Clase 10 - Listas

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

03-10-2019

Resumen de la clase

- **Comparación de strings:** Para comparar dos *strings* distintos, se ocupa el operador “<”,“>”,”==“ ó “!=“.
  Por ejemplo:

```
s1 = “a”
s2 = “b”
print(s1 < s2) #True
```

- Python tiene muchas funciones para operar con *strings*, a continuación veremos algunas de ellas.

Resumen de la clase

Asumamos que `s`, `x` e `y` son *strings* distintos.

- **`s.count(x):`** Retorna la cantidad de ocurrencias de `x` en `s`.
- **`s.find(x):`** Retorna la primera ocurrencia de `x` en `s`.
- **`s.replace(x,y):`** Reemplaza `x` por `y` en `s`.
- **`s.upper():`** Transforma todos los caracteres de `s` a mayúsculas.
- **`s.lower():`** Transforma todos los caracteres de `s` a mayúsculas.
- **`x in s:`** Devuelve `True` o `False` dependiendo si `x` se encuentra en `s`.
- **`s.strip():`** Elimina todos los espacios o caracteres especiales (como "\n") al principio y final de `s`.
- **`s.isdigit():`** Devuelve True si el string son solo dígitos y False en caso contrario. No considera el separador decimal.

Contenidos

1. Definición de listas
2. Obtener elementos
3. Editar elementos
4. Agregar elementos
5. Cantidad de elementos de una lista
6. `for` sobre lista

Listas

- Como vimos anteriormente, una variable de tipo texto, o *string* en realidad es una secuencia de caracteres.

Listas

- Como vimos anteriormente, una variable de tipo texto, o *string* en realidad es una secuencia de caracteres.
- Estos caracteres ocupan cierta posición. Esto nos permitía poder extraer caracteres de forma individual, solo al saber su posición.

Listas

- Como vimos anteriormente, una variable de tipo texto, o *string* en realidad es una secuencia de caracteres.
- Estos caracteres ocupan cierta posición. Esto nos permitía poder extraer caracteres de forma individual, solo al saber su posición.
- Esto no es solo aplicable para *strings*, sino que es abstraíble a cualquier tipo de datos. Es decir, es posible crear variables que almacenen **más de un elemento**. Estas variables no solo almacenarán más de un elemento, sino que le asignarán una posición específica a estos elementos. Todo esto nos permitirá hacer operaciones más complejas sobre ellas. **Las listas nos permiten almacenar más de un dato en una sola variable**

Listas

- Como vimos anteriormente, una variable de tipo texto, o *string* en realidad es una secuencia de caracteres.
- Estos caracteres ocupan cierta posición. Esto nos permitía poder extraer caracteres de forma individual, solo al saber su posición.
- Esto no es solo aplicable para *strings*, sino que es abstraíble a cualquier tipo de datos. Es decir, es posible crear variables que almacenen **más de un elemento**. Estas variables no solo almacenarán más de un elemento, sino que le asignarán una posición específica a estos elementos. Todo esto nos permitirá hacer operaciones más complejas sobre ellas. **Las listas nos permiten almacenar más de un dato en una sola variable.**
- Las variables anteriormente nombradas se denominan **listas**.

Listas

- Las listas son variables en Python que guardan una secuencia de valores. Estos valores se denominan *elementos*.
- La forma más simple de crear una lista es de la siguiente manera:

```
lista1 = [1,2,3]
lista2 = ["t","e","x","t","o"]
```

Listas

```
lista1 = [1,2,3]
```

- En el caso de la variable `lista1`, creamos una lista que tiene tres elementos de tipo *int*. El 1, que es el primer elemento, está en la posición 0. El 2, que es el segundo elemento, está en la posición 1, y el 3, que es el tercer elemento, está en la posición 2.

Listas

- Asimismo, podemos crear listas con elementos de distinto tipo. Por ejemplo:

```
variable_tipo_lista = ["texto",100,-9.346,0,0.53,True]
```

Listas

```
lista2 = ["t","e","x","t","o"]
```

- En el caso de la variable `lista1`, creamos una lista que tiene 5 elementos de tipo *string*

¿A qué se parece esta lista?

Listas

- Recordemos de *strings* que podemos obtener el carácter en cualquier posición. Esto se hacía de la siguiente manera:

```
s = "texto"
print(s[0]) #h
```

- En este caso, lo que se imprime en consola es la letra “h”. Esto se debe a que la “h” está en la posición 0 del texto que tiene la variable s.
- ¿Cómo podríamos hacer lo mismo en listas? En realidad, un *string* no es más que una lista de caracteres. Por lo tanto, muchas de las características que vimos en *strings* anteriormente aplican a las listas

Obtener elementos

Para poder obtener algún elemento de una posición determinada un una lista, se hace de la siguiente forma:

```
elemento_en_posición_x = variable_de_tipo_lista[x]
```

Donde elemento_en_posición_x es la variable que almacenará el elemento que está en la posición x de la lista que está almacenada en la variable_de_tipo_lista.

Obtener elementos

Veamos un ejemplo. Queremos obtener el tercer elemento de una lista:

```
lista3 = ["administración","ventas","operaciones","post-venta"]
elemento = lista3[2]
print(elemento) #operaciones
```

En este caso, como extraemos el elemento en la posición 2 y lo almacenamos en la variable elemento, al imprimir en consola la variable elemento obtenemos el *string* "operaciones"

Si no queda claro, veamos esta imagen que representa a la lista "lista1".

"administración"	"ventas"	"operaciones"	"post-venta"
0	1	2	3

Slice

Al igual que en *strings*, también podemos ocupar el concepto de *slice* en listas. En este caso, esto nos serviría para obtener una nueva lista desde nuestra lista original. Esta nueva lista tendrá ciertos elementos que dependerá de cómo hicimos el *slice.* Se hace de la siguiente manera:

```
l[i:j]
```

Donde obtenemos los elementos desde la posición `i` hasta `j-1`. Por ejemplo, queremos obtener los dos primeros elementos de la misma lista del ejemplo anterior.

```
lista3 = ["administración","ventas","operaciones","post-venta"]
elemento = lista3[0:2]
print(elemento)
```

```
['administración', 'ventas']
```

Editar elementos

- Así como podemos obtener ciertos elementos de una lista, también podemos editarlos. Digamos que tenemos la misma lista del ejemplo anterior. No obstante, el departamento de “post-venta” cambió el nombre a “servicio al cliente” ¿Cómo podríamos editarlo?
- Veamos un ejemplo:

```
#ejemplo 1 edición elemento de una lista
lista3 = ["administración","ventas","operaciones","post-venta"]
lista3[2] = "servicio al cliente"
print(lista3)
```

```
['administración', 'ventas', 'servicio al cliente', 'post-venta']
```

Editar elementos

- También podemos editar varios elementos simultáneamente, por ejemplo:

```
#ejemplo 1 edición slice
lista3 = ["administración","ventas","operaciones","post-venta"]
lista3[1:3] = ["servicio al cliente","finanzas"]
print(lista3)
```

```
['administración', 'servicio al cliente', 'finanzas', 'post-venta']
```

Añadir elementos

- ¿Cómo podríamos agregar nuevos elementos a nuestra lista? Para poder agregar elementos a una lista, ocupamos la función `append()`. Digamos que tenemos a una lista `l`. Si queremos agregar un elemento `x`, entonces escribimos:

```
l.append(x)
```

- Veamos cómo un ejemplo:

```
lista1 = ["administración","ventas","operaciones","servicio al cliente"]
print(lista1)
lista1.append("marketing")
print(lista1)
```

```
['administración', 'ventas', 'operaciones', 'servicio al cliente', 'marketing']
```

Largo de una lista

- ¿Cómo podríamos saber la cantidad de elementos de una lista? Igual que en un *string*, mediante la función `len()`.
- Veamos un ejemplo:

```
#ejemplo 1 len()
lista1 = ["administración","ventas","operaciones","servicio al cliente"]
print(len(lista1))
```

```
4
```

`for` sobre una lista

- Al igual que en un string, podemos iterar sobre los elementos de una lista. Si tenemos una lista de nombre `l`:

```
for i in l:
    i
```

for sobre una lista

- En este caso, `i` representa a cada uno de los elementos de la lista `l`. Veamos un ejemplo:

```
#ejemplo 1 for
lista1 = ["administración","ventas","operaciones","servicio al cliente"]

for elemento in lista1:
  print(elemento)
```

for sobre una lista

- También podemos hacer un for sobre una lista en base a las posiciones de los elementos. Veamos un ejemplo:

```
#ejemplo 2 for
lista1 = ["administración","ventas","operaciones","servicio al cliente"]

for i in range(len(lista1)):
  print(lista1[i])
```

Ejercicio

- Implemente un diccionario en Python. El diccionario es un directorio donde se crean relaciones entre llaves y valores. Por ejemplo, en un diccionario español-inglés la “llave” es la palabra en español y el valor es la palabra en inglés.
- Su diccionario debe ser capaz de recibir un string en el formato “llave,valor”. En los elementos de índice par se almacenarán las llaves, y en los elementos de índice impar se almacenará el valor.
- Si se recibe una llave ya existente, se debe cambiar el valor correspondiente a esa llave.
- Su programa debe ser capaz de recibir pares “llave,valor” de forma indefinida hasta que el usuario indique lo contrario.
- Puede asumir que el usuario siempre ingresará el input correcto.

Bibliografía

- http://runest.ing.puc.cl/list.html
- A. B. Downey. Think Python: How to think like a computer scientist. Green Tea Press, 2013 -> Capítulo 10

Links

- https://repl.it/@FelipeLopez/IIC1103Listas que contiene todos los ejemplos de la clase.

Resumen de la clase

- Las listas son variables en Python que guardan una secuencia de valores. Estos valores se denominan *elementos*.
- Por ejemplo:

```
lista1 = [1,2,3]
```

- Podemos crear listas vacías también, de la siguiente forma:

```
lista2 = []
```

- Para poder obtener algún elemento de una posición determinada un una lista, se hace de la siguiente forma:

```
elemento_en_posición_x = variable_de_tipo_lista[x]
```

Donde elemento_en_posición_x es la variable que almacenará el elemento que está en la posición x de la lista que está almacenada en la variable_de_tipo_lista.

Resumen de la clase

- **Slice:** Es la acción de obtener una nueva lista desde otra lista. Esta nueva lista tendrá ciertos elementos que dependerá de cómo hicimos el *slice.* Si `l` es una lista, se hace de la siguiente manera:

```
l[i:j]
```

Donde obtenemos los elementos desde la posición `i` hasta `j-1`.

- **Edición de elementos:** Podemos editar elementos de una lista. Si `l` es una lista y k es una posición de la lista:

```
l[k]=nuevo elemento
```

Donde la información que estaba en la posición k se sobreescribió con el nuevo elemento. Recordemos que este nuevo elemento puede ser de cualquier tipo.

Resumen de la clase

- **Agregar elementos:** Se pueden agregar nuevos elementos a una lista. Si `l` es una lista, se hace de la siguiente manera:

```
l.append(x)
```

Donde se agrega el elemento x a la lista.

- **Cantidad de elementos de una lista:** Se puede obtener la cantidad de elementos de una lista. Si `l` es una lista, se hace de la siguiente manera:

```
len(l)
```

- **`for` sobre una lista:** Podemos "recorrer" todos los elementos de una lista mediante un `for`. Si `l` es una lista, se hace de la siguiente manera:

```
for i in s1:
    i #i representa a cada uno
      #de los elementos de l.
```

- **`for` sobre una lista:** Podemos "recorrer" todas las posiciones de una lista mediante un `for`. Si tenemos una lista de nombre lista, se hace de la siguiente manera:

```
for i in range(len(lista)):
    lista[i]
```

Clase 10 - Listas

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

03-10-2019