Diccionarios

Algoritmos y Programación Básica (CC2005) - 2026

Diccionarios

Semestre 02, 2026

El problema con las listas

Tenemos una lista de estudiantes con sus notas.

```
estudiantes = ["Ana", "Carlos", "María"]
notas = [85, 90, 78]
```

¿Cómo sé a quién pertenece cada nota?

Solo por el índice — y eso es frágil.

El riesgo

Si se agrega un estudiante en la posición incorrecta:

```
estudiantes.insert(1, "Luis")
# notas[1] ya no corresponde a Carlos — es de Luis
```

El índice ya no tiene significado real.

Los datos están vinculados solo por su posición.

Lo que realmente queremos

Asociar un **nombre** con un **valor**.

```
"Ana"    →  85
"Carlos" →  90
"María"  →  78
```

Eso es exactamente lo que hace un diccionario.

Diccionarios

Un diccionario almacena pares **clave: valor**.

Se accede a los valores por su **clave**, no por su posición.

```
notas = {
    "Ana": 85,
    "Carlos": 90,
    "María": 78
}
```

- Las llaves `{ }` delimitan el diccionario.
- Cada par va separado por coma.
- La clave y el valor se separan con `:`.

Acceder a un valor

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

print(notas["Ana"])      # 85
print(notas["Carlos"])   # 90
```

Se accede por **clave**, no por índice.

`notas["Ana"]` significa: dame el valor asociado a "Ana".

Comparación directa

| | Lista | Diccionario | | --- | --- | --- | | Acceso | `notas[0]` — ¿qué es índice 0? | `notas["Ana"]` — claro y legible | | Significado | Depende de la posición | La clave describe el valor | | Orden | Importa | No importa para el acceso |

Las claves pueden ser de distintos tipos

```
config = {
    "debug": True,
    "version": 3,
    "nombre": "Mi App",
    "intentos_max": 5
}
```

Los valores también pueden ser de cualquier tipo.

```
persona = {
    "nombre": "Ana",
    "edad": 22,
    "activa": True,
    "cursos": ["CC2005", "CC3062"]
}
```

Operaciones básicas

Agregar o modificar una clave

```
notas = {"Ana": 85, "Carlos": 90}

# Modificar
notas["Ana"] = 92

# Agregar nueva clave
notas["María"] = 78

print(notas)   # {"Ana": 92, "Carlos": 90, "María": 78}
```

Si la clave ya existe → se actualiza el valor.

Si la clave no existe → se agrega el par.

Eliminar una clave

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

del notas["Carlos"]

print(notas)   # {"Ana": 85, "María": 78}
```

Verificar si una clave existe

```
notas = {"Ana": 85, "Carlos": 90}

if "Ana" in notas:
    print("Ana está registrada")

if "Luis" not in notas:
    print("Luis no está registrado")
```

Longitud del diccionario

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

print(len(notas))   # 3
```

`len()` devuelve el número de pares clave-valor.

Acceso seguro con get

Acceder con `[ ]` produce un error si la clave no existe.

```
notas = {"Ana": 85}

print(notas["Luis"])   # ERROR: KeyError
```

Solución: el método get

```
notas = {"Ana": 85}

print(notas.get("Luis"))          # None — sin error
print(notas.get("Luis", 0))       # 0 — valor por defecto
print(notas.get("Ana", 0))        # 85 — la clave sí existe
```

`get(clave, valor_por_defecto)` devuelve el valor si la clave existe, o el valor por defecto si no.

Cuándo usar get vs [ ]

| Situación | Método | | --- | --- | | La clave siempre debe existir | `diccionario["clave"]` | | La clave puede no existir | `diccionario.get("clave", defecto)` |

Métodos de diccionarios

keys — todas las claves

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

print(notas.keys())   # dict_keys(["Ana", "Carlos", "María"])
```

values — todos los valores

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

print(notas.values())   # dict_values([85, 90, 78])
```

items — todos los pares

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

print(notas.items())
# dict_items([("Ana", 85), ("Carlos", 90), ("María", 78)])
```

update — actualizar con otro diccionario

```
base = {"Ana": 85, "Carlos": 90}
nuevos = {"María": 78, "Ana": 92}

base.update(nuevos)

print(base)   # {"Ana": 92, "Carlos": 90, "María": 78}
```

Las claves que ya existen se actualizan.

Las claves nuevas se agregan.

pop — eliminar y obtener el valor

```
notas = {"Ana": 85, "Carlos": 90}

valor = notas.pop("Carlos")

print(valor)   # 90
print(notas)   # {"Ana": 85}
```

Tabla de métodos

| Método | ¿Qué hace? | | --- | --- | | `keys()` | Devuelve todas las claves | | `values()` | Devuelve todos los valores | | `items()` | Devuelve pares (clave, valor) | | `get(k, d)` | Devuelve el valor de `k` o `d` si no existe | | `update(d)` | Fusiona con otro diccionario | | `pop(k)` | Elimina y devuelve el valor de `k` |

Iterar un diccionario

Iterar sobre las claves

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

for nombre in notas:
    print(nombre)
```

Resultado:

```
Ana
Carlos
María
```

Iterar directamente sobre el diccionario recorre las **claves**.

Iterar sobre claves y valores

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

for nombre, nota in notas.items():
    print(f"{nombre}: {nota}")
```

Resultado:

```
Ana: 85
Carlos: 90
María: 78
```

`items()` devuelve cada par como una tupla — se desempaqueta directamente en dos variables.

Calcular promedio iterando

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}
total = 0

for nota in notas.values():
    total = total + nota

promedio = total / len(notas)
print(f"Promedio: {promedio}")
```

Resultado:

```
Promedio: 84.33333333333333
```

La mejor estructura: lista de diccionarios

Una lista de diccionarios representa una tabla de datos.

Cada diccionario es una **fila**.

Las claves son los **encabezados de columna**.

Ejemplo: tabla de estudiantes

```
estudiantes = [
    {"nombre": "Ana",    "nota": 85, "seccion": "A"},
    {"nombre": "Carlos", "nota": 90, "seccion": "B"},
    {"nombre": "María",  "nota": 78, "seccion": "A"},
    {"nombre": "Luis",   "nota": 62, "seccion": "B"},
]
```

| nombre | nota | seccion | | --- | --- | --- | | Ana | 85 | A | | Carlos | 90 | B | | María | 78 | A | | Luis | 62 | B |

¿Por qué es la mejor estructura?

- Cada fila es independiente — un diccionario completo.
- Los campos tienen nombre — no dependen del índice.
- Fácil de leer, escribir y mantener.
- Directamente compatible con CSV, JSON y bases de datos.

Acceder a un campo

```
print(estudiantes[0]["nombre"])   # Ana
print(estudiantes[2]["nota"])     # 78
```

Primero el índice de la fila, luego la clave del campo.

Iterar la tabla

```
for estudiante in estudiantes:
    print(f"{estudiante['nombre']}: {estudiante['nota']}")
```

Resultado:

```
Ana: 85
Carlos: 90
María: 78
Luis: 62
```

Filtrar filas

```
aprobados = []

for estudiante in estudiantes:
    if estudiante["nota"] >= 61:
        aprobados.append(estudiante)

for e in aprobados:
    print(e["nombre"], "—", e["seccion"])
```

Resultado:

```
Ana — A
Carlos — B
María — A
Luis — B
```

Agregar una fila nueva

```
nuevo = {"nombre": "Sofía", "nota": 95, "seccion": "A"}
estudiantes.append(nuevo)

print(len(estudiantes))   # 5
```

Modificar un campo

```
estudiantes[0]["nota"] = 88

print(estudiantes[0])   # {"nombre": "Ana", "nota": 88, "seccion": "A"}
```

Archivos CSV

CSV = **Comma Separated Values** (valores separados por coma).

Es el formato más usado para compartir datos tabulares.

Se abre en Excel, Google Sheets, o cualquier editor de texto.

Ejemplo de archivo CSV

`estudiantes.csv`:

```
nombre,nota,seccion
Ana,85,A
Carlos,90,B
María,78,A
Luis,62,B
```

La primera fila son los **encabezados**.

Cada fila siguiente es un registro.

Leer CSV con Python puro

```
archivo = open("estudiantes.csv", "r")
lineas = archivo.readlines()
archivo.close()
```

```
encabezados = lineas[0].strip().split(",")

estudiantes = []

for linea in lineas[1:]:
    valores = linea.strip().split(",")

    fila = {}
    for i in range(len(encabezados)):
        fila[encabezados[i]] = valores[i]

    estudiantes.append(fila)

print(estudiantes[0])
```

Resultado:

```
{"nombre": "Ana", "nota": "85", "seccion": "A"}
```

- `readlines()` lee todas las líneas como una lista de strings.
- `strip()` elimina el salto de línea al final.
- `split(",")` divide por coma.
- Se construye un diccionario por fila y se agrega a la lista.

Limitación del método manual

Los valores siempre se leen como **strings**.

Para usar la nota como número hay que convertirla:

```
fila["nota"] = int(fila["nota"])
```

Hay que hacerlo campo por campo.

Pandas

Pandas es una biblioteca de Python para análisis de datos.

Diseñada para trabajar con datos tabulares de forma eficiente.

Maneja conversiones de tipos, datos faltantes y operaciones sobre columnas automáticamente.

Instalar pandas

```
pip install pandas
```

Importar pandas

```
import pandas as pd
```

`pd` es el alias estándar — se usa en todo el ecosistema de ciencia de datos.

Leer un CSV con pandas

```
import pandas as pd

df = pd.read_csv("estudiantes.csv")

print(df)
```

Resultado:

```
  nombre  nota seccion
0    Ana    85       A
1  Carlos    90       B
2   María    78       A
```

```
3   Luis    62       B
```

Una sola línea — pandas detecta encabezados, tipos y estructura automáticamente.

¿Qué es un DataFrame?

`df` es un **DataFrame** — la estructura principal de pandas.

Es una tabla bidimensional con:

- **filas** numeradas automáticamente (índice).
- **columnas** nombradas (los encabezados del CSV).

Es la versión profesional de nuestra lista de diccionarios.

Acceder a una columna

```
print(df["nombre"])
```

Resultado:

```
0      Ana
1    Carlos
2     María
3      Luis
Name: nombre, dtype: object
```

Acceder a una fila

```
print(df.iloc[0])   # primera fila por índice
```

Resultado:

```
nombre     Ana
nota        85
seccion      A
Name: 0, dtype: object
```

Estadísticas en una línea

```
print(df["nota"].mean())    # 78.75 — promedio
print(df["nota"].max())     # 90 — máximo
print(df["nota"].min())     # 62 — mínimo
print(df["nota"].sum())     # 315 — suma
```

Filtrar filas

```
aprobados = df[df["nota"] >= 61]
print(aprobados)
```

Resultado:

```
  nombre  nota seccion
0    Ana    85       A
1  Carlos    90       B
2   María    78       A
3   Luis    62       B
```

Información general del DataFrame

```
print(df.info())
```

Resultado:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
```

```
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   nombre   4 non-null      object
 1   nota     4 non-null      int64
 2   seccion  4 non-null      object
```

Muestra tipos de datos, cantidad de filas y valores nulos.

Convertir DataFrame a lista de diccionarios

```
estudiantes = df.to_dict(orient="records")

print(estudiantes[0])
# {"nombre": "Ana", "nota": 85, "seccion": "A"}
```

`to_dict("records")` convierte cada fila en un diccionario.

Puente entre pandas y la estructura que ya conocemos.

Python puro vs pandas

| Aspecto | Python puro | Pandas | | --- | --- | --- | | Líneas de código | Muchas | Pocas | | Conversión de tipos | Manual | Automática | | Datos faltantes | Manejo manual | Automático | | Estadísticas | Programar desde cero | `mean()`, `max()`, `sum()` | | Velocidad en datasets grandes | Lenta | Muy rápida | | Curva de aprendizaje | Ninguna (ya lo sabemos) | Requiere aprendizaje |

Ejemplo completo

Leer un CSV de productos y calcular el valor total del inventario.

`productos.csv`:

```
nombre,precio,cantidad
Laptop,5000,10
Monitor,1200,25
Teclado,350,50
Mouse,150,100
```

Con pandas

```
import pandas as pd

df = pd.read_csv("productos.csv")

df["total"] = df["precio"] * df["cantidad"]

print(df)
print(f"\nValor total del inventario: Q{df['total'].sum()}")
```

Resultado:

```
    nombre  precio  cantidad   total
0    Laptop    5000        10   50000
1   Monitor    1200        25   30000
2   Teclado     350        50   17500
3     Mouse     150       100   15000

Valor total del inventario: Q112500
```

Con Python puro

```
archivo = open("productos.csv", "r")
lineas = archivo.readlines()
archivo.close()

encabezados = lineas[0].strip().split(",")
productos = []
```

```
for linea in lineas[1:]:
    valores = linea.strip().split(",")
    fila = {encabezados[i]: valores[i] for i in range(len(encabezados))}
    fila["precio"] = int(fila["precio"])
    fila["cantidad"] = int(fila["cantidad"])
    fila["total"] = fila["precio"] * fila["cantidad"]
    productos.append(fila)

total_inventario = 0
for p in productos:
    total_inventario = total_inventario + p["total"]
    print(f"{p['nombre']}: Q{p['total']}")

print(f"\nValor total: Q{total_inventario}")
```

Mismo resultado — más líneas de código.

Errores comunes

Error 1: KeyError — clave inexistente

```
notas = {"Ana": 85}

print(notas["Luis"])   # ERROR: KeyError: 'Luis'
```

Solución: usar `get` o verificar con `in`.

```
print(notas.get("Luis", 0))

if "Luis" in notas:
    print(notas["Luis"])
```

Error 2: confundir listas y diccionarios

```
# Lista — acceso por índice
frutas = ["manzana", "naranja"]
print(frutas[0])       # manzana

# Diccionario — acceso por clave
stock = {"manzana": 10, "naranja": 5}
print(stock["manzana"])   # 10
print(stock[0])           # ERROR: 0 no es una clave
```

Error 3: olvidar que get devuelve None

```
notas = {"Ana": 85}

resultado = notas.get("Luis")
print(resultado + 10)   # ERROR: no se puede sumar None + 10
```

Siempre proveer un valor por defecto cuando se usará el resultado.

```
resultado = notas.get("Luis", 0)
print(resultado + 10)   # 10
```

Error 4: modificar el diccionario al iterar

```
notas = {"Ana": 85, "Carlos": 90, "María": 78}

for nombre in notas:
    if notas[nombre] < 80:
        del notas[nombre]   # ERROR: no se puede modificar durante iteración
```

Solución: construir una lista de claves a eliminar primero.

```
a_eliminar = [n for n in notas if notas[n] < 80]
for nombre in a_eliminar:
    del notas[nombre]
```

Buenas prácticas

- Usar diccionarios cuando los datos tienen **nombre** — no solo posición.
- Usar lista de diccionarios para representar tablas de datos.
- Preferir `get` sobre `[ ]` cuando la clave puede no existir.
- Usar pandas para leer CSV — maneja tipos y casos especiales automáticamente.
- Usar `to_dict("records")` para convertir un DataFrame a lista de diccionarios.

¿Lista o diccionario?

| Situación | Estructura | | --- | --- | | Colección de elementos del mismo tipo | Lista | | Datos con nombre (campos) | Diccionario | | Tabla de registros | Lista de diccionarios | | Análisis de datos de un CSV | pandas DataFrame |