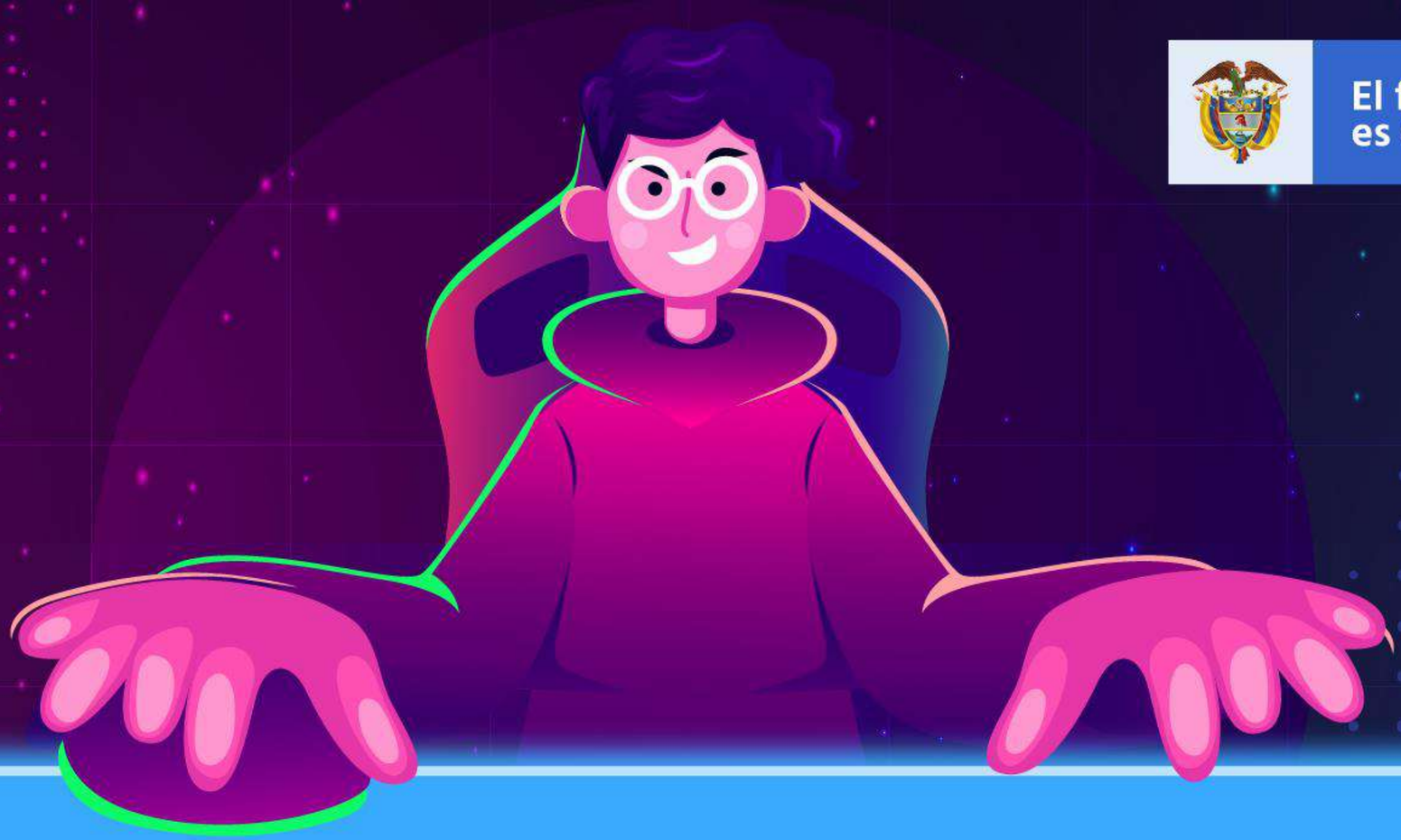


El futuro digital es de todos

MinTIC

Listas

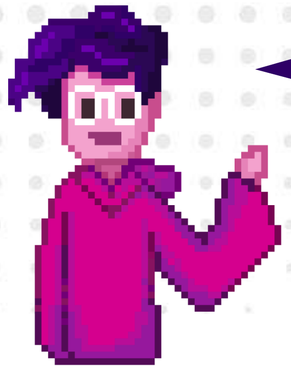
OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Listas



Python tiene varios tipos de datos compuestos y dentro de las secuencias, un tipo muy importante de secuencia son las **listas**.

Entre las secuencias, el más versátil, es la lista. Para definir una, usted debe escribir es entre corchetes cuadrados, separando sus elementos con comas cada uno.

La lista en Python son variables que almacenan arrays, internamente cada posición puede ser un tipo de datos distinto.

Las listas en Python son:

heterogéneas: pueden estar conformadas por elementos de distintos tipo, incluidos otras listas.
mutables: sus elementos pueden modificarse. Una lista en Python es una estructura de datos formada por una secuencia ordenada de objetos.

Los elementos de una lista pueden accederse mediante su índice, siendo 0 el índice del primer elemento.

In [18]:
```
lista = [1, 2.5, 'DevCode', [5,6] ,4]

print(lista[0]) # 1
print(lista[1]) # 2.5
print(lista[2]) # DevCode
print(lista[3]) # [5,6]
print(lista[3][0]) # 5
print(lista[3][1]) # 6
print(lista[1:3]) # [2.5, 'DevCode']
print(lista[1:6]) # [2.5, 'DevCode', [5, 6], 4]
print(lista[1:6:2]) # [2.5, [5, 6]]
```

```
1
2.5
DevCode
[5, 6]
5
6
[2.5, 'DevCode']
[2.5, 'DevCode', [5, 6], 4]
[2.5, [5, 6]]
```

Una lista que no contiene ningún elemento se denomina lista vacía:

```
In [19]: lista = [ ]
         lista

Out[19]: []
```

Al definir una lista se puede hacer referencia a otras variables.

Como siempre, hay que tener cuidado al modificar una variable que se ha utilizado para definir otras variables, porque esto puede afectar al resto de variables:

• Si se trata objetos inmutables, el resto de variables no resultan afectadas, como muestra el siguiente ejemplo:

```
In [22]: nombre = "Pepe"
         edad = 25
         listas = [nombre, edad]
         listas

Out[22]: ['Pepe', 25]

In [23]: nombre = "Juan" # Queda la memoria cargada
         listas

Out[23]: ['Pepe', 25]
```

Pero si se trata de objetos mutables y al modificar la variable se modifica el objeto, el resto de variables sí resultan afectadas, como muestra el siguiente ejemplo:

```
In [24]: nombres = ["Ana", "Bernardo"]
         edades = [22, 21]
         lista = [nombres, edades]
         lista

Out[24]: [['Ana', 'Bernardo'], [22, 21]]

In [25]: nombres += ["Cristina"]
         lista

Out[25]: [['Ana', 'Bernardo', 'Cristina'], [22, 21]]

In [26]: factura = ['pan', 'huevos', 100, 1234]
         factura

Out[26]: ['pan', 'huevos', 100, 1234]

In [27]: factura[0]

Out[27]: 'pan'

In [28]: factura[3]

Out[28]: 1234
```

La función len() devuelve la longitud de la lista (su cantidad de elementos).

```
In [29]: len(factura)
Out[29]: 4
```

Los índices de una lista inicia entonces de 0 hasta el tamaño de la lista menos uno (len(factura) - 1):

```
In [30]: len(factura) - 1
Out[30]: 3
```

Pueden usarse también índices negativos, siendo -1 el índice del último elemento.

```
In [31]: factura[-1]
Out[31]: 1234
```

Los índices negativos van entonces de -1 (último elemento) a -len(factura) (primer elemento).

```
In [32]: factura[-len(factura)]
Out[32]: 'pan'
```

A través de los índices, pueden cambiarse los elementos de una lista en el lugar.

```
In [33]: factura[1] = "carne"
         factura
Out[33]: ['pan', 'carne', 100, 1234]
```

De esta forma se cambia el valor inicial de un elemento de la lista lo cual hacen una la lista mutable

Métodos

El el objeto de tipo lista integra una serie de métodos integrados a continuación:

append()

Este método agrega un elemento al final de una lista.

```
In [34]: versiones_plone = [2.5, 3.6, 4, 5]
         versiones_plone

Out[34]: [2.5, 3.6, 4, 5]

In [35]: versiones_plone.append(6)
         versiones_plone

Out[35]: [2.5, 3.6, 4, 5, 6]
```

count()

Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista.

```
In [36]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]

In [37]: print("6 ->", versiones_plone.count(6))
6 -> 1

In [38]: print("5 ->", versiones_plone.count(5))
5 -> 1

In [39]: print( "2.5 ->", versiones_plone.count(2.5))
2.5 -> 1
```

extend()

Este método extiende una lista agregando un iterable al final.

```
In [40]: versiones_plone = [2.1, 2.5, 3.6]
         versiones_plone

Out[40]: [2.1, 2.5, 3.6]

In [41]: versiones_plone.extend([4])
         versiones_plone

Out[41]: [2.1, 2.5, 3.6, 4]

In [42]: versiones_plone.extend(range(5,7))
         versiones_plone

Out[42]: [2.1, 2.5, 3.6, 4, 5, 6]
```

index()

Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la lista.

```
In [43]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6, 4]
         print(versiones_plone.index(4))
```

```
3
```

El método admite como argumento adicional un índice inicial a partir de donde comenzar la búsqueda, opcionalmente también el índice final.

```
In [44]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6, 4]
         versiones_plone[2]
```

```
Out[44]: 3.6
```

```
In [45]: print(versiones_plone.index(4, 2))
```

```
3
```

```
In [46]: versiones_plone[3]
```

```
Out[46]: 4
```

```
In [47]: print(versiones_plone.index(4, 5))
```

```
6
```

```
In [48]: versiones_plone[6]
```

```
Out[48]: 4
```

El método devuelve un excepción ValueError si el elemento no se encuentra en la lista, o en el entorno definido.

```
In [1]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6, 4]

In [2]: print(versiones_plone.index(9))
```

```
---------------------------------------------------------------------
---
ValueError                                Traceback (most recent call la
st)
<ipython-input-2-c3c11fc57eef> in <module>
----> 1 print(versiones_plone.index(9))

ValueError: 9 is not in list
```

insert()

Este método inserta el elemento x en la lista, en el índice i.

```
In [51]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]
         print(versiones_plone)
```

```
[2.1, 2.5, 3.6, 4, 5, 6]
```

```
In [52]: versiones_plone.insert(2, 3.7)
         print(versiones_plone)
```

```
[2.1, 2.5, 3.7, 3.6, 4, 5, 6]
```

pop()

Este método devuelve el último elemento de la lista, y lo borra de la misma.

```
In [53]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]

         print(versiones_plone.pop())
```

```
6
```

```
In [3]: print(versiones_plone)
```

```
[2.1, 2.5, 3.6, 4, 5, 6, 4]
```

Opcionalmente puede recibir un argumento numérico, que funciona como índice del elemento (por defecto, -1)

```
In [4]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]

In [5]: print(versiones_plone.pop(2))
3.6

In [6]: print(versiones_plone)
[2.1, 2.5, 4, 5, 6]
```

remove()

Este método recibe como argumento un elemento, y borra su primera aparición en la lista.

```
In [7]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]
print(versiones_plone)
[2.1, 2.5, 3.6, 4, 5, 6]

In [58]: versiones_plone.remove(2.5)
print(versiones_plone)
[2.1, 3.6, 4, 5, 6]
```

El método devuelve un excepción **ValueError** si el elemento no se encuentra en la lista.

```
In [59]: versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]
         print(versiones_plone)

[2.1, 2.5, 3.6, 4, 5, 6]

In [60]: versiones_plone.remove(7)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-60-09e9ae60eb3f> in <module>
----> 1 versiones_plone.remove(7)

ValueError: list.remove(x): x not in list
```

reverse()

Este método invierte el orden de los elementos de una lista.

In [61]:
```
versiones_plone = [2.1, 2.5, 3.6, 4, 5, 6]
print(versiones_plone)
```

```
[2.1, 2.5, 3.6, 4, 5, 6]
```

In [62]:
```
versiones_plone.reverse()
print(versiones_plone)
```

```
[6, 5, 4, 3.6, 2.5, 2.1]
```

sort()

Este método ordena los elementos de una lista.

```
In [63]: versiones_plone = [4, 2.5, 5, 3.6, 2.1, 6]
         print(versiones_plone)

[4, 2.5, 5, 3.6, 2.1, 6]
```

```
In [64]: versiones_plone.sort()
         print(versiones_plone)

[2.1, 2.5, 3.6, 4, 5, 6]
```

El método sort() admite la opción reverse, por defecto, con valor False. De tener valor True, el ordenamiento se hace en sentido inverso.

```
In [65]: versiones_plone.sort(reverse=True)
         print(versiones_plone)

[6, 5, 4, 3.6, 2.5, 2.1]
```

Ejemplo de iterar sobre una lista:

Para separar una cadena en frases, los valores pueden separarse con la función integrada split().

```
In [75]: mensaje = "Hola, como estas tu?"
         mensaje.split() # retorna una lista

Out[75]: ['Hola,', 'como', 'estas', 'tu?']
```

```
In [76]: for palabra in mensaje.split():
             print(palabra)

Hola,
como
estas
tu?
```

Ejemplo de iterar sobre dos o más secuencias

Para iterar sobre dos o más secuencias al mismo tiempo, los valores pueden emparejarse con la función integrada zip().

In [77]:
```
preguntas = ['nombre', 'objetivo', 'sistema operativo']
respuestas = ['Leonardo', 'aprender Python y Plone', 'Linux']

for pregunta, respuesta in zip(preguntas, respuestas):
    print('¿Cual es tu {0}?, la respuesta es: {1}.'.format(
          pregunta, respuesta))
```

```
¿Cual es tu nombre?, la respuesta es: Leonardo.
¿Cual es tu objetivo?, la respuesta es: aprender Python y Plone.
¿Cual es tu sistema operativo?, la respuesta es: Linux.
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022