Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

viernes, 31 de enero de 2014

Cadenas, listas, tuplas, diccionarios y conjuntos (set)

Las **cadenas**, **listas** y **tuplas** son distintos tipos de **secuencias**. Una secuencia es un tipo de objeto que almacena datos y que permite el acceso a una parte determinada de su información utilizando índices.

Las **listas**, **tuplas**, **diccionarios** y **conjuntos** (**set**) son estructuras que permiten trabajar con colecciones de datos. El primer elemento de una lista o de una tupla ocupa la posición 0.

Listas

Una lista es una estructura de datos que contiene una colección o secuencia de datos. Los datos o elementos de una lista deben ir separados con una coma y todo el conjunto entre corchetes. Se dice que una lista es una estructura mutable porque además de permitir el acceso a los elementos, pueden suprimirse o agregarse nuevos.

ListaEstaciones = [“Invierno”, “Primavera”, “Verano”, “Otoño”] # Declara lista

Para definir una lista vacía, a la que con posterioridad se podría agregar elementos, existen dos posibilidades:

lista = []
lista = **list()**

Tuplas

Una **tupla** permite tener agrupados un conjunto inmutable de elementos, es decir, en una tupla no es posible agregar ni eliminar elementos. Las tuplas se declaran separando los elementos por comas y éstos, opcionalmente, pueden ir entre paréntesis. Se recomienda el uso de paréntesis para evitar ambigüedades del tipo: **print**(9, 8, 7) y **print**((9, 8, 7)).

TuplaDiasSemana = (“LU”, “MA”, “MI”, “JU”, “VI”, “SA”, “DO”) # Declara tupla

Operaciones con cadenas y listas

```
cadena1 = 'tengo una yama que Yama se llama'  # declara variable
lista1 = ['pera', 'manzana', 'naranja', 'uva']  # declara lista
longitud = len(cadena1)  # 32, devuelve longitud de la cadena
elem = len(lista1)  # 4, devuelve nº elementos de la lista
cuenta = cadena1.count('yama')  # 1, cuenta apariciones de 'yama'
print(cadena1.find('yama'))  # 10, devuelve posición de búsqueda
cadena2 = cadena1.join('***')  # inserta cadena1 entre caracteres
lista1 = cadena1.split(' ')  # divide cadena por separador → lista
tupla1 = cadena1.partition(' ')  # divide cadena por separador → tupla
cadena2 = cadena1.replace('yama','cabra',1) # busca/sustituye términos
numero = 3.14  # asigna número con decimales
cadena3 = str(numero)  # convierte número a cadena
if cadena1.startswith("tengo"):  # evalúa si comienza por “tengo”
if cadena1.endswith("llama"):  # evalúa si termina por “llama”
if cadena1.find("llama") != -1:  # evalúa si contiene “llama”
cadena4 = 'Python'  # asigna una cadena a una variable
print(cadena4[0:4])  # muestra desde la posición 0 a 4: "Pyth"
print(cadena4[1])  # muestra la posición 1: "y"
print(cadena4[:3] + '-' + cadena4[3:])  # muestra "Pyt-hon"
print(cadena4[:-3])  # muestra todo menos las tres últimas: "Pyt"

# declara variable con cadena alfanumérica
cadena5 = "  abc;123  "

# suprime caracteres en blanco (y \t\n\r) por la derecha
print(cadena5.rstrip())  # "  abc;123"

# suprime caracteres en blanco (y \t\n\r) por la izquierda
```

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

```
print(cadena5.lstrip())  # "abc;123  "

# suprime caracteres en blanco (y \t\n\r) por derecha e izquierda
print(cadena5.strip())  # "abc;123"

# suprime caracteres del argumento por la derecha e izquierda
print(cadena5.strip("123456790; "))  # "abc"

cadena6 = "Mar"   # declara una variable
print(cadena5.upper())   # convierte a mayúsculas: "MAR"
print(cadena5.lower())   # convierte a minúsculas: "mar"
```

Operaciones con listas y tuplas

```
lista1 = ['uno', 2, True]  # declara una lista heterogénea
lista2 = [1, 2, 3, 4, 5]  # declara lista numérica (homogénea)
lista3 = ['nombre', ['ap1', 'ap2']]  # declara lista dentro de otra
lista4 = [54,45,44,22,0,2,99]  # declara una lista numérica
print(lista1)  # ['uno', 2, True], muestra toda la lista
print(lista1[0])  # uno, muestra el primer elemento de la lista
print(lista2[-1])  # 5, muestra el último elemento de la lista
print(lista3[1][0])  # calle, primer elemento de la lista anidada
print(lista2[0:3:1])  # [1,2,3], responde al patrón inicio:fin:paso
print(lista2[::-1])  # devuelve la lista ordenada al revés
lista1[2] = False  # cambia el valor de un elemento de la lista
lista2[-2] = lista2[-2] + 1  # 4+1 → 5 - cambia valor de elemento
lista2.pop(0)  # borra elemento indicado o último si no indica
lista1.remove('uno')  # borra el primer elemento que coincida
del lista2[1]  # borra el segundo elemento (por índice)
lista2 = lista2 + [6]  # añade elemento al final de la lista
lista2.append(7)  # añade un elemento al final con append()
lista2.extend([8, 9])  # extiende lista con otra por el final
lista1.insert(1, 'dos')  # inserta nuevo elemento en posición
del lista2[0:3]  # borra los elementos desde:hasta
lista2[:] = []  # borra todos los elementos de la lista
print(lista1.count(2))  # cuenta el nº de veces que aparece 2
print(lista1.index("dos"))  # busca posición que ocupa elemento
lista3.sort()  # ordena la lista
lista3.sort(reverse=True)  # ordena la lista en orden inverso
lista5 = sorted(lista4)  # ordena lista destino
tupla1 = (1, 2, 3)  # declara tupla (se usan paréntesis)...
tupla2 = 1, 2, 3  # ...aunque pueden declararse sin paréntesis
tupla3 = (100,)  # con un elemento hay terminar con ",”
tupla4 = tupla1, 4, 5, 6  # anida tuplas
tupla5 = ()  # declara una tupla vacía
tupla6 = tuple([1, 2, 3, 4, 5])  # Convierte una lista en una tupla
tupla2[0:3]  # (1, 2, 3), accede a los valores desde:hasta
```

Diccionarios o matrices asociativas

Los **diccionarios** son objetos que contienen una lista de parejas de elementos. De cada pareja un elemento es la **clave**, que no puede repetirse, y, el otro, un **valor** asociado. La **clave** que se utiliza para acceder al valor tiene que ser un dato inmutable como una cadena, mientras que el valor puede ser un número, una cadena, un valor lógico (**True**/**False**), una lista o una tupla.

Los pares **clave-valor** están separados por dos puntos, las parejas por comas y todo el conjunto se encierra entre llaves.

Ejemplos:

capitales = {'Chile':'Santiago', 'España':'Madrid', 'Francia':'París'}

Para definir un diccionario vacío hay dos opciones:

capitales = {}
capitales = **dict()**

En el siguiente ejemplo se realizan algunas operaciones típicas con un diccionario:

```
# declara diccionario
capitales = {'Chile':'Santiago',
             'España':'Madrid',
             'Francia':'París'}

print('La capital de Chile es', capitales['Chile'])  # 'Santiago'
del capitales['Francia']  # borra el par Francia:París
print('\nHay {0} países\n'.format(len(capitales)))  # 'Hay 2 países'
for pais, capital in capitales.items():  # recorre diccionario
```

que permiten obtener de distintos modos números a...

Archivo

enero 2014 (10)

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

```
    print('Capital de {0}: {1}'.format(pais, capital))  # muestra par

capitales['Portugal'] = 'Lisboa'  # agrega par Portugal:Lisboa
if 'Portugal' in capitales:  # comprueba si existe clave
    print('\nCapital Portugal:', capitales['Portugal']) # 'Lisboa'
```

Operaciones con diccionarios

```
dic1 = {'Lorca':'Escritor', 'Goya':'Pintor'} # declara diccionario
print(dic1) # {'Goya': 'Pintor', 'Lorca': 'Escritor'}
dic2 = dict((('mesa',5), ('silla',10))) # declara a partir de tupla
dic3 = dict(ALM=5, CAD=10) # declara a partir de cadenas simples
dic4 = dict([(z, z**2) for z in (1, 2, 3)]) # declara a partir patrón
print(dic4)  # muestra {1: 1, 2: 4, 3: 9}
print(dic1['Lorca'])  # escritor, acceso a un valor por clave
print(dic1.get('Gala', 'no existe'))  # acceso a un valor por clave
if 'Lorca' in dic1: print('está')  # comprueba si existe una clave
print(dic1.items())  # obtiene una lista de tuplas clave:valor
print(dic1.keys())  # obtiene una lista de las claves
print(dic1.values())  # obtiene una lista de los valores
dic1['Lorca'] = 'Poeta'  # añade un nuevo par clave:valor
dic1['Amenabar'] = 'Cineasta'  # añade un nuevo par clave:valor
dic1.update({'Carreras' : 'Tenor'})  # añadir con update()
del dic1['Amenábar']  # borra un par clave:valor
print(dic1.pop('Amenabar', 'no está'))  # borra par clave:valor
```

Recorrer secuencias y diccionarios con for...in

```
artistas = {'Lorca':'Escritor', 'Goya':'Pintor'} # diccionario
paises = ['Chile','España','Francia','Portugal'] # declara lista
capitales = ['Santiago','Madrid','París','Lisboa']  # declara lista
for c, v in artistas.items(): print(c,':',v)  # recorre diccionario
for i, c in enumerate(paises): print(i,':',c)  # recorre lista
for p, c in zip(paises, capitales): print(c,' ',p) # recorre listas
for p in reversed(paises): print(p,)  # recorre en orden inverso
for c in sorted(paises): print(c,)  # recorre secuencia ordenada
```

Importante: A partir de Python 3.6 la implementación de los diccionarios ha cambiado, manteniéndose en todo momento el orden en que fueron agregados los elementos a un diccionario cuando son recorridos o consultados.

Relacionado:

- Con OrderedDict el orden ha llegado
- Bucles eficaces con Itertools

Recorrer rangos con for... in range()

```
for num in range(7): print(num)  # recorre de 0 a 6
for num in range(1,8): print(num)  # recorre de 1 a 7
for num in range(10,50,5): print(num) # de 10 a 45 de 5 en 5
for num in range(0,-10,-1): print(num)  # de 0 a -9 de -1 en -1

lista = ["Chorizo","Jamón","Morcilla","Salchichón"]  # lista
for elemento in range(len(lista)):  # recorre elementos de lista
    print (elemento, lista[elemento])  # muestra posición y elemento
```

Operadores para secuencias: in, not in, is, is not

```
cadena = 'Python'  # asigna cadena a variable
lista = [1, 2, 3, 4, 5]  # declara lista
if 'y' in cadena: print('"y" está en "Python"')  # contiene
if 6 not in lista: print('6 no está en la lista') # no contiene
if 'abcabc' is 'abc' * 2: print('Son iguales')  # son iguales
```

Conjuntos: set

Un **conjunto** es una lista de elementos donde ninguno de ellos está repetido. A partir de una lista, en la que pueden haber elementos repetidos, con **set** es posible crear otra lista con todos los elementos pero sin repetir ninguno. Además, si tenemos varias listas podemos realizar operaciones de conjuntos de unión, diferencia, intersección y diferencia simétrica:

- diferencia, | unión, & intersección y ^ diferencia simétrica

Operaciones con conjuntos

```
conjunto = set()  # Define un conjunto vacío
lista = ['vino', 'cerveza', 'agua', 'vino']  # define lista
bebidas = set(lista)  # define conjunto a partir de una lista
print('vino' in bebidas)  # True, 'vino' está en el conjunto
print('anis' in bebidas)  # False, 'anis' no está en el conjunto
print(bebidas)  # imprime {'agua', 'cerveza', 'vino'}
bebidas2 = bebidas.copy()  # crea nuevo conjunto a partir de copia
print(bebidas2)  # imprime {'agua', 'cerveza', 'vino'}
bebidas2.add('anis')  # añade un nuevo elemento
print(bebidas2.issuperset(bebidas)) # True, bebidas es subconjunto
bebidas.remove('agua')  # borra elemento
print(bebidas & bebidas2)  # imprime elementos comunes
tapas = ['croquetas', 'solomillo', 'croquetas']  # define lista
conjunto = set(tapas)  # crea conjunto (sólo una de croquetas)
if 'croquetas' in conjunto:  # evalúa si croquetas está
conjunto1 = set('Python')  # define conjunto: P,y,t,h,o,n
conjunto2 = set('Pitonisa')  # define conjunto: P,i,t,o,n,s,a
print(conjunto2 - conjunto1)  # aplica diferencia: s, i, a
print(conjunto1 | conjunto2)  # aplica unión: P,y,t,h,o,n,i,s,a
print(conjunto1 & conjunto2)  # intersección: P,t,o,n
print(conjunto1 ^ conjunto2)  # diferencia simétrica: y,h,i,s,a
```

Relacionado:

- Tipos de cadenas: Unicode, Byte y Bytearray
- Iteradores y generadores
- Evaluar, ejecutar y compilar cadenas

Ir al índice del tutorial de Python

Publicado por Pherkad en 10:07

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua