



Estructuras de datos y funciones

Introducción a estructuras de datos

Utilizar estructuras de datos apropiadas para la elaboración de un algoritmo que resuelve un problema acorde al lenguaje Python.

Codificar un programa utilizando funciones para la reutilización de código acorde al lenguaje Python.

- Unidad 1:
Introducción a Python
- Unidad 2:
Sentencias condicionales e iterativas
- Unidad 3:
Estructuras de datos y funciones



Te encuentras aquí



¿Qué aprenderás en esta sesión?

- *Identifica las características de las distintas estructuras de datos para la resolución de problemas.*

¿Qué entendemos por
estructura de datos?



`/* Listas */`

Introducción a listas

Son contenedores que permiten almacenar múltiples datos.

```
# Estos elementos pueden ser de un mismo tipo de datos, por ejemplo solo strings:  
animales = ['gato', 'perro', 'raton']  
  
# O pueden ser de distintos tipos de dato.  
lista_heterogenea = [1, "gato", 3.0, False]
```

Los elementos de la lista se pueden modificar, ya sea cambiando los valores de éstos, agregando o eliminando elementos. Por eso es que se dice que las listas son **mutables**.

Definir una lista

Para crear una lista utilizaremos la siguiente sintaxis:

```
a = [1, 2, 3, 4]
```

Elementos de la lista

Son todo lo que esté dentro de los corchetes [], separados por coma.

Mostrar una lista

Para mostrar una lista podemos ocupar **print** o **llamar a su objeto contenedor**.

```
print(a) # mostramos los valores de la lista a definida arriba  
print([1, 2, 'hola', 4]) # mostramos directo una lista
```

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

```
[1, 2, 'hola', 4]
```


Índice

Cada elemento en la lista tiene una posición específica, la que se conoce como índice, el que permite acceder al elemento que está dentro de la lista.

En Python los índices parten en **cero** y van hasta **$n - 1$** , donde n es la cantidad de elementos en la lista.



Índice

Ejemplo

En una lista que contiene 4 elementos:

- el primer elemento está en la posición cero
- el último en la posición 3

Para acceder al elemento de una posición específica de la lista, se debe escribir el nombre de la lista, y entre paréntesis de corchetes, el índice de la posición.

```
colores = ["verde", "rojo", "rosa", "azul"]  
print(colores[0])  
print(colores[1])  
print(colores[3])
```

```
verde  
rojo  
azul
```

Índice

más allá de los límites

En caso de que el índice sea mayor o igual a la cantidad de elementos en la lista, Python arrojará un **IndexError: list index out of range**, que indica que el índice se posiciona fuera del rango de la lista.

```
colores[8]
```

```
-----  
  
IndexError                                Traceback (most recent call last)  
<ipython-input-4-cf0cdf800f9c> in <module>()  
----> 1 colores[8]  
IndexError: list index out of range
```

Índice

negativo

Los índices también se pueden utilizar con números negativos y de esta forma referirse a los elementos desde el último al primero.

En Python, el índice del último elemento también se puede denotar con **-1** y el primer elemento corresponde al elemento **-n**.

```
a = [1, 2, 3, 4, 5]  
a[-1]
```

5

ARGV

Hasta el momento vimos que **input()** es una instrucción que nos permite ingresar datos por parte del usuario para poder ser utilizados dentro de nuestros programas.

Ahora presentaremos **argv**, los cuales son **argumentos** que se pueden ingresar desde la terminal al momento de ejecutar nuestro programa.

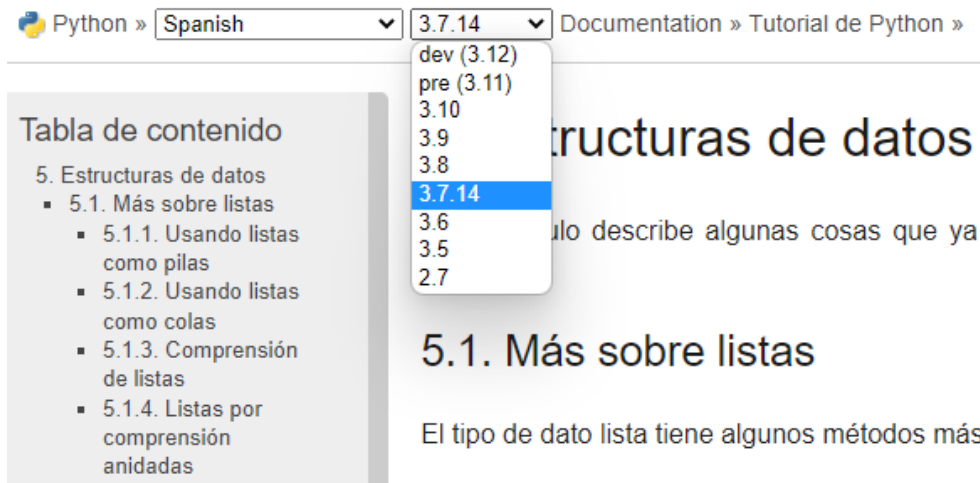
El inconveniente que tiene `sys.argv` es que es poco intuitivo para el usuario, ya que si no está familiarizado con el código, no podrá entender cada argumento inicialmente. Pero es fundamental cuando existen procesos automáticos agendados por el computador el cual ejecuta un archivo `.py` para realizar procesos claves para las empresas.



Leyendo la documentación de listas

La información oficial de Python sobre las listas (y de manera más general de estructuras de datos) se encuentra en docs.python.org.

Importante: debes asegurarte que la versión que se esté consultando sea la misma con la que se está trabajando.



The screenshot shows the Python documentation website interface. At the top, there are navigation links: "Python »", a language dropdown set to "Spanish", a version dropdown set to "3.7.14", and "Documentation » Tutorial de Python »". The version dropdown menu is open, showing a list of versions: "dev (3.12)", "pre (3.11)", "3.10", "3.9", "3.8", "3.7.14" (highlighted in blue), "3.6", "3.5", and "2.7". On the left, a "Tabla de contenido" (Table of contents) lists sections under "5. Estructuras de datos", including "5.1. Más sobre listas" and its sub-sections. The main content area shows the title "Estructuras de datos" and the start of section "5.1. Más sobre listas", which begins with the text "El tipo de dato lista tiene algunos métodos más."

`/* Métodos */`

Métodos aplicables a listas

Cuando definimos una lista en Python, el intérprete infiere cuál es la **mejor representación de la expresión**. En base a este punto, también le delega a la expresión una **serie de acciones** que pueda realizar, las que se conocen como métodos de lista.

Cuando generamos una nueva lista y la asignamos a una variable, su generación viene con una serie de atributos y métodos asociados.

La forma tradicional para llamar a un método fue vista anteriormente y se llama notación de punto: **objeto.método(argumentos)**.

Métodos aplicables a listas

Ejemplo

Al generar un objeto llamado **lista_de_numeros** que se compone de los números del 1 al 10, donde Python le concederá una serie de acciones dado que infiere que su mejor representación es una lista, donde se puede ver cuáles son todas las posibles acciones utilizando el atributo **__dir__()**.

```
lista_de_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
print(lista_de_numeros.__dir__())
```

```
['__repr__', '__hash__', '__getattribute__', '__lt__', '__le__', '__eq__', '__ne__', '__gt__', '__ge__', '__iter__',  
'__init__', '__len__', '__getitem__', '__setitem__', '__delitem__', '__add__', '__mul__', '__rmul__', '__contains__',  
'__iadd__', '__imul__', '__new__', '__reversed__', '__sizeof__', 'clear', 'copy', 'append', 'insert', 'extend', 'pop',  
'remove', 'index', 'count', 'reverse', 'sort', '__doc__', '__str__', '__setattr__', '__delattr__', '__reduce_ex__',  
'__reduce__', '__subclasshook__', '__init_subclass__', '__format__', '__dir__', '__class__']
```

Aquellos elementos que están envueltos por `__` se conocen como `magic built-in` o `dunders`, y buscan generar flexibilizaciones en el comportamiento de las clases.



Método `append(x)`

Agrega elementos al final de la lista

Ejemplo:

Si queremos agregar el celeste a nuestra lista de colores, se hace de la siguiente forma:

```
colores = ['verde', 'rojo', 'rosa', 'azul']  
colores.append("celeste")
```

```
# pedimos una representación actualizada de la lista  
print(colores)
```

```
['verde', 'rojo', 'rosa', 'azul', 'celeste']
```

Cabe destacar que no es necesario declarar de forma explícita la asignación al objeto, dado que `append` es una función inherente al objeto, opera y lo actualiza dentro de ella.

Método insert(i,x)

Agrega el elemento x en la posición i específica

Ejemplo:

Insertemos el número 20 donde corresponde, la posición 11, ya que partimos desde cero.

```
lista_de_numeros.insert(11, 20)
```

```
print(lista_de_numeros)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 20, 13]
```

Método pop()

Sacar el último elemento dentro de una lista

También es posible pasar como argumento de la función una posición específica, de esta forma, se buscará dentro de la lista el elemento en esa posición, y este será eliminado y entregado.

```
colores = ['verde', 'rojo', 'rosa', 'azul']  
colores.pop(3)
```

```
'azul'
```

```
color = colores.pop(0)  
print(color)
```

```
'verde'
```

Método remove()

Remover un elemento específico

En caso que el elemento no esté presente en la lista, Python arrojará un error **ValueError**.

```
colores = ['rojo', 'rosa']  
colores.remove("rojo")  
print(colores)
```

```
'rosa'
```

```
colores.remove("azul")
```

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

Método reverse()

Invertir el orden de los elementos de una lista

```
numeros = [100, 20, 70, 500]
animales = ["perro", "gato", "hurón", "erizo"]
numeros.reverse()
animales.reverse()
print(numeros)
print(animales)
```

```
[500, 70, 20, 100]
['erizo', 'hurón', 'gato', 'perro']
```


Método sort()

Ordenar los elementos de forma ascendente

En caso de que se trate de strings, se ordenan de forma ascendente en el abecedario.

```
animales.sort()  
numeros.sort()  
print(animales)  
print(numeros)
```

```
['erizo', 'gato', 'hurón', 'perro']  
[20, 70, 100, 500]
```

Al trabajar con listas en Python todas estas funciones se realizaron sobre la misma lista. Por tanto, si no se guarda la lista original en algún objeto separado, esta información se pierde.



Método sorted()

Obtiene los mismos resultados del método sort, y en el caso de querer ordenar de manera descendente se utiliza el argumento reverse = True

```
# ordenamiento ascendente  
sorted([3,6,7,4,1])
```

```
[1, 3, 4, 6, 7]
```

```
# ordenamiento descendente  
sorted([3,6,7,4,1], reverse = True)
```

```
[7, 6, 4, 3, 1]
```

Método index()

Retorna el índice (de cero al que corresponda dependiendo del largo de la lista)

```
print(animales.index('gato'))
```

2

```
print(numeros.index(500))
```

0

/* Operaciones */

Concatenación de listas

```
# definamos dos listas de animales
animales = ['Gato', 'Perro', 'Tortuga']
animales_2 = ['Hurón', 'Hamster', 'Erizo de Tierra']

# Si las concatenamos, podremos obtener una lista de mascotas
mascotas = animales + animales_2
# Veamos algunas características
print(animales)
print(len(animales))
print(animales_2)
print(len(animales_2))
print(mascotas)
print(len(mascotas))
```

```
['Gato', 'Perro', 'Tortuga']
3
['Hurón', 'Hamster', 'Erizo de Tierra']
3
['Gato', 'Perro', 'Tortuga', 'Hurón', 'Hamster', 'Erizo de Tierra']
6
```

Repitiendo listas

```
animales_actualizados = animales * 4

mascotas = animales_actualizados + animales_2

# Veamos algunas características
print(animales_actualizados)
print(len(animales_actualizados))
print(animales_2)
print(len(animales_2))
print(mascotas)
print(len(mascotas))
```

```
['Gato', 'Perro', 'Tortuga', 'Gato', 'Perro', 'Tortuga', 'Gato', 'Perro', 'Tortuga',
 'Gato', 'Perro', 'Tortuga']
12
['Hurón', 'Hamster', 'Erizo de Tierra']
3
['Gato', 'Perro', 'Tortuga', 'Gato', 'Perro', 'Tortuga', 'Gato', 'Perro', 'Tortuga',
 'Gato', 'Perro', 'Tortuga', 'Hurón', 'Hamster', 'Erizo de Tierra']
15
```

¿Cuáles son los métodos que acabamos de aprender?



¿Cómo podemos ordenar
esta lista en orden
descendente (de la Z a la A)?





Próxima sesión...

- *Utiliza operaciones de creación y acceso a los elementos de una estructura de datos acorde al lenguaje Python para resolver un problema.*
- *Utiliza operaciones para la agregación, modificación y eliminación de elementos de una estructura de datos acorde al lenguaje Python para resolver un problema.*

{desafío}
latam_

*Academia de
talentos digitales*

