TEL101 - Iniciación a la Programación

Clase 5: Listas

Prof. Nicolás Torres
nicolas.torresr@usm.cl
Ingeniería Civil Telemática
Departamento de Electrónica
Universidad Técnica Federico Santa María

Listas

Las listas son secuencias **mutables** compuestas por elementos de cualquier tipo. Se utilizan como "contenedores" para almacenar múltiples valores en una sola variable.

Creación

Las listas se declaran por medio de los paréntesis corchetes `[ ]`, y sus elementos se separan por comas.

Lista vacía

In [1]:
```
lista = []
```

Lista con elementos

In [2]:
```
a = [1, 3.141516, 7, -5, 0.25]

b = [1, a, 'a', 'hola']
```

In [3]:
```
print("La lista a:",a)
print("La lista b:",b)
```

```
La lista a: [1, 3.141516, 7, -5, 0.25]
La lista b: [1, [1, 3.141516, 7, -5, 0.25], 'a', 'hola']
```

Además, el constructor `list()` recibe un iterable y construye una lista cuyos elementos son los mismos y en el mismo orden que en el iterable. Si el argumento es una lista, se genera una copia; y si el argumento no es provisto, se retorna una lista vacía.

In [4]:
```
list("abc")
```

Out[4]:
```
['a', 'b', 'c']
```

Índices

Al igual que los strings, las listas tienen elementos ordenados posicionalmente desde el primero hasta el último, con índices positivos $(0, 1, 2, \ldots)$, y desde el último hasta el primero, a través de índices negativos ($-1, -2, -3, \ldots$).

In [5]:
```
letras = ['a', 'b', 'c', 'd', 'e']
```

In [6]:
```
letras[0]
```

Out[6]:
```
'a'
```

In [7]:
```
letras[1]
```

Out[7]:
```
'b'
```

In [8]:
```
letras[2]
```

Out[8]:
```
'c'
```

In [9]:
```
letras[-1]
```

Out[9]:
```
'e'
```

In [10]:
```
letras[-2]
```

Out[10]:
```
'd'
```

In [11]:
```
letras[-3]
```

Out[11]:
```
'c'
```

Rebanadas (Slicing)

Al igual que los strings, existe el operador slicing.

`lista[i:j]` entrega los elementos de la `lista` desde el índice `i` hasta el índice `j-1`.

In [12]:
```
letras = ['a', 'b', 'c', 'd', 'e']
```

In [13]:
```
letras[1:4]
```

Out[13]:
```
['b', 'c', 'd']
```

Las variantes `lista[i:]`, `lista[:j]`, `lista[i:j:k]` también están disponibles.

Mutabilidad

A diferencia de los strings, las listas son un tipo de dato **mutable**. Por lo tanto, sus elementos se pueden modificar.

En consecuencia, `lista[i] = x`, reemplaza el elemento en la posición `i` de la `lista` por el valor `x`.

In [14]:
```
x = [1, 2, 3]
```

```
x[1] = 'a'
print(x)
```

```
[1, 'a', 3]
```

Operaciones básicas sobre listas

Concatenación

El operador concatenador (`+`) retorna la unión entre secuencias.

In [15]:
```
['a', 'b'] + ['c', 'd']
```

Out[15]:
```
['a', 'b', 'c', 'd']
```

Repetición

El operador repetidor (`*`) retorna los elementos de la secuencia repetidos.

In [16]:
```
['a', 'b'] * 3
```

Out[16]:
```
['a', 'b', 'a', 'b', 'a', 'b']
```

Membresía

En las listas, `x in y` es `True` si y solo si `x` es un elemento de `y` .

In [17]:
```
'pollo' in ['repollo', 'gallinero', 'rechicken']
```

Out[17]:
```
False
```

In [18]:
```
2 in [1,2,3]
```

Out[18]:
```
True
```

In [19]:
```
'2' in [1,2,3]
```

Out[19]:
```
False
```

Funciones básicas sobre listas

Longitud de una lista

La función `len(s)` retorna el número de elementos en la lista `s` .

In [20]:
```
len(['Hola', 'Mundo'])
```

Out[20]:
```
2
```

Mínimo y Máximo

La función `min(s)` retorna el menor elemento en la lista `s`.

La función `max(s)` retorna el mayor elemento en la lista `s`.

El criterio que utiliza para comparar caracteres es el **orden lexicográfico**.

Todos los elementos deben ser comparables entre sí.

In [21]:
```
min([6, 4, 1, 9, 5])
```

Out[21]: 1

In [22]:
```
max([6, 4, 1, 9, 5])
```

Out[22]: 9

In [23]:
```
min(['a', 'e', 'i', 'o', 'u'])
```

Out[23]: 'a'

In [24]:
```
max(['a', 'e', 'i', 'o', 'u'])
```

Out[24]: 'u'

Función `sum`

La función `sum(s)` retorna la suma de todos los elementos en la lista `s`. Todos los elementos de la lista `s` deben ser numéricos; de lo contrario, ocurre una excepción.

In [25]:
```
sum([1, 2, 3, 4, 5])
```

Out[25]: 15

In [26]:
```
sum([0.25, 1.5, -1, 3.75])
```

Out[26]: 4.5

Métodos de listas

Las listas implementan una serie de métodos enfocados en agregar, eliminar y cambiar el orden de sus elementos. A continuación, se detallan los métodos más comunes.

Método `index`

El método `list.index(e)` retorna el índice del primer elemento cuyo valor es igual a `e` en la lista `list`. Si el elemento no se encuentra, se produce una excepción.

In [27]:
```
letras = ['a', 'a', 'b', 'a', 'b']
```

In [28]:
```
letras.index("b")
```

Out[28]:
```
2
```

In [29]:
```
letras.index('c')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [29], in <cell line: 1>()
----> 1 letras.index('c')

ValueError: 'c' is not in list
```

Método `count`

El método `list.count(e)` retorna el número de ocurrencias del elemento `e` en la lista `list`.

In [30]:
```
vocales = ['a', 'e', 'i', 'o', 'i', 'u']
```

In [31]:
```
vocales.count('i')
```

Out[31]:
```
2
```

In [32]:
```
vocales.count('b')
```

Out[32]:
```
0
```

Método `append`

El método `list.append(e)` agrega el elemento `e` al final de la lista `list`. Este método solo acepta un argumento a la vez. En otras palabras, no es posible agregar varios elementos al mismo tiempo.

In [33]:
```
x = [1, 2, 3]
x.append('a')
print(x)
```

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

Método `insert`

El método `list.insert(i,e)` agrega el elemento `e` en la posición `i` de la lista `list`.

In [34]:
```
x = [1, 2, 3]
x.insert(1,'a')
print(x)
```

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

Método `remove`

El método `list.remove(e)` elimina el primer elemento de la lista `list` cuyo valor sea igual a `e`. Si el elemento `e` no se encuentra, se produce una excepción.

In [35]:
```
x = [1, 3, 4, 1]
x.remove(1)
print(x)
```

```
[3, 4, 1]
```

Hay una forma de eliminar un elemento de una lista dado su índice en lugar de su valor: la instrucción `del`.

In [36]:
```
lista = [1, 3, 4, 1]
del lista[1]
print(lista)
```

```
[1, 4, 1]
```

Método `sort`

El método `list.sort()` ordena los elementos de la lista `list` de menor a mayor. Los elementos de la lista deben ser comparables entre sí, y de manera opcional, se puede cambiar el criterio de orden. El parámetro `reverse`, cuyo valor por defecto es `False`, permite ordenar de forma descendente si se establece en `True`.

In [37]:
```
lista = [6, 8, 1, 4, 3]
lista.sort()
print(lista)
```

```
[1, 3, 4, 6, 8]
```

In [38]:
```
lista = ['b', 'e', 'a', 'd', 'c']
lista.sort(reverse=True)
print(lista)
```

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

Método `reverse`

El método `list.reverse()` invierte el orden de los elementos de la lista `list`.

In [39]:
```
lista = [1, 2, 3, 4, 5]
lista.reverse()
print (lista)
```

```
[5, 4, 3, 2, 1]
```

Estructura de Repetición `for`

In [40]:
```
colores = ["verde", "azul", "amarillo", "rojo", "negro"]
for color in colores:
    print ('El color es',color)
```

```
El color es verde
El color es azul
El color es amarillo
El color es rojo
El color es negro
```

Equivalencia entre ciclo `while` y ciclo `for`.

In [41]:
```
colores = ["verde", "azul", "amarillo", "rojo", "negro"]
i = 0
while i < len(colores):
    print('El color en la posición',i,'es',colores[i])
    i += 1
```

```
El color en la posición 0 es verde
El color en la posición 1 es azul
El color en la posición 2 es amarillo
El color en la posición 3 es rojo
El color en la posición 4 es negro
```

Rangos

La función `range()` entrega una secuencia de enteros basada en un valor de inicio, un valor de término y un salto entre valores. Solo es obligatorio señalar el valor de término, ya que por defecto el valor de inicio es `0` con un salto o incremento de `1`. El valor de término no está incluido en la secuencia, así `range(n)` genera el intervalo 0, 1, 2, ..., `n-1`.

In [42]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [43]:
```
list(range(10))
```

Out[43]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [44]:
```
list(range(1, 11))
```

Out[44]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [45]:
```
list(range(0, 30, 5))
```

Out[45]:
```
[0, 5, 10, 15, 20, 25]
```

In [46]:
```
list(range(0, -10, -1))
```

Out[46]:
```
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Recorrer una lista a través de su rango de índices

```
for índice in range(len(lista)):
    elemento = lista[índice]
```

Ejemplo

In [47]:
```
colores = ["verde", "azul", "amarillo", "rojo", "negro"]
for i in range(len(colores)):
    print ('El color en la posición',i,'es',colores[i])
```

```
El color en la posición 0 es verde
El color en la posición 1 es azul
El color en la posición 2 es amarillo
El color en la posición 3 es rojo
El color en la posición 4 es negro
```