

# Listas

Una lista es una secuencia de datos. Ya sé que esta descripción es un poco incompleta, pero ten paciencia y confórmate de momento con esa idea.

```
In [2]: ▶ # Algunos datos de una persona, en una lista:

datos_amigo = ["Santiago", 56, 1.74, ["Música", "Lectura", "Ajedrez"]]

print(datos_amigo[0], datos_amigo[1], datos_amigo[2], datos_amigo[3])
print(datos_amigo[3][0])
print(len(datos_amigo))

Santiago 56 1.74 ['Música', 'Lectura', 'Ajedrez']
Música
4
```

Si la longitud de una lista es 4, sus elementos son el 0, 1, 2 y 3.

Las listas en Python son heterogéneas, porque sus elementos son de tipos arbitrarios y posiblemente distintos: en el ejemplo, un string, un entero, un real, o incluso una lista.

```
In [3]: ▶ # Los elementos de una lista pueden recorrerse con un índice que recorre

datos_amigo = ["Santiago", 56, 1.74, ["Música", "Lectura", "Ajedrez"]]

for i in range(len(datos_amigo)):
    print(datos_amigo[i])

print()

# Pero es más cómodo, y altamente recomendable hacerlo directamente sin
# Se recomienda fuertemente hacerlo así cuando sea posible.

for elem in datos_amigo:
    print(elem)

Santiago
56
1.74
['Música', 'Lectura', 'Ajedrez']

Santiago
56
1.74
['Música', 'Lectura', 'Ajedrez']
```

## Un ejemplo de partida: cálculo de la media

Queremos calcular la media de unos cuantos números reales:

```
In [3]: ▶ def media(numeros):
        """
        Esta función calcula la media de una lista de reales

        Parameters
        -----
        numeros : [float]
            Una lista de números reales no vacía

        Returns
        -----
        float
            La media de la lista

        Example
        -----
        >>> media([4.0, 6.0, 7.0, 3.0, 2.0])
        4.4
        """
        suma = 0.0
        for x in numeros:
            suma = suma + x
        return suma / len(numeros)

media([4.0, 6.0, 7.0, 3.0, 2.0])
```

Out[3]: 4.4

## Segundo ejemplo: polinomios

Podemos representar un **polinomio** como una lista:

$$3x^3 - 6x + 2$$

Lo representamos con 4 reales, los coeficientes de menor a mayor orden:

```
In [2]: ▶ poly = [2.0, -6.0, 0.0, 3.0]

# Acceso a cada elemento:
print(poly[0], poly[1], poly[2], poly[3])

# Longitud de una lista:
print(len(poly))

2.0 -6.0 0.0 3.0
4
```

Las posiciones de la lista poly van desde 0 hasta len(poly)-1 .

Ojo a los accesos ilegales!!

```
In [3]: poly[4]
```

```
-----  
-----  
IndexError                                Traceback (most recent call 1  
ast)  
<ipython-input-3-d87bb9df8586> in <module>  
----> 1 poly[4]  
  
IndexError: list index out of range
```

Se puede cambiar el contenido de una posición de una lista

```
In [4]: poly[3] = -7.0  
poly
```

```
Out[4]: [2.0, -6.0, 0.0, -7.0]
```

```
In [5]: poly[4] = 9.0
```

```
-----  
-----  
IndexError                                Traceback (most recent call 1  
ast)  
<ipython-input-5-c08954f25ba8> in <module>  
----> 1 poly[4] = 9.0  
  
IndexError: list assignment index out of range
```

Los elementos de una lista se pueden usar en cualquier contexto:

```
In [6]: a = poly[2] + 1 # en una expresión  
abs(poly[3]) # como argumento de una llamada a función
```

```
Out[6]: 7.0
```

Se pueden añadir más elementos a una lista

```
In [7]: poly.append(9.0)  
poly
```

```
Out[7]: [2.0, -6.0, 0.0, -7.0, 9.0]
```

Vamos a diseñar una función que evalúe un polinomio en un valor,  $x$ , de la variable.

```
In [9]:  x = 2
        suma = 0.0
        potencia = 1
        suma = suma + potencia * poly[0]
        potencia = potencia * x
        suma = suma + potencia * poly[1]
        potencia = potencia * x
        suma = suma + potencia * poly[2]
        potencia = potencia * x
        suma = suma + potencia * poly[3]
        suma
```

Out[9]: -66.0

Hagámoslo ahora con un bucle, para que funcione con un polinomio de cualquier grado.

```
In [12]: poly = [2.0, -6.0, 0.0, 3.0]
        x = 2
        suma = 0.0
        potencia = 1
        long = len(poly)
        for i in range(long):
            suma = suma + potencia * poly[i]
            potencia = potencia * x
        suma
```

Out[12]: 14.0

```
In [12]: ► def eval_poly(poly, x):
        """Esta función evalúa un polinomio para un valor dado de la variable x

        Parameters
        -----
        poly: [float]
            El polinomio, dado por la lista de sus coeficientes,
            donde poly[i] -> coeficiente de grado i
        x : float
            Valor de la variable x

        Returns
        -----
        float
            Valor del polinomio poly para el valor de la variable x

        Example
        -----
        >>> eval_poly( [1.0, 1.0], 2)
        3.0
        """
        suma = 0.0
        potencia = 1
        long = len(poly)
        i = 0
        for i in range(long):
            suma = suma + poly[i] * potencia
            potencia = potencia * x
        return suma

eval_poly([2.0, -6.0, 0.0], 2.0), eval_poly( [1.0, 1.0], 2)
```

Out[12]: (-10.0, 3.0)

### Ejemplo 3: Descomposición en factores primos de un número

Ejemplos:

60 -> [2, 2, 3, 5]

15 -> [3, 5]

```

In [13]: ► def factores(n):
          """
          Genera la lista de factores de n

          Parameters
          -----
          n : int
              Número que se desea descomponer, n > 1

          Returns
          -----
          [int]
              Factores de n

          Example
          -----
          >>> factores(256)
          [2, 2, 2, 2, 2, 2, 2, 2]
          """
          fct = 2 # el primer factor primo posible
          lista_factores = []
          while n > 1:
              if n % fct == 0:
                  lista_factores.append(fct)
                  n = n // fct
              else:
                  fct += 1
          return lista_factores

factores(3**2 * 2**4 * 7**5 * 5**2 * 3 * 7)

```

```

Out[13]: [2, 2, 2, 2, 3, 3, 3, 5, 5, 7, 7, 7, 7, 7]

```

Si queremos calcular la multiplicidad de cada divisor, podemos hacer que devuelva una lista de tuplas:

```
In [14]: ▶ def factores_exponentes(n):
        """
        Genera la lista de factores de n,
        cada uno de ellos con su grado de multiplicidad

        Parameters
        -----
        n : int
            Número que se desea descomponer, n > 1

        Returns
        -----
        [(int, int)]
            Factores (factor, exponente) de n

        Example
        -----
        >>> factores_exponentes(256)
        [(2, 8)]
        """
        fct = 2
        lista_factores = []
        while n > 1:
            if n % fct == 0:
                exp = 1
                n = n // fct
                while (n % fct) == 0:
                    exp += 1
                    n = n // fct
                lista_factores.append([fct, exp])
                # aquí, n % fct != 0, de manera que incrementamos fct
                fct = fct + 1
        return lista_factores

        print(60, " -> ", factores_exponentes(60))
        print(96, " -> ", factores_exponentes(96))

60 -> [[2, 2], [3, 1], [5, 1]]
96 -> [[2, 5], [3, 1]]
```

## Listas intensionales

```
In [15]: ▶ [(i, i**2) for i in range(2, 10)]
```

```
Out[15]: [(2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
```

```
In [16]: ▶ # Descomponemos 120 en dos factores de todos los modos posibles:
```

```
print([(i, 120//i) for i in range(1, 120+1) if 120 % i == 0])
```

```
# Seguro que tú puedes diseñar una función que hace esto con cualquier e
```

```
[(1, 120), (2, 60), (3, 40), (4, 30), (5, 24), (6, 20), (8, 15), (10, 12), (12, 10), (15, 8), (20, 6), (24, 5), (30, 4), (40, 3), (60, 2), (120, 1)]
```

```
In [17]: # Combinación de generadores (for) y filtros (if):  
  
print([(i, j) for i in range(10) for j in range(i) if i % 4 == 0 if j >  
      [(8, 4), (8, 5), (8, 6), (8, 7)]
```

```
In [18]: # Lo siguiente usa la descomposición en factores definida antes:  
  
[(i, factores_exponentes(i)) for i in range(2, 40)]
```

```
Out[18]: [(2, [[2, 1]]),  
          (3, [[3, 1]]),  
          (4, [[2, 2]]),  
          (5, [[5, 1]]),  
          (6, [[2, 1], [3, 1]]),  
          (7, [[7, 1]]),  
          (8, [[2, 3]]),  
          (9, [[3, 2]]),  
          (10, [[2, 1], [5, 1]]),  
          (11, [[11, 1]]),  
          (12, [[2, 2], [3, 1]]),  
          (13, [[13, 1]]),  
          (14, [[2, 1], [7, 1]]),  
          (15, [[3, 1], [5, 1]]),  
          (16, [[2, 4]]),  
          (17, [[17, 1]]),  
          (18, [[2, 1], [3, 2]]),  
          (19, [[19, 1]]),  
          (20, [[2, 2], [5, 1]]),  
          (21, [[3, 1], [7, 1]]),  
          (22, [[2, 1], [11, 1]]),  
          (23, [[23, 1]]),  
          (24, [[2, 3], [3, 1]]),  
          (25, [[5, 2]]),  
          (26, [[2, 1], [13, 1]]),  
          (27, [[3, 3]]),  
          (28, [[2, 2], [7, 1]]),  
          (29, [[29, 1]]),  
          (30, [[2, 1], [3, 1], [5, 1]]),  
          (31, [[31, 1]]),  
          (32, [[2, 5]]),  
          (33, [[3, 1], [11, 1]]),  
          (34, [[2, 1], [17, 1]]),  
          (35, [[5, 1], [7, 1]]),  
          (36, [[2, 2], [3, 2]]),  
          (37, [[37, 1]]),  
          (38, [[2, 1], [19, 1]]),  
          (39, [[3, 1], [13, 1]])]
```