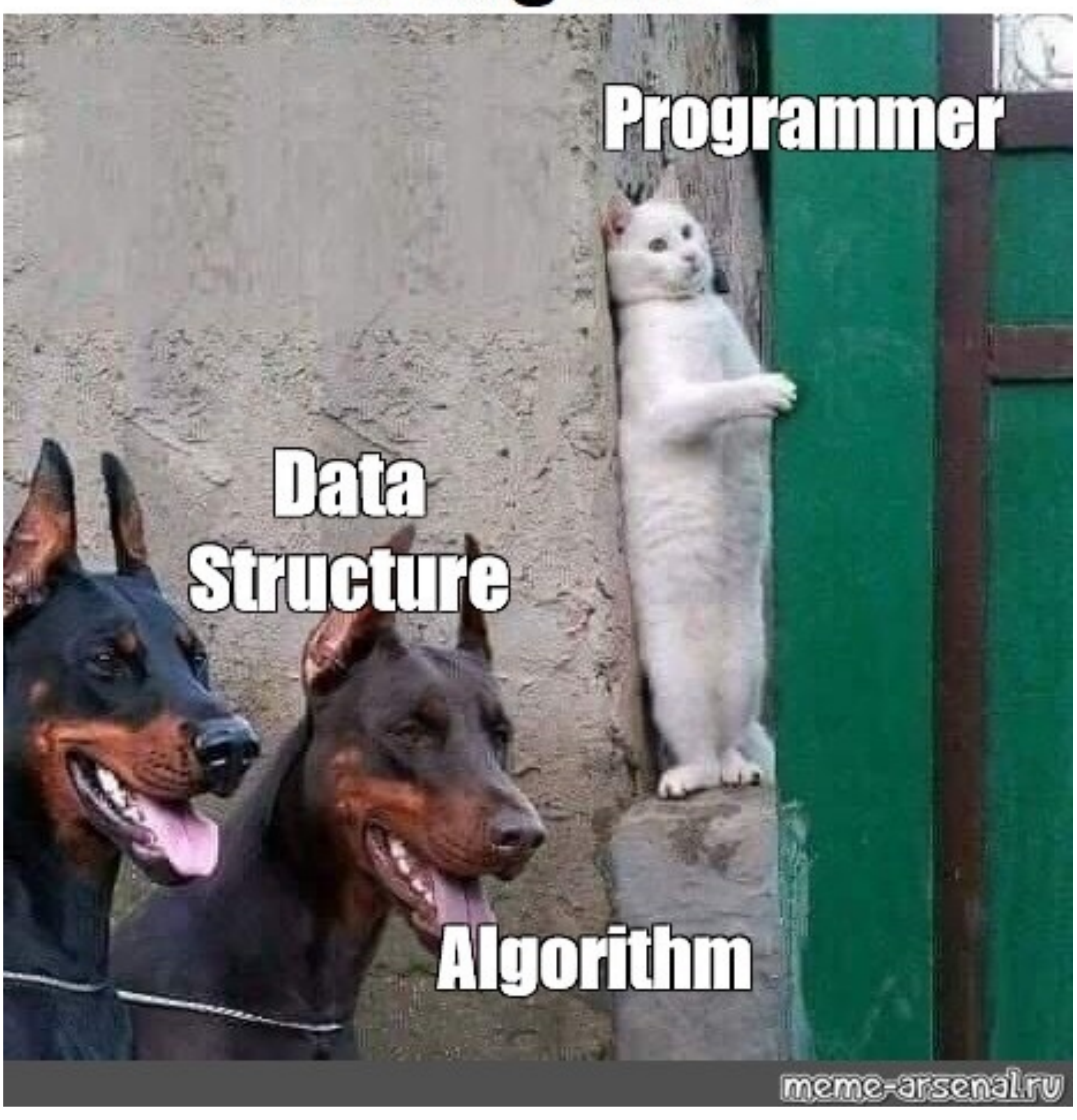
Introducción a Python

estructuras de datos

Miguel Carrasco
miguel.carrasco@uai.cl

- Estructuras de control
- Listas
  - Definición y ejemplos

Avengers 4



```
for  i in coleccion:
    #instrucciones de ciclo for
```

El ciclo for recorre los elementos que forman parte de la colección uno a uno.

Una colección puede ser:

1. un texto (recorrerá cada letra),
2. un conjunto de elementos (como entrega range) o
3. una lista

```
for i in 'hola':
    print(i) #imprime cada letra de hola
```

```
for i in range(1,11):
    print(i) #imprime los nros del 1 al 10
```

¿qué es una lista?

Una lista es una variable que puede guardar muchas cosas al mismo tiempo dentro.

Usamos como analogía una cajonera que puede contener variadas cosas.

¿Qué es una lista?

Algunas Características de las listas son:

Tienen un nombre

Cada espacio se identifica con un número que comienza en 0. Es decir el primer elemento es el lugar 0 de la lista. A este identificador lo llamaremos Índice.

Puedes guardar lo que quieras en cada espacio. Números, palabras o frases

El número que indentifica la posición en la lista lo llamaremos índice

¿Para qué sirve?

Para ahorrar código teniendo varios valores en una sola variable.

- Relacionar valores a un concepto en particular por ejemplo, el listado de las notas.
- Ordenar valores
- Trabajar con múltiples valores de forma ordenada y dinámica.

¿Cómo se ve?

```
data = [45,5,3,"hola","Otra cosa",22]
```

Leyendo valores

Los valores que se asignan a una lista, van separados por comas y entre corchetes.

```python
data = [45,5,3,"hola","Otra cosa"]
print(data[0])
```

```
45
```

El formato es el siguiente:

```
nombre_lista = [elementos]
```

Para obtener un elemento debo indicar su índice:

```python
print(data[0])
```

```python
data = [45,5,3,"hola","Otra cosa"]
print(data[3])
```

```
hola
```

Para mostrar todos los valores aprovechamos la estructura del ciclo for, de tal forma de escribir el código necesario y así recorrer toda la lista.

```
data = [45,5,3,"hola","Otra cosa"]
for elemento in data:
    print(elemento)
```

Esta forma recorre todos los elementos de la lista y permite realizar acciones con cada uno. El formato es el siguiente:

```
for elemento in lista:
    <<acciones>>
```

- **elemento** es una variable que guardará cada elemento de la lista
- **lista** es el nombre de la lista
- <<acciones>> es código que se ejecuta para cada elemento de la lista

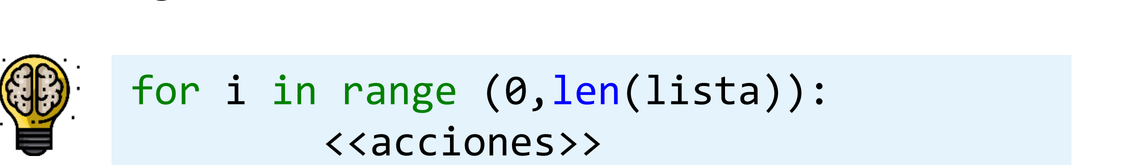
Valores de una lista

Otra forma de observar cada valor de la lista es usando el índice como identificador.

```
data = [45,5,3,"hola","Otra cosa"]
for i in range (0,len(data)):
    print(data[i])
```

```
45
5
3
Hola
Otra cosa
```

Dado que el índice está ordenado y sabemos el largo de la lista, fabricamos el rango en donde la variable i debe moverse. El formato es el siguiente:



- **i** es una variable que avanza de uno en uno
- **`len()`** es una función que nos entrega el largo de la lista
- <<acciones>> es código que se ejecuta para cada elemento de la lista

Ejemplo 1

Llene una lista con números enteros y calcule la suma de ellos

```python
lista = [3,56,5,38,2,3,765]
suma = 0
for elemento in lista:
    suma = suma + elemento
print(suma)
```

```python
lista = [3,56,5,38,2,3,765]
suma = 0
for i in range (0,len(lista)):
    suma = suma + lista[i]
print(suma)
```

```
872
```

Importante:

Si la lista tiene valores con texto, no podrá sumarlos. Ejemplo: lista = [1,2,4, “Hola”]

Ejemplo 2

Dada una lista con números enteros calcule, cuántos números pares hay en la misma

```python
lista = [3,56,5,38,2,3,765]
cont = 0

for elemento in lista:
    if elemento%2==0:
        cont = cont + 1
print("no. de pares es", cont)
```

```python
lista = [3,56,5,38,2,3,765]
cont = 0

for i in range(0,len(lista)):
    if lista[i]%2==0:
        cont = cont + 1
print("no. de pares es", cont)
```

```
no. de pares es 3
```

Ejemplo 3

Dada una lista con números enteros, encuentre el número menor de esta

```python
lista = [3,56,5,38,2,3,765]
menor = lista[0]

for elemento in lista:
    if element < menor:
        menor = elemento
print("menor elemento: ", menor)
```

```python
lista = [3,56,5,38,2,3,765]
menor = lista[0]

for i in range (0,len(lista)):
    if lista[i] < menor:
        menor = lista[i]
print("menor elemento: ", menor)
```

```
menor elemento: 2
```

Ejemplo 4

Dada una lista con números enteros, encuentre el número mayor de esta

```
lista = [3,56,5,38,2,3,765]
mayor = lista[0]

for elemento in lista:
    if elemento > mayor:
        mayor = elemento
print("mayor elemento: ", mayor)
```

```
lista = [3,56,5,38,2,3,765]
mayor = lista[0]

for i in range (0,len(lista)):
    if lista[i] > mayor:
        mayor = lista[i]
print("mayor elemento: ", mayor)
```

```
menor elemento: 2
```

- Estructuras de control
- Listas
  - Definición
  - Ingreso de datos

```
try:
    start = date(int(self.start_year.g
                self.months.index(se
                int(self.start_day.ge

    end = date(int(self.end_year.get()
                self.months.index(se
                int(self.end_day.get(
```

Esta bien ingresar datos por código, pero ¿cómo podemos hacer para que el usuario ingrese los datos?

Usamos los ciclos para hacerlo de forma dinámica porque no sabemos cuántos datos ingresarán.

```
print("ingrese el número de elementos: ")
num = int(input())

lista = []
for i in range (0,num):
    print("ingrese es elemento",i)
    elem = int(input())
    lista.append(elem)
```

lista vacía nos sirve para decirle al computador que usaremos una lista, esta queda vacía esperando a que ingresemos información

***.append*(<contenido>)** es un método que nos permite agregar un valor al final de la lista.

Esta bien ingresar datos por código, pero ¿cómo podemos hacer para que el usuario ingrese los datos?

Usamos los ciclos para hacerlo de forma dinámica porque no sabemos cuántos datos ingresarán.

```
print("ingrese el número de elementos: ")
num = int(input())

lista = []
for i in range (0,num):
    print("ingrese es elemento",i)
    elem = int(input())
    lista.append(elem)

print ("la lista es:")
for i in range (0,len(lista)):
    print(lista[i])
```

recorremos el listado posición por posición

```python
print("ingrese el num. de elementos: ")
num = int(input())

lista = []
for i in range (0,num):
    print("ingrese numero",i)
    lista.append(input())

print ("la lista es:")
for i in range (0,len(lista)):
    print(lista[i])
```

```
ingrese el num. de elementos
5
ingrese numero 0
4
ingrese numero 1
56
ingrese numero 2
786
ingrese numero 3
54
ingrese numero 4
5678
la lista es:
4
56
786
54
5678
```

Tiempo : 10 minutos

Genere una lista con 10 números enteros ingresados por el usuario y calcule su promedio

```python
print("ingrese numero de elementos ")
num = int(input())
lista = []

for i in range (0,num):
    print("ingrese numero",i)
    lista.append(input())

print ("la lista es:")
for i in range (0,len(lista)):
    print(lista[i])

suma=0
for elemento in lista:
    suma = suma + elemento

promedio = suma/len(lista)
print ("promedio es:", promedio)
```

```
ingrese número de elementos:
4
ingrese numero 0
3
ingrese numero 1
4
ingrese numero 2
3
ingrese numero 3
4
la lista es:
3
4
3
4
promedio es: 3.5
```

Tiempo : 10 minutos

Genere una lista con 10 números enteros ingresados por el usuario y muestre los valores en forma ordenada

```
ingrese número de elementos:
4
ingrese numero 0
6
ingrese numero 1
4
ingrese numero 2
3
ingrese numero 3
9
la lista ordenada es:
3
4
6
9
```

```python
n = int (input("ingrese numeros:"))

data =[]
for i in range(n):
  print("ingrese numero {}: ".format(i+1))
  data.append(int(input()))

data.sort()
print(data)
```

En resumen, gráficamente, un arreglo unidimensional, se puede ver gráficamente:

Observe que todas los elementos del arreglo tienen el mismo nombre

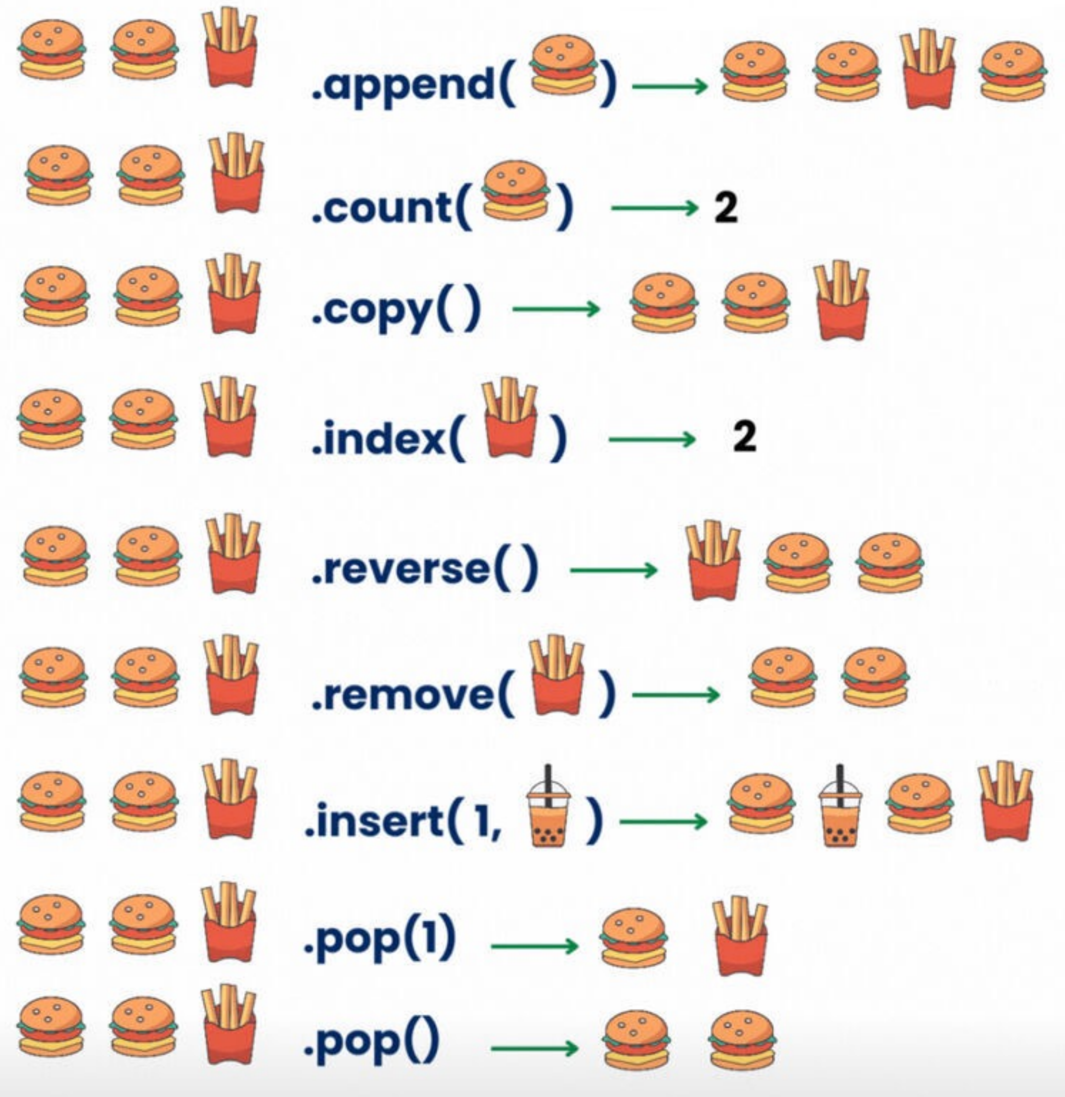
Índice del elemento en el arreglo A

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
A=	45	6	2	67	85	23	41	9

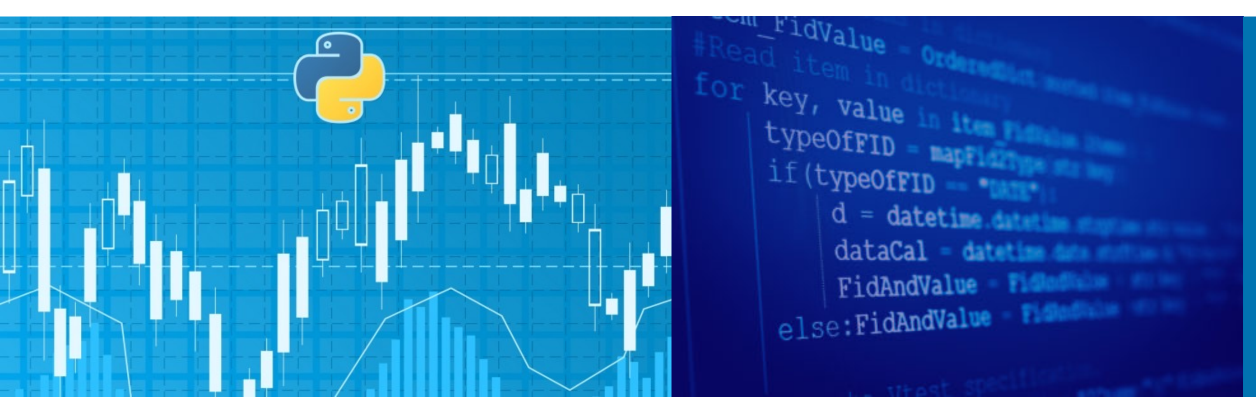
Otros Métodos

Las listas permiten realizar otras funciones además de insertar al final de la lista. A continuación se muestran algunas de

`clear()`	*remueve todos los elementos de la lista*
`copy()`	*retorna una copia de la lista*
`count()`	*retorna el número de elementos según un valor indicado (value)*
`extend()`	*agrega elementos al final de una lista (o iterable)*
`index()`	*retorna el índice del primer elemento según un valor indicado*
`insert()`	*inserta un elemento en una posición específica*
`pop()`	*remueve un elemento en una posición específica (opcional)*
`remove()`	*remueve el primer elemento según un valor específico*
`reverse()`	*reversa el orden de la lista*
`sort()`	*ordena la lista*


PYTHON LIST METHODS
.append( )
.count( )
2
.copy()
.index( )
2
.reverse()
.remove( )
.insert(1,)
.pop(1)
.pop()

- Estructuras de control
- Listas
- Diccionarios
  - Definición



Una diccionario es una tipo de datos estructurado que permite almacenar un conjunto de datos no ordenados de forma llave-valor de forma única

fuente: from https://www.redeszone.net/app/uploads-redeszone.net/2017/07/diccionario.jpg

Todos valores definidos en el diccionario están acompañados por una "llave" y un "valor" asociado

llave:valor

```python
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}

print(dcc)
```

"korea": "kia"

la llave es única, no pueden haber dos llaves con el mismo nombre. En caso que existan dos llaves con el mismo nombre se selecciona el ultimo valor

el valor puede ser un número, texto, una lista, o incluso otro diccionario

Podemos modificar un valor, accediendo a través de la llave:

```
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

```
dcc["francia"] = "peugeot"
print(dcc["francia"])
```

```
peugeot
```

Al indicar el nombre de la llave, podemos asignar un nuevo valor en esa llave

Podemos acceder a un valor del diccionario a través del nombre de las llaves, o bien a través del método get:

```
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

```
x = dcc.get("japon")
print(x)
```

```
toyota
```

.get es un método seguro ya que si existe la llave retornará none

Podemos agregar un valor, accediendo a través de la llave:

```python
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

```python
dcc["alemania"] = "bmw"
print(dcc)
```

```
{'usa': 'ford', 'japon':
'toyota', 'francia':
'citroen', 'italia': 'fiat',
'korea': 'kia', 'alemania':
'bmw'}
```

Al indicar el nombre de la llave, podemos asignar un nuevo valor en esa llave

Podemos eliminar un valor/llave, accediendo a través de la llave:

```
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

```
del dcc["japon"]
print(dcc)
```

```
{'usa': 'ford', 'francia': 'citroen', 'italia': 'fiat', 'korea': 'kia'}
```

con el comando **del** y el nombre de la llave podemos borrar el valor

Podemos recorrer el diccionario de forma de obtener las llaves, los valores o ambos según un determinado método que debemos definir:

```
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

A través del método **values** podemos acceder a los valores

```
for llave in dcc:
    print(llave)
```

```
usa
japon
francia
italia
korea
```

```
for valor in dcc.values():
    print(valor)
```

```
ford
toyota
citroen
fiat
kia
```

Podemos recorrer el diccionario de forma de obtener las llaves, los valores o ambos según un determinado método que debemos definir:

```
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

```
for llave, valor in dcc.items():
    print(llave, valor)
```

observe que el método items retorna dos resultados

```
usa ford
japon toyota
francia citroen
italia fiat
korea kia
```

Podemos recorrer el diccionario de forma de obtener las llaves, los valores o ambos según un determinado método que debemos definir:

```
dcc ={
"usa": "ford",
"japon" : "toyota",
"francia": "citroen",
"italia": "fiat",
"korea": "kia"
}
```

```
for llave in dcc:
    print(llave, dcc[llave])
```

```
usa ford
japon toyota
francia citroen
italia fiat
korea kia
```

En este otro método recorremos por las llaves obteniendo el mismo resultado

Podemos crear diccionarios de diccionarios a través de dos formas:

```
dcc ={
  "rock": {
    "autor": "Elvis",
    "year" : 1971
         },
  "clasica":{
    "autor": "Mozart",
    "year" : 1780
  },
  "jazz":{
    "autor": "Pat Metheny",
    "year": 1990
  }
}
```

```
tipo_1={
    "autor": "Elvis",
    "year" : 1971
}
tipo_2 ={
    "autor": "Mozart",
    "year" : 1780
}
tipo_3 = {
    "autor": "Pat Metheny",
    "year": 1990
}

dcc_nested = {
     "rock"    : tipo_1,
     "clasica": tipo_2,
     "jazz”    : tipo_3
}
```

Podemos recorrer el diccionario de forma de obtener las llaves, los valores o ambos según un determinado método que debemos definir:

```
dcc ={
  "rock": {
    "autor": "Elvis",
    "year" : 1971
          },
  "clasica":{
    "autor": "Mozart",
    "year" : 1780
  },
  "jazz":{
    "autor": "Pat Metheny",
    "year": 1990
  }
}
```

```
for o_key, o_value in dcc.items():
    print(o_value)
    for i_key, i_value in o_value.items():
         print(i_key, i_value)
```

Tiempo : 10 minutos

Empleando el siguiente diccionario, busque el autor que tenga el mayor año según lo señalado en el diccionario

```
dcc ={
  "rock": {
    "autor": "Elvis",
    "year" : 1971
         },
  "clasica":{
    "autor": "Mozart",
    "year" : 1780
  },
  "jazz":{
    "autor": "Pat Metheny",
    "year": 1990
  }
}
```

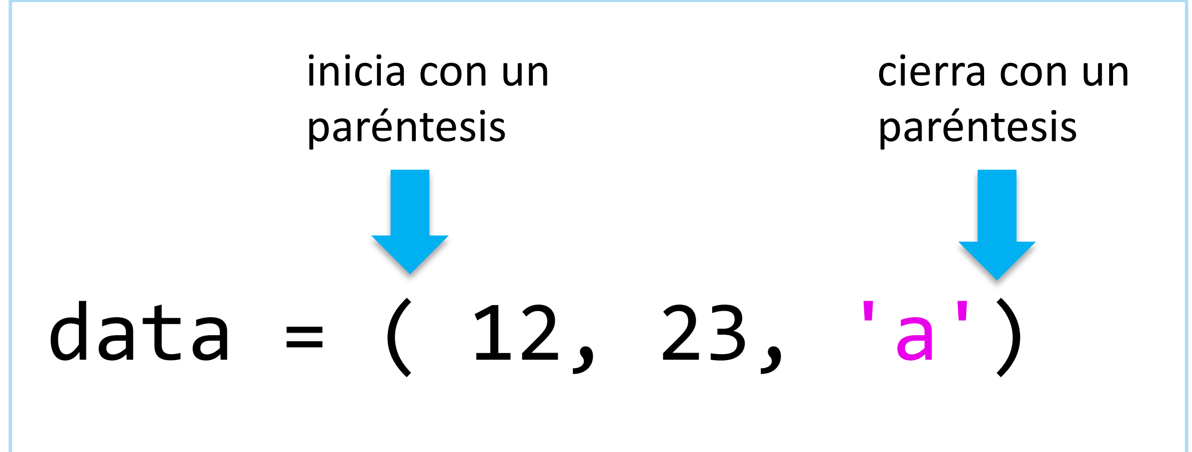
```
mx = dcc["rock"]["year"]

for o_key, o_value in dcc.items():
   for i_key, i_value in o_value.items():
      if (i_key=="year"):
         if (i_value>mx):
            mx = i_value

print("año mayor:", mx)
```

- Estructuras de control
- Listas
- Diccionarios
- Tuplas
  - Definición

Una tupla es otro tipo de datos estructurado definido internamente por Python. Las Tuplas nos permiten almacenar datos en una única variable. Sin embargo, la restricción más relevante de las tuplas es que sus datos NO pueden ser cambiados una vez creada



Los valores definidos en la tupla están separados por una coma. Los datos pueden ser caracteres, o números, o bien otra tupla

Las tuplas son

- ordenadas
- inmutables, es decir, no se pueden cambiar, borrar o insertar datos una vez creada.

```
data = ('Chile', 'Peru', 'Brasil')

print(data)
```

```
pib = (18, 15, 50)

print(pib)
```

Para acceder a los valores de las tuplas simplemente podemos indicar una posición, al igual que lo hacemos con las listas

Las tuplas son
- ordenadas
- inmutables, es decir, no se pueden cambiar, borrar o insertar datos una vez creada.

```
data = ('Chile', 'Perú', 'Brasil')

print(data[0])
```

→ Chile

```
sudamerica = ('Chile', 'Perú', 'Brasil')
norteamerica = ('Canada', 'Mexico', 'USA')

america = (sudamerica, norteamerica)
print(america[0][1])
```

→ Perú

Tupla: unpacking

Para acceder a los valores de las tuplas simplemente podemos indicar una posición, al igual que lo hacemos con las listas

Las tuplas son
- ordenadas
- inmutables, es decir, no se pueden cambiar, borrar o insertar datos una vez creada.

```
data = ('Chile', 'Perú', 'Brasil')

a, b, c = data

print(a) ──> Chile
print(b) ──> Perú
print(c) ──> Brasil
```

Las tuplas son objetos iterables. Esto significa que podemos recorrerlas por sus valores.

Como las tuplas son iterables, podemos recorrerlas

```
data = ('Chile', 'Peru', 'Brasil')

for pais in data:
    print(data)
```

Tupla: Unión

podemos unir dos tuplas a través del símbolo '+'

```
sudamerica = ('Chile', 'Perú', 'Brasil')
norteamerica = ('Canada', 'Mexico', 'USA')

america = sudamerica + norteamerica

print(america)
```

```
data = (1, 3, 5)

new_data = data*2

print(new_data)
```

(1,3,5,1,3,5)

Método count() permite contar el número de veces que un determinado elemento se encuentra en la tupla

```
nombres = ('Miguel', 'Carlos', 'Ana', 'Miguel', 'Andres')

contador = nombres.count('Miguel')

print(contador)
```

Método index() permite encontrar la posición de un determinado elemento dentro de la tupla. Solo indica la primera aparición

```
nombres = ('Miguel', 'Carlos', 'Ana', 'Miguel', 'Andres')

id = nombres.index('Carlos')

print(id)
```

Método in podemos determinar si existe un determinado valor en la tupla

```
nombres = ('Miguel', 'Carlos', 'Ana', 'Miguel', 'Andres')

if 'Miguel' in nombres:
    print('Miguel está en la lista')
else:
    print('No está en la lista')
```

Tuplas

COMBINE A LIST OF TUPLES
AND REGULAR LIST INTO DICTIONARY
THEN SUM BY UNIQUE VALUE OF TUPLE

imgflip.com

Temario

- Estructuras de control
- Listas
- Diccionarios
- Tuplas
- Conjuntos
  - Definición

```
try:
    start = date(int(self.start_year.g
                 self.months.index(se
                 int(self.start_day.ge

    end = date(int(self.end_year.get()
               self.months.index(se
               int(self.end_day.get(
```

sets (conjuntos)

Un set es otro tipo de datos definido internamente por Python. Los sets son la cuarta estructura de datos de Python. Entre sus principales propiedades se encuentran que no tiene orden, es posible agregar y eliminar items pero no podemos cambiarlos, y no tiene índices.

Las sets son

- no tienen orden
- no tienen índices
- no es posible cambiar valores

inicia con una llave

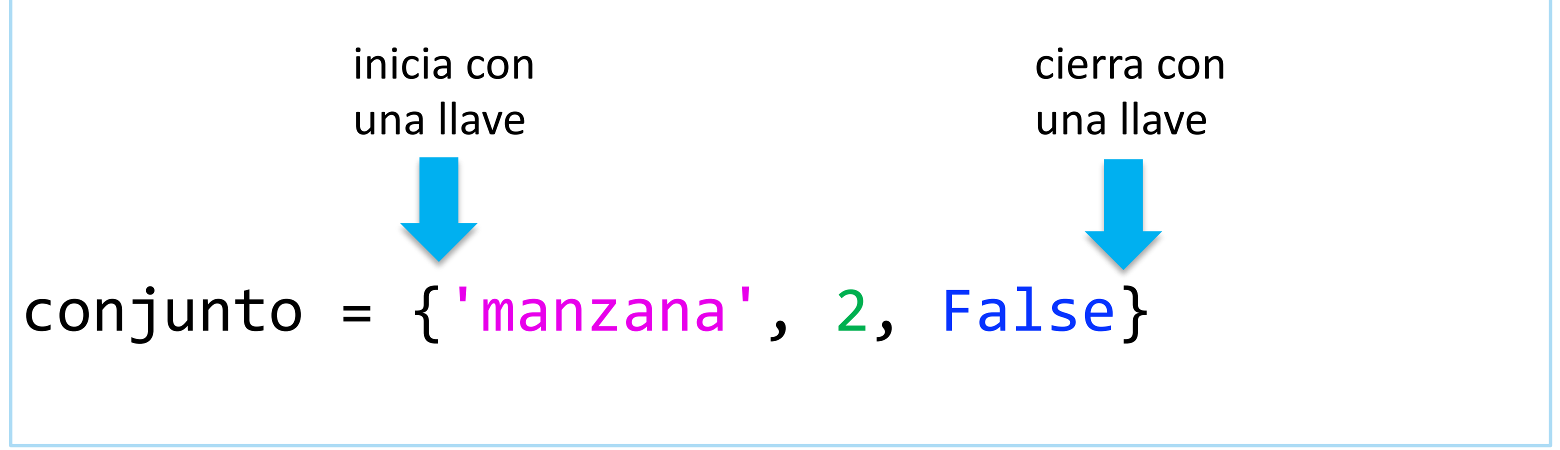
cierra con una llave

```
conjunto = {'manzana', 'pera', 'sandía'}
```

Al igual que otros tipos de datos, cada elemento de los conjuntos puede ser de otro tipo de datos (enteros, flotantes, strings, booleanos)

Las sets son

- no tienen orden
- no tienen índices
- no es posible cambiar valores



sets (conjuntos)

Podemos construir un conjunto a partir de una tupla, lista inclusive un diccionario, solo que en este último quedan almacenadas las llaves y no el contenido

Las sets son

- no tienen orden
- no tienen índices
- no es posible cambiar valores

```
lista = ['uno', False, 4]
tupla = ('uno', False, 4)
dcc = {0:'cero', 1:'uno'}

print(set(lista))   → {False, 4, 'uno'}
print(set(tupla))   → {False, 4, 'uno'}
print(set(dcc))     → {0, 1}
```

Los sets (conjuntos) son objetos iterables. Esto significa que podemos recorrerlas por sus valores.

Como loson iterables, podemos recorrerlas

```
conjunto = {'manzana', 2, False}

for data in conjunto:
    print(data)
```

2024 © INTRODUCCIÓN A PYTHON MIGUEL CARRASCO

Podemos unir dos o más conjuntos con el método update. Además los objetos a unir pueden ser otra estructura de datos (lista, tuplas, diccionarios)

```python
sudamerica = {'Chile', 'Perú', 'Brasil'}
norteamerica = {'Canada', 'Mexico', 'USA'}

america = set()

america.update(sudamerica)
america.update(norteamerica)

print(america)
```

conjunto vacío

Existen dos métodos para borrar elementos: remove y discard. El método remove genera un error si el elemento no se encuentra en el conjunto, en cambio discard no genera un error si el elemento no está presente

```
sudamerica = {'Chile', 'Perú', 'Brasil'}
norteamerica = {'Canada', 'Mexico', 'USA'}

sudamerica.remove('Chile')
norteamerica.discard('Colombia')

print(sudamerica)
print(norteamerica)
```

Con el método union podemos unir dos conjuntos. La ventaja de la unión es que cualquier elemento duplicado es eliminado.

```
sud_A = {'Chile', 'Perú', 'Brasil'}
sud_B = {'Chile', 'Colombia', 'Bolivia'}

sudamerica = sud_A.union(sud_B)

print(sudamerica)
```

```
{'Bolivia', 'Perú',
'Brasil', 'Colombia',
'Chile'}
```

Con el método intersecion podemos intersectar dos conjuntos, es decir, mantener los elementos que existen en ambos conjuntos

```
sud_A = {'Chile', 'Perú', 'Brasil'}
sud_B = {'Chile', 'Colombia', 'Bolivia'}

sudamerica = sud_A.intersection(sud_B)

print(sudamerica)
```

→ {'Chile'}

Con el método symmetric_difference podemos buscar todos los elementos que NO están presentes en ambos conjunto, es decir, el conjunto complemento de la intersección

```
sud_A = {'Chile', 'Perú', 'Brasil'}
sud_B = {'Chile', 'Colombia', 'Bolivia'}

sudamerica =
sud_A.symmetric_difference(sud_B)

print(sudamerica)
```

```
{'Bolivia',
'Brasil',
'Perú',
'Colombia'}
```

Otros Métodos

Los conjuntos permiten realizar otras operaciones. A continuación se muestran algunas de éstas

Método	Descripción
`clear()`	*remueve todos los elementos del conjunto*
`copy()`	*retorna una copia del conjunto*
`difference()`	*retorna un conjunto con las diferencias entre dos o más conjuntos*
`discard()`	*remueve un item específico del conjunto*
`intersection()`	*retorna un conjunto que es la intersección de dos o más conjuntos*
`isdisjoint()`	*retorna si dos conjuntos tienen una intersección o no*
`issubset()`	*retorna si un conjunto está contenido en otro o no*
`pop()`	*remueve un elemento del conjunto*
`union()`	*retorna la unión de dos conjuntos*
`update()`	*actualiza el conjunto con otro conjunto*

ME AFTER 10 LINES OF CODING
Enough For Today!