El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

CICLO 1

EJE TEMÁTICO 5

FUNDAMENTOS DE PROGRAMACIÓN

Estructuras de datos en Python

Universidad Industrial de Santander

UIS

MisiónTIC2022

Fundamentos de programación

Estructuras de datos en Python

Ruta de aprendizaje

Estructuras de datos en Python

Este capítulo presenta las estructuras de datos más comunes usadas en el lenguaje de programación Python.

Se enfatiza en la creación, uso, métodos y funciones de los respectivos tipos estructurados manejados por el lenguaje.

Palabras clave: Conjuntos, Diccionarios, Listas, Tuplas.

Universidad Industrial de Santander
UIS
MisiónTIC2022

Introducción

Imagina que tienes que ir a la tienda por los ingredientes de una receta que quieres preparar. ¿Qué tal una rica hamburguesa? En este caso, necesitas llevar toda la información de los ingredientes a la tienda para que no se te olviden. Buscas una pequeña libreta y por cada hoja escribes un ingrediente. Al final tienes 8 hojas que debes llevar a la tienda para comprar los ingredientes. Seguramente, estás pensando que esto resultaría muy incómodo e ineficiente. De seguro, habrás fruncido el ceño y expresado lo siguiente: *sería más fácil escribir todo en una hoja de la siguiente manera:*

1. Carne molida
2. Tomate
3. Pan
4. Cebolla
5. Aceite
6. Queso tajado
7. Lechuga
8. Tocineta

Introducción

¿Se te hace "agua" la boca? Volvamos al plan. Cada elemento tiene una posición en la lista y una palabra o valor. Así tienes todo lo que necesitas en un solo lugar y puedes tener acceso a cada elemento de la lista. En programación, a este tipo de acciones se le identifica como una estructura de datos conocida como lista o vector.

Hasta ahora has usado cadenas de caracteres o *strings* en *Python* muchas veces. Pero, ¿sabes cómo se implementan? Es decir, ¿sabes cómo se estructuran los datos internamente o cómo se implementan las distintas operaciones?

Las **estructuras de datos** son agrupaciones de variables simples que conforman un conjunto de datos más complejo con el cual puedes dar soluciones eficientes a situaciones más cercanas a la vida práctica, como lo son por ejemplo: el manejo de calificaciones de estudiantes de un curso o la gestión de nómina de los empleados de una empresa.

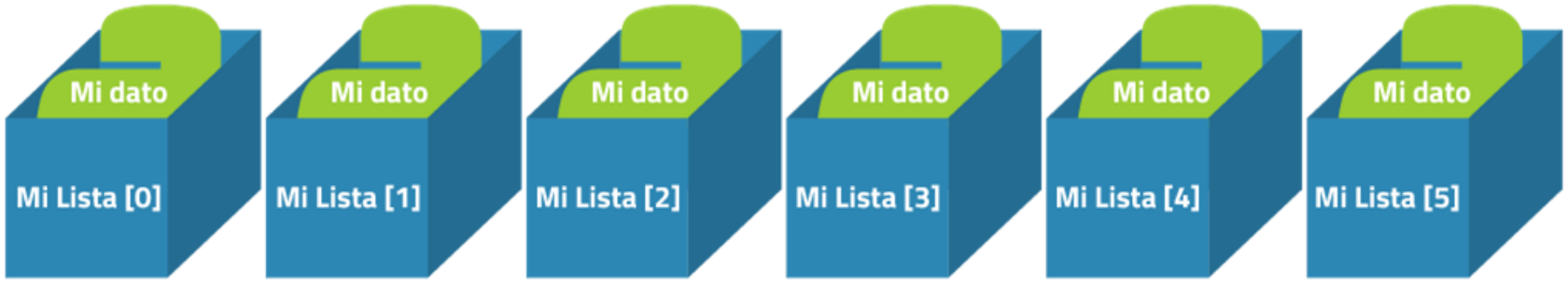
Definiciones

Variable: Es un espacio de memoria que **contiene un dato simple** de tipo cadena, numérico, booleano, etc.

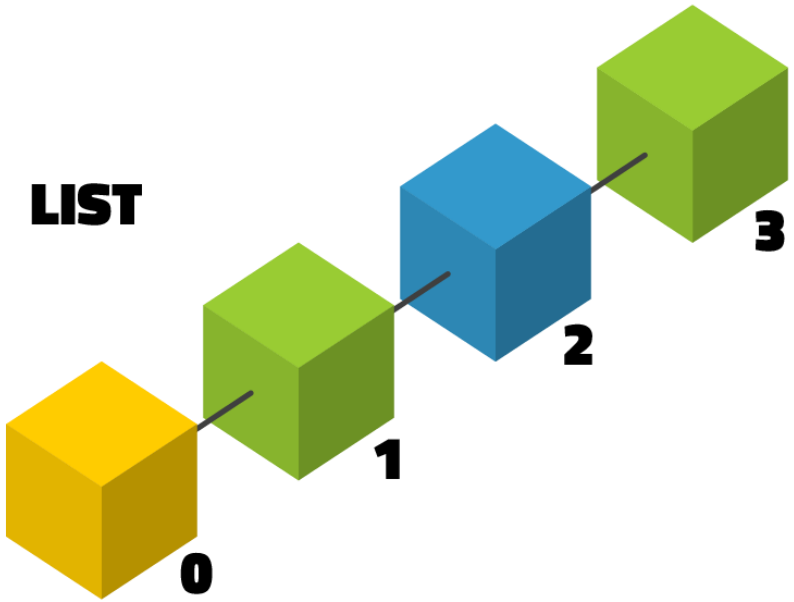
Al contenido de este espacio de memoria, se accede a través de lo que llamamos un **identificador o Nombre de Variable.**

Definiciones

Variable Estructurada: es un agrupamiento, empaquetamiento o colección de varios espacios de memoria, a los cuales se accede a través de un único identificador**.**



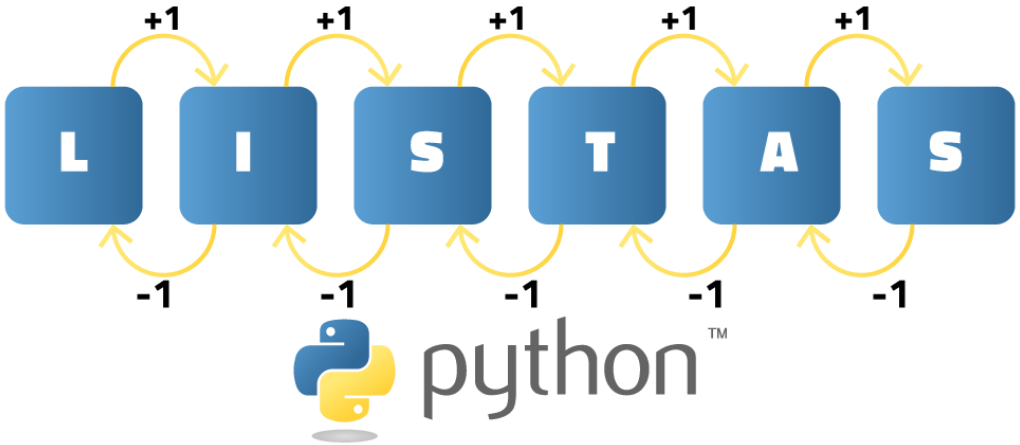
Estructuras de tipo Listas y Tuplas



Las **listas** y **tuplas** se refieren a **áreas de almacenamiento adyacente o contiguo en memoria**, donde podemos reunir o guardar distintos tipos de elementos bajo un mismo nombre de variable.

Estructuras de tipo Listas y Tuplas

Las listas son estructuras que permiten ser modificadas a lo largo de la ejecución de un programa usando algunos métodos y operadores. Este comportamiento le da la caracterización a las listas de ser **estructuras mutables**.



Estructuras de tipo Listas y Tuplas

Las tuplas son estructuras que, una vez creadas o definidas, NO permiten modificarse a lo largo de la ejecución de un programa, lo cual les da la caracterización de ser **estructuras inmutables**.

Revisión de lo aprendido

De acuerdo con lo visto en la presente sección del módulo responde las siguientes preguntas:

1. Un identificador se puede definir como:

   a) El nombre que se la da a una variable en un programa
   b) El código especial con el que se calcula un dato particular
   c) El nombre del archivo que contiene el código fuente de un programa python

Revisión de lo aprendido

2. Las variables estructuradas son:

a) Varios espacios de memoria con un identificador único
b) Un conjunto de datos que tienen un identificador respectivo para cada dato
c) Instrucciones que soportan la estructura lógica de un programa

Revisión de lo aprendido

3. ¿Cuál de las afirmaciones es correcta?:

a) Las tuplas son mutables y las listas son inmutables.
b) Las tuplas y las listas son inmutables.
c) Las tuplas son inmutables y las listas son mutables.

Creación de una Lista en Python

Para crear una lista en Python, se nombra la variable y se asigna con el signo igual el conjunto de datos que conforman la lista, separados por comas y encerrados entre corchetes.

Ejemplo:

```
dato_v = [ 10, 20, 30, 40 ]
dato_x = [ “Juan”, “Pedro”, “José”, “Silvia” ]
dato_s = [ “Camisas”, “Pantalones”, “Zapatos”, “Corbata” ]
```

Las listas dato_s y dato_x contienen ítems de tipo texto o cadena de caracteres (string). Es por esto que, cada dato de tipo *string* debe ir entre comillas.

Creación de una Lista en Python

Otra manera de crear listas de manera más fácil y efectiva, es usando la función `range(`,el cual crea un conjunto de valores según los parámetros de inicio, final y paso.

```
x = list(range(2, 20, 2))
print(x) #Resultado [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

El primer parámetro de range inicia la secuencia en 2. El segundo parámetro termina la secuencia hasta el número 20, **sin incluirlo,** es decir hasta 19. El tercer parámetro será el paso. La secuencia final irá desde 2 hasta 20 de 2 en 2.

```
x = list(range(16, 0, -2))
print(x) #Resultado [16, 14, 12, 10, 8, 6, 4, 2]
```

Creación de una Tupla en Python

Las tuplas usan paréntesis para definir sus elementos.

Ejemplo:

Cosas = ("Apartamento", "Ascensor", "Batería”, ”Libro", "Alfombra")

Esta tupla recién creada contiene ítems de texto o cadena de caracteres (*string*). Por esta razón, cada elemento de tipo *string* debe ir entre comillas.

Es importante recordar que cuando se crean las tuplas se debe, de forma obligatoria, especificar cuáles elementos se van a guardar, ya que no podemos modificar la tupla en tiempo de ejecución. No tiene sentido crear una tupla vacía.

Acceso a los elementos de los elementos de Listas y Tuplas

Para acceder a los elementos contenidos en listas y tuplas, se debe tener en cuenta el índice en el que se encuentra el elemento deseado.

Un índice es un número entero que muestra la ubicación que un elemento ocupa en una lista o tupla. Siendo CERO (0) el índice usado para la primera posición.

Ejemplo:

Objetos = ["Celular", "Vehículo", "Escritorio"]

Objetos =	**[“Celular”,**	**“Vehículo”,**	**“Escritorio”]**
	0	**1**	**2**

Acceso a múltiples elementos

Para acceder a más de un elemento de una lista o de una tupla, se debe especificar, entre corchetes, el índice de inicio y el índice final separados por dos puntos.

Los elementos obtenidos de la lista serán los correspondientes desde el índice de inicio hasta el índice final menos 1.

Ejemplos:

```
lista = [1, 2.5, 'Code', [5,6] ,4]

print (lista[1:3])          # devuelve [2.5, 'Code']
print (lista[1:6])          # devuelve [2.5, 'Code', [5, 6], 4]
print (lista[1:6:2])        # devuelve [2.5, [5, 6]]
print (lista[::-1])         # devuelve la lista invertida
```

Para tener en cuenta

Cuando se accede a múltiples elementos de una lista o una tupla:

- El índice inicial siempre debe ser menor que el índice final.
- Si no se especifica el índice inicial, se asume que es 0.
- Si no se especifica el índice final, se asume que es el tamaño de la lista o tupla menos uno.
- En la selección de múltiples elementos, si el índice final sobrepasa el tamaño de la lista, no se produce error sino que se toma hasta el último elemento de la lista.

Recorrer una Lista o una Tupla

Los elementos de una Lista o de una Tupla, pueden obtenerse utilizando una instrucción iterativa for.

```
lista = [1, 2.5, 'Enya Isabel', [5,6] ,99]
for element in lista:
    print (element)
```

La salida del anterior segmento de código es:

```
1
2.5
"Enya Isabel"
[5,6]
99
```

Métodos para modificar Listas

Podemos crear una lista y luego modificar sus elementos mientras se ejecuta el código.

Para esto, las listas tienen un conjunto de métodos y funciones que realizan acciones y operaciones sobre una lista en particular.

Algunos de estos métodos son:

append, extend, insert, pop, remove

Método append

El método append permite añadir un ítem al final de una lista.

```
my_list = [2, 5, 'DevCode', 1.2, 5]
my_list.append(10)                          # [2, 5, 'DevCode',
1.2, 5, 10]
my_list.append([3,9])                  # [2, 5, 'DevCode', 1.2, 5,
[3, 9]]
```

Podemos agregar cualquier tipo de elemento a una lista, pero tenga en cuenta lo que pasa cuando agregamos una lista dentro de otra, la lista se agrega como uno y solo un elemento.

Método extend

El método extend se utiliza para agregar elementos interables como un *string* u otra lista separando sus elementos.

Extend nos permite agregar ítems dentro de una lista, pero a diferencia de *append*, cada ítem de la nueva lista se adiciona como un elemento individual dentro de la lista afectada.

Ejemplo:

```
my_list = [2, 5, 'DevCode', 1.2, 5]
my_list.extend([3,9])                    # [2, 5, 'DevCode', 1.2, 5, 3, 9]
```

Método insert

El método insert permite añadir un ítem en una posición o índice específico.

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']
Nombres.insert(1, 'Ivon')
print(Nombres)          #Resultado  ['David',  'Ivon',  'Elkin',  'Juliana',
'Silvia']

Nombres.insert(1, 'Carmen')
print(Nombres)          #Resultado  ['David',   'Carmen',  'Ivon',  'Elkin',
'Juliana', 'Silvia']

Nombres.insert(5, 'Joel')
print(Nombres)          #Resultado  ['David',   'Carmen',  'Ivon',  'Elkin',
'Juliana',  'Joel', 'Silvia']

Nombres.insert(99, 'Rafael')
print(Nombres)          #Resultado  ['David',   'Carmen',  'Ivon',  'Elkin',
'Juliana',  'Joel', 'Silvia', 'Rafael']
```

Método pop

El método pop quita un elemento de la lista dado su índice.

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']
Nombres.pop(1)
print(Nombres) #Resultado ['David', 'Juliana', 'Silvia']
```

En este ejemplo, el elemento con índice 1 (‘Elkin’) se elimina de la lista.
Si no se le indica el índice, se eliminará el último elemento de la lista.

```
lista = [10,20,30,40,50]
lista.pop()
print(lista) #Resultado [10, 20, 30, 40]
```

Método remove

El método remove.

Si fuera requerido remover un ítem de una lista basado en el valor, el método remove se podría utilizar de la siguiente manera.

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']
Nombres.remove('Elkin')
print(Nombres) #Resultado ['David', 'Juliana', 'Silvia']
```

Si el valor a eliminar no se encuentra en la lista, se mostrará un error.

Revisión de lo aprendido

1. Las tuplas se crean utilizando paréntesis

   a) Verdadero
   b) Falso

Revisión de lo aprendido

2. ¿Cuántos elementos de lista crea la línea de Código

```
x = list(range(2, 20, 2))?
```

a) 9

b) 8

c) Genera error de sintaxis

Revisión de lo aprendido

3. La línea de código `x = list[range(16, 0, -2)]`

a) Genera error de sintaxis

b) Produce una lista de números en orden descendente

c) Produce una lista de números negativos

Revisión de lo aprendido

4. Para mostrar una lista invertida de 5 elementos se puede usar una de las siguientes líneas de código:

a) `print (lista[::-1])`

b) `print (lista[5:0:-1])`

c) `print (lista[5:4:-1])`

Revisión de lo aprendido

5. Al ejecutar la secuencia de instrucciones:

```
my_list = [2, 5, 'cima', 12]
my_list.append([10,20])
```

la variable estructurada my_list contiene los siguientes datos:

a) my_list = [2, 5, 'cima', 12, 10, 20]

b) my_list = [2, 5, 'cima', 12, [10, 20]]

c) my_list = [2, 5, 'cima', 12, 10, 20, 10, 20]

Revisión de lo aprendido

6. Al ejecutar la secuencia de instrucciones:

```
my_list = [2, 5, 'cima', 12]
my_list.extend([10,20])
```

la variable estructurada my_list contiene los siguientes datos:

a) my_list = [2, 5, 'cima', 12, 10, 20]

b) my_list = [2, 5, 'cima', 12, [10, 20]]

c) my_list = [2, 5, 'cima', 12, 10, 20, 10, 20]

Revisión de lo aprendido

7. Al ejecutar la secuencia de instrucciones:

```
my_list = [10,20,30,40,50]
lismy_list ta.pop()
```

la variable estructurada my_list contiene los siguientes datos:

a) amy_list = [10, 20, 30,40, 50]

b) my_list = [10, 20, 30, 40]

c) my_list = []

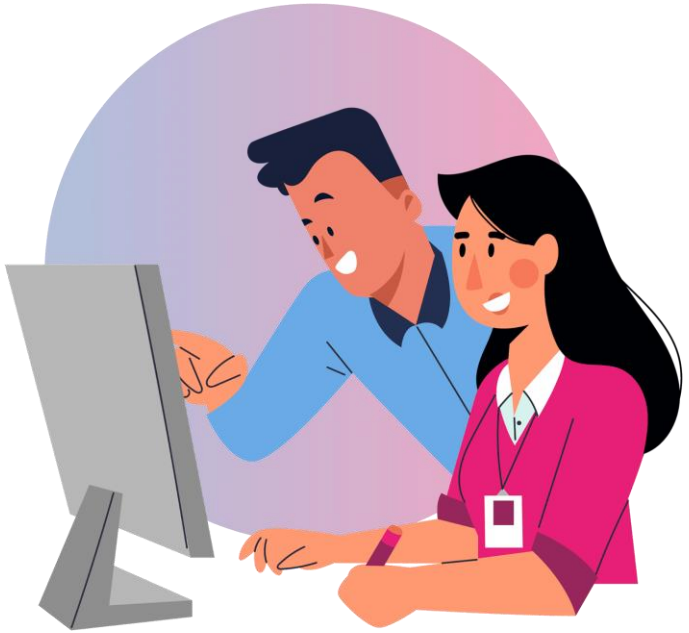
Conjuntos

Un conjunto es una **colección desordenada** de elementos.

Cada elemento del conjunto **es único** (sin duplicados) y debe ser inmutable (no se puede cambiar).

Sin embargo, un conjunto en sí mismo es mutable, ya que podemos agregarle o quitarle elementos completos.

La estructura de conjuntos en Python también se pueden utilizar para realizar operaciones como unión, intersección, diferencia y diferencia simétrica.



Creación de Conjuntos

Un conjunto se crea colocando todos los elementos (elementos) entre llaves {}, separados por comas o usando la función incorporada set(). Puede tener cualquier número de elementos y pueden ser de diferentes tipos (entero, flotante, tupla, cadena, etc.). Sin embargo, un conjunto no puede contener elementos mutables como listas, conjuntos o diccionarios.

```
mi_conjunto = {1, 2, 3}
print(mi_conjunto)# Resultado: {3, 1, 2}
```

Es importante recordar que, los elementos de un conjunto no tiene un orden predefinido como las listas o las tuplas; por lo tanto, las posiciones de los elementos no importan y **pueden variar entre la definición y la impresión del conjunto.**

Creación de Conjuntos

Uso de la instrucción set:

```
mi_conjunto = set([1, 2, 3, 2])
print(mi_conjunto)
# Resultado: {1, 2, 3}

mi_conjunto = {1, 2, [3, 4]}
# Mostrará un error
```

Esta última línea mostrará un error porque no podemos tener elementos mutables en un conjunto como lo es una lista.

Diccionarios

Los diccionarios son estructuras de datos que nos permiten almacenar **valores** indexados, no mediante números, sino mediante **claves**. Esto nos permite ordenar datos de manera más natural y sencilla.

Para crear un diccionario, empaquetamos entre llaves { } cada par de elementos **Clave:Valor** separadas por comas, así:


```
mi_diccionario = {'nombre':'david', 'apellido':'Cortez', 'edad': 27,
'genero': 'masculino'}
```

Acceso a elementos de Diccionarios

Una vez que almacenamos los datos en el diccionario, vamos a acceder a ellos.

```
print(mi_informacion['apellido'])                    # Resultado: Cortez
```

Con el **método get()** de un diccionario, podemos obtener el valor de una clave, pero si no existe la clave devolver un mensaje en *string* como respuesta.

```
print(mi_informacion.get('apellido', "No tiene un apellido"))
# Resultado: Cortez
print(mi_informacion.get('celular', "No tiene un celular"))
# Resultado: No tiene un celular
```

Acceso a elementos de Diccionarios

También es posible almacenar elementos complejos como listas dentro de un diccionario

```
mi_informacion = {
        'nombre' : 'David',
        'apellido' : 'Cortez',
        'sobrenombre' : 'DD',
        'padres' : ["Marina gomez", "Julián Cortez"],
        'edad' : 27,
        'genero' : 'masculino',
        'estado Civil' : 'Soltero',
        'hijos' : 2,
        'mascotas' : 'Perro',
        'nombres de mascotas' : ["chato"]
        }
```

Modificar Diccionarios

Para agregar valores nuevos a un diccionario, se debe realizar de la siguiente forma:

```
mi_informacion['salario'] = 200000
```

De esta forma, la clave será “salario” y el valor 200000.

Si ahora queremos modificar un valor ya existente, se debe realizar de la siguiente forma:

```
mi_informacion['edad'] = 38
```

Al ser una clave existente se modificará solo el valor.

Modificar Diccionarios

Si se requiere **eliminar** un elemento **clave:valor** se debe utilizar:

```
mi_informacion.pop('hijos')
del(mi_informacion['mascotas'])
```

En la primera línea, utilizamos pop() como una operación del diccionario; y en la segunda línea, usamos la función predefinida del().

De esta manera, se habrá eliminado las **claves** de hijos y mascotas con sus respectivos **valores**.

Revisión de lo aprendido

1. Una estructura tipo conjunto es una estructura de datos desordenados

   a) Verdadero
   b) Falso

Revisión de lo aprendido

2. Los diccionarios son estructuras de datos a las que se accede a través de un par clave:valor.

a) Verdadero

b) Falso

Revisión de lo aprendido

3. Una de las siguientes instrucciones es una forma CORRECTA de crear un diccionario:

a)
```
mi_diccionario = {'nombre', 'Juan', 'ocupacion','Soldador',
'edad',27}
```

b)
```
mi_diccionario = {'nombre':'Juan', 'ocupacion':'Soldador',
'edad': 27}
```

c)
```
mi_diccionario = ['nombre':'Juan', 'ocupacion':'Soldador',
'edad': 27]
```

Revisión de lo aprendido

4. Una de las siguientes instrucciones NO se puede usar para remover elementos de un diccionario:

a) `remove`

b) `pop`

c) `del`

El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

CICLO 1

EJE TEMÁTICO 5

FUNDAMENTOS DE PROGRAMACIÓN

Estructuras de Datos en Python

Universidad Industrial de Santander

MisiónTIC 2022