

Tipos de datos estructurados

Fundamentos de la Programación

Salvador Sánchez

Universidad de Alcalá

Octubre de 2018

Los contenidos de esta presentación pueden ser copiados y redistribuidos en cualquier medio o formato, así como adaptados, remezclados, transformados y servir de base para la creación de nuevos materiales a partir de ellos, según la licencia Atribución 4.0 Unported (CC BY 4.0)



Definición

Tipos de datos que permiten agrupar elementos.

- Secuencias en Python: listas, tuplas, diccionarios, conjuntos...
- Las secuencias son por naturaleza heterogéneas: pueden contener elementos de tipos distintos.
- Cada posición puede referenciar un elemento de tipo simple (entero, real, booleano, string) o compuesto (otras secuencias, instancias de una clase, archivos, etc.).

Definición

Aquellos objetos cuyo valor puede cambiar se dice que son **mutables**, a diferencia de aquellos cuyo valor no puede cambiar una vez creados, los cuales se denominan **inmutables**.

- Mutables en Python: listas, diccionarios.
- Inmutables: números, cadenas y tuplas.

Definición

Conjunto mutable y organizado de elementos.

- Se declara y representa encerrando los miembros entre corchetes.
- Los elementos de una lista son variables, pudiendo añadir y eliminar elementos en cualquier momento.
- Se pueden modificar sus elementos.
- Rango de los elementos: 0 a longitud-1
- Puede tener elementos repetidos

$a = [0.056, 38.65, -6.09, 1.267, -51.2]$

- Representación interna:

a	0.056	38.65	-6.09	1.267	-51.2
	0	1	2	3	4

Operaciones enfocadas a elementos

- `lista.index(elem)` - Retorna la posición del elemento en la lista. Si no está se produce una excepción.
- `lista[i]` - Accede al elemento en la posición `i`.
- `elem in lista` - Determina si un elemento está o no en la lista.
- `lista.count(elem)` - Número de veces que aparece un elemento en la lista.

Operaciones sobre listas

- `len(lista)` - retorna el tamaño de la lista (num. elementos).
- `lista.append(elemento)` - añade un elemento al final de la lista.
- `L1.extend(L2)` - añade a L1 todos los elementos de L2.
- `lista1 + lista2` - produce una lista con los elementos de ambas.
- `lista.insert(pos,elem)` - inserta un elemento en una posición.
- `lista.remove(elem)` - Elimina la primera aparición del elemento.
- `lista.sort()` - Ordena la lista
- `lista.reverse()` - Invierte el orden de los elementos de la lista
- `lista.pop()` - Elimina y retorna el último elemento
- `+` (concatenar) y `*` (repetir): Ej. `lista1 = lista2 * 3`

- Las listas/tuplas son el elemento base de la iteración con for.

```
frutero = [ 'pera', 'higo', 'fresa', 'caqui' ]  
for fruta in frutero:  
    print ( "Una pieza de fruta: ", fruta )
```

- Forma abreviada de obtener subsecuencias a partir de secuencias existentes
- Sintaxis: `secuencia [inicio:fin:salto]`
- Si no se especifica inicio, se entiende desde 0, si no se especifica fin, se entiende hasta el final de la secuencia
- Ejemplos:

```
preposiciones = ['a', 'ante', 'bajo', 'cabe', 'con', 'contra', 'de', 'desde', 'en']  
a = preposiciones [1:5] # copia de la posicion 1 a la 4  
a=['ante', 'bajo', 'cabe', 'con']  
b = preposiciones [4:] # copia de la posicion 4 al final  
b=['con', 'contra', 'de', 'desde', 'en']  
c = preposiciones [:3] # copia de la posicion 0 a la 2  
c=['a', 'ante', 'bajo']  
d = preposiciones [:] # hace un duplicado
```

Definición

Conjunto inmutable y ordenado de elementos.

- Se declara y representa encerrando los miembros entre paréntesis.
- Número fijo de elementos: sólo se pueden añadir y eliminar reasignando la tupla completa.
- Ejemplo:

$a = (0.056, 38.65, -6.09, 1.267, -51.2)$

a	0.056	38.65	-6.09	1.267	-51.2
	0	1	2	3	4

Operaciones con tuplas

- Sus métodos son todos de *solo-lectura*.
 - No pueden añadirse elementos (no tienen métodos `append` ni `extend`).
 - No pueden eliminarse elementos (no tienen `remove` ni `pop`).
- Pueden utilizarse como claves en un diccionario (las listas no).
- Pueden ‘convertirse’ en listas: `list()` toma una tupla y devuelve una lista con los mismos elementos
 - Es posible también ‘convertir’ una lista a tupla: `tuple()`
 - Se dice que `tuple` “congela” una lista, y que a su vez `list` “descongela” una tupla.

Definición

Colección homogénea cuyos elementos son caracteres de texto.

- Número fijo de elementos: no se pueden añadir ni eliminar.
- Sus elementos no se pueden modificar.
- Creación: texto entre comillas simples o dobles.
- Ejemplo:

```
cadena1 = "Universidad de Alcala"  
cadena2 = 'Alcala de Henares'
```

Operaciones con cadenas

- `lista.upper()` - Pone en mayúsculas
- `lista.lower()` - Pone en minúsculas
- `lista.find(car)` - Posición del carácter `car` en la cadena
- `lista.replace(car,car2)` - Sustituye un carácter por otro
- `lista.split()` - Retorna una lista con las partes de la cadena separadas (usando espacios)
- Se puede usar `split()` con cualquier separador si se indica: `split('.')`, `split(':')`, etc.
- Ya conocidas: `+` (concatenar cadenas), `*`*i* (repetir *i* veces la cadena)

Definición

Estructura común en los lenguajes de programación formada por un número **fijo** de elementos **homogéneos** almacenados conjuntamente en memoria y a los que es posible acceder mediante índices

- Según sus dimensiones da soporte a conceptos matemáticos distintos:
 - Array de dimensión 1: vector
 - Array de dimensión 2: matriz
- Python no proporciona un constructor nativo para arrays
 - Implementaciones posibles: listas, tuplas, utilización de bibliotecas específicas (numpy), etc.

Definición

Tipo de datos estructurado que permite agrupar un número finito de datos, denominados campos, bajo un mismo nombre.

- Cada campo tiene un tipo (que pueden ser distintos).
- Acceso a campos: notación punto (Ej. `alumno.nombre = 'Angela'`)
- Python no proporciona soporte nativo: implementación mediante clases sin método, tuplas con nombre (*namedtuples*), listas, etc.

Definición

Construcción de los lenguajes que modela no solo datos sino también comportamiento (las operaciones asociadas a esos datos).

- Da soporte al concepto de TAD (Tipo Abstracto de Datos)
- $\text{TAD} = \text{Datos} + \text{Operaciones}$
- Una clase que no implementa operaciones (métodos), es un registro
- Al menos una operación es obligatoria: la inicialización (constructor)

Registros como clases sin métodos

```
class Fecha:
    def __init__(self, dd, mm):
        self.dia = dd
        self.mes = mm
```

Probador

```
f = Fecha(8,12)
print(f.dia, '/', f.mes)
```

Registros como clases sin métodos

Los registros también pueden formar parte de secuencias.

```
class Fecha:
    def __init__(self, dd, mm, aa):
        self.dia = dd
        self.mes = mm
        self.anyo = aa

class Alumno:
    def __init__(self, nom, ap, fecha_nac, age):
        self.nombre = nom
        self.apellidos = ap
        self.fecha_nacimiento = fecha_nac
        self.edad = age

# Probador
f = Fecha(1,12,1997)
a = Alumno('Pedro', 'Martinez', f, 17)
lista_alumnos = []
lista_alumnos.append(a)
f = Fecha(3,4,1995)
lista_alumnos.append(Alumno('Maria', 'Perez', f, 20))
for alumno in lista_alumnos:
    print(alumno.nombre)
    print(alumno.fecha_nacimiento.dia, '/', alumno.fecha_nacimiento.mes)
```

Definición

Estructura de datos sin orden entre los elementos, y en la que el acceso viene determinado por una clave única que se asocia a cada valor.

- La clave es a menudo una cadena de texto, si bien puede ser cualquiera de los tipos inmutables de Python.
- Tipos de claves: : boolean, integer, float, tupla, string ...
- Los diccionarios son mutables: se pueden agregar, eliminar y cambiar sus elementos.

Crear diccionarios

- Para crear un diccionario se emplean corchetes (`{}`) alrededor de pares `clave:valor` separados por comas.
- El diccionario más simple es un diccionario vacío, el cual no contiene ninguna clave o valor en absoluto:

```
>>> empty_dict = {}  
>>> empty_dict  
{}
```

Ejemplo de diccionarios

- Ejemplo preliminar de diccionario con citas del Diccionario del Diablo de Ambrose Bierce:

```
>>> bierce = {  
    "dia": "Periodo de veinticuatro horas, casi todas desperdiciadas",  
    "paciencia": "forma menor de desesperanza, a menudo disfrazada de virtud",  
    "paz": "En asuntos internacionales, tiempo de mentiras entre dos periodos de  
        lucha",  
}  
>>>
```

- Al escribir el nombre del diccionario en el intérprete se imprimirán sus claves y valores:

```
>>> bierce  
{'dia': 'Periodo de veinticuatro horas, casi todas desperdiciadas', 'paciencia': '  
    forma menor de desesperanza, a menudo disfrazada de virtud', 'paz': 'En  
    asuntos internacionales, tiempo de mentiras entre dos periodos de lucha'}
```

Operaciones con diccionarios

- **Agregar** un elemento: basta con referirse al ítem por su clave y asignarle un valor.
 - Si la clave ya estaba en el diccionario, el valor existente se reemplaza por el nuevo, si la clave es nueva, se agrega al diccionario con su valor.
 - Imposible errar por especificar un índice fuera de rango.
- **Eliminar** un elemento: utilizar `del`. Ejemplo: `del bierce['dia']`
- **Reiniciar** eliminando todos los elementos: `clear`. Ejemplo: `bierce.clear()`
- **Pertenencia** de una clave: `in`. Ejemplo: `'dia' in bierce`
- **Número de elementos**: `len(diccionario)`
- **Réplica**: `nuevo_diccionario = diccionario.copy()`

- **Lista de claves:** `keys`. Ejemplo: `bierce.keys()`
 - Retorna un objeto de tipo `dict_keys`, que es una forma iterable de lista. Puede convertirse a lista con `list()`.
- **Lista de valores:** `values`. Ejemplo: `bierce.values()`
 - Retorna un objeto de tipo `dict_values`, una forma iterable de lista.
Ej: `for item in x.keys(): print item`
- **Lista de items:** al igual que `values` y `keys`, es iterable:
`diccionario.items()`

- Una lista es una colección de elementos mutable (número de elementos variable + elementos modificables)
- Una tupla es una colección de elementos inmutable (número de elementos fijo + elementos no modificables)
- El acceso a las posiciones y muchas otras operaciones sobre secuencias son comunes a todas las secuencias
- Las cadenas de texto son secuencias inmutables de elementos homogéneos
- Los registros son datos heterogéneos cuyos valores tienen nombre (campos).
- Python no proporciona soporte nativo para los conceptos de array ni de registro: diversos modos de implementación
- Los diccionarios contienen parejas clave-valor