

# INTRODUCCIÓN A LA PROGRAMACIÓN

CON  python

DICCIONARIOS

## Estructuras de datos

Una **estructura de datos** es la **colección de datos** que se caracterizan por su **organización** y las **operaciones** que se definen en ella.

# Estructuras de datos

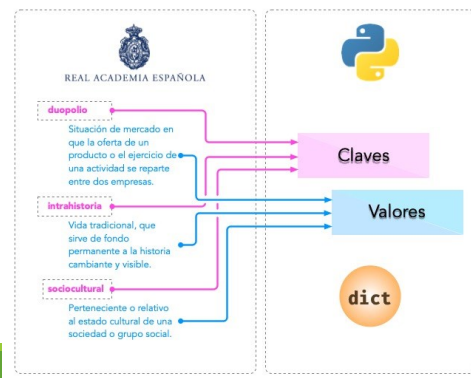
En general tenemos:

- cadenas
- listas
- tuplas
- **diccionarios**
- conjuntos

Nombre	Tipo	Ejemplo	Descripción
Lista	<code>list</code>	<code>[1, 2, 3]</code>	Secuencia heterogénea de datos mutable
Tupla	<code>tuple</code>	<code>1, 2, 3</code> o <code>(1, 2, 3)</code>	Secuencia heterogénea de datos inmutable
Rango	<code>range</code>	<code>range(1, 20, 2)</code>	Secuencia de enteros inmutable
Cadena de caracteres	<code>str</code>	<code>'Hola'</code>	Secuencia de caracteres inmutable
Diccionario	<code>dict</code>	<code>{'a':1, 'b':2, 'c':3}</code>	Tabla asociativa de valores únicos (clave, valor)
Conjunto	<code>set</code>	<code>{1, 2, 3}</code>	Colección sin orden de valores únicos

## Diccionarios

Podemos trasladar el concepto de diccionario de la vida real al de diccionario en Python. Al fin y al cabo un diccionario es un objeto que contiene palabras, y cada palabra tiene asociado un significado. Haciendo el paralelismo, diríamos que en Python un diccionario es también una colección indexada por claves (las palabras) que tienen asociados unos valores (los significados).



# Diccionarios

---

Los diccionarios en Python tienen las siguientes *características*:

- Mantienen el **orden** en el que se insertan las claves.
- Son **mutables**, con lo que admiten añadir, borrar y modificar sus elementos.
- Las **claves** deben ser **únicas**. A menudo se utilizan las *cadena de texto* como claves, pero en realidad podría ser cualquier tipo de datos inmutable: enteros, flotantes, tuplas (entre otros).
- Tienen un **acceso muy rápido** a sus elementos, debido a la forma en la que están implementados internamente.

# Diccionarios

---

En otros lenguajes de programación, a los diccionarios se les conoce como *arrays asociativos*, «*hashes*» o «*mapas*».



## Diccionarios. Creación

---

Un diccionario es una colección de elementos que son pares **clave:valor**

Para crear un diccionario usamos llaves {} rodeando pares clave: valor que están separados por comas.

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}
print(lenguajes)
```

Este ejemplo almacena los lenguajes de programación junto con su fecha de creación. Por pantalla se mostrará

```
{'C': 1972, 'Python': 1991, 'Java': 1996}
```

## Diccionarios. Creación

---

En lugar de crearlo a lo largo

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}
```

Otra manera de organizarlo en el código puede ser

```
lenguajes = {'C':1972,
             'Python':1991,
             'Java':1996}
```

## Diccionarios. Creación

---

Podemos crear un diccionario vacío

```
lenguajes = {}
```

Dado un diccionario vacío podemos construir un diccionario de la siguiente manera:

```
lenguajes['C'] = 1972  
lenguajes['Python'] = 1991  
lenguajes['Java'] = 1996  
print(lenguajes)
```

La ejecución del código anterior mostrará en pantalla:

```
{'C': 1972, 'Python': 1991, 'Java': 1996}
```

## Diccionarios. Creación

---

Podemos crear diccionarios a partir de listas asignando a todos el mismo valor:

```
lenguajes_antiguos = ['Assembler', 'Lisp', 'Pascal']  
lenguajes = dict.fromkeys(lenguajes_antiguos, 0)  
print(lenguajes)
```

La ejecución produce la siguiente salida:

```
{'Assembler': 0, 'Lisp': 0, 'Pascal': 0}
```

## Diccionario. Recuperar un valor

Para obtener un elemento de un diccionario basta con escribir la **clave** entre corchetes.

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
print(lenguajes['C'])
```

La ejecución mostrará:



Si la clave no existe

```
print(lenguajes['Kotlin'])
```

Abortará el programa con un error **KeyError**

## Diccionario. Recuperar un valor

Podemos evitar que se aborte el programa usando el método **get(clave)** que devolverá **None** si la clave no está en el diccionario.

```
lenguajes = { 'C':1972, 'python':1991, 'Java':1996}  
print(lenguajes.get('Kotlin'))
```

La ejecución mostrará:



## Diccionario. Recuperar un valor

Podemos pasarle al método `get` un argumento que se devolverá como valor por defecto si la clave no existe:

```
lenguajes = { 'C':1972, 'python':1991, 'Java':1996}
print(lenguajes.get('Kotlin', 'año desconocido'))
```

La ejecución mostrará:

```
año desconocido
```

## Diccionario. Añadir o modificar un elemento

Para añadir un elemento a un diccionario sólo es necesario hacer referencia a la clave y asignarle un valor:

- Si la clave **ya existía** en el diccionario, **se reemplaza** el valor existente por el nuevo.
- Si la clave **es nueva**, **se añade** al diccionario con su valor.

```
lenguajes = {'C':1972, 'Python':1991, 'Java':1996}
lenguajes['Kotlin'] = 2016
print(lenguajes)
```

Producirá la salida:

```
{'C': 1972, 'Python': 1991, 'Java': 1996, 'Kotlin': 2016}
```

## Diccionarios. Actividad

---

1. Crea una **función** que recibiendo de una cadena de texto con el siguiente formato:

<ciudad>:<habitantes>;<ciudad>:<habitantes>;<ciudad>:<habitantes>;....

Devuelve un diccionario con los siguientes elementos:

- Claves: ciudades.
- Valores: habitantes (como enteros)

Ejemplo:

**Entrada:** Tokyo:38\_140\_000;Delhi:26\_454\_000;Shanghai:24\_484\_000;Mumbai:21\_357\_000

**Salida:** {'Tokyo': 38140000, 'Delhi': 26454000, 'Shanghai': 24484000, 'Mumbai': 21357000, 'São Paulo': 21297000}

## Solución

---

```
data = 'Tokyo:38_140_000;Delhi:26_454_000;Shanghai:24_484_000;Mumbai:21_357_000'

cities = {}
for record in data.split(';'):
    city, population = record.split(':')
    cities[city] = int(population)

print(cities)
```



## Diccionarios. Longitud

---

La función `len()` devolverá la cantidad de elementos del diccionario pasado como argumento

```
lenguajes = { 'C':1972, 'python':1991, 'Java':1996}  
print(len(lenguajes))
```

La ejecución mostrará

```
3
```

## Diccionarios. Pertenencia de una clave

---

Para determinar si un elemento está o no en una lista se utilizan los operadores `in` y `not in`. Obtendremos como resultado un booleano, siendo `True` si el elemento está en la lista y `False` en caso contrario.

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
print('Java' in lenguajes)
```

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
print('Kotlin' not in lenguajes)
```

## Diccionarios. Actividad

---

2. Usando un diccionario, cuenta el número de veces que se repite cada letra en una cadena de texto dada.

Ejemplo:

- Entrada: 'boom'

- Salida: {'b': 1, 'o': 2, 'm': 1}

## Solución

---

```
sentence = 'supercalifragilisticexpialidocious'
letter_counter = {}
for letter in sentence:
    if letter in letter_counter:
        letter_counter[letter] += 1
    else:
        letter_counter[letter] = 1
print(letter_counter)
```

## Diccionarios. Eliminar un elemento

---

Para eliminar un elemento mediante la función `del()` indicando su clave:

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
del(lenguajes['Python'])  
print(lenguajes)
```

La ejecución mostrará:

```
{'C': 1972, 'Java': 1996}
```

## Diccionarios. Acceder y eliminar

---

Mediante el método `pop()` podemos extraer un elemento del diccionario por su clave. Vendría a ser una combinación de `get()` + `del()`:

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
print(lenguajes.pop('Python'))  
print(lenguajes)
```

Produce la salida

```
1991  
{'C': 1972, 'Java': 1996}
```

## Diccionarios. Borrado completo

Utilizando el método `clear()`

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}
lenguajes.clear()
print(lenguajes)
```

Produce la salida: `{}`

O simplemente asignando un diccionario vacío:

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}
lenguajes = {}
print(lenguajes)
```

## Diccionarios. Recorrer

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}
for clave in lenguajes:
    print(clave)
```

Produciendo la siguiente salida

```
C
Python
Java
```

A este proceso de recorrer una colección se le denomina enumerar o iterar.

## Diccionarios. Recorrer. Todos los valores

---

También podemos iterar por los valores usando el método `values()`. Aunque lo que devuelve no es una lista podemos usarla como tal

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
for valor in lenguajes.values():  
    print(valor)
```

La ejecución mostrará:

```
1972  
1991  
1996
```

## Diccionario. Lista de todos los valores

---

Si quiero todos los valores en una lista

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
valores = list(lenguajes.values())  
print(valores)
```

La ejecución mostrará:

```
[1972, 1991, 1996]
```

## Diccionarios. Recorrer. Todas las claves

---

Del mismo modo que hemos accedido a todos los valores almacenados en el diccionario, también podemos acceder a todas las claves usando el método `keys()`. Aunque lo que devuelve no es una lista podemos usarla como tal

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
for clave in lenguajes.keys():  
    print(clave)
```

```
C  
Python  
Java
```

## Diccionarios. Lista de todas las claves

---

Si quiero todas las claves en una lista

```
lenguajes = { 'C':1972, 'Python':1991, 'Java':1996}  
claves = list(lenguajes.keys())  
print(claves)
```

La ejecución mostrará:

```
['C', 'Python', 'Java']
```

## Diccionarios. Recorrer. Todos los elementos

---

Podemos iterar por ambos a la vez usando el método `items()`

```
for clave, valor in lenguajes.items():  
    print(clave, '->', valor)
```

Produciendo la siguiente salida:

```
C -> 1972  
Python -> 1991  
Java -> 1996
```

## Diccionarios. Actividad

---

3. Escribe una función `calcula_porcentajes()` que reciba el diccionario de ciudades y poblaciones ya visto, y suponiendo que estas ciudades son las únicas que existen en el planeta, calcule el porcentaje de población relativo de cada una de ellas con respecto al total devolviendo el diccionario correspondiente.

Ejemplo:

**Entrada:** {'Tokyo': 38140000, 'Delhi': 26454000, 'Shanghai': 24484000, 'Mumbai': 21357000, 'São Paulo': 21297000}

**Salida:** {'Tokyo': 28.95, 'Delhi': 20.08, 'Shanghai': 18.59, 'Mumbai': 16.21, 'São Paulo': 16.17}

# Soluciones

```
data = {
    'Tokyo': 38140000,
    'Delhi': 26454000,
    'Shanghai': 24484000,
    'Mumbai': 21357000,
    'São Paulo': 21297000,
}

# straightforward way
# total_population = sum(data.values())

total_population = 0
for population in data.values():
    total_population += population

for city in data:
    data[city] = (data[city] / total_population) * 100

print(data)
```

```
entrada = {'Tokyo': 38140000,
           'Delhi': 26454000,
           'Shanghai': 24484000,
           'Mumbai': 21357000,
           'São Paulo': 21297000}

lista = list(entrada.values())
suma = sum(lista)

salida = {}
for key, values in entrada.items():
    salida[key] = round(values/suma*100,2)
print(salida)
```

## Diccionarios. Combinar

Dados dos (o más) diccionarios, es posible «mezclarlos» para obtener una combinación de los mismos. Esta combinación se basa en dos premisas:

- Si la clave no existe, se añade con su valor.
- Si la clave ya existe, se añade con el valor del «último» diccionario en la mezcla.

Python ofrece dos mecanismos para realizar esta combinación. Vamos a partir de los siguientes diccionarios para ejemplificar su uso:

```
lenguajes1 = {'C':1972, 'Java':1990}
lenguajes2 = {'Kotlin': 2016, 'Java': 1996, 'Python':1991}
```



## Diccionarios. Combinar

Sin modificar los diccionarios originales:

```
lenguajes1 = {'C':1972, 'Java':1990}
lenguajes2 = {'Kotlin': 2016, 'Java': 1996, 'Python':1991}
lenguajes = lenguajes1 | lenguajes2
print(lenguajes)
```

Produce la salida:

```
{'C': 1972, 'Java': 1996, 'Kotlin': 2016, 'Python': 1991}
```

## Diccionarios. Combinar

Modificando los diccionarios originales:

```
lenguajes1 = {'C':1972, 'Java':1990}
lenguajes2 = {'Kotlin': 2016, 'Java': 1996, 'Python':1991}
lenguajes1.update(lenguajes2)
print(lenguajes1)
```

Produce la salida:

```
{'C': 1972, 'Java': 1996, 'Kotlin': 2016, 'Python': 1991}
```

 Nota

Tener en cuenta que el orden en el que especificamos los diccionarios a la hora de su combinación (mezcla) es relevante en el resultado final. En este caso *el orden de los factores sí altera el producto*.

## Diccionarios. Copia

Al igual que ocurría con las listas, para hacer una copia de un diccionario en otro debemos usar el método `copy()`

```
lenguajes = {'C':1972, 'Java':1996}
copia_lenguajes = lenguajes.copy()
lenguajes['C'] = 1970
print(lenguajes)
print(copia_lenguajes)
```

Al ejecutarse produce la salida:

```
{'C': 1970, 'Java': 1990}
{'C': 1972, 'Java': 1990}
```

## Diccionarios. Diccionarios por comprensión

De forma análoga a cómo se escriben las listas por comprensión, podemos aplicar este método a los diccionarios usando llaves `{ }`.

Veamos un ejemplo en el que creamos un diccionario por comprensión donde las claves son palabras y los valores son sus longitudes:

```
palabras = ['sun', 'space', 'rocket', 'earth']
longitud_palabras = {palabra:len(palabra) for palabra in palabras}
print(longitud_palabras)
```

Produce la salida:

```
{'sun': 3, 'space': 5, 'rocket': 6, 'earth': 5}
```

## Diccionarios. Diccionarios por comprensión

Si tengo las claves en una lista y los valores en otra puedo crear un diccionario con estas dos listas:

```
lenguajes = ['C', 'Java', 'Python']
años = [1972, 1996, 1991]
diccionario_lenguajes = {clave:valor for clave,valor in zip(lenguajes,años)}
print(diccionario_lenguajes)
```

## Diccionarios. Actividad

4. Tenemos almacenadas las notas de un examen en un diccionario. Es necesario separar al alumnado que aprobó y al que suspendió en dos diccionarios. Igualmente habrá que pasar a mayúsculas el nombre del alumnado que aprobó y a minúsculas el nombre del alumnado que suspendió. Escribe un programa en Python que realice esta operación usando diccionarios por comprensión.

Ejemplo

**Entrada:** {'John': 4, 'Marc': 7, 'Meryl': 2, 'Anthony': 8, 'Carol': 3, 'Andreas': 3, 'Sarah': 6}

**Salida:**

Aprobaron: {'MARC': 7, 'ANTHONY': 8, 'SARAH': 6}

Suspendieron: {'john': 4, 'meryl': 2, 'carol': 3}

## Solución

```
marks = {
    'John': 4,
    'Marc': 7,
    'Meryl': 2,
    'Anthony': 8,
    'Carol': 3,
    'Sarah': 6,
}

passed = {student.upper(): mark for student, mark in marks.items() if mark >= 5}
failed = {student.lower(): mark for student, mark in marks.items() if mark < 5}

print(f'Aprobaron: {passed}')
print(f'Suspendieron: {failed}')
```

## Diccionarios. Ordenación

En este apartado se ha podido observar que el diccionario se recorre en el orden que ha sido creado. Pero muchas veces es más interesante ordenar este por la clave o el valor. Para ello se puede emplear el función `sorted()`, al aplicarlo a un diccionario se obtendrán las claves ordenadas.

```
lenguajes = {'Python':1991, 'C':1972, 'Java':1996}
lenguajes_ordenados = sorted(lenguajes.items())
print(lenguajes_ordenados)
```

La ejecución de este código produce la salida:

```
[('C', 1972), ('Java', 1990), ('Python', 1991)]
```

Nótese que la función `sorted()` devuelve siempre una lista de tuplas. En esta la clave es el primer elemento y el valor el segundo. Por este motivo, para acceder a las claves es necesario utilizar dos índices. Para eso tenemos que **convertir a diccionario lo que devuelve la función `sorted()`**

```
lenguajes = {'Python':1991, 'C':1972, 'Java':1996}
lenguajes_ordenados = dict(sorted(lenguajes.items()))
print(lenguajes_ordenados)
```

## Anidamiento. Lista de diccionarios

El diccionario `animal_1` contiene la información acerca de un animal:

```
animal_1 = {'especie': 'perro', 'nombre': 'marco', 'edad': 13, 'sexo': 'M'}
```

Pero no tiene lugar para almacenar otro animal ni mucho menos un grupo de animales. ¿Cómo puedo manejar un grupo de animales? Una forma es hacer una lista de animales, dónde cada animal sea un diccionario que contenga información acerca del animal. Por ejemplo, el siguiente código construye una lista de tres animales.

```
animal_1 = {'especie': 'perro', 'nombre': 'marco', 'edad': 13, 'sexo': 'M'}
animal_2 = {'especie': 'perro', 'nombre': 'chica', 'edad': 2, 'sexo': 'H'}
animal_3 = {'especie': 'gato', 'nombre': 'felix', 'edad': 5, 'sexo': 'M'}

refugio = [animal_1, animal_2, animal_3]
for animal in refugio:
    print(animal)
```

## Anidamiento. Lista de diccionarios

Un ejemplo más realista involucra un refugio de más de 3 animales. Veamos de crear un refugio de 30 animales:

```
#creo una lista vacía para almacenar los animales del refugio
refugio = []

#creo 30 animales
for _ in range(30):
    especie = input("Especie: ")
    nombre = input("Nombre: ")
    edad = int(input("Edad: "))
    sexo = input("Sexo: ")
    animal = {'especie': especie, 'nombre': nombre, 'edad': edad, 'sexo': sexo}
    refugio.append(animal)

#muestro los 5 primeros animales
for animal in refugio[:5]:
    print(animal)

#muestro cuantos animales han sido creados
print('Número total de animales:', len(refugio))
```

## Anidamiento. Diccionario de listas

En lugar de poner un diccionario dentro de una lista a veces es útil poner una lista dentro de un diccionario. Por ejemplo, pensemos en una pizza cuando alguien pide una. Si fueses a utilizar solo una lista podrías almacenar en realidad los ingredientes. Con un diccionario, los ingredientes son una parte de la pizza que estás describiendo. En el siguiente ejemplo dos tipos de información estamos almacenando para la pizza: tamaño e ingredientes. La lista de ingredientes es un valor asociado a la clave 'ingredientes'.

```
#almacena información de la pizza que está siendo pedida
pizza = {'tamaño': 'pequeña',
        'ingredientes': ['setas', 'queso extra']}

#resumen del pedido
print("Has pedido una pizza",pizza['tamaño'], "con los siguientes ingredientes:")
for ingrediente in pizza['ingredientes']:
    print(ingrediente)
```

Este código produce la salida:

```
Has pedido una pizza pequeña con los siguientes ingredientes:
setas
queso extra
```

## Anidamiento. Diccionario de listas

Se puede anidar una lista dentro de un diccionario siempre que se quiera asociar más de un valor asociado a una sola clave en un diccionario. Otro ejemplo, puede ser querer almacenar los lenguajes de programación favoritos de distintas personas.

```
lenguajes_favoritos = {'Gaby': ['Python', 'Java'],
                      'Pedro': ['Kotlin', 'C++', 'Python'],
                      'Carmen': ['C']}

for nombre, lenguajes in lenguajes_favoritos.items():
    print("Los lenguajes favoritos de",nombre,"son: ")
    for lenguaje in lenguajes:
        print(lenguaje)
```


Este código produce la salida:

```
Los lenguajes favoritos de Gaby son:
Python
Java
Los lenguajes favoritos de Pedro son:
Kotlin
C++
Python
Los lenguajes favoritos de Carmen son:
C
```

# Anidamiento

---

No debes anidar las listas y los diccionarios en demasiada profundidad. Si estás anidando elementos mucho más profundos de lo que se ve en los ejemplos anteriores o estás trabajando con el código de otra persona con niveles significativos de anidación, lo más probable es que exista una más simple para resolver el problema.




## Anidamiento. Diccionario de diccionario

---

Puedes anidar un **diccionario dentro de otro diccionario**, pero su código puede complicarse rápidamente cuando lo haces.

Por ejemplo, si tienes varios usuarios para un sitio web, cada uno con un nombre de usuario único, puedes utilizar los nombres de usuario como claves en un diccionario. A continuación, puedes almacenar información sobre cada usuario utilizando un **diccionario** como el valor asociado a su **nombre** de usuario.



## Anidamiento. Diccionario de diccionario

En el siguiente listado, almacenamos tres piezas de información sobre cada usuario: **su nombre, apellido y ubicación**. Accederemos a esta información recorriendo los nombres de usuario y el diccionario de información asociada a cada nombre de usuario:

```
usuarios = {
    'aleros': {
        'nombre': 'Alejandro',
        'apellido': 'Rosales',
        'ubicacion': 'Chiclana de la Frontera'
    },
    'carpal': {
        'nombre': 'Carmen',
        'apellido': 'Palacios',
        'ubicacion': 'Cádiz'
    }
}

for nombre_usuario, info_usuario in usuarios.items():
    print("Nombre de usuario:", nombre_usuario)
    print("\tNombre y apellido:", info_usuario['nombre'], info_usuario['apellido'])
    print("\tUbicación:", info_usuario['ubicacion'])
```

```
Nombre de usuario: aleros
Nombre y apellido: Alejandro Rosales
Ubicación: Chiclana de la Frontera
Nombre de usuario: carpal
Nombre y apellido: Carmen Palacios
Ubicación: Cádiz
```

## Actividad:

- Haz un diccionario llamado **lugares\_favoritos**. Piensa en tres nombres de personas para utilizarlos como claves en el diccionario, y almacena de uno a tres lugares favoritos para cada persona. Para hacer este ejercicio un poco más interesante, pide a algunos amigos que nombren algunos de sus lugares favoritos. Haz un bucle en el diccionario e imprime el nombre de cada persona y sus lugares favoritos.
- Haz un diccionario llamado **ciudades**. Utiliza los nombres de tres ciudades como claves en tu diccionario. Crea un diccionario con información sobre cada ciudad e incluye el país en el que se encuentra la ciudad, su población aproximada y un dato sobre esa ciudad. Las claves del diccionario de cada ciudad deben ser algo así como país, población y dato. Imprime el nombre de cada ciudad y toda la información que has almacenado sobre ella.



## Diccionarios. Autoevaluación

---

1. ¿Cómo es un diccionario vacío?
2. ¿Cómo es un diccionario con clave 'María' y valor 42?
3. ¿Cuál es la principal diferencia entre diccionarios y listas?
4. ¿Qué pasa si accedes a ciudades['Londres'] si tu diccionario es {'Roma':123456}?
5. Si ciudades es un diccionario ¿cuál es la diferencia entre 'Roma' en ciudades y 'Roma' en ciudades.keys()?
6. Si ciudades es un diccionario ¿cuál es la diferencia entre 'Roma' en ciudades y 'Roma' en ciudades.values()?