



# Curso Intensivo de Python - Tema 5

Agustín Arenas



## Curso Intensivo de Python - Tema 5

- 1) Estructuras de datos
  - a) Listas
  - b) Tuplas
  - c) Diccionesarios
  - d) Conjuntos

# Estructura de datos





# Estructura de datos

- Si imaginamos la memoria como los casilleros, vemos que los valores ocupan un determinado lugar
- El número de casillero, representa el índice del dato y el casillero el dato propiamente dicho
- Cuando programamos, podemos necesitar hacer referencia a un conjunto de valores a través de una sola variable, y es por tal motivo que existen estas estructuras de datos
- Podemos tener múltiples valores, ordenarlos, buscar valores determinados, etc. Python ofrece estructuras de datos de diversos sabores y con diversas aplicaciones, cada una tiene sus fortalezas y debilidades



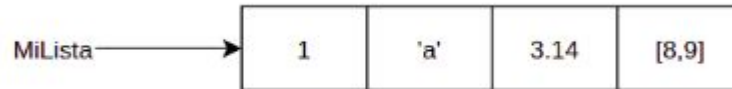
# Listas

- La **lista** es la más conocida y posiblemente más utilizada
- Consiste en una secuencia de valores, de **diversos tipos, mutable y dinámica**
- Es decir, una lista puede contener datos enteros, flotantes, cadenas u otra lista conviviendo sin problemas
- Al ser mutable, estos datos pueden modificarse si es necesario y que sea dinámica quiere decir que podemos solicitar más espacio para almacenar datos cuando querramos



# Listas

- La lista se declara con []
- Podemos darle una representación de forma gráfica:





# Listas

- Es posible definir una lista mediante el operador de repetición
- Este operador es el `*` y se utiliza como

`[lista]* numeroRepeticiones`



# Listas

```
listaVacia = [] #Se declara una lista vacía, sin elementos
listaConValores = [1,'a',3.14,[8,9]] #Valores declarados por el
programador
listaConRepeticion = [0]*10 #Se crea una lista con 10 elementos 0s.
print(listaVacia)
print(listaConValores)
print(listaConRepeticion)
```





# Listas

- Los elementos de una lista se pueden acceder mediante un ciclo for, debido a que es una secuencia de datos
- También mediante el operador de **indexación** (`[]`), con el cuál accedemos mediante el índice, es decir, la posición del dato
- **¡Importante!** En la mayoría de los lenguajes, excepto Matlab, los índices arrancan en **0** y terminan en **LongitudDato - 1**

# Listas

```
import random as rd
lista = list(range(20))
#Range devuelve una secuencia de 0 a 19
#Esa secuencia se pasa a la función list que la transforma
#en una lista
print('Lista desde for')
for dato in lista:
    print('El dato es {0}'.format(dato))
#El resultado sería igual a
print('\nLista desde for accedido por indice')
for indice in range(len(lista)): #len(secuencia) nos devuelve la longitud
    #de una secuencia, al ser un valor entero, range lo toma y calcula la
    #secuencia desde 0 hasta longitudSecuencia - 1
    print('El dato es {0}'.format(lista[indice])) #El operador de indexación es []
print('-----')
#A través de dicho operador se puede llevar a cabo las modificaciones
indiceAleatorio = rd.randint(0,len(lista)-1) #0,19
numeroAleatorio = rd.randint(-len(lista),len(lista)) #-20,20
print('Lista antes de modificarse: {0}'.format(lista))
lista[indiceAleatorio] = numeroAleatorio
print('Lista despues de modificarse: {0} en {1} con el valor {2}'.format
(lista,indiceAleatorio,numeroAleatorio))
```



# Concatenación y copia

- Las listas se pueden concatenar (unir o join) mediante el operador +
- El operador de repetición internamente hace eso: Crea N copias de listas y las concatena en una lista final
- Además de concatenarse, pueden copiarse, pero... ¡Cuidado!

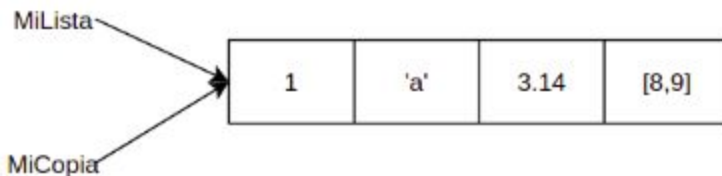


# Concatenación y copia

```
listaPares = list(range(0,10,2))
listaImpares = list(range(1,9,2))
print('Lista pares {0} \nE impares {1}'.format(listaPares,listaImpares))
listaTotal = listaPares + listaImpares
print(listaTotal)
print('\n-----\n')
#Creo una copia
copiaLista = listaTotal
print('Lista original {0} \nCopia: {1}'.format(listaTotal,copiaLista))
print('-----')
listaTotal[0] = 10
copiaLista[1] = 20
copiaLista[2] = listaTotal[4]
print('Lista original {0} \nCopia: {1}'.format(listaTotal,copiaLista))
#¿Qué pasó?
```

# Cortar

- Asignar una lista a una nueva lista hace que ambas variables hacen referencia a la misma posición de memoria, es decir, a la misma lista



- Para evitar esto, debemos hacer una copia elemento a elemento de la lista original a la nueva lista. O concatenar la lista original con una **lista vacía** y asignar esa concatenación a la nueva lista



# Cortar

- A partir de una lista original, además de copiar los elementos, es posible generar una sublista de los elementos a través del **slicing**
- La expresión de slicing selecciona un rango de elementos de una secuencia, un **slice** es un generador de elementos que son tomados de una secuencia



# Cortar

```
lista = list(range(10))
print(lista) #Lista original
primerosTres = lista[0:3] #[0,1,2]
#El slicing se compone de la forma:
# lista[inicio : fin]
#Y toma los elementos desde inicio, hasta
# (fin-1)
ultimosTres = lista[-1:5]
#La ultima posición es -1, y desde esa posición
#hacia adelante se decrementa en 1
print(primerosTres)
print(ultimosTres)
tomaDeADos = lista[5:2] #Si no se establece
#el inicio o fin, se toma la primera y ultima
#posición de la lista
print(tomaDeADos)
```



# Búsqueda

- El uso de índices inválidos durante el slicing no **genera una excepción**
- Sino que:
  - Si la posición final está más allá del final de la lista, Python usará la longitud de la lista
  - Si la posición inicial está antes de la posición inicial de la lista, Python usará la posición 0
  - Si el inicio es mayor al final, el slicing retornará una lista vacía.
- Es posible buscar por un elemento en una lista haciendo uso del operador **in**, o **not in** para buscar un elemento que no está en la lista
- Python provee también métodos para trabajar con listas





# Búsqueda

```
dias =  
['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo']  
buscarDia = input('Ingrese día: ')  
if buscarDia not in dias:  
    print('404 - Calendario not found')  
else:  
    print('El día existe')
```



# Métodos

```
lista = list(range(5))
print('Lista original: {0}'.format(lista))
print('Método append')
#append(x) añade un elemento al final de la
lista
lista.append(5)
print('Lista modificada: {0}'.format(lista))
print('Método insert')
#insert(i,x) inserta un elemento en una
posición
#determinada. Si existe elemento en esa
posición,
#se desplazan los otros elementos
lista.insert(6,6)
print('Lista modificada: {0}'.format(lista))
print('Método remove')
#remove(x) remueve el PRIMER elemento cuyo
valor
#es igual a x
```



# Métodos

```
lista.insert(6,6)

print('Lista modificada: {0}'.format(lista))

lista.remove(6)

print('Lista modificada: {0}'.format(lista))

print('Método count')

#Cuenta la cantidad de veces que x aparece

print('Cuenta {0}'.format(lista.count(6)))

print('Lista modificada: {0}'.format(lista))

print('Método index')

#Retorna el indidce del primer elemento cuyo valor

#es igual x. PUEDE GENERAR UN ValueError sino

#se encuentra

print('Valor en índice {0}:

{1}'.format(6,lista.index(6)))

print('Método clear')

#Borra todos los elementos de la lista

lista.clear()

print('Lista modificada: {0}'.format(lista))

print('-----Fin programa-----' )
```



# Lista anidada

```
import random as rd
FILAS = 4
COLUMNAS = 3
matrix = [[0] * COLUMNAS] * FILAS
print('Matrix vacia {0}'.format(matrix))
for i in range(FILAS):
    for j in range(COLUMNAS):
        matrix[i][j] = rd.randint(0, (FILAS * COLUMNAS))
        # Para matrices se usa el primer
        # [] para fila y el segundo [] la columna
    print('Matrix modificada {0}'.format(matrix))
print(matrix[2][1])
```



# Tuplas

- Las **tuplas** no difieren de las listas, una tupla es una secuencia inmutable, lo cual significa que su contenido no puede modificarse
- Se pueden crear iniciando o usando la función `tuple()` que toma una secuencia y la transforma en tupla.
- Las tuplas soportan todas las operaciones de las lista, excepto aquellas que cambian su contenido. Es decir:
  - Acceso por indexación
  - Métodos como `index`
  - Built-in como `len`, `min` y `max`
  - Slicing
  - Operador `in`
  - Operador `+` y `*`



# Tuplas

- El porqué de las tuplas es que tienen una mejor performance que las listas debido a su característica de inmutabilidad. Otra razón es que son seguras, debido a que no permiten cambiar los datos



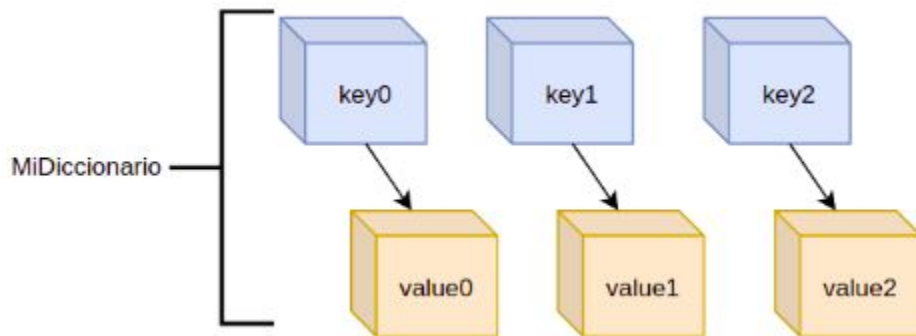
# Tuplas

```
tupla = tuple(range(10))
print('Tupla {0}'.format(tupla))
tuplaVacia = ()
print('Tupla vacía {0}'.format(tuplaVacia))
generador = (0,) * 10
print('Generador {0}'.format(generador))
concatenacion = generador + tupla
print('Concatenacion {0}'.format(concatenacion))
print(concatenacion[18]) #Acceso elemento
print(tupla[1:5]) #Slicing
print(tupla.__len__()) #Longitud
print(max(tupla)) #Máximo
print(min(concatenacion)) #Mínimo

copia = tupla[:] #Copia
print('Copia {0}'.format(copia))
```

# Diccionarios

- Los diccionarios forman es una estructura de datos del tipo mapa, donde una llave está relacionada con un valor, en una relación 1 a 1







# Diccionarios

- La llave se usa para acceder al valor
- Para crear un diccionario se deben encerrar los valores entre `{}`, separando los valores por `,` y escribiendo **llave:valor**
- Las llaves dentro del diccionario deben ser objetos inmutables, ejemplo: cadenas, enteros, números flotantes, tuplas pero no listas



# Diccionarios

```
from cgi import print_form

agenda = {'Batman':'555-1111','Iron Man':'555-2222','Wonder Woman':'555-33333'}

print(agenda)

#Para acceder a los datos del diccionario se usa el operador
#[]. SI LA LLAVE NO EXISTE GENERA UN KEYERROR

print('El teléfono de Batman es {0}'.format(agenda['Batman']))

#Para añadir un dato, hacemos uso del mismo operador
agenda['Superman'] = '555-5555'
agenda['Batman'] = '555-6666'

print(agenda)

#Nota: Las comparaciones de cadenas son sensibles a mayúsculas
#Puede usarse in y not in para prevenir los KeyError

if 'Guason' in agenda:
    print('HAAAAAAAAHAHAHA')
else:
    print('Falta the joker')
```



# Diccionarios

```
#Para eliminar un elemento hacemos uso de la funcion del
del agenda['Iron Man']
print(agenda)

#La funcion len nos dice cuantos elementos tiene el diccionario
print('Diccionario de {0} elementos'.format(len(agenda)))

#Los diccionarios pueden tener multiples valores
dicc = {'A':1, 'B':3.14, 'C':[0,1,2], 'D':'Z', 'E':(1,2,3)}
print(dicc)

#Un diccionario puede crearse vacío
diccionarioVacio = {}

#O con la funcion dict()
diccionarioDict = dict(m=8, n=9)

#Algunos métodos son
print(dicc.clear()) #Elimina todos los elementos
print(dicc.get('Batman', 'No key')) #Obtiene un valor por la llave
#Y si no existe devuelve el valor por defecto sin generar
#excepción
```



# Diccionarios

```
dicc = {'A':1, 'B':3.14, 'C':[0,1,2], 'D':'Z', 'E':(1,2,3), 'Batman':'555-1111'}  
print(dicc.items()) #Retorna el conjunto de llave-valor  
print(dicc.keys()) #Retorna todas las llaves del diccionario en una secuencia  
print(dicc.pop('Batman', 'No key')) #Similar a get para remover elemento  
print(dicc)  
print(dicc.values()) #Retorna todas los valores del diccionario en una secuencia
```



# Conjuntos

- Un **set** contiene una colección de valores únicos y trabaja de forma similar a un conjunto matemático
- Los datos que almacenan deben ser **únicos**, de forma **no ordenada** y pueden ser **de diferentes tipos**



# Conjuntos

```
myConjunto = set() #Crea un conjunto vacío, puede tomarse
#un objeto iterable (lista,tupla,cadena)
myConjunto = set('aaabc')
print(myConjunto)
myConjuntoDos = set(['a','b','c'])
myConjuntoTres = set(['a','d','e'])
print('Longitud del conjunto {0}'.format(len(myConjunto)))
myConjunto.add('d') #add(x) añade el elemnto x al conjunto
print(myConjunto)
myConjunto.update(['abc','def']) #update(iterable) añade una secuencia de valores
print(myConjunto)
myConjunto.discard('d') #discard(x) remueve x del conjunto
print(myConjunto)
#Para iterar se puede usar un ciclo for
for dato in myConjunto:
    print(dato)
```



# Conjuntos

```
#Y in o not in para buscar un valor
if 'abc' in myConjunto:
    print('abc!')

#Los conjuntos se pueden "concatenar" mediante la union
print(myConjuntoDos.union(myConjuntoTres))

#Los elementos son todos los del conjutuno uno y dos
#Se puede hacer una intersección tambien
print(myConjuntoDos.intersection(myConjuntoTres))

#Solo tienen los valores que tienen en común ambos conjuntos
#La diferencia retorna los elementos que están en el primer
#conjunto y no en el segundo
print(myConjuntoDos.difference(myConjuntoTres))

#La diferencia simétrica devuelve los elementos que
#se encuentran en el primer o segundo conjunto pero no
#en ambos
print(myConjuntoDos.symmetric_difference(myConjuntoTres))

#Se puede saber si un subconjunto está dentro de un conjunto
#O superconjunto
print(myConjuntoDos.issubset(myConjuntoTres))
print(myConjuntoDos.issuperset(myConjuntoTres))
```

# Unicode

UN Emoji Charts

## Emoji List, v15.0

[Index & Help](#) | [Images & Rights](#) | [Spec](#) | [Proposing Additions](#)












This chart provides a list of the Unicode emoji characters and sequences, with single image and annotations. Clicking on a Sample goes to the emoji in the full list. The ordering of the emoji and the annotations are based on Unicode CLDR data. Emoji sequences have more than one code point in the **Code** column. Recently-added emoji are marked by a © in the name and outlined images.

Emoji with skin-tones are not listed here: see [Full Skin Tone List](#).

For counts of emoji, see [Emoji Counts](#).

While these charts use a particular version of the Unicode Emoji data files, the images and format may be updated at any time. For any production usage, consult those data files. For information about the contents of each column, such as the **CLDR Short Name**, click on the column header. For further information, see [Index & Help](#).



| Smileys & Emotion |        |   |                                 |  |
|-------------------|--------|---|---------------------------------|--|
| face-smiling      |        |   |                                 |  |
| Nr                | Code   | Sample  | CLDR Short Name                 | Other Keywords   |
| 1                 | U+2700 |  | grinning face                   | face   grin   grinning face  |
| 2                 | U+2701 |  | grinning face with big eyes     | face   grinning face with big eyes   mouth   open   smile                    |
| 3                 | U+2702 |  | grinning face with smiling eyes | eye   face   grinning face with smiling eyes   mouth   open   smile          |
| 4                 | U+2703 |  | beaming face with smiling eyes  | beaming face with smiling eyes   eye   face   grin   smile                   |
| 5                 | U+2704 |  | grinning squinting face         | face   grinning squinting face   laugh   mouth   satisfied   smile           |
| 6                 | U+2705 |  | grinning face with sweat        | cold   face   grinning face with sweat   open   smile   sweat                |
| 7                 | U+2706 |  | rolling on the floor laughing   | face   floor   laugh   roll   rolling   rolling on the floor laughing   rofl |
| 8                 | U+2707 |  | face with tears of joy          | face   face with tears of joy   joy   laugh   tear                           |
| 9                 | U+2708 |  | slightly smiling face           | face   slightly smiling face   smile   |
| 10                | U+2709 |  | upside-down face                | face   upside-down   |
| 11                | U+270A |  | melting face                    | disappear   dissolve   liquid   melt   melting face                          |

Disponible aquí



# Unicode

(👁️👁️)  
SYM名L



Symbol of The Day

Symbols  
Unicode®  
Alphabets  
Emoji  
Collections  
Tools  
Codes ▶  
Holidays  
Art 🎨👉▶

Search

— WHEN TEXT  
IS NOT ENOUGH

7

Arabic  
Numerals



Hot Symbols



Top-50 Emoji



Symbols for  
Steam



Symbols for  
Nickname



Stars



Quotation  
Marks



Punctuation

19

Disponibile aquí



¿Preguntas?