

TEMA 3. ESTRUCTURAS DE DATOS

Diccionarios

- También denominados “matrices asociativas”
- Relacionan una clave y un valor, es decir, el acceso a los elementos se realiza a través de una clave en lugar de un índice
- Operaciones básicas: creación y asignación de nuevas entradas

```
#Crear un diccionario vacío  
agenda = {}  
  
#Añadir nuevos elementos  
agenda['Juan'] = '623458356'  
agenda['Maria'] = '673513895'  
print(agenda)
```

```
print(agenda['Juan'])  
print(agenda.get('Maria'))  
print(agenda['Marta'])
```

```
{'Juan': '623458356', 'Maria': '673513895'}
```

```
623458356  
673513895  
KeyError: 'Marta'
```

TEMA 3. ESTRUCTURAS DE DATOS

Diccionarios

- Recuperación de claves y recuperación de valores

`keys()`

`values()`

```
print(agenda.keys())  
print(agenda.values())
```

```
dict_keys(['Juan', 'Maria'])  
dict_values(['623458356', '673513895'])
```

- Comprobación de clave: `in`

```
'Marta' in agenda
```

False

- Eliminar entradas: `pop(key)`

```
agenda.pop('Juan')  
print(agenda)
```

```
{'Maria': '673513895'}
```

- List comprehension sobre diccionarios

```
agenda['Marta'] = '621337248'  
[agenda[k] for k in agenda]
```

```
['673513895', '621337248']
```

TEMA 3. ESTRUCTURAS DE DATOS

Diccionarios: datos anidados

```
dicc = {'tamano': 'mediana',  
        'precio': 15.67,  
        'toppings': ['champinones', 'queso extra', 'pepperoni', 'albahaca'],  
        'cliente':  
            {'nombre': 'Jane Doe',  
             'telefono': '455-344-234',  
             'correo': 'janedoe@email.com'}}
```

Cuando se accede a un dato de un diccionario, se procesa como ese tipo de dato (lista, diccionario, tipo básico, etc.)

```
print(dicc['precio'])  
print(dicc['toppings'])  
print(len(dicc['toppings']))  
print(dicc['toppings'][2])  
print(dicc['cliente'])  
print(dicc['cliente']['telefono'])
```

15.67

['champinones', 'queso extra', 'pepperoni', 'albahaca']

4

pepperoni

{'nombre': 'Jane Doe', 'telefono': '455-344-234', 'correo': 'janedoe@email.com'}

455-344-234

TEMA 3. ESTRUCTURAS DE DATOS

JSON

- **JavaScript Object Notation**, es un formato que utiliza texto interpretable por humanos para transmitir objetos de datos (alternativa a **XML**).
- **Consiste en pares de atributo:valor**, entre llaves { }. Los pares atributo:valor están separados por comas
- **Ejemplo:**

```
{  
    "tamano": "mediana",  
    "precio": 15.67,  
    "toppings": ["champinones", "pepperoni", "albahaca"],  
    "queso_extra": false,  
    "delivery": true,  
    "cliente": {  
        "nombre": "Jane Doe",  
        "telefono": null,  
        "correo": "janedoe@email.com"  
    }  
}
```

TEMA 3. ESTRUCTURAS DE DATOS

JSON: tipos básicos

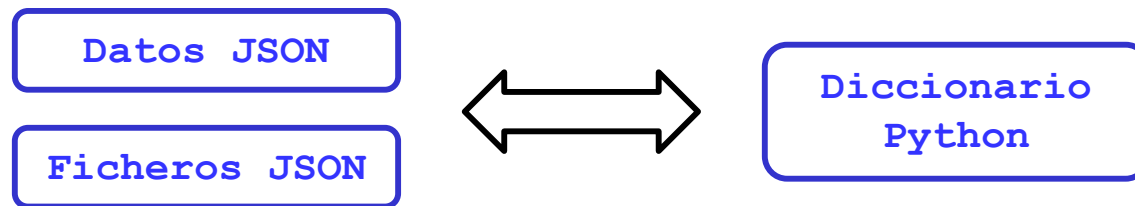
- **Number, String, Boolean**
- **Array:** lista ordenada de cero o más valores, entre corchetes y separados por comas
- **Object:** matriz asociativa no-ordenada (pares atributo:valor), entre llaves y separados con comas; las claves son strings y deben ser distintas de otras claves dentro del mismo objeto.
- **null**

```
{
  String      "tamano": "mediana",
  Number (float) "precio": 15.67,
  Array      "toppings": ["champinones", "pepperoni", "albahaca"],
  Boolean     "queso_extra": false,
  Boolean     "delivery": true,
  Object     "cliente": {
    "nombre": "Jane Doe",
    "telefono": null,
    "correo": "janedoe@email.com"
  }
}
```

TEMA 3. ESTRUCTURAS DE DATOS

JSON y Python

- **La estructura del formato JSON y de los diccionarios de Python es similar**
- **El módulo json permite leer, escribir y procesar datos y ficheros JSON en Python fácilmente, como si fueran diccionarios**



- **En primer lugar, es necesario importarlo en el programa:**

```
import json
```

TEMA 3. ESTRUCTURAS DE DATOS

JSON como cadena de caracteres

- **loads(str):** Convertir un string en formato JSON en un diccionario
- **Podemos trabajar con este diccionario como con cualquier diccionario en Python**

```
datos_JSON = """
{
    "tamano": "mediana",
    "precio": 15.67,
    "toppings": ["champinones", "queso extra", "pepperoni", "albahaca"],
    "cliente": {
        "nombre": "Jane Doe",
        "telefono": "455-344-234",
        "correo": "janedoe@email.com"
    }
}
"""

datos_diccionario = json.loads(datos_JSON)
datos_diccionario
```

```
Out[2]: {'tamano': 'mediana',
        'precio': 15.67,
        'toppings': ['champinones', 'queso extra', 'pepperoni', 'albahaca'],
        'cliente': {'nombre': 'Jane Doe',
                     'telefono': '455-344-234',
                     'correo': 'janedoe@email.com'}}
```

TEMA 3. ESTRUCTURAS DE DATOS

JSON como cadena de caracteres

- **dumps(str): Convertir un diccionario en un string en formato JSON**

```
cliente = {  
    "nombre": "Nora",  
    "telefono": "455-266-123",  
    "correo": "nora@mimail.com",  
    "delivery": False  
}  
  
# Obtener una cadena de caracteres JSON  
cliente_JSON = json.dumps(cliente)  
cliente_JSON
```

```
Out[5]: '{"nombre": "Nora", "telefono": "455-266-123",  
        "correo": "nora@mimail.com", "delivery": false}'
```

```
# Obtener una cadena de caracteres JSON  
cliente_JSON = json.dumps(cliente, indent=3, sort_keys=True)  
print(cliente_JSON)  
  
{  
    "correo": "nora@mimail.com",  
    "delivery": false,  
    "nombre": "Nora",  
    "telefono": "455-266-123"  
}
```

JSON	Python
object	dict
array	list
string	str
número (int)	int
número (real)	float
true	True
false	False
null	None

TEMA 3. ESTRUCTURAS DE DATOS

JSON como fichero

- Generalmente, el formato **JSON** se utilizar para almacenar información en ficheros. El módulo `json` facilita el procesamiento de estos ficheros.
- Leer un fichero **JSON** y guardarlo en un diccionario:

```
with open(nombre_fichero) as f:  
    dicc = json.load(f)
```

- Escribir un diccionario en un fichero de texto en formato **JSON**:

```
with open(nombre_fichero, "w") as f:  
    json.dump(dicc, f)
```

TEMA 3. ESTRUCTURAS DE DATOS

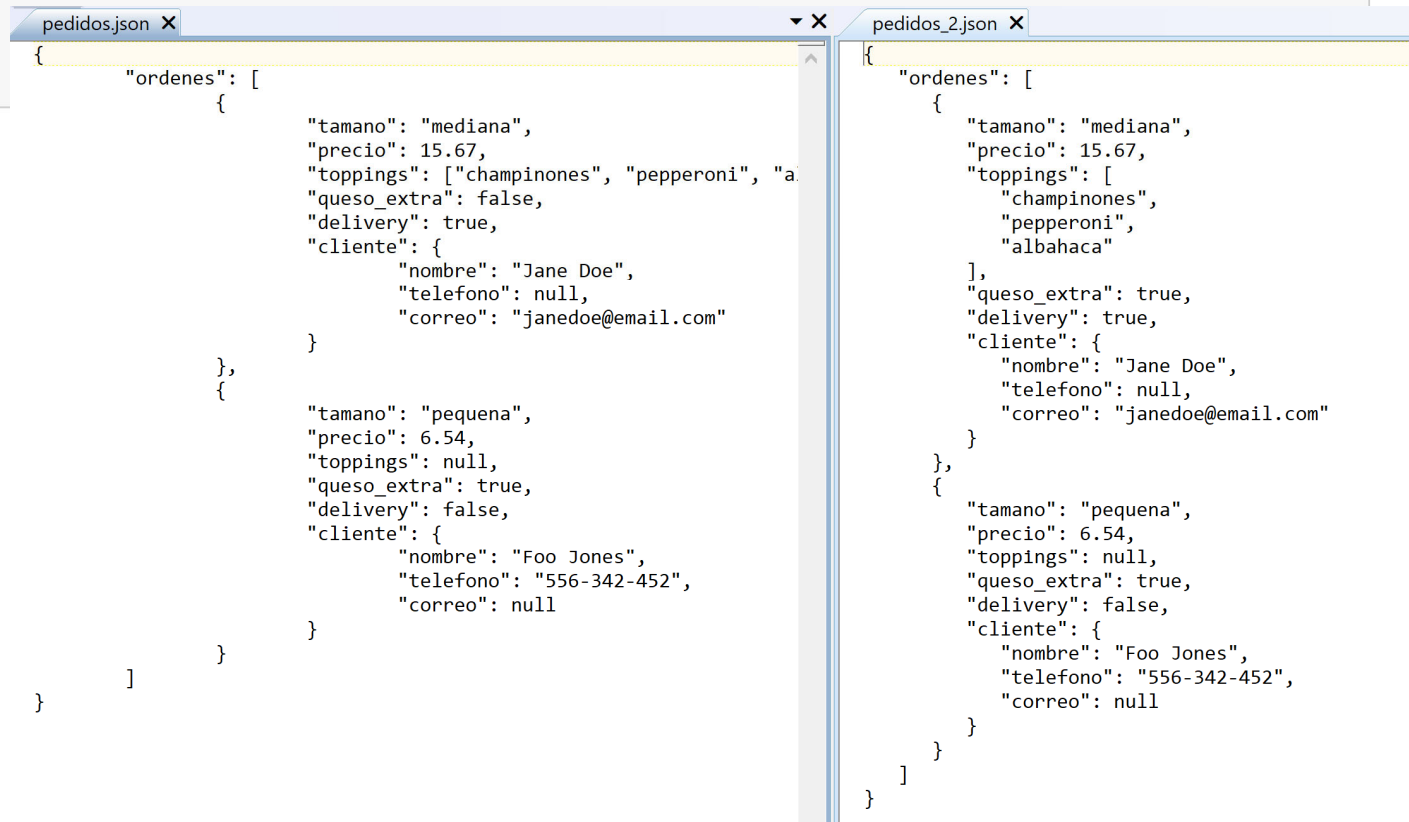
JSON como fichero

```
with open("pedidos.json") as f:
    dicc = json.load(f)

print(dicc)
```

```
{'ordenes': [{ 'tamano': 'mediana', 'precio': 15.67, 'toppings': ['champinones', 'pepperoni', 'albahaca'], 'queso_extra': False,
'delivery': True, 'cliente': { 'nombre': 'Jane Doe', 'telefono': None, 'correo': 'janedoe@email.com'}}, { 'tamano': 'pequena', 'p
recio': 6.54, 'toppings': None, 'queso_extra': True, 'delivery': False, 'cliente': { 'nombre': 'Foo Jones', 'telefono': '556-342
-452', 'correo': None}}]}
```

```
dicc['ordenes'][0]['queso_extra']=True
with open("pedidos_2.json", "w") as f:
    json.dump(dicc, f, indent=3)
```



```
pedidos.json {
  "ordenes": [
    {
      "tamano": "mediana",
      "precio": 15.67,
      "toppings": ["champinones", "pepperoni", "a
      "queso_extra": false,
      "delivery": true,
      "cliente": {
        "nombre": "Jane Doe",
        "telefono": null,
        "correo": "janedoe@email.com"
      }
    },
    {
      "tamano": "pequena",
      "precio": 6.54,
      "toppings": null,
      "queso_extra": true,
      "delivery": false,
      "cliente": {
        "nombre": "Foo Jones",
        "telefono": "556-342-452",
        "correo": null
      }
    }
  ]
}

pedidos_2.json {
  "ordenes": [
    {
      "tamano": "mediana",
      "precio": 15.67,
      "toppings": [
        "champinones",
        "pepperoni",
        "albahaca"
      ],
      "queso_extra": true,
      "delivery": true,
      "cliente": {
        "nombre": "Jane Doe",
        "telefono": null,
        "correo": "janedoe@email.com"
      }
    },
    {
      "tamano": "pequena",
      "precio": 6.54,
      "toppings": null,
      "queso_extra": true,
      "delivery": false,
      "cliente": {
        "nombre": "Foo Jones",
        "telefono": "556-342-452",
        "correo": null
      }
    }
  ]
}
```

TEMA 3. ESTRUCTURAS DE DATOS

GeoJSON

- Formato para codificar una variedad de estructuras de datos geográficos.
- Tipos de objetos **GeoJSON**:
 - Geometría: **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, **GeometryCollection**
 - Feature: contiene una geometría y propiedades adicionales
 - Colección de Features: lista de Features
- <http://geojson.org/geojson-spec.html>

TEMA 3. ESTRUCTURAS DE DATOS

GeoJSON: Ejemplo

```
{ "type": "Feature",  
  "properties": {  
    "nombre": "IGLESIA PARROQUIAL DE SAN ESTEBAN",  
    "numpol": "5",  
    "idnotes": "004197",  
    "codvia": "67590",  
    "telefono": "963918276",  
    "ruta": "5" },  
  "geometry": {  
    "type": "Point",  
    "coordinates": [ -0.373245739750917, 39.476100796615739 ]  
  }  
}
```