados * 100 / total

porcentaje_aprobados = porc_aprobados(48, A = 22, B = 25, C = 21)
print(porcentaje_aprobados)
```

Funciones que devuelven más de un valor

La función **elemento_quimico** recibe un símbolo químico y devuelve el número atómico del elemento correspondiente y su denominación. Para ello, utiliza un diccionario en el que las claves son los símbolos químicos y los valores son cadenas que contienen para cada elemento su número atómico y denominación, unidos por un guión. Mediante el símbolo se accede a la cadena que luego es dividida con **split** en dos partes (utilizando como separador el propio guión '-'). **split** devuelve una lista con las dos partes. En lista[0] queda el número atómico y en lista[1] la denominación, los dos valores que devuelve esta función.

```
def elemento_quimico(simbolo):
    ''' Devuelve número atómico y denominación del elemento '''

    elementos = {'H':'1-Hidrógeno', 'He':'2-Helio', 'Li':'3-Litio'}
    elemento = elementos[simbolo]
    lista = elemento.split('-')
    return (lista[0], lista[1])

num_atomico, denomina = elemento_quimico('He')
print('Núm. Atómico:', num_atomico)
print('Denominación:', denomina)
```

Funciones sin return

Una función sin **return** devuelve **None** si es asignada a una variable o llamada desde un **print()**. Por lo demás, funcionan igual que cualquier otra función.

```
def repite(caracter='-', repite=3):
    print(caracter * repite)

repite('=', 20)
```

Funciones generadas a partir de otras

A partir de un función existente es posible generar una nueva. Después, ambas podrán usarse de igual forma.

```
at = area_triangulo  # la función calcula área de un triángulo
print at(10,4)  # la nueva función usa los argumentos base y altura
```

Relacionado:

- Funciones que imponen nombrar o no parámetros
- Programación Orientada a Objetos

que permiten obtener de distintos modos números a...

Archivo

febrero 2014 (17)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

- Programación funcional: funciones de orden superior
- Diccionario de variables locales y globales

Publicado por Pherkad en 4:10

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua