Programación I

Funciones de utilidad y Matrices

Profesor: Ignacio Lincolao Venegas

Ingeniería Civil Informática, UCT

Uso del in en listas y cadenas de caracteres

```
# Uso del in en un carácter
palabra = “casa”
caracter = “c”
if caracter in palabra:
    print(“El carácter c se encuentra en la palabra”)



# Uso del in en la listas
lista = [ “manzana”, “naranja”, “limón” ]
if “manzana” in lista:
    print(“La palabra manzana se encuentra en la lista”)



# uso del in en la lista con for
lista = [ “manzana”, “naranja”, “limón” ]
for x in lista:
    print(x)
> imprime manzana, naranja, limón
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Remover todos los valores específicos de una lista

Existen ocasiones donde es necesario remover un valor en específico de una lista. La función remove() permite remover específico de la lista,pero ¿Qué hacemos cuando uno quiere remover todos las instancias con ese valor en la lista?. Ejemplo queremos remover la palabra “tomate” de la siguiente lista

```
lista = [‘cebolla’, ‘tomate’, ‘manzana’, ‘tomate’, ‘cebolla’, ‘manzana’, ‘naranja’, ‘plátano’, ‘tomate’]
lista.remove(‘tomate’)
print(lista) #[‘cebolla’, ‘manzana’, ‘tomate’, ‘cebolla’, ‘manzana’, ‘naranja’, ‘plátano’, ‘tomate’]
```

Para buscar todas las palabras tomate sin conocer cuántas hay en la lista podemos utilizar un While de la siguiente manera:

```
lista = [‘cebolla’, ‘tomate’, ‘manzana’, ‘tomate’, ‘cebolla’, ‘manzana’, ‘naranja’, ‘plátano’, ‘tomate’]

while ‘tomate’ in lista:
    lista.remove(‘tomate’)
```

La linea “`while ‘tomate’ in lista:`” procede ejecutar el while con la condición que mientras exista ‘tomate’ en la lista se cumple la condición.

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Caracteres especiales

Existen caracteres especiales llamados espacios en blanco, esto se refiere a caracteres que no se imprimen, como los espacios, tabulaciones, algunos símbolos de fin de línea. Estos espacios en blanco se pueden ingresar dentro de la cadena de caracteres permitiendo expresar mejor las los mensajes o organizar información.

```
## Las tabulaciones son con \t
a = “Prueba de espacio en blanco”
print(a)
>“Prueba de espacio en blanco”
a = “\tPrueba de espacio en blanco”
print(a)
>    “Prueba de espacio en blanco”
a = “\t\tPrueba de espacio en blanco”
print(a)
>        “Prueba de espacio en blanco”
## Los saltos de líneas son con \n
a = “Prueba de \nespacio en \nblanco”
print(a)
>Prueba de
>espacio en
>blanco
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Eliminar espacios en blancos

Los espacios en blanco pueden traer problemas a la hora de querer comparar cadenas de texto ya que parecen iguales pero son diferente, no es lo mismo la cadena de caracteres “prueba” que “prueba ”, “ prueba” o “ prueba ”, todas estas cadenas son distintas. Un ejemplo es cuando se solicita el nombre del usuario y el cliente en vez de sólo escribir su nombre “Juan” escribe “Juan ”, luego si se quiere comparar su nombre con algún valor en específico o buscarlo por “Juan” no se podrá encontrar, por eso existen funciones que permiten eliminar los espacios adicionales en los lados derecho e izquierdo de una cadena, asegurándonos que no exista un espacio en blanco . **Nota: Los espacios en blanco también son \n \t y otros similares.**

```
#rstrip() elimina los espacios al lado derecho
a = “palabra ”
print(a)
> “palabra”
#lstrip() elimina los espacios al lado izquierdo
a = “ palabra”
print(a)
> “palabra”
#strip() Ambos lados
a = “ palabra ”
print(a)
> “palabra”
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

print()

```
print(“hola”)
> hola
print(“hola”)
print(“mundo”)
> hola
> mundo
print(“hola” end= “”) # end evita que se ingrese un salto de línea al final del print reemplazando el salto por lo indicado
print(“mundo”)
> holamundo
print(“hola” end= “ ”)
print(“mundo”)
> hola mundo
texto = “primer”
print(“Mi”, end=f“ {texto} ”) # Nótese los espacios
print(“hola mundo”)
> Mi primer hola mundo
# Agregar comillas dentro de un texto
print(“ \”Hola mundo\” ”) # Comilla doble \”
print(“ \’Hola mundo\’ ”) # Comilla simple \’
> “hola mundo”
> ‘hola mundo’
#Utilizar las cadenas f se puede usar desde python 3.6
texto = “mundo”
print(f“Hola {texto}”)
> Hola mundo
```

Lista de comprensiones

Una lista de comprensión permite la generación de una lista en una sola linea de codigo, combinando el bucle for y la creación de nuevos elementos en la misma línea, agregando automáticamente cada elemento nuevo. Para realizarla se debe crear la lista abriendo los corchetes y definir la expresión de valor que uno quiere agregar ejemplo valor*3, esta expresión devolverá múltiplos de 3. Luego escribir el bucle para generar los números que se desea aplicar a la expresión. Ejemplo:

```
lista =  [ valor*3 for valor in range(10)]
print(lista)
>[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Tuplas

Las listas pueden poseer valores que podemos ir cambiando a lo largo del programa, pero en ocasiones necesitamos que estos valores no cambian a lo largo de la vida del programa. Las tuplas cumplen esta función, Los valores que no pueden ser cambiados son inmutables y una lista inmutable es llamado una tupla.

```
cuadrado = (20,10)
print(cuadrado[0])
print(cuadrado[1])
> 20
> 10
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Listas

Algunas formas para acceder a los valores de las lista son la siguiente.

```
lista = [“a”, “BA”, 23, 54, [“A”,2], “CASA”, 3]

print(lista[0]) # “a”
print(lista[1]) # “BA”
print(lista[4]) # [“A”, 2]
print(lista[4][0]) # “A”
print(lista[4][1]) # 2
print(lista[1:3]) # [“BA”,23]
print(lista[1:5]) # [“BA”,23,54,[“A”,2]]
print(lista[1:6:2]) # [“BA”,54,”CASA”]
print(lista[0:6:2]) # [“a”,23,[“A”,2]]
```

Listas funciones de utilidad

```
# append()
lista = [1,2,3,4,5]
lista.append(6) # [1,2,3,4,5,6]
lista.append([4,5]) #[1,2,3,4,5,6,[4,5]]
# extend() # permite agregar los elementos de una lista en otra
lista = [1,2,3,4,5,6]
lista.extend([4,5]) #[1,2,3,4,5,6,4,5]
# remove() Elimina el elemento que se indica
lista = [1,2,3,4,5]
lista.remove(2) # [1,3,4,5]
# index() devuelve el número de índice del elemento que pasamos como parámetro
lista = [“Manzana”, “Naranja”, “Banana”]
lista.index(“Naranja”) # 1
# count() permite ver cuántas veces un elemento se repite en la lista
lista = [1,2,3,4,5,6,5,4,3,3,1,2,3,3]
lista.count(3) # 5
```

Actividad

1- Imprime todos los caracteres de una cadena de caracteres usando un for
2-Crea una lista de 5 frutas, de esas 5 elige una y pregunta si existe dentro de la lista deberá devolver verdadero, luego prueba con una fruta que no existe dentro
3- Elimina todos los 0 de la lista lista (debe ser con el uso de while): lista = [0,5,1,0,3,5,4,8,9,0,2,1,0,6,0,0,1,6,8]
4- imprime por pantalla, una sola línea que diga “hola mundo” utilizando 2 print
5- Muestra por pantalla los números de la lista
lista = [0,5,1,0,3,5,4,8,9,0,2,1,0,6,0,0,1,6,8]
Los números a mostrar son desde el tercer número (índice 2) hasta el octavo número (índice 7) sin usar range.
6- lista = [0,5,1,0,3,5,4,8,9,0,2,1,0,6,0,0,1,6,8]
Recorre e imprime los valores de la lista avanzando de 3 en 3, el resultado debería ser el siguiente [0, 0, 4, 0, 0, 0, 8]
7- lista = [0,5,1,0,3,5,4,8,9,0,2,1,0,6,0,0,1,6,8]
Muestra por pantalla cuantas veces aparece el 0

fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Introducción a Matrices

Las matrices no son una estructura propia del lenguaje Python, si no que una matriz es una lista de listas que nosotros interpretamos. Por ejemplo la estructura matriz = [[4,5] , [8,10]]

```
matriz = [ [4,5], [8,10] ]
print(matriz)
> [ [4,5], [8,10] ]
```

Para poder recorrer sus valores normalmente se utilizan bucles anidados, donde el primer ciclo itera a través del número de fila, y el segundo ciclo recorre los elemento de la fila.

```
matriz = [ [4,5], [8,10], [16,2] ]
for y in range(len(matriz)):
    for x in range(len(matriz[y])):
        print(matriz[y][x], end=(‘ ’)) # end ‘’ para que no realice un salto de línea
    print() # para que coloque el salto de línea
> 4  5
> 8  10
> 16  2
```

fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Introducción a Matrices

Para imprimir matrices bidimensionales existen otras formas aparte del range(), estas pueden ser con el uso directamente del for in.

```
matriz = [ [4,5], [8,10], [16,2] ]
for row in matriz:
    for elemento in row:
        print(elemento, end= ‘ ’)
    print()
```

Otro método es con el uso del join para generar la linea completa de una vez combinado con la lista de comprensión

```
matriz = [ [4,5], [8,10], [16,2] ]
for row in matriz:
    print(‘ ’.join([str(elemento) for elemento in row]))
    print()
```

fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Introducción a Matrices

En el caso de querer realizar operaciones como sumas multiplicación, o ir recorriendo la matriz e ir realizando operaciones se realiza con los métodos anteriores

```
matriz = [ [4,5], [8,10], [16,2] ]
suma = 0
for row in matriz:
    for elemento in row:
        suma += elemento
```

```
matriz = [ [4,5], [8,10], [16,2] ]
suma = 0
for y in len(matriz):
    for x in len(y):
        suma += matriz[y][x]
```

fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Introducción a Matrices

La forma correcta de hacerlo es crear una lista con n valores, y luego recorrer la lista e ir agregando una lista dentro del valor.

```
n = 5
m = 4
lista = [0] * n # Genera una lista de 5 valores iniciada en 0
for y in range(len(lista)):
    lista[y] = [0] * m
```

Otra forma es con el append o con la lista de comprensión

```
n = 5
m = 4
lista = [0] * n # Genera una lista de 5 valores iniciada en 0
for y in lista:
    lista.append([0] * m)

lista_2 = [[0] * m for i in range(n)]
```

fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Introducción a Matrices

Un error común al querer generar una matriz es generarla de la siguiente manera

```
lista = [m * [0]] * n # Donde generamos una lista de n x m rellenada con 0
# Crea una lista anidad de 3 columnas y 5 filas
for y in lista
    for x in y:
        print(x,end=“ ”)
    print()

lista[2][0] = 20

for y in lista
    for x in y:
        print(x,end=“ ”)
    print()
```

Esto devuelve solo una referencia a una lista de m ceros, **pero no una lista.**

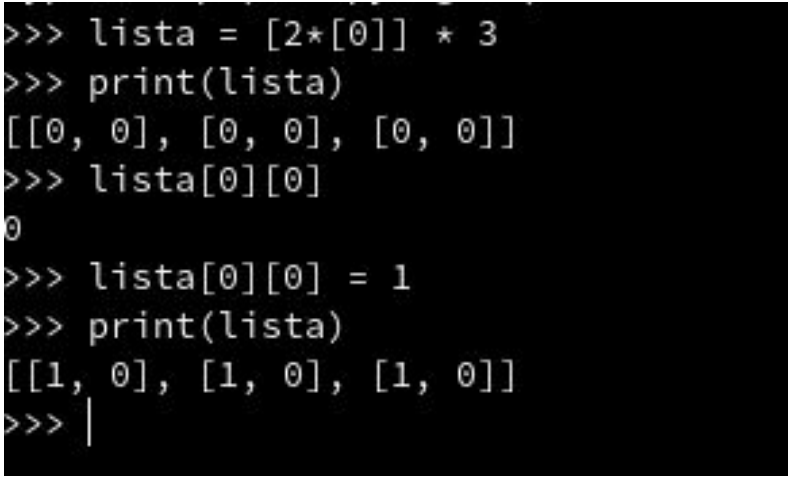
fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Introducción a Matrices

Respecto a la diapositiva anterior aquí un ejemplo:

-Al crear la matriz de esa manera sólo estamos haciendo una lista de referencia, donde cada fila hace referencia al mismo espacio en memoria.

Si yo modifico el valor de lista[0][0] afectará a toda su columna ya que estas hacen referencia a la misma dirección



fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Actividad

1- Genera una matriz de 5x10 iniciada en 0
2- Genera una matriz de 10x 10 con numeros aleatorios
3- Calcular y mostrar por pantalla la suma de todos los números de la matriz anterior
4- Genera una matriz de 5x5 con números aleatorios, luego fila por medio asignar 0 a los valores: Ejemplo

```
0  0  0   0   0
1 23 54 54 1
0  0   0   0  0
12 5  13 1  2
0   0   0   0  0
```

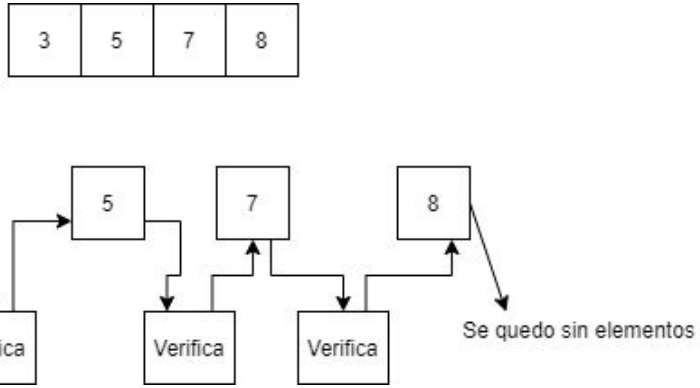
5- Genera una matriz de 5x5 con números aleatorios, luego columna por medio asignar 0 a los valores: Ejemplo
6- crea una matriz de 6 x 6 usando listas de comprensión

fuente: https://www.snakify.org/es/lessons/two_dimensional_lists_arrays/

Algoritmo de búsqueda secuencial

Los elementos almacenados una colección, un ejemplo de una colección es una lista, se le puede llamar que tiene una relación lineal o secuencial. Cada elemento de dato se almacena en una posición que es relativa a los demás, estas posiciones relativas es el valor del índice de cada elemento. Con esto índices de forma ordenada es posible revisarlos en secuencia, a este proceso se le llama búsqueda secuencial.

La búsqueda comienza desde el primer elemento de la lista, verificamos si el valor que se busca, en el caso de no serlo continuamos con el siguiente, repetimos proceso hasta que se encuentre lo que se busca o simplemente si se nos acaban los elementos de la lista, en este caso se da por informado que el elemento no existe en la lista.



Algoritmo de búsqueda Binaria

Para poder realizar una búsqueda más eficiente se puede utilizar la búsqueda binaria, el problema de esto esque solo se puede aplicar en listas ordenadas. La búsqueda binaria comienza desde el elemento central, si es el elemento que estamos buscando el algoritmo termina. Si no es el elemento correcto, se puede utilizar el orden que tiene la lista a nuestro favor, así eliminando la mitad de los elementos restantes, si el elemento que buscamos es mayor al elemento del centro, la lista inferior se puede ignorar y podemos continuar la mitad superior. Luego se puede repetir el proceso con la mitad superior, se comienza con el ítem central y se compara con el valor buscado, si no se encuentra el valor se vuelve a dividir a la mitad

Algoritmo de búsqueda Binaria