



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Programación para aplicaciones geoespaciales

Introducción al Lenguaje Python

LAURA SEBASTIÁ

TEMA 3. ESTRUCTURAS DE DATOS

Contenidos



1. Cadenas

2. Listas

3. Diccionarios

Bibliografía:

Introducción a la programación con Python 3

Tema 5, apartados 5.1 y 5.2

Tema 7, apartado 7.5

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas

- Una cadena es una sucesión de caracteres encerrada entre comillas simples o dobles.
- Ya hemos estudiado algunas funciones (float, int, str) y métodos predefinidos (lower, upper, replace).
- Función len: longitud de una cadena

<pre>print(len('abc'))</pre>	3
<pre>print(len(''))</pre>	0
<pre>print(len(' '))</pre>	1
<pre>print(len('abc'*4))</pre>	12

- Indexación: acceso a los caracteres de una cadena

0	1	2	3	4	5	6	7	8	9	10
H	o	l	a	,		m	u	n	d	o

```
a='Hola, mundo'
print (a[2])
print (a[len(a)-1])
print (a[-1])
print (a[len(a)])
```

```
l
o
o
IndexError: string index out of range
```

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas: Slicing (operador de corte)

- Se denota con dos puntos (:) que separan dos índices dentro de los corchetes del operador de indexación.
- La expresión `[i:j]` significa que se desea obtener la subcadena formada por los caracteres `a[i]`, `a[i+1]`, ..., `a[j-1]`

```
a='ejemplo'  
print (a[2:5])  
print (a[-5:5])  
print (a[2:-2])  
print (a[-5:-2])
```

La salida en todos los casos es:

emp

- Cada índice tiene un valor por defecto: `a[:j]` equivale a `a[0:j]` y `a[i:]` equivale a `a[i:len(a)]`
- Se puede utilizar un tercer valor que indica el incremento del índice en cada iteración

```
a='ejemplo'  
print (a[0:len(a):2])
```

eepp

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas: Modificación

- Las cadenas son inmutables. No se pueden modificar sus elementos. Para ello, es necesario crear una nueva cadena.

```
a='ejemplo'  
a[0]='E'
```

TypeError: 'str' object does not support item assignment

```
a='ejemplo'  
b='E'+a[1:]  
print (b)
```

Ejemplo

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas: Otros métodos

- **Búsqueda**
 - **Operador in:** `a in b` devuelve `True` si la subcadena `a` se encuentra en la cadena `b`
 - **Métodos `index` y `find`:** devuelven la posición donde comienza la subcadena

```
a='ejemplo'
if 'je' in a:
    print (a+' contiene je')
print ('Indice de je:')
print (a.find('je'), a.index('je'))
print ('Indice de jeee: ')
print (a.find('jeee'))
print (a.index('jeee'))
```

```
ejemplo contiene je
Indice de je:
1 1
Indice de jeee:
-1
ValueError: substring not found
```

- **Contador: `count()`**

```
s = "Hola mundo"
s.count("Hola")
```

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas: Otros métodos

- Comprobaciones: `isdigit()`, `isalnum()`, `isalpha()`, `islower()`, `isupper()`, `isspace()`

```
# Determina si todos los caracteres son dígitos.  
print("1234".isdigit())  
# Determina si todos los caracteres son alfanuméricos.  
print("abc123".isalnum())  
# Determina si todos los caracteres son alfabéticos.  
print("abcdef".isalpha())  
print("abc123".isalpha())  
# Determina si todas las letras son minúsculas.  
print("abcdef".islower())  
# Mayúsculas.  
print("ABCDEF".isupper())  
# Determina si la cadena contiene solo espacios.  
print("Hola mundo".isspace())  
print("    ".isspace())
```

True

True

True

False

True

True

False

True

TEMA 3. ESTRUCTURAS DE DATOS

Listas

- Python nos permite definir secuencias de valores de cualquier tipo mediante el uso de listas.
- Los valores de una lista se representan encerrados entre corchetes y separados por comas.

```
a=[1,2,3.5]  
b=['Juan', 'Maria', 'Luis']  
c=[1, 1+1, 6/3]  
print (a)  
print (b)  
print (c)
```

```
[1, 2, 3.5]  
['Juan', 'Maria', 'Luis']  
[1, 2, 2]
```

- Las listas son mutables, es decir, se puede modificar su contenido (`a[1]=3`)
- Operadores y funciones que ya conocemos: `+`, `*`, `[]`, `[:]`, `len`, `in`, `index`

TEMA 3. ESTRUCTURAS DE DATOS

Listas: Ejemplos

```
a=[1,2,3]
print ('a:', a)
b=a
b[0]=7
print ('a:', a, 'b:', b)
c=a+[10,20]
print ('c:', c)
d=[0]*10
print ('d:', d)
e=c[1:3]
print ('e:', e)
f=[1,2,3,4,5,6,7,8]
print (f[::2])
print (len(f))
print (10 in c)
print (c.index(10))
print (c.index(25))
```

```
a: [1, 2, 3]

a: [7, 2, 3] b: [7, 2, 3]

c: [7, 2, 3, 10, 20]

d: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

e: [2, 3]

[1, 3, 5, 7]
8
True
3
ValueError: 25 is not in list
```

TEMA 3. ESTRUCTURAS DE DATOS

Listas: Adición de elementos

- **Operador de concatenación +** : Une las listas, creando una nueva lista
- **Método append(elemento)** : Añade “elemento” al final de la lista, modificando la lista original
- **Método insert(índice, elemento)**: Inserta “elemento” en la posición “índice”, modificando la lista original

```
a=[1,2,3]
print ('a:',a)
b=a+[4]
print ('b:',b)
c=b
c.append(5)
print ('b:',b,'c:',c)
c.insert(1, 25)
print ('b:',b,'c:',c)
```

a: [1, 2, 3]

b: [1, 2, 3, 4]

b: [1, 2, 3, 4, 5] c: [1, 2, 3, 4, 5]

b: [1, 25, 2, 3, 4, 5] c: [1, 25, 2, 3, 4, 5]

TEMA 3. ESTRUCTURAS DE DATOS

Listas: Eliminación de elementos

- **Método pop():** Elimina el último elemento de la lista
- **Método pop(índice):** Elimina el elemento en la posición “índice”
- **Método remove(elemento):** Elimina la primera ocurrencia de “elemento”
- **En todos los casos, se modifica la lista original**

```
print ('c:',c)
x=c.pop()
print ('x:',x,'c:',c)
x=c.pop(1)
print ('x:',x,'c:',c)
c.remove(3)
print ('c:',c)
```

```
c: [1, 2, 3, 4, 5]
x: 5 c: [1, 25, 2, 3, 4]
x: 25 c: [1, 2, 3, 4]
c: [1, 2, 4]
```

TEMA 3. ESTRUCTURAS DE DATOS

Listas: Ordenación de elementos

- **Método sort():** Ordena la lista de forma ascendente
- **Método sort(reverse=True):** Ordena la lista de forma descendente
- **En ambos casos, se modifica la lista original**

```
nombres_masculinos=['Jose','Jose','Ricky','Jacinto','David','Alvaro','Ricky']  
print (nombres_masculinos)  
nombres_masculinos.sort()  
print (nombres_masculinos)  
nombres_masculinos.sort(reverse=True)  
print (nombres_masculinos)
```

```
['Jose', 'Jose', 'Ricky', 'Jacinto', 'David', 'Alvaro', 'Ricky']  
['Alvaro', 'David', 'Jacinto', 'Jose', 'Jose', 'Ricky', 'Ricky']  
['Ricky', 'Ricky', 'Jose', 'Jose', 'Jacinto', 'David', 'Alvaro']
```

TEMA 3. ESTRUCTURAS DE DATOS

Listas: Recorrido de elementos

- **List comprehension:** mecanismo que se utiliza cuando se desea crear una nueva lista basada en los valores de una lista existente.

Sintaxis: corchetes que contienen una expresión que se ejecuta para cada elemento de la lista (iteración indicada por el bucle for)

```
newlist = [expression for item in iterable if condition == True]
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
newlist = [x.upper() for x in fruits]  
print(newlist)
```

```
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

Sin if

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
newlist = [x for x in fruits if "a" in x]  
print(newlist)
```

```
['apple', 'banana', 'mango']
```

Con if

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas y Listas

- Las cadenas también se pueden recorrer utilizando List Comprehension

Generar una lista con los caracteres de una cadena:

```
l = [c for c in 'cadena']  
l
```

```
['c', 'a', 'd', 'e', 'n', 'a']
```

Generar una lista con los caracteres de una cadena que no son vocales:

```
l = [c for c in 'cadena' if c not in ['a', 'e', 'i', 'o', 'u']]  
l
```

```
['c', 'd', 'n']
```

TEMA 3. ESTRUCTURAS DE DATOS

Cadenas y Listas: Otros métodos

- **Separación: split()**

```
"Hola mundo!\nHello world!".split()  
['Hola', 'mundo!', 'Hello', 'world!']
```

```
"Hola mundo!\nHello world!".split(' ')  
['Hola', 'mundo!\nHello', 'world!']
```

- **Unión: join()**

```
" ".join(["Hola", "mundo"])
```

```
'Hola mundo'
```

```
", ".join(["C", "C++", "Python", "Java"])
```

```
'C, C++, Python, Java'
```