Unit 6 Lesson 3: Working with JSON

Quick Reference Guide

📦 Import

```
import json
```

📖 Reading JSON Files

Basic File Reading

```
# Read JSON file
with open("employee.json", "r") as file:
    data = json.load(file)

# data is now a Python dictionary or list
print(data["name"])
print(data["salary"])
```

Reading JSON with Lists

```
# employees.json contains an array
with open("employees.json", "r") as file:
    employees = json.load(file)

# employees is a list of dictionaries
for emp in employees:
    print(f"{emp['name']}: ${emp['salary']}")
```

✍️ Writing JSON Files

Basic File Writing

```
# Create Python data
employee = {
    "employee_id": "E001",
    "name": "Jennifer Martinez",
    "department": "Sales",
    "salary": 65000,
    "active": True
}

# Write to JSON file
with open("employee.json", "w") as file:
    json.dump(employee, file, indent=4)
```

Writing Lists

```
employees = [
    {"employee_id": "E001", "name": "Jennifer Martinez", "salary": 65000},
    {"employee_id": "E002", "name": "Michael Chen", "salary": 70000}
]

with open("employees.json", "w") as file:
    json.dump(employees, file, indent=4)
```

🔤 Working with JSON Strings

Parse String to Python

```
# JSON from an API or user input
json_string = '{"customer": "Acme Corp", "total": 1500.00}'

# Convert string to Python dictionary
data = json.loads(json_string)  # Note the 's'!
print(data["customer"])
```

Convert Python to String

```
data = {"customer": "Acme Corp", "total": 1500.00}

# Convert dictionary to JSON string
json_string = json.dumps(data, indent=4)  # Note the 's'!
print(json_string)
```

📚 JSON Methods Summary

Method	Purpose	Input	Output
`json.load(file)`	Read from file	File object	Python object
`json.loads(string)`	Parse string	JSON string	Python object
`json.dump(data, file)`	Write to file	Python object	JSON file
`json.dumps(data)`	Create string	Python object	JSON string

Memory Tip: The 's' stands for "string"

- `load` / `dump` = files
- `loads` / `dumps` = strings

📊 JSON Data Types

JSON Type	Python Type	Example
`string`	`str`	`"Jennifer Martinez"`
`number`	`int` or `float`	`65000` or `3.14`
`boolean`	`bool`	`true` or `false`
`null`	`None`	`null`
`array`	`list`	`["Sales", "Marketing"]`
`object`	`dict`	`{"name": "John"}`

Important: JSON uses lowercase `true`, `false`, and `null`

🔍 Accessing Nested Data

```
company = {
    "name": "TechCorp",
    "location": {
        "city": "San Francisco",
        "state": "CA"
    },
    "departments": ["Sales", "IT", "Marketing"]
}

# Access nested dictionary
print(company["location"]["city"])  # San Francisco

# Access list items
print(company["departments"][0])  # Sales

# Safely access with .get()
email = company.get("email", "Not provided")
```

Pretty Printing (indent parameter)

```
data = {"name": "John", "salary": 65000}

# Without indent - compact but hard to read
json.dump(data, file)
# Output: {"name":"John","salary":65000}

# With indent=4 - readable and formatted
json.dump(data, file, indent=4)
# Output:
# {
#     "name": "John",
#     "salary": 65000
# }
```

Best Practice: Always use `indent=4` for files humans will read

Read-Modify-Write Pattern

```
import json

# 1. READ - Load existing data
with open("inventory.json", "r") as file:
    products = json.load(file)

# 2. MODIFY - Change the data
for product in products:
    product["price"] = product["price"] * 1.10  # 10% increase

# 3. WRITE - Save changes back
with open("inventory.json", "w") as file:
    json.dump(products, file, indent=4)
```

⚠️ Common Mistakes

Mistake 1: Confusing load/loads

```
# ❌ Wrong - trying to load file with loads
with open("data.json") as f:
    data = json.loads(f)  # Error!

# ✅ Correct
with open("data.json") as f:
    data = json.load(f)  # No 's' for files
```

Mistake 2: Forgetting indent

```
# ❌ Hard to read
json.dump(data, file)

# ✅ Readable
json.dump(data, file, indent=4)
```

Mistake 3: JSON syntax errors

```
# ❌ Python syntax (won't work in JSON)
{
    'name': 'John',  # Single quotes not allowed
    name: 'John',    # Keys must be quoted
    True: 'value'    # Must be lowercase: true
}

# ✅ Valid JSON
{
    "name": "John",
    "active": true
}
```

🛡️ Error Handling

```
import json

# Handle file not found
try:
    with open("config.json", "r") as file:
        config = json.load(file)
except FileNotFoundError:
    print("Config file not found, using defaults")
    config = {"theme": "default", "language": "en"}

# Handle invalid JSON
try:
    data = json.loads(json_string)
except json.JSONDecodeError:
    print("Invalid JSON format!")
```

💼 Business Use Cases

Configuration Files

```
config = {
    "app_name": "SalesTracker Pro",
    "database": {
        "host": "db.company.com",
        "port": 5432
    },
    "features": {
        "email_notifications": True,
        "export_excel": True
    }
}

with open("config.json", "w") as file:
    json.dump(config, file, indent=4)
```

API Responses

```
# Simulated API response
api_response = '{"status": "success", "data": {"revenue": 45000, "customers": 127}}'

# Parse the response
result = json.loads(api_response)
print(f"Revenue: ${result['data']['revenue']:,}")
```

User Profiles

```
user_profile = {
    "user_id": "U12345",
    "name": "Sarah Johnson",
    "preferences": {
        "theme": "dark",
        "notifications": True,
        "language": "en"
    },
    "recent_orders": [
        {"order_id": "ORD001", "total": 250.00},
        {"order_id": "ORD002", "total": 180.50}
    ]
}
```

Product Catalog

```
products = [
    {
        "product_id": "P001",
        "name": "Laptop",
        "price": 1200.00,
        "specs": {
            "ram": "16GB",
            "storage": "512GB SSD",
            "processor": "Intel i7"
        },
        "in_stock": True
    }
]
```

🆚 CSV vs JSON Comparison

Feature	CSV	JSON
Structure	Flat table	Nested objects
Data Types	Text only	Multiple types
Readability	Simple	More complex
File Size	Smaller	Larger
Use Case	Reports, logs	APIs, configs
Nested Data	❌ No	✅ Yes
Excel Compatible	✅ Yes	⚠️ Limited

When to use CSV:

- Tabular data (rows and columns)
- All records have same fields
- Working with spreadsheets
- Large datasets (more efficient)

When to use JSON:

- Complex nested structures
- Different fields per record
- API communication
- Configuration files
- Hierarchical data

🎯 Quick Tips

1. **Always use** `indent=4` for human-readable JSON files
2. **Remember the 's'**: `loads` / `dumps` for strings, `load` / `dump` for files
3. **JSON keys must be strings** (unlike Python dicts)
4. **Use lowercase** for `true`, `false`, `null` in raw JSON

5. **Validate JSON** online at jsonlint.com if you're editing by hand
6. **Use try-except** when loading JSON from external sources

---

📖 Complete Example

```
import json

# Create business data
company_data = {
    "company": "TechCorp Inc",
    "year_founded": 2018,
    "employees": [
        {
            "employee_id": "E001",
            "name": "Jennifer Martinez",
            "department": "Sales",
            "salary": 65000,
            "skills": ["negotiation", "CRM", "presentations"]
        },
        {
            "employee_id": "E002",
            "name": "Michael Chen",
            "department": "IT",
            "salary": 78000,
            "skills": ["Python", "SQL", "Cloud"]
        }
    ],
    "offices": {
        "headquarters": {
            "city": "San Francisco",
            "state": "CA"
        },
        "branch": {
            "city": "Austin",
            "state": "TX"
        }
    }
}

# Save to file
with open("company.json", "w") as file:
    json.dump(company_data, file, indent=4)

print("✅ Data saved to company.json")
```

```python
# Load from file
with open("company.json", "r") as file:
    loaded_data = json.load(file)

# Access nested data
print(f"Company: {loaded_data['company']}")
print(f"HQ: {loaded_data['offices']['headquarters']['city']}")
print(f"Employees: {len(loaded_data['employees'])}")

# Modify and save
loaded_data["year_founded"] = 2019
with open("company.json", "w") as file:
    json.dump(loaded_data, file, indent=4)

print("✅ Data updated")
```