Python Essentials for DevOps: File Handling, Error Handling, OS Interaction & More

2.6 File Handling

Reading and Writing Files

Python provides built-in functions for file operations. Always use context managers (`with` statement) for automatic resource cleanup.

Basic Syntax

```
# Reading a file
with open('file.txt', 'r') as file:
    content = file.read()
    # or read line by line: lines = file.readlines()

# Writing to a file
with open('file.txt', 'w') as file:
    file.write('Hello, World!')
```

File Modes

- `'r'` - Read (default)
- `'w'` - Write (overwrites existing file)
- `'a'` - Append
- `'x'` - Exclusive creation (fails if file exists)
- `'b'` - Binary mode
- `'t'` - Text mode (default)
- `'+'` - Read and write

Working with Different File Formats

Text Files (.txt)

```
# Reading
with open('data.txt', 'r') as file:
    for line in file:
        print(line.strip())  # strip() removes newline characters

# Writing
with open('output.txt', 'w') as file:
    file.write('Line 1\n')
    file.write('Line 2\n')
```

Library	Function / Method	Application / Usage
csv (built-in)	`csv.reader(file)`	Read rows from a CSV file as lists.
	`csv.DictReader(file)`	Read rows into dictionaries (keys from header row).
	`csv.writer(file)`	Write rows to a CSV file (as lists).
	`csv.DictWriter(file, fieldnames)`	Write rows as dictionaries with specified field names.
	`writer.writerow(row)`	Write a single row.
	`writer.writerows(rows)`	Write multiple rows.
json (built-in)	`json.load(file)`	Parse JSON from a file into Python objects.
	`json.loads(string)`	Parse JSON from a string.
	`json.dump(obj, file)`	Serialize Python object to JSON and write to file.
	`json.dumps(obj)`	Serialize Python object to JSON string.
	`indent` (argument)	Pretty-print JSON with indentation.
	`sort_keys=True`	Sort keys alphabetically in output JSON.
yaml (PyYAML)	`yaml.safe_load(file_or_string)`	Load YAML into Python objects (safe version, recommended).
	`yaml.load(file_or_string, Loader=yaml.FullLoader)`	Load YAML with a specific loader.
	`yaml.safe_load_all(file)`	Load multiple YAML documents from one file.
	`yaml.dump(obj)`	Serialize Python object to YAML string.
	`yaml.dump(obj, file)`	Write serialized Python object as YAML to file.
	`yaml.safe_dump(obj)`	Safe version of dump (avoids arbitrary code execution).
	`default_flow_style=False` (arg in `dump`)	Output YAML in block style (human-readable).

JSON Files (.json)

```python
import json

# Reading JSON
with open('data.json', 'r') as file:
    data = json.load(file)
```

```python
# Writing JSON
data = {'name': 'John', 'age': 30, 'city': 'New York'}
with open('data.json', 'w') as file:
    json.dump(data, file, indent=4)  # indent for pretty printing
```

YAML Files (.yaml/.yml)

```python
import yaml  # requires PyYAML: pip install pyyaml

# Reading YAML
with open('config.yml', 'r') as file:
    config = yaml.safe_load(file)

# Writing YAML
config = {'database': {'host': 'localhost', 'port': 5432}}
with open('config.yml', 'w') as file:
    yaml.dump(config, file, default_flow_style=False)
```

CSV Files (.csv)

```python
import csv

# Reading CSV
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)

# Reading as dictionary
with open('data.csv', 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row['column_name'])

# Writing CSV
data = [['Name', 'Age'], ['Alice', 30], ['Bob', 25]]
with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

# Writing as dictionary
fieldnames = ['Name', 'Age']
data = [{'Name': 'Alice', 'Age': 30}, {'Name': 'Bob', 'Age': 25}]
with open('output.csv', 'w', newline='') as file:
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(data)
```

2.7 Error and Exception Handling

Basic Exception Handling

```
try:
    # Code that might raise an exception
    result = 10 / 0
except ZeroDivisionError:
    # Handle specific exception
    print("Cannot divide by zero!")
except (TypeError, ValueError) as e:
    # Handle multiple exceptions
    print(f"Error: {e}")
except Exception as e:
    # Handle any exception
    print(f"Unexpected error: {e}")
else:
    # Executes if no exception was raised
    print("Operation completed successfully")
finally:
    # Always executes, used for cleanup
    print("This always runs")
```

Raising Exceptions

```
def validate_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative")
    if age > 150:
        raise ValueError("Age seems unrealistic")
    return age

try:
    validate_age(-5)
except ValueError as e:
    print(f"Validation error: {e}")
```

Custom Exceptions

```
class ConfigError(Exception):
    """Custom exception for configuration errors"""
    pass
```

```python
def load_config(filepath):
    if not os.path.exists(filepath):
        raise ConfigError(f"Config file not found: {filepath}")
    # Load configuration...
```

3.1 Working with the Operating System

The os Module

```python
import os

# Working with directories
os.mkdir('new_dir')  # Create directory
os.makedirs('path/to/nested/dir', exist_ok=True)  # Create nested directories
os.rmdir('empty_dir')  # Remove empty directory

# File operations
os.rename('old.txt', 'new.txt')  # Rename file
os.remove('file.txt')  # Remove file

# Path operations
current_dir = os.getcwd()  # Get current working directory
os.chdir('/path/to/dir')  # Change directory
file_list = os.listdir('.')  # List directory contents

# Environment variables
home_dir = os.environ.get('HOME')
os.environ['CUSTOM_VAR'] = 'value'  # Set environment variable

# Path manipulation
file_path = os.path.join('dir', 'subdir', 'file.txt')  # Platform-independent path
dirname = os.path.dirname('/path/to/file.txt')  # /path/to
basename = os.path.basename('/path/to/file.txt')  # file.txt
```

The sys Module

```python
import sys

# Command-line arguments
script_name = sys.argv[0]  # First argument is script name
arguments = sys.argv[1:]   # Remaining arguments

# Standard streams
sys.stdout.write('Output to stdout\n')
sys.stderr.write('Output to stderr\n')
user_input = sys.stdin.readline()
```

```python
# System information
python_version = sys.version
platform = sys.platform

# Exiting the program
sys.exit(0)  # Exit with success code
sys.exit(1)  # Exit with error code
```

The `pathlib` Module (Modern Approach)

```python
from pathlib import Path

# Creating paths
current_dir = Path.cwd()
home_dir = Path.home()
file_path = Path('dir') / 'subdir' / 'file.txt'  # Platform-independent

# File operations
path = Path('file.txt')
path.exists()  # Check if path exists
path.is_file()  # Check if it's a file
path.is_dir()   # Check if it's a directory

# Reading/writing
content = path.read_text()  # Read file as text
path.write_text('Hello!')   # Write text to file

# Directory operations
dir_path = Path('my_dir')
dir_path.mkdir(exist_ok=True)  # Create directory if it doesn't exist

# Iterating over directory contents
for file in Path('.').iterdir():
    print(file.name)

# Finding files
py_files = list(Path('.').glob('*.py'))  # All Python files in current directory
all_files = list(Path('.').rglob('*'))   # All files recursively
```

3.2 Command-Line Execution and Scripting

The `subprocess` Module

```python
import subprocess
```

```python
# Basic command execution
result = subprocess.run(['ls', '-l'], capture_output=True, text=True)
print(result.returncode)  # Exit status
print(result.stdout)      # Standard output
print(result.stderr)      # Standard error

# With shell expansion
result = subprocess.run('echo $HOME', shell=True, capture_output=True, text=True)

# Check for errors (raises CalledProcessError if returncode   0)
try:
    result = subprocess.run(['false'], check=True, capture_output=True, text=True)
except subprocess.CalledProcessError as e:
    print(f"Command failed with exit code {e.returncode}")

# Using Popen for more control
process = subprocess.Popen(['python', 'script.py'],
                          stdin=subprocess.PIPE,
                          stdout=subprocess.PIPE,
                          stderr=subprocess.PIPE,
                          text=True)

stdout, stderr = process.communicate(input='some input')
returncode = process.wait()

# Real-time output processing
process = subprocess.Popen(['tail', '-f', 'logfile.txt'],
                          stdout=subprocess.PIPE,
                          text=True)

for line in process.stdout:
    print(f"LOG: {line.strip()}")
    if 'ERROR' in line:
        process.terminate()  # Stop the process
```

3.3 Date and Time Operations

The `datetime` Module

```python
from datetime import datetime, date, time, timedelta
import time as time_module  # For the time module

# Current date and time
now = datetime.now()
today = date.today()
current_time = datetime.now().time()
```

```python
# Creating specific dates
dt = datetime(2023, 12, 25, 15, 30, 45)  # Year, month, day, hour, minute, second
d = date(2023, 12, 25)
t = time(15, 30, 45)

# Formatting dates
formatted = now.strftime('%Y-%m-%d %H:%M:%S')  # 2023-12-25 15:30:45
parsed = datetime.strptime('2023-12-25', '%Y-%m-%d')  # String to datetime

# Date arithmetic
tomorrow = today + timedelta(days=1)
last_week = today - timedelta(weeks=1)
difference = datetime(2023, 12, 31) - datetime(2023, 1, 1)  # Returns timedelta

# Timezone handling (requires pytz or Python 3.9+ zoneinfo)
from datetime import timezone
utc_now = datetime.now(timezone.utc)

# Timestamps
timestamp = datetime.now().timestamp()  # Seconds since epoch
dt_from_ts = datetime.fromtimestamp(timestamp)

# Sleeping (from time module)
time_module.sleep(2.5)  # Sleep for 2.5 seconds
```

Practice Exercises

File Handling Exercises

1. Text File Processing: Write a script that reads a text file, counts the occurrences of each word, and writes the results to a new file.
2. JSON Configuration: Create a function that reads a JSON configuration file, allows modifying specific values, and writes the updated configuration back to the file.
3. CSV Data Analysis: Write a script that reads a CSV file containing sales data, calculates the total sales per month, and outputs the results to a new CSV file.
4. YAML Parser: Create a function that reads a YAML file, converts it to JSON format, and saves it as a new file with a .json extension.
5. File Backup Utility: Write a script that creates a backup of all files in a directory with a specific extension (e.g., .txt) by copying them to a backup folder with a timestamp.

Error Handling Exercises

6. Robust File Reader: Create a function that attempts to read a file but handles various exceptions (FileNotFoundError, PermissionError, etc.) gracefully with appropriate error messages.
7. Input Validation: Write a function that prompts the user for a number and uses exception handling to ensure valid input, with custom error messages for different invalid inputs.
8. API Response Handler: Create a function that simulates making API calls (you can use random success/failure) and implements comprehensive exception handling for different types of failures.

OS Interaction Exercises

9. Directory Cleaner: Write a script that removes all files in a directory that are older than 30 days, using the `os` or `pathlib` module to check file creation/modification dates.
10. Environment Setup: Create a script that checks if required environment variables are set and creates necessary directory structure for an application if it doesn't exist.
11. File Organizer: Write a script that organizes files in a directory by moving them into subdirectories based on their file extensions.

Command-Line Exercises

12. Git Status Checker: Create a script that uses subprocess to run `git status` and parses the output to determine if there are any uncommitted changes.
13. Process Monitor: Write a script that lists all running processes (using `ps` on Unix or `tasklist` on Windows) and filters them based on a user-provided search term.
14. Command Runner with Timeout: Create a function that runs a command with subprocess but terminates it if it runs longer than a specified timeout.

Date/Time Exercises

15. Log File Analyzer: Write a script that reads a log file with timestamps, filters entries from the last 24 hours, and counts occurrences of different log levels (ERROR, WARNING, INFO).

"'