2.7 Tipos estructurados: listas y cadenas

- 2.1 Abstracción de problemas para su programación. Conceptos fundamentales
- 2.2 Variables, expresiones, asignación
- 2.3 Uso de entrada/salida por consola
- 2.4 Manejo de estructuras básicas de control de flujo: secuencial, alternativa y repetitiva
- 2.5 Definición y uso de subprogramas y funciones. Ámbito de variables
- 2.6 Entrada/salida a ficheros
- 2.7 Tipos y estructuras de datos básicas: arrays

2.7 Tipos estructurados: listas y cadenas

En Python existen tipos estructurados de datos. Una variable de uno de esos tipos, con un solo nombre, representa varios valores de los tipos básicos.

En este curso vamos a ver:

- Listas -> secuencias de elementos de distintos tipos
- Cadenas de texto -> secuencias de caracteres

Listas

Una lista es una secuencia de datos de cualquier tipo.

El número de elementos puede cambiar durante la ejecución de un programa.

Una lista se representa mediante sus valores entre corchetes, separados por comas.

```
>>> lista = ["velocidad", 17, 3.1416]
>>> type(lista)
<type 'list'>
>>> len(lista)
3
```

Listas. Acceso a los elementos

Los elementos individuales de una lista pueden ser accedidos a través de un índice, que comienza en 0, el índice del primer elemento de la lista.

```
>>> lista = ["velocidad", 17, 3.1416]
>>> print(lista[0])
velocidad
>>> type(lista[2])
<type 'float'>
>>> i = 1
>>> print(lista[i])
17
```

2.7 Tipos estructurados: listas y cadenas

Operadores

- Operador de concatenación `+`
- Operador de repetición `*`
- Operador de corte (slicing) `[:]`
- Comparaciones, pertenencia, identidad `==` `!=` `in` `is`

El resultado de la aplicación de un operador o función a una lista se puede asignar a una nueva lista: se crea un nuevo objeto.

```
>>> p = [1, 2, 3, 4]
>>> print(p+p)
[1, 2, 3, 4, 1, 2, 3, 4]
>>> q=p*3
>>> print(q)
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>> print(q[3:7])
[4, 1, 2, 3]
>>> print(3 in p and p!=q)
True
```

Corte o "Slicing"

Selecciona los elementos de la lista desde inicio hasta fin-1 de paso en paso

```
lista[inicio:fin:paso]
```

Si se omite inicio o fin se toman el primer o último elemento.

Valores negativos significan posiciones a contar desde el final (-1) hacia atrás.

Si se omite el paso se supone igual a 1.

```
numeros = ["uno","dos","tres","cuatro","cinco"]

numeros[1:4]    ⇒    ['dos', 'tres', 'cuatro']
numeros[2:-1]   ⇒    ['tres', cuatro']
numeros[:]      ⇒    ['uno', 'dos', 'tres', 'cuatro', 'cinco']
numeros[::2]    ⇒    ['uno', 'tres', 'cinco']
numeros[::-1]   ⇒    ['cinco', 'cuatro', 'tres', 'dos', 'uno']
```

Funciones

- Longitud `len()`
- Creación de listas numéricas `list(range())`
- Funciones para listas numéricas `sum()` `max()` `min()`
- Ordenación de sus elementos `sorted()`

```
>>> p = list(range(1,5))
>>> print(p, len(p))
[1, 2, 3, 4] 4
>>> print(sum(p), max(p), min(p))
10 4 1
>>> print(sorted(p*3))
[1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]
>>> nombres = ["Paula", "Luis", "Juan", "Ines"]
>>> print(sorted(nombres))
['Ines', 'Juan', 'Luis', 'Paula']
```

Modificación de listas

Las listas son variables mutables. Esto quiere decir que los métodos pueden cambiar la información de una variable sin tener que crear una nueva modificada.

- Asignación directa a un elemento p[1] = 7
- Métodos de inserción .append() .extend() .insert()
- Métodos de borrado .pop() .remove()

```
>>> numeros = ["uno", "dos", "tres", "cuatro"]
>>> numeros[2] = "cinco"
>>> print(números)
['uno', 'dos', 'cinco', 'cuatro']
>>> numeros.append("seis")
>>> numeros.extend(["siete",  "ocho",  "nueve"])
>>> print(números)
['uno','dos','cinco','cuatro','seis','siete','ocho','nueve']
```

Modificación de listas

```
>>> numeros.pop()
'nueve'
>>> print(numeros)
['uno', 'dos', 'cinco', 'cuatro', 'seis', 'siete', 'ocho']
>>> numeros.pop(2)
'cinco'
>>> print(numeros)
['uno', 'dos', 'cuatro', 'seis', 'siete', 'ocho']
>>> numeros.remove("siete")
>>> print(numeros)
['uno', 'dos', 'cuatro', 'seis', 'ocho']
>>> numeros.insert(2, "tres")
>>> print(numeros)
['uno', 'dos', 'tres', 'cuatro', 'seis', 'siete', 'ocho']
```

Recorrido de listas

El bucle for nos permite acceder a los elementos de una lista de manera secuencial. Aunque también es posible acceder mediante los índices a los elementos específicos de una lista.

```
>>> p = [1, 2, 3, 5, 7, 11, 13, 17]
>>> for n in p:
...     print(n**2,end=‘ ‘)
...
1 4 9 25 49 121 169 289
>>>
>>> for i in range(len(p)):
...   print(p[i]**2,end=‘ ‘)
...
1 4 9 25 49 121 169 289
```

Cadenas

Una cadena es un tipo especial de lista, donde sus elementos son caracteres (dígitos, letras, signos y caracteres especiales de control como tabuladores o saltos de línea) y se representan encerrados entre comillas. Las comillas simples o dobles son equivalentes.

Son inmutables así que no se puede cambiar su contenido directamente.

```
>>> nombre = "Patricia"
>>> type(nombre)
<type 'str'>
>>> print (nombre[2], len(nombre))
't' 8
>>> nombre[2]="p"
TypeError: 'str' object does not support item assignment
```

Caracteres especiales

Los caracteres especiales se representan con un código después de \.

```
\n       ⇒    salto de línea
\t       ⇒    tabulador
\\       ⇒    barra invertida \
\' o \"  ⇒    Comillas simples o dobles
```

```
>>> frase = "primera linea \n segunda \t \" linea \" "
>>> print(frase)
primera linea
 segunda        " linea "
>>> frase
'primera linea \n segunda \t " linea " '
```

Operadores

- Los mismos que para las listas `+` `*` `[:]` `==` `!=`
- Pertenencia `in` (permite buscar caracteres o subcadenas)

```
>>> nombre = "Patricia"
>>> print("tri" in nombre)
True
>>> otro_nombre="Mort"+nombre[4:8]
>>> print(otro_nombre)
'Morticia'
```

Funciones y métodos

- Longitud `len(cadena)`
- Lectura `input()`
- Conversión números/cadenas `int()` `float()` `str()`
- Métodos `.join(secuencia)`, `.split(separador)`

```
>>> print (int("123") + float("12.3"), "123" + str(123))
135.3 123123
>>> dni="00000014-Z"
>>> num_y_letra=dni.split("-")
>>> print(num_y_letra)
['00000014', 'Z']
>>> dni_sin_guion="".join(num_y_letra)
>>> print(dni_sin_guion)
'00000014Z'
```

Listas de listas rectangulares, matrices

Una lista puede formar parte de otra lista como elemento.
Esto permite construir listas de listas.
En este curso las usaremos para representar matrices.

```
>>> x=[[1.2,3.4,2.6],[0.1,30,5.5]]
>>> print(x[0])
[1.2, 3.4, 2.6]
>>> print(x[0][0])
1.2
>>> x[1][1]
30
>>> x[1][2]
5.5
```

Cada elemento de x es una lista de 3 números.

x[0] es la primera fila.

x[0][0] es el primer elemento de esa fila, la esquina sup. izq. de la matriz.

x[1][2] es el tercer elemento de la segunda fila.

Listas de listas rectangulares, matrices

Para procesar cada elemento de una matriz por separado, se recorre usando dos bucles anidados

```
>>> for i in range(len(x)):
...     for j in range(len(x[i])):
...         print (x[i][j]+1,end=' ‘)
...     print()
...
2.2 4.4 3.6
1.1 31 6.5
>>>
```

len(x) es el número de filas, luego i tomará los valores 0 y después 1.

len(x[i]) es el número de columnas, luego j valdrá 0, 1, 2, en ese orden.

El bucle más interno se ejecuta *primero* para i con el valor 0, como si hubiésemos escrito `x[0][j]+1` y *después* con el valor 1, como si hubiésemos escrito `x[1][j]+1`. Cada vez, con j valiendo primero 0, después 1 y finalmente 2.

De este modo se obtienen las parejas de índices `[0][0]`, `[0][1]`, `[0][2]`, `[1][0]`, `[1][1]`, `[1][2]`, *dentro* del bucle más interno

Listas de listas rectangulares, matrices

El comportamiento de funciones con listas de listas es igual que con listas simples.
En el siguiente fragmento se definen dos funciones.
La primera devuelve una matriz de ceros de filas x columnas.
La segunda suma dos matrices y devuelve el resultado.

```
def zeros(filas,columnas):
        a = []
        for i in range(filas):
                a.append([0]*columnas)
        return a
def suma_matriz(a,b):
        c=zeros(len(a),len(a[0]))
        for i in range(len(a)):
                for j in range(len(a[0])):
                        c[i][j]=a[i][j]+b[i][j]
        return c
```