RELATORIA II

Programación II

Jhanth Carlo Castillo Pérez

La correspondencia relacionada a esta relatoría es dirigida a Cristian Pachón
Facultad de Ciencias Exactas y Naturales
Universidad Nacional de Colombia Sede Manizales

jhcastillop@unal.edu.co

Jueves, 23 de marzo del 2023

CICLO FOR

Cuando se conoce el número de ciclos a ejecutar o la secuencia a recorrer, el ciclo For es el adecuado para realizar iteraciones

Para aclarar, los ciclos For (o ciclos *para*) son una estructura de control cíclica que nos permiten ejecutar una o varias líneas de código de forma iterativa (o repetitiva), pero teniendo cierto control y conocimiento sobre las iteraciones. En el dicho ciclo es necesario tener un valor inicial y un valor final, y opcionalmente podemos hacer uso del tamaño del "paso" entre cada "giro" o iteración del ciclo.

 # Ejemplo con un argumento

```
for i in range(5):
    print(i, end=", ") # prints: 0, 1, 2, 3, 4,
```

 # Ejemplo con dos argumentos

```
for i in range(-1, 5):
    print(i, end=", ") # prints: -1, 0, 1, 2, 3, 4,
```

 # Ejemplo con tres argumentos

```
for i in range(-1, 5, 2):
    print(i, end=", ") # prints: -1, 1, 3,
```

 # Ejercicio realizado sobre la serie Fibonacci

```
fib1, fib2 = 1, 1
print(fib1, fib2, sep = "-", end="-")
for turno in range(98):
    fib1, fib2 = fib2, fib1 + fib2
    print(fib2, end = "-") #prints: los primeros 100 números de la serie Fibonacci.
```

Martes, 28 de marzo del 2023

MÉTODOS

Los métodos son acciones o funciones que puede realizar un objeto. Para clasificar los métodos, se deben identificar los objetos en Python: strings lista, diccionarios y tuplas.

Métodos de strings o cadenas

Las cadenas o strings son un tipo inmutable que permite almacenar secuencias de caracteres. Para crear una, es necesario incluir el texto entre comillas dobles ". Puedes obtener más ayuda con el comando help(str).

Las cadenas no están limitadas en tamaño, por lo que el único límite es la memoria de tu ordenador. Una cadena puede estar también vacía.

Descripción de métodos de strings

- capitalize() se aplica sobre una cadena y la devuelve con su primera letra en mayúscula.
- upper() convierte todos los caracteres alfabéticos en mayúsculas.
- lower() convierte todos los caracteres alfabéticos en minúscula.
- swapcase() convierte los caracteres alfabéticos con mayúsculas en minúsculas y viceversa.
- upper() convierte todos los caracteres alfabéticos en mayúsculas.
- El método count() permite contar las veces que otra cadena se encuentra dentro de la primera. Permite también dos parámetros opcionales que indican donde empezar y acabar de buscar.
- isalnum() devuelve True si la cadena está formada únicamente por caracteres alfanuméricos, False de lo contrario. Caracteres como @ o & no son alfanumericos.
- isalpha() devuelve True si todos los caracteres son alfabéticos, False de lo contrario.
- strip() elimina a la izquierda y derecha el carácter que se le introduce. Si se llama sin parámetros elimina los espacios. Muy útil para limpiar cadenas.

Clasificación de métodos de strings, por utilidad

- Formateo: capitalize(), upper(), lower(), title(), center(spaces), strip()
- Operaciones: count(subcadena), find(subcadena), replace(old, new)
- Verificación: isalnum(), isalpha(), isdigit, isdecimal
- Indexado: [índice]
- Slicing: [indice1: indice2: salto]

 # Ejemplo de método de strings

```
cadena = "hola mundo cruel"

print("mayusculas =>", cadena.upper())

print("capitalizar =>", cadena.capitalize())

print("conteo =>", cadena.count("o"))
```

Métodos de listas

Las listas son un tipo de dato que permite almacenar datos de cualquier tipo. Son mutables y dinámicas, lo cual es la principal diferencia con los sets y las tuplas; son uno de los tipos o estructuras de datos más versátiles del lenguaje, ya que permiten almacenar un conjunto arbitrario de datos. Es decir, podemos guardar en ellas prácticamente lo que sea.

Descripción de métodos de listas

- append() añade un elemento al final de la lista.
- extend() permite añadir una lista a la lista inicial.
- insert() añade un elemento en una posición o índice determinado.
- remove() recibe como argumento un objeto y lo borra de la lista.
- pop() elimina por defecto el último elemento de la lista, pero si se pasa como parámetro un índice permite borrar elementos diferentes al último.
- reverse() inverte el órden de la lista.
- sort() ordena los elementos de menos a mayor por defecto.
- index() recibe como parámetro un objeto y devuelve el índice de su primera aparición. Como hemos visto en otras ocasiones, el índice del primer elemento es el 0.

Clasificación de métodos de listas, por utilidad

- Operaciones: append(value), insert(index, value), pop(index), remove(value), count(value)
- Ordenado: sort(), reverse()
- Almacenamiento: clear(), copy()
- Indexado: [indice]
- Slicing: [indice1: indice2: salto]

 # Ejemplo de método de listas

```
lista = [10,20,30,40,50, 50]

lista.append(1000)

print("agregar elementos =>", lista)

lista.pop(2)  #Elimino 3er elemento (30)

print("eliminar elemento =>", lista)

print("contar un elemento =>", lista.count(50))

lista.insert(0, 300)

print("Agregar un elemento al principio =>", lista)
```

Métodos de tuplas

Las tuplas son un tipo o estructura de datos que permite almacenar datos de una manera muy parecida a las listas, con la salvedad de que son inmutables; son muy similares a las listas, pero con dos diferencias: no pueden ser modificadas una vez declaradas, y en vez de inicializarse con corchetes se hace con (). Dependiendo de lo que queramos hacer, las tuplas pueden ser más rápidas.

Descripción de métodos de tuplas

- count() cuenta el número de veces que el objeto pasado como parámetro se ha encontrado en la lista.
- index() busca el objeto que se le pasa como parámetro y devuelve el índice en el que se ha encontrado. En el caso de no encontrarse, se devuelve un ValueError. También acepta un segundo parámetro opcional, que indica a partir de que índice empezar a buscar el objeto.

Clasificación de métodos de tuplas, por utilidad

- Operaciones: index(value), count(value)
- Indexado: [indice]
- Slicing: [indice1: indice2: salto]

Métodos de diccionarios

Los diccionarios son una estructura de datos que permite almacenar su contenido en forma de llave y valor; es una colección de elementos, donde cada uno tiene una llave key y un valor value. Los diccionarios se pueden crear con paréntesis {} separando con una coma cada par key: value. En el siguiente ejemplo tenemos tres keys que son el nombre, la edad y el documento.

Descripción de métodos de diccionarios

- clear() elimina todo el contenido del diccionario.
- get() nos permite consultar el value para un key determinado. El segundo parámetro es opcional, y en el caso de proporcionarlo es el valor a devolver si no se encuentra la key.
- items() devuelve una lista con los keys y values del diccionario. Si se convierte en list se puede indexar como si de una lista normal se tratase, siendo los primeros elementos las key y los segundos los value.
- keys() devuelve una lista con todas las keys del diccionario.
- values() devuelve una lista con todos los values o valores del diccionario.
- pop() busca y elimina la key que se pasa como parámetro y devuelve su valor asociado. Daría un error si se intenta eliminar una key que no existe.
- popitem() elimina de manera aleatoria un elemento del diccionario.
- update() se llama sobre un diccionario y tiene como entrada otro diccionario. Los value son actualizados y si alguna key del nuevo diccionario no está, es añadida.

Clasificación de métodos de diccionarios, por utilidad

- Extraccion: items(), keys(), values(), get(key)
- Eliminar: pop(key)
- Almacenamiento: clear(), copy()
- Indexado: [key]

 # Ejemplo de método de diccionarios

diccEstudiantes = {"Juan Chica": 4.2, "Manuela Segura": 3.5, "Santiago Tabares": 3.0, "Cristian Pachon": 2.0}

print("indexado para diccEstudiantes => ", diccEstudiantes["Santiago Tabares"])

diccEstudiantes["Miguel Gomez"] = 4.5

diccEstudiantes["Juan Aya"] = 3.9

diccEstudiantes["Carlos Castillo"] = 4.0

Jueves, 13 de abril del 2023

FUNCIONES

Una función es un bloque de líneas de código o un conjunto de instrucciones cuya finalidad es realizar una tarea específica. Puede reutilizarse a voluntad para repetir dicha tarea.

Para definir una función nueva se utiliza la palabra reservada def, seguida del nombre que le quieres dar a la función. A continuación, añade los paréntesis con los parámetros de entrada de la función seguidos de dos puntos (:). Después, normalmente en otra línea de código, agrega el bloque de instrucciones que quieres que se ejecuten cada vez que se invoca a la función.

Dentro de este bloque de instrucciones (casi siempre al final y de forma opcional) se coloca una sentencia de retorno que no es más que el valor que deseas que devuelva la función: para esto usa la palabra reservada return. Una función en siempre devolverá un valor; es decir, si no se añade una sentencia de retorno (return), se devolverá el valor predeterminado None.

 # Definición de una función

def <NombreFuncion> (<parametro1>, <parametro2>, ...):

  <sentencias>

  return <valor de retorno>

 # Ejemplo de una función

def suma(a, b):

return a + b

result = suma(8,5) # result = 13