Fundamentos de Programación y Estructura de Datos

Sesión Conceptual 2

{desafío} latam_

- Objetivo de la sesión
- Activación de conceptos clave

- Conceptualización
- Ejercicios
- Quiz

- Cierre

Inicio

10 minutos

{desafío} latam_

¿Qué aprenderás?

A reconocer los tipos de datos usados en el entorno Python y su uso para la construcción de un programa, las diferencia entre ellos y sus usos más típicos.

{desafío} latam_

Desarrollo

70 minutos

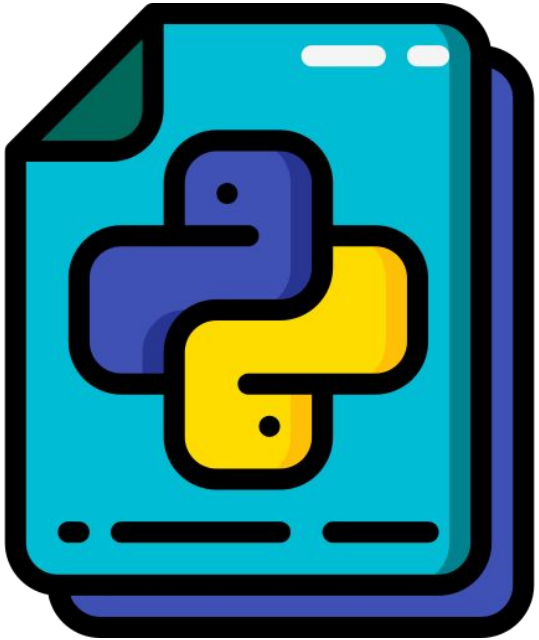
{desafío} latam_

/* Introducción a Estructuras de Datos */

{desafío}
latam_

Aspectos generales

Dentro de las estructuras nativas de Python están las Tuplas, los Sets, las Listas y los Diccionarios. Y en este capítulo nos enfocaremos principalmente en las Listas que son una de las estructuras de datos más importantes en Python.



{desafío} latam_

Introducción a las listas

Las listas son contenedores que permiten almacenar múltiples datos. En lugar de guardar un solo número o un solo string en una variable, ahora podemos almacenar en un **objeto** varios valores simultáneamente, tal como se observa a continuación:

```
# 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]
```

Definir y mostrar una lista:
Se definen con los paréntesis de corchete `[]`; Todo lo que esté dentro de los `[]`, separados por coma, son los elementos de la lista.

```
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]
```

{desafío} latam_

ÍNDICES

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.

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

verde
rojo
azul
```

ÍNDICES

Índices 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.

Índices negativos

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

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

5
```

Ejercicio de aplicación 1

En base a la lista `a` que se define a continuación, ¿qué se muestra en cada petición con los índices?

```
a = [1, 2, 3, 4, 'hola', 8]
```

- `a[0]`.
- `a[7]`.
- `a[a[0]]`.
- `a[4]`.
- `a[-1]`.

{desafío} latam_

ARGV

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

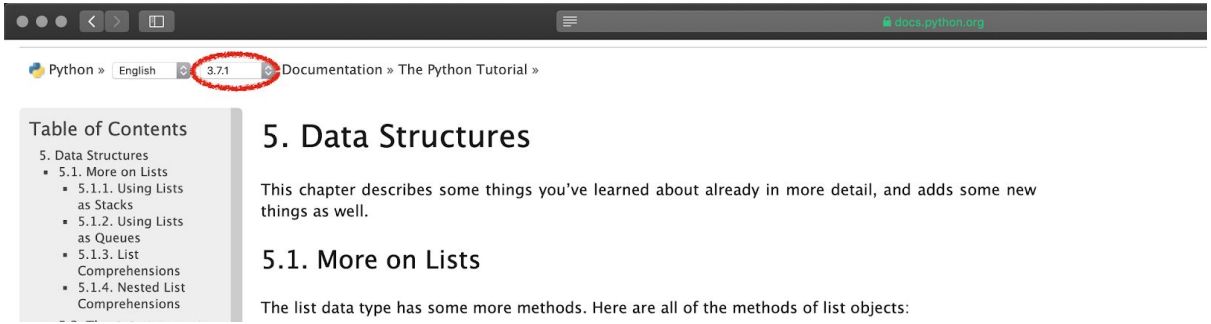
Ejercicio guiado

"Mini presentador"

{desafío} latam_

Operaciones y funciones básicas en una lista

Las listas de Python tienen una serie de funciones que permiten realizar operaciones básicas, entre ellas, ordenar los elementos de una lista, saber cuántas veces se encuentra un elemento allí, además de, borrar y agregar elementos. Por lo mismo, es muy importante saber leer la documentación asociada a éstas.



{desafío} latam_

Métodos aplicables a las 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**.

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

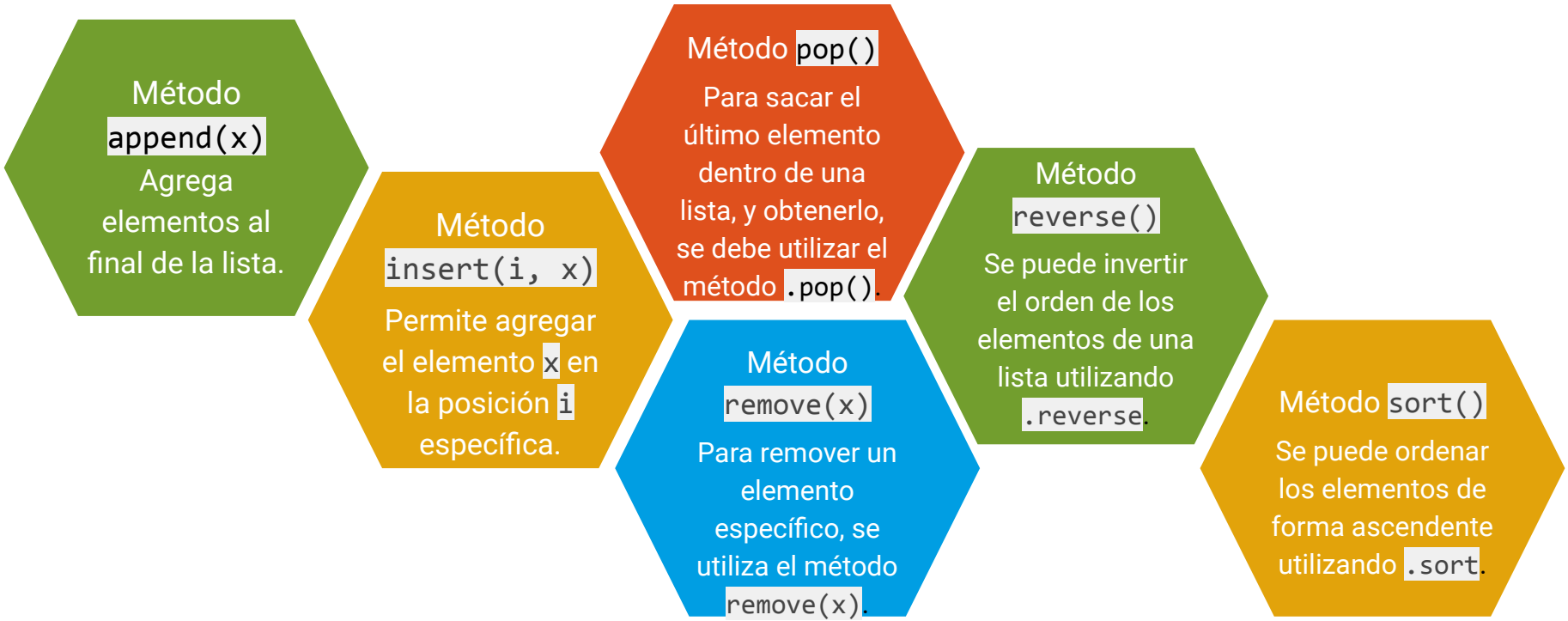
Al generar un objeto llamado `lista_de_numeros` que se compone de los números del 1 al 10, donde además se puede ver cuáles son todas las posibles acciones utilizando el atributo `__dir__()`.

```
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__']
```

Métodos aplicables a las listas



{desafío} latam_

Ejercicio de aplicación 2: STRINGS, es una lista, pero no tanto

En base al `string texto` que se define a continuación ¿Qué se muestra en cada petición con los índices?

```
texto = "este es un string"
```

- `texto[:4]`
- `texto[-5:]`
- `texto[5:7]`
- `texto[::-1]`
- `texto.append(" en Python")`
- `texto + " en Python"`

{desafío} latam_

Operaciones: Concatenación de listas

Es importante entender una lista como “una estructura de tipo lista”, por lo tanto, hay algunos comportamientos que podrían sorprendernos, ya que no es lo que esperaríamos, donde un ejemplo de esto es la operación `+`.

```
# 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
```

Operaciones: Repitiendo listas

Al utilizar el operador `*` con un string, se genera el mismo resultado que con los strings.

```
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
```

Ejercicio de aplicación 3: Trabajando con listas

En base a las listas que se definen a continuación, calcule que la salida de cada petición, además indique qué tipo de dato o estructura de dato es cada salida.

```
animales_domesticos = ['perro','gato','canario']
animales_salvajes = ['león','elefante','jirafa']
```

- `animales_domesticos + animales_salvajes`
- `animales_domesticos.append(animales_salvajes)`
- `animales_domesticos[3][2]`
- `3 * animales_domesticos[3]`
- `3 * animales_domesticos[3][2]`

{desafío} latam_

Quiz

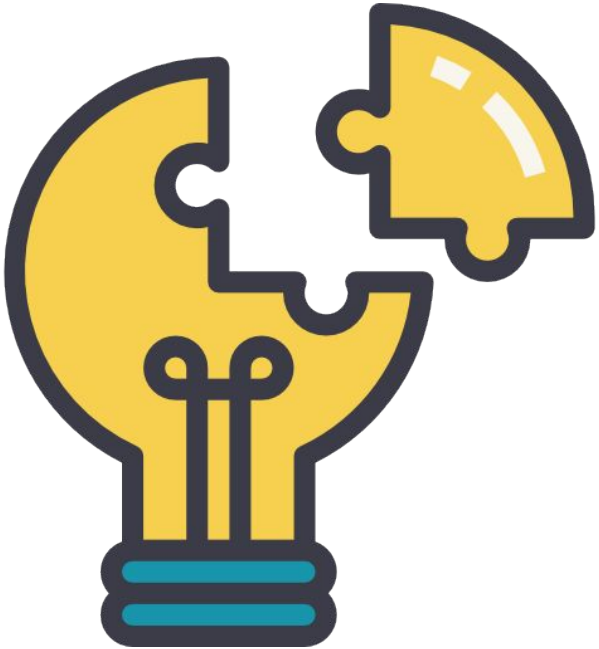
{desafío}
latam_

/* Introducción a los diccionarios */

{desafío}
latam_

Diccionarios

Los diccionarios son una estructura de datos compuesta por pares de `clave:valor`, donde cada clave se asocia con un elemento del diccionario, y como toda estructura de datos, permiten almacenar una gran cantidad de estos en una sola variable.



Lista versus Diccionario

Para acceder a los elementos de una lista, se hace a través del índice asociado al elemento, mientras que en un diccionario, se hace por medio de la clave.

≠

En una lista, los índices se generan automáticamente para cada elemento, mientras que las claves de un diccionario no.

≠

La lista es un elemento ordenado donde el orden lo entrega el índice, en cambio, en un diccionario es un elemento no ordenado.

Diccionarios: ¿Para qué sirven?

Principalmente cuando se requiere identificar rápidamente un elemento a través de un valor único, la clave y para generar conversiones.

Diccionarios

Crear un diccionario

En Python, un diccionario (vacío) se define con llaves: `{}`. Cada par de clave y valor se asocia mediante `:`, en la forma `clave: valor`.

```
notas = {"Camila": 7, "Antonio": 5,
"Felipe": 6, "Antonia": 7}
```

Acceder a un elemento dentro de un diccionario

Se accede a un elemento específico utilizando la **clave** de ese **valor**, donde necesariamente se debe utilizar la misma clave en la que está almacenado el elemento.

```
notas["Felipe"] # 6
notas["felipe"] # KeyError: 'felipe'
```

Diccionarios

La clave tiene que ser única

En un diccionario, solo puede haber un valor asociado a una clave.

```
duplicados = {"clave": 1, "clave": 2}
print(duplicados) # {"clave": 2}
```

Agregando un elemento a un diccionario

Para agregar un elemento a un diccionario, hace falta especificar una clave nueva, de forma que el nuevo valor ingresado sea identificado con dicha clave.

```
diccionario["llave 2"] = 9
print(diccionario) # {"llave 1": 5, "llave 2": 9}
```

Diccionarios

Cambiando un elemento dentro de un diccionario

Podemos actualizar el valor de un elemento, redefiniendo el valor asociado a la clave.

```
diccionario = {"llave 1": 5, "llave 2": 7}
diccionario["llave 2"] = 9
print(diccionario) # {"llave 1": 5, "llave 2": 9}
```

Eliminar elementos de un diccionario

Podemos eliminar una llave de un diccionario, junto a su valor, de dos formas: usando el método `pop` del diccionario o utilizando `del`.

Diccionarios

Unir diccionarios

Se puede lograr a través de la siguiente expresión.

```
diccionario_a = {"nombre": "Alejandra",
"apellido": "López", "edad": 33, "altura":
1.55}
diccionario_b = { "mascota":"miti",
"ejercicio":"bicicleta"}

# Union de diccionario_a y diccionario_b
diccionario_a.update(diccionario_b)

# Notar que la unión queda en el primer
diccionario
print(diccionario_a)
```

Cuidado con las colisiones

Cuando ambos diccionarios tienen una clave en común, el valor del segundo diccionario sobreescribe al del primero.

Otros Métodos para diccionarios

Otras Estructuras de Datos

Tuplas: Una tupla es un par ordenado inmutable, es decir, no se pueden modificar partes de ella.

```
# definición una tupla
tupla_ej = ("Abril", 2021)
type(tupla_ej)

tuple
```

Sets: Es una estructura de datos que permite trabajar similar a lo que es la teoría de conjuntos.

```
# definición de un set
# se pueden ver que existen valores repetidos
muchos_animales = {'Gato', 'Perro', 'Tortuga',
                   'Gato', 'Perro', 'Tortuga',
                   'Gato', 'Perro', 'Tortuga',
                   'Gato', 'Perro', 'Tortuga',
                    'Hurón', 'Hamster', 'Erizo de
Tierra'}
```

Convertir estructuras

- Convertir un diccionario en una lista: para lograr esto, se debe utilizar la función `items()`. Cada par (clave, valor) será una tupla.
- Convertir una lista en un diccionario: para invertir la transformación se utiliza la función `dict`.

Ejercicio guiado

"Efemérides"

{desafío} latam_

Quiz

{desafío}
latam_

Cierre

10 minutos

{desafío} latam_

Preguntas de cierre

1. ¿Qué son las estructuras de Datos en Python?
2. ¿Cuáles son las estructuras de Datos principales en Python?
3. ¿Qué son los métodos y qué elementos los poseen?
4. ¿Qué ventajas me entregan las estructuras de datos por sobre las variables comunes y corrientes?

{desafío} latam_

Academia de talentos digitales