Universidad Andrés Bello®

Unab Online

Listas y Strings

León, R. (2021). *Listas y Strings* [apunte].

Chile. UNAB

LISTAS Y STRINGS

1. Listas

Una *lista* es una colección ordenada de valores. Una lista puede contener cualquier tipo de dato. En Python el tipo de datos que representa a la lista se denomina **list**.

Las dos formas principales para crear una lista son las siguientes: (i) usar una lista literal, es decir, se indican los elementos de la lista entre paréntesis de corchete, o (ii) utilizando la sentencia list aplicada sobre un iterable. En la Figura 1 se muestra un ejemplo para cada forma.

Figura 1
Ejemplo de creación de listas.

Forma (i)

```
>>> primos = [2, 3, 5, 7, 11]
>>> primos
[2, 3, 5, 7, 11]
>>> [1 + 2.0, 3.5 + 4.5, 9 + 1 - 4]
[3.0, 8.0, 6]
>>> ["hola" + " " + "mundo", 12 * 34, True or False]
['hola mundo', 408, True]
```

Forma (ii)

```
>>> list("chao")
['c', 'h', 'a', 'o']
>>> list(range(7))
[0, 1, 2, 3, 4, 5, 6]
>>> list()
[]
```

2. Operaciones sobre listas

Existen distintas operaciones que se pueden realizar sobre las listas:

len(l) entrega la cantidad de elementos que se encuentran en la lista

Figura 2
Ejemplo de la operación len(l).

```
>>> paises = ["Bolivia", "Chile", "Peru", "Uruguay"]
>>> len(paises)
4
>>> len([True, False, True])
3
>>> len([])
0
```

l[i] entrega el elemento en la posición i de la lista. El valor de i se denomina **índice**. Hay que tener en cuenta que los índices comienzan desde el valor cero.

Figura 3
Ejemplo del uso del índice en una lista.

```
>>> paises = ["Bolivia", "Chile", "Peru", "Uruguay"]
>>> paises[0]
'Bolivia'
>>> paises[3]
'Uruguay'
```

El valor del elemento en la posición i de la lista es posible modificarlo mediante una instrucción de asignación.

Figura 4
Ejemplo de modificación de un elemento en una lista.

```
>>> paises = ["Bolivia", "Chile", "Peru", "Uruguay"]
>>> paises[2] = "Ecuador"
>>> paises
['Bolivia', 'Chile', 'Ecuador', 'Uruguay']
```

Si se indica una posición i de la lista donde no existe un elemento, se produce un **error de índice.**

Figura 5
Ejemplo de error de índice en una lista.

```
>>> paises = ["Bolivia", "Chile", "Peru", "Uruguay"]
>>> paises[4]
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    paises[4]
IndexError: list index out of range
```


Si se indica una posición i negativa, entonces los elementos se cuentan desde el final hacia atrás.

Figura 6
Ejemplo de posición negativa.

```
>>> paises = ["Bolivia", "Chile", "Peru", "Uruguay"]
>>> paises[-2]
'Peru'
>>> paises[-4]
'Bolivia'
```

l.**append**(x) agrega el elemento x al final de la lista

Figura 7
Ejemplo de la operación l.append(x).

```
>>> paises = ["Bolivia", "Chile", "Peru", "Uruguay"]
>>> paises.append("Brasil")
>>> paises.append("Colombia")
>>> paises
['Bolivia', 'Chile', 'Peru', 'Uruguay', 'Brasil', 'Colombia']
```

sum(l) entrega la suma de los elementos de la lista.

Figura 8
Ejemplo de la función sum(l).

```
>>> valores = [2, 1 , -2, 3, -4, 5]
>>> sum(valores)
5
>>> sum([])
0
```


l1 + l2 concatena las listas l1 y l2.

Figura 9
Ejemplo de concatenación de listas.

```
>>> list("python") + [3, 6, 2]
['p', 'y', 't', 'h', 'o', 'n', 3, 6, 2]
>>> [1, 3] + [99, 55]
[1, 3, 99, 55]
```

l * n repite n veces la lista l.

Figura 10
Ejemplo de repetición de listas.

```
>>> l = list("waka")
>>> l
['w', 'a', 'k', 'a']
>>> l * 2
['w', 'a', 'k', 'a', 'w', 'a', 'k', 'a']
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> [1, 2, 3] * 0
[]
```

Para saber si un elemento x está en la lista l, se utiliza x **in** l. La versión negativa es **not** x **in** l.

Figura 11
Ejemplo de pertenencia en una lista.

```
>>> numeros = list(range(1, 17, 2))
>>> numeros
[1, 3, 5, 7, 9, 11, 13, 15]
>>> 11 in numeros
True
>>> 14 in numeros
False
>>> 6 not in numeros
True
```

l[i:j] es el operador de *rebanado*. Entrega una nueva lista que contiene los elementos desde la posición i hasta justo antes de la posición j.

Figura 12
Ejemplo de rebanado en una lista.

```
>>> letras = list("programacion")
>>> letras
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'c', 'i', 'o', 'n']
>>> letras[3:7]
['g', 'r', 'a', 'm']
>>> letras[-5:-2]
['a', 'c', 'i']
>>> letras[:4]
['p', 'r', 'o', 'g']
>>> letras[-6:]
['m', 'a', 'c', 'i', 'o', 'n']
>>> letras[5:]
['a', 'm', 'a', 'c', 'i', 'o', 'n']
>>> letras[:-7]
['p', 'r', 'o', 'g', 'r']
```

l.count(x) cuenta la cantidad de veces que el elemento x se encuentra en la lista l.

Figura 13
Ejemplo para contar elementos en una lista.

```
>>> letras = list("programacion")
>>> letras
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'c', 'i', 'o', 'n']
>>> letras.count("a")
2
>>> letras.count("v")
0
```

l.index(x) entrega la primera posición encontrada del elemento x en la lista l.

Figura 14
Ejemplo para obtener la primera posición de un elemento en una lista.

```
>>> letras = list("programacion")
>>> letras
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'c', 'i', 'o', 'n']
>>> letras.index("a")
5
>>> letras.index("g")
3
>>> letras.index("v")
Traceback (most recent call last):
  File "<pyshell#105>", line 1, in <module>
    letras.index("v")
ValueError: 'v' is not in list
```

l.remove(x) elimina el primer elemento x encontrado de la lista l.

Figura 15
Ejemplo para remover un elemento de una lista.


```
>>> letras = list("programacion")
>>> letras
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'c', 'i', 'o', 'n']
>>> letras.remove("a")
>>> letras
['p', 'r', 'o', 'g', 'r', 'm', 'a', 'c', 'i', 'o', 'n']
>>> letras.remove("r")
>>> letras
['p', 'o', 'g', 'r', 'm', 'a', 'c', 'i', 'o', 'n']
```

Se puede eliminar también un elemento de la lista indicando la posición mediante **del** l[i].

Figura 16
Ejemplo para eliminar un elemento de una lista indicando la posición.

```
>>> letras = list("programacion")
>>> letras
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'a', 'c', 'i', 'o', 'n']
>>> del letras[5]
>>> letras
['p', 'r', 'o', 'g', 'r', 'm', 'a', 'c', 'i', 'o', 'n']
```

l.reverse() invierte la lista.

Figura 17
Ejemplo para invertir una lista.

```
>>> numeros = [2, 5, 3, 1, 7, 9, 2, 8]
>>> numeros.reverse()
>>> numeros
[8, 2, 9, 7, 1, 3, 5, 2]
```


l.sort() ordena la lista.

Figura 18
Ejemplo para ordenar una lista.

```
>>> numeros = [2, 5, 3, 1, 7, 9, 2, 8]
>>> numeros.sort()
>>> numeros
[1, 2, 2, 3, 5, 7, 8, 9]
```

Para recorrer los elementos de una lista, simplemente se utiliza la sentencia *for*, la cual permite que la variable de control obtenga el valor de cada elemento a medida que se recorren. Por ejemplo, en la Figura 19 se muestra un programa donde se utiliza la sentencia *for* para recorrer una lista y ejecutar ciertas instrucciones con cada elemento. En cada iteración del ciclo *for* la variable de control i obtiene un valor, luego calcula el cuadrado y el cubo de tal valor y lo imprime por pantalla.

Figura 19
Ejemplo para recorrer una lista.

Programa

```
valores = [2, 6, 1, 9]
for i in valores:
    cuad = i ** 2
    cubo = i ** 3
    print(f"{i} elevado a 2 es {cuad}")
    print(f"{i} elevado a 3 es {cubo}")
```

Salida del programa

```
2 elevado a 2 es 4
2 elevado a 3 es 8
6 elevado a 2 es 36
6 elevado a 3 es 216
1 elevado a 2 es 1
1 elevado a 3 es 1
9 elevado a 2 es 81
9 elevado a 3 es 729
```


3. Cadenas (strings)

Un texto siempre es un string (cadena), que puede tener una longitud determinada y ser tan complejo como uno desee. Estas cadenas se pueden manipular, ya sea para extraer información, para convertir un texto en otro, o para codificar información en un string.

En Python, el tipo de datos para las cadenas es str, sobre el cual se pueden aplicar ciertas acciones (métodos) convenientes para procesar algún texto.

Comencemos indicando que ya hemos visto algunos operadores que también se pueden aplicar a los strings, como se muestra en la Figura 20.

Figura 20
Operaciones simples sobre strings.

Programa

```
s = "avion"
t = "tren"

#concatenacion de strings
print("no voy en " + t + " voy en " + s)

#elemento en la posicion i de un string
print(s[1])

#pertenencia
print("en" in t)
print("x" in s)
```

Salida del programa

```
no voy en tren voy en avion
v
True
False
```


El método s.replace(x, y) busca en el string s todas las apariciones del texto x y lo reemplaza por el texto y.

Figura 21
Ejemplo del método replace.

Programa

```
s = "la mar estaba serena"
print(s)

t = s.replace("a","e")
print(t)

w = t.replace("e","o")
print(w)
```

Salida del programa
la mar estaba serena le mer estebe serene lo mor ostobo sorono

El método s.split(sep) separa el string s en varios strings, usando el texto sep como separador. Si el separador no se indica, se asume como espacio. Este método entrega el resultado a través de una lista.

Figura 22
Ejemplo del método split.

Programa

```
s = "la mar estaba serena"
print(s)

lista1 = s.split()
print(lista1)

lista2 = s.split("e")
print(lista2)
```

Salida del programa
la mar estaba serena ['la', 'mar', 'estaba', 'serena'] ['la mar ', 'staba s', 'r', 'na']

Este método es muy útil al momento de solicitar varios valores al usuario. En la Figura 23 se puede observar el uso de este método permitiendo al usuario ingresar varios valores en una sola línea.

Figura 23
Ejemplo del método split para el ingreso de varios valores.

Programa

```
datos = input("Ingrese base y altura del triangulo: ")

valores = datos.split()

base = float(valores[0])
altura = float(valores[1])

area = (base * altura) / 2
print(f"El area es {area}")
```

Salida del programa

```
Ingrese base y altura del triangulo: 2.5 10.2
El area es 12.75
```

El método sep.join(l) une todos los strings de la lista l utilizando el separador sep. Este método deja el resultado en un nuevo string.

Figura 24
Ejemplo del método join.

Programa

```
valores = ["1", "2", "3", "4", "5", "6", "7"]

cadena1 = " ".join(valores)
print(cadena1)

cadena2 = " -> ".join(valores)
print(cadena2)

cadena3 = "".join(valores)
print(cadena3)
```

Salida del programa

```
1 2 3 4 5 6 7
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
1234567
```

"Para complementar la información y contenidos presentados, puedes dirigirte a":

Bibliografía

1. **Downey, A. (2013). Think Python. Green Tea Press.** Accesado el 2 de diciembre de 2019 en https://greenteapress.com/wp/think-python-2e/

2. **Python software foundation, Python v3 Documentation.** Accesado el 2 de diciembre de 2019 en https://docs.python.org/3/

3. Marzal Varó, A., Gracia Luengo, I., & García Sevilla, P. (2014). Introducción a la programación con Python 3. Universitat Jaume I. https://doi.org/10.6035/sapientia93 (Capítulo 1, secciones 1.1 y 1.2, páginas: 11 - 16)

4. Hinojosa, A. (2016). Python paso a paso. RA-MA Editorial y Publicaciones.

5. Marzal, A., Gracia, I., García, P. (2014). Introducción a la Programación con Python 3. Universitat Jaumé Publicaciones.

6. Bonvallet, R. (2013). Apuntes de Programación. Editorial USM.