12. Working with different data formats

September 17, 2025

0.1 1. Working with the file system (os, os.path)

0.1.1 os

The os module contains functions to get information on local directories, files, processes, and environment variables.

`os.getcwd()` - returns the current working directory

[40]:
```
import os
current_path = os.getcwd()
print(current_path)
```

```
/Users/iulia/PycharmProjects/python-training-mar2025/docs
```

`os.listdir(path)` - returns a list of all the entries in the directory given by `path`

[41]:
```
os.listdir(current_path)
```

[41]:
```
['my_module.py',
 '09. More data structures.ipynb',
 'images',
 '01. Introduction.ipynb',
 'output.json',
 '13. Working with databases.ipynb',
 'users.json',
 '__pycache__',
 '12. Working with different data formats.ipynb',
 'output.csv',
 'data.csv',
 '14. Decorators.ipynb',
 'data.json',
 '07. Modules.ipynb',
 'books.csv',
 'clients.json',
 '06. Methods on known data types.ipynb',
 '15. Object-Oriented Programming.ipynb',
 '02. Python basics.ipynb',
 'file_example_out.txt',
 '.ipynb_checkpoints',
 '08. Introduction to File Handling.ipynb',
 '04. Lists and Tuples.ipynb',
 'transactions.csv',
 '11. Introduction to Object-Oriented Programming.ipynb',
 'ceva',
 '05. Functions.ipynb',
 '10. Exception Handling.ipynb',
 '99. Project.ipynb',
 'file_example_in.txt',
 '03. Control Flow.ipynb']
```

`os.mkdir(path)` - creates a directory

`os.makedirs(path)` - creates directory recursively, by adding eventual missing directories

[42]:
```
os.mkdir('testdir')
assert 'testdir' in os.listdir(current_path)
```

`os.chdir()` - changes the current working directory

[43]:
```
os.chdir('testdir')
print('Items in testdir:', os.listdir())
os.chdir(current_path)
```

```
Items in testdir: []
```

`os.rename(source, dest)` - renames the file or directory

[44]:
```
os.rename('testdir', 'new_testdir')
assert 'testdir' not in os.listdir(current_path)
assert 'new_testdir' in os.listdir(current_path)
```

`os.remove(path)` - removes a file

`os.rmdir(path)` - removes the directory path

`os.removedirs(path)` - Removes directories recursively

[45]:
```
os.rmdir('new_testdir')
assert 'new_testdir' not in os.listdir(current_path)
```

`os.walk(path)` - Directory tree generator. For each directory in the directory tree rooted at top, yields a 3-tuple `dirpath, dirnames, filenames`:

- `dirpath` is a string, the path to the directory.
- `dirnames` is a list of the names of the subdirectories in `dirpath` (excluding '.' and '..').
- `filenames` is a list of the names of the non-directory files in `dirpath`.

[46]:
```
for dirpath, dirnames, filenames in os.walk('.'):
    print(dirpath, dirnames, filenames)
```

```
. ['images', '__pycache__', '.ipynb_checkpoints', 'ceva'] ['my_module.py', '09.
More data structures.ipynb', '01. Introduction.ipynb', 'output.json', '13.
Working with databases.ipynb', 'users.json', '12. Working with different data
formats.ipynb', 'output.csv', 'data.csv', '14. Decorators.ipynb', 'data.json',
'07. Modules.ipynb', 'books.csv', 'clients.json', '06. Methods on known data
types.ipynb', '15. Object-Oriented Programming.ipynb', '02. Python
basics.ipynb', 'file_example_out.txt', '08. Introduction to File
Handling.ipynb', '04. Lists and Tuples.ipynb', 'transactions.csv', '11.
Introduction to Object-Oriented Programming.ipynb', '05. Functions.ipynb', '10.
Exception Handling.ipynb', '99. Project.ipynb', 'file_example_in.txt', '03.
Control Flow.ipynb']
./images ['.ipynb_checkpoints'] ['M-X.png', 'python_installation_windows.png',
'pycharm_installation_mac.png', 'python_installation.png', 'A-L.png',
'pycharm_installation_windows.png', 'Y-Z.png']
./images/.ipynb_checkpoints [] ['A-L-checkpoint.png', 'M-X-checkpoint.png']
./__pycache__ [] ['my_module.cpython-313.pyc', 'my_module.cpython-312.pyc']
./.ipynb_checkpoints [] ['02. Python basics-checkpoint.ipynb', '13. Working with
databases-checkpoint.ipynb', '09. More data structures-checkpoint.ipynb', '08.
Introduction to File Handling-checkpoint.ipynb', '01. Introduction-
checkpoint.ipynb', '06. Methods on known data types-checkpoint.ipynb', '11.
Introduction to Object-Oriented Programming-checkpoint.ipynb', '14. Decorators-
checkpoint.ipynb', '07. Modules-checkpoint.ipynb', 'file_example_in-
checkpoint.txt', 'file_example_out-checkpoint.txt', '10. Exception Handling-
checkpoint.ipynb', '05. Functions-checkpoint.ipynb', '15. Object-Oriented
Programming-checkpoint.ipynb', '12. Project-checkpoint.ipynb', '12. Working with
different data formats-checkpoint.ipynb', '03. Control Flow-checkpoint.ipynb',
'04. Lists and Tuples-checkpoint.ipynb']
./ceva [] []
```

0.1.2 `os.path`

`os.path` contains functions for manipulating filenames and directory names.

`os.path.exists(path)` - test whether a path exists

[47]:
```
os.path.exists(current_path)
```

[47]:
```
True
```

`os.path.isfile(path)` - test whether a path is a regular file

[48]:
```
os.path.isfile(current_path)
```

[48]:
```
False
```

`os.path.isdir(path)` - return true if the pathname refers to an existing directory

[49]:
```
os.path.isdir(current_path)
```

[49]:
```
True
```

`os.path.split(path)` - split a pathname; returns tuple `(head, tail)` where `tail` is everything after the final slash

[50]:
```
os.path.split(current_path)
```

[50]:
```
('/Users/iulia/PycharmProjects/python-training-mar2025', 'docs')
```

`os.path.join(path, "new_var")` - join two or more pathname components, inserting `os.sep` as needed.

[51]:
```
os.path.join(current_path, 'testdir', 'innerdir')
```

[51]:
```
'/Users/iulia/PycharmProjects/python-training-mar2025/docs/testdir/innerdir'
```

0.2 Exercises 1

1. Write a Python program that creates a directory `outdir` at the current location and a directory `innerdir` inside `outdir`. Create an empty file inside `innerdir`. Use `os.walk()` to print the directory tree for `outdir`. Remove the directories and the file.

0.3 2. Working with paths in the file system (`pathlib`)

The `pathlib` module provides an object-oriented approach to handling file system paths. It allows you to work with file and directory paths in a more intuitive and Pythonic way than traditional string manipulation.

The main class in this module is `Path`, which represents a file or directory path:

[52]:
```
from pathlib import Path
```

Using this class, we can instantiate it to create paths:

[53]:
```
current_dir = Path()
root_path = Path("/")
relative_path = Path("images/pycharm_installation_mac.png")
```

`Path` objects attributes:

[54]:
```
relative_path.parent  # parent directory
```

[54]:
```
PosixPath('images')
```

[55]:
```
relative_path.stem  # final path component, without its suffix
```

[55]:
```
'pycharm_installation_mac'
```

[56]:
```
relative_path.suffix  # file extension
```

[56]:
```
'.png'
```

`Path` objects can be used to build new paths:

[57]:
```
python_path = root_path / "usr" / "bin" / "python3"
python_path
```

[57]:
```
PosixPath('/usr/bin/python3')
```

Methods of `Path` objects to check various properties:

[58]:
```
python_path.exists()
```

[58]:
```
True
```

[59]:
```
python_path.is_file()
```

[59]:
```
True
```

[60]:
```
python_path.is_dir()
```

[60]:
```
False
```

The `Path` class also implements functions in `os` module used for directory manipulation, as methods:

[61]:
```
new_dir = current_dir / "new_dir"
new_dir.mkdir(exist_ok=True)

for directory in current_dir.iterdir():
    if directory.is_dir():
        print(directory)
```

```
images
__pycache__
new_dir
.ipynb_checkpoints
ceva
```

[62]:
```
for txt_file in current_dir.glob("**/*.txt"):
    print(txt_file)
```

```
file_example_out.txt
file_example_in.txt
.ipynb_checkpoints/file_example_in-checkpoint.txt
.ipynb_checkpoints/file_example_out-checkpoint.txt
```

[63]:
```
new_dir.rmdir()
```

[64]:
```
for dirpath, dirnames, filenames in current_dir.walk():
    print(dirpath, dirnames, len(filenames))
```

```
. ['images', '__pycache__', '.ipynb_checkpoints', 'ceva'] 27
images ['.ipynb_checkpoints'] 7
images/.ipynb_checkpoints [] 2
__pycache__ [] 2
```

```
.ipynb_checkpoints [] 18
ceva [] 0
```

0.4 Exercises 2

1. Solve the same exercises above (Exercises 1), but using `pathlib` module.

0.5 3. Working with JSON format

0.5.1 What is JSON?

JSON (JavaScript Object Notation) is a lightweight data format used for exchanging data between systems. It is easy for humans to read and write and easy for machines to parse and generate. JSON is based on key-value pairs and supports basic data types such as strings, numbers, arrays, and objects.

Example:

```
{
  "name": "Alice",
  "age": 30,
  "isEmployed": true,
  "skills": ["Python", "SQL", "JavaScript"],
  "address": {
    "city": "New York",
    "zipcode": "10001"
  }
}
```

0.5.2 json module

The `json` module in Python provides methods to **encode** (convert Python objects to JSON) and **decode** (convert JSON to Python objects).

- `json.loads()` - parses a JSON string and converts it into a Python object.

[65]:
```
import json

json_string = '{"name": "Alice", "age": 30}'
data = json.loads(json_string)
```

[66]:
```
data
```

[66]:
```
{'name': 'Alice', 'age': 30}
```

[67]:
```
type(data)
```

[67]:
```
dict
```

- `json.dumps()` - converts a Python object into a JSON string.

```
[68]: json_string = json.dumps(data)
```

```
[69]: json_string
```

```
[69]: '{"name": "Alice", "age": 30}'
```

- `json.load()` - reads JSON data from a file and converts it into a Python object.

```
[70]: with open("data.json", "r") as f:
          data = json.load(f)
```

```
[71]: data
```

```
[71]: {'name': 'Mia', 'hobbies': ['painting', 'jogging']}
```

```
[72]: data["age"] = 20
      data["hobbies"].append("cooking")
      data
```

```
[72]: {'name': 'Mia', 'hobbies': ['painting', 'jogging', 'cooking'], 'age': 20}
```

- `json.dump()` - writes a Python object as JSON data to a file.

```
[73]: with open("output.json", "w") as f:
          json.dump(data, f)
```

- Formatting with `indent` and `sort_keys`: both `dump` and `dumps` methods also receive `indent` and `sort_keys` optional parameters to add indentation or sort keys alphabetically.

```
[74]: formatted_json = json.dumps(data, indent=4, sort_keys=True)
      print(formatted_json)
```

```
{
    "age": 20,
    "hobbies": [
        "painting",
        "jogging",
        "cooking"
    ],
    "name": "Mia"
}
```

0.6 Exercises 3

1. Using `users.json` file:
   - open it and decode the Python object inside it
   - filter users with `"email"` key and encode the resulting object in a JSON string; print the string to the console
   - filter users with ages between 20 and 40 and encode the resulting object in a JSON file, using `indent` and `sort_keys` parameters.

0.7 4. Working with CSV format

0.7.1 What is CSV?

CSV (Comma-Separated Values) is a simple file format used to store tabular data, such as a spreadsheet or database. Each line in a CSV file represents a data record, and each record consists of fields separated by a delimiter (commonly a comma).

Example:

```
Name,Age,City
Alice,30,New York
Bob,25,Los Angeles
Charlie,35,Chicago
```

0.7.2 csv module

The `csv` module in Python provides functionality to read, write, and manipulate CSV files easily. It supports different delimiters, quoting styles, and file encoding.

- `csv.reader()` - reads a CSV file and returns an iterable object for processing each row as a list.

[75]:
```python
import csv

with open("data.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

```
['Name', 'Age', 'City']
['Alice', '30', 'New York']
['Bob', '25', 'Los Angeles']
['Charlie', '35', 'Chicago']
```

- `csv.writer()` - writes data to a CSV file row by row.

[76]:
```python
data = [
    ["Name", "Age", "City"],
    ["Alice", 30, "New York"],
    ["Bob", 25, "Los Angeles"],
    ["Charlie", 35, "Chicago"]
]

with open("output.csv", "w", newline="") as file:
    writer = csv.writer(file)
    # writer.writerows(data)
    for line in data:
        writer.writerow(line)
        print(f"{line} written to CSV file.")
```

```
['Name', 'Age', 'City'] written to CSV file.
['Alice', 30, 'New York'] written to CSV file.
['Bob', 25, 'Los Angeles'] written to CSV file.
['Charlie', 35, 'Chicago'] written to CSV file.
```

- `csv.DictReader()` - reads a CSV file and converts each row into a dictionary, with column headers as keys.

[77]:
```
with open("data.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row)
```

```
{'Name': 'Alice', 'Age': '30', 'City': 'New York'}
{'Name': 'Bob', 'Age': '25', 'City': 'Los Angeles'}
{'Name': 'Charlie', 'Age': '35', 'City': 'Chicago'}
```

- `csv.DictWriter()` - writes dictionaries to a CSV file, using specified `fieldnames` as headers.

[78]:
```
data = [
    {"Name": "Alice", "Age": 30, "City": "New York"},
    {"Name": "Bob", "Age": 25, "City": "Los Angeles"},
    {"Name": "Charlie", "Age": 35, "City": "Chicago"}
]

with open("output.csv", "w", newline="") as file:
    fieldnames = ["Name", "Age", "City"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(data)
```

0.8 Exercises 4

1. Using `books.csv`, do the following:
    - read the CSV file
    - create two other CSV files: `mathematics_books.csv` and `computer_science_books.csv`, containing only books in each genre (*Genre* column should be equal to *mathematics* or *computer_science*, respectively), with all columns in *books.csv* except *Genre*