Softy EDUCATION

&

Taki Academy
www.takiacademy.com

python™

contact@softyeducation.com

Day 6

Content :

1. Error handling
2. File manipulation
3. JSON
4. Working with folders
5. Exercises

Softy EDUCATION

01 Error handling

python™

01- Error handling

In Python, errors can occur for various reasons, such as a division by zero, accessing a non-existent key in a dictionary, or opening a non-existent file. Error handling helps in dealing with these unexpected errors and provides a way to execute code smoothly even if an error occurs.

01- Error handling

The try and except Blocks

The basic structure for error handling in Python involves a try block followed by one or more except blocks.

```
try:
    # code that might cause an error
except SomeErrorType:
    # code to execute if SomeErrorType occurs
```

01- Error handling

Example 1: Division by Zero

Here's a common example, where an attempt to divide by zero will raise a ZeroDivisionError.

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Example 2: Handling Multiple Exceptions

You can handle different exceptions in separate blocks, as shown here:

```
try:
    result = 10 / int(input("Enter a number: "))
except ZeroDivisionError:
    print("You can't divide by zero!")
except ValueError:
    print("That's not a valid number!")
```

The else and finally Clauses

You can also use else and finally clauses within error handling.

- The else block is executed if no error occurs in the try block.
- The finally block is always executed, whether an error occurs or not.

01- Error handling

```python
try:
    result = 10 / 2
except ZeroDivisionError:
    print("You can't divide by zero!")
else:
    print("Division successful")
finally:
    print("This will always be printed")
```

Softy EDUCATION

02 File manipulation

python™

02 - File manipulation

File manipulation is a fundamental skill in programming, allowing you to read from and write to files, which is essential for data storage and retrieval. Python provides several built-in functions and methods that make file manipulation straightforward. In this article, we'll explore opening files, reading and writing data, and more.

Opening Files

To manipulate a file in Python, you first need to open it using the open function.

```
file_object = open('filename.txt', 'mode')
```

02 - File manipulation

Here, **'filename.txt'** is the name of the file, and 'mode' defines how the file will be opened.

```
file = open('example.txt', 'r')
```

- **'r'**: Read (default). Opens the file for reading.
- **'w'**: Write. Opens the file for writing (creates a new file or truncates an existing file).
- **'a':** Append. Opens the file for writing (creates a new file or appends to an existing file).
- **'b':** Binary. Read/Write in binary mode.

Reading Files

Once a file is open, you can read its content using various methods.

02 - File manipulation

read()

This method reads the entire file.

```
content = file.read()
print(content)
```

readline()

This method reads one line at a time.

```
line = file.readline()
print(line)
```

readlines()

This method reads all the lines and returns them as a list.

```
lines = file.readlines()
for line in lines:
    print(line)
```

Writing to Files

You can write to a file using the write and writelines methods.

02 - File manipulation

write()

This method writes a string to the file.

```
file = open('example.txt', 'w')
file.write('Hello, World!')
```

writelines()

This method writes a list of strings.

```
lines = ['Hello, World!', 'Welcome to Python!']
file.writelines(lines)
```

Closing Files

It's crucial to close a file when you're done with it to free up system resources.

```
file.close()
```

writelines()

This method writes a list of strings.

```
lines = ['Hello, World!', 'Welcome to Python!']
file.writelines(lines)
```

Using with Statement

The with statement automatically takes care of closing the file, making it a safer way to handle files.

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

Error Handling

Sometimes, errors can occur during file manipulation (e.g., file not found). You can handle these using a try-except block.

```
try:
    with open('non_existent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print('File not found.')
```

Softy EDUCATION

03 JSON

python™

03 - JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It's commonly used for exchanging data between a web server and client, but also for storing configuration and other data.

03 - JSON

Python has a built-in package called json, which can be used to work with JSON data. Here's an article to help you understand how to work with JSON in Python, including reading, writing, parsing, and more.

03 - JSON


JSON
JSON
{
"name": "John Doe",
"age": 32,
"address": {
"street": "123 Main St",
"city": "Anytown",
"state": "CA"
},
"tags": ["Finance", "Admin"]
}
Python
{
"name": "John Doe",
"age": 32,
"address": {
"street": "123 Main St",
"city": "Anytown",
"state": "CA"
},
"tags": ["Finance", "Admin"]
}
ScrapingBee

Introduction to JSON

JSON is a text-based data format that represents structured data based on JavaScript object syntax. It is often used for asynchronous browser/server communication. A JSON object contains data in the form of key/value pairs.

03 - JSON

A simple example of a JSON object is:

```json
{
  "name": "John",
  "age": 30,
  "city": "New York"
}
```

Using JSON in Python

Python has a built-in package called json, which can be used to work with JSON data. Here's how you can use it:

Parsing JSON

You can convert a JSON string into a Python object using the json.loads() method.

```python
import json

json_string = '{"name": "John", "age": 30, "city": "New York"}'
python_obj = json.loads(json_string)
print(python_obj["name"])  # Output: John
```

Converting Python Object to JSON

You can also convert a Python object into a JSON string using the json.dumps() method.

```
import json

python_obj = {
  "name": "John",
  "age": 30,
  "city": "New York"
}
json_string = json.dumps(python_obj)
print(json_string)  # Output: {"name": "John", "age": 30, "city": "New York"}
```

Reading JSON from a File

You can read JSON data from a file using the json.load() method.

```python
import json

with open('data.json', 'r') as file:
    data = json.load(file)
    print(data['name'])  # Output: John
```

Writing JSON to a File

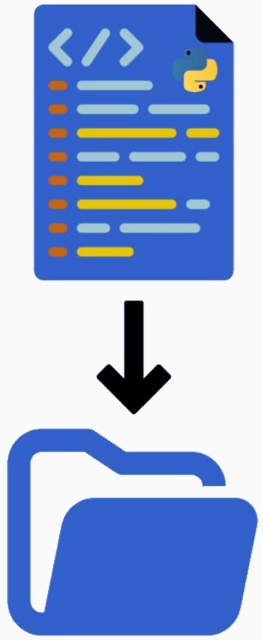
You can write JSON data to a file using the json.dump() method.

```python
import json

data = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

with open('data.json', 'w') as file:
    json.dump(data, file)
```

04 Working with folders

04 - Working with folders

Python offers various built-in libraries like os and shutil to work with directories and files.



Python OS Module

04 - Working with folders

1. Importing Necessary Libraries

Before you can work with folders, you need to import the os and shutil modules.

```
import os
import shutil
```

2. Getting the Current Working Directory

You can find out the current working directory using the os.getcwd() method.

```
current_directory = os.getcwd()
print("Current Directory:", current_directory)
```

3. Changing the Current Working Directory

You can change the working directory using the os.chdir(path) method.

```
new_directory = "/path/to/new/directory"
os.chdir(new_directory)
```

4. Creating a New Directory

Create a new directory using the os.mkdir(path) method.

```
new_folder_path = "new_folder"
os.mkdir(new_folder_path)
print(f"'{new_folder_path}' has been created")
```

5. Listing the Contents of a Directory

Use the os.listdir(path) method to get a list of filenames in a directory.

```
new_folder_path = "new_folder"
os.mkdir(new_folder_path)
print(f"'{new_folder_path}' has been created")
```

5. Listing the Contents of a Directory

Use the os.listdir(path) method to get a list of filenames in a directory.

```
new_folder_path = "new_folder"
os.mkdir(new_folder_path)
print(f"'{new_folder_path}' has been created")
```

Softy EDUCATION

05 Mini project

python™

Project: Personal Contact Manager

The goal of this project is to create a contact manager where users can:

- Add New Contacts: Store name, phone number, and email.
- View All Contacts: See a list of all stored contacts.
- Search for a Contact: Find a contact by name.
- Update a Contact: Edit an existing contact's details.
- Delete a Contact: Remove a contact from the file.

Softy EDUCATION

Thank you

python™