Estructuras de datos y funciones

Métodos con listas

{desafío} latam_

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

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Aplica métodos para la manipulación de listas*

{desafío}
latam_

¿Qué entendemos por estructura de datos?

{desafío} latam_

/* 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 un 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'
```

{desafío} latam_

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
```

{desafío} latam_

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
```

max, min, sum

Funciones de uso intuitivo

```
numeros = [5, 2, 9, 1, 7]

maximo = max(numeros)
minimo = min(numeros)
suma = sum(numeros)

print("El máximo valor en la lista es:" , maximo)
print("El mínimo valor en la lista es:" , minimo)
print("La suma de todos los elementos en la lista es:" , suma)
```

/* Operaciones */

Concatenación de listas

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

mascotas = animales + animales_2

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...

- *Aplica métodos para la manipulación de diccionarios*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales