More Python for Videogames

Videogames Technology
Asignatura transversal

Departamento de Automática

Universidad de Alcalá

Objectives

1. Being able to manipulate files in Python.
2. Understand and apply Python serialization (*pickles* and JSON).
3. Being able to handle exceptions.

Bibliography

- The Python Tutorial. *Section 7.2: Reading and writing files.* (Link)
- The Arcade Library. *Sound.* (Link)
- Learn Arcade. *Chapter 20: Sound.* (Link)

Table of Contents

Path

Definition

```
root
 ├── home
 │    ├── rick
 │    │    ├── foo.py
 │    │    └── music.mp3
 │    └── morthy
 │         └── ...
 ├── bin
 └── ...
```

Path

A string that identifies a file in a file system

Two types of paths:

- Absolute: Address from the root directory
- Relative: Address from the working directory

Working directory: Folder where your program is running from

The path separator is operating system dependent

- Linux/macOS/Android: Uses forward slash: /
- Windows: Uses backslash: \

Path

Paths in Linux

```
/
├── home/
│   ├── rick/
│   │   ├── foo.py
│   │   └── music.mp3
│   └── morthy/
│           └── ...
├── bin/
└── ...
```

On Linux, the absolute path is:

```
path = '/home/rick/music.mp3'
```

If we are in `/home/`, the relative path is:

```
path = 'rick/music.mp3'
```

Path

Paths in Windows

```
C:\
├── Users\
│   ├── rick\
│   │   ├── foo.py
│   │   └── music.mp3
│   └── morthy\
│       └── ...
└── ...
```

On **Windows**, the absolute path is:

`'C:\Users\rick\music.mp3'`

If we are in `C:\Users\`, the relative path is:

`'rick\music.mp3'`

And it is represented in Python by:

```
path = 'C:\\Users\\rick\\music.mp3'
```

But by also using *raw string*:

```
path = r'C:\Users\rick\music.mp3'
```

Path

Portable code with the os module

The `os` module provides old fashioned tools to deal with paths

- `os.path.join("folder", "subfolder", "file.txt")`
- `os.sep`

Recommended

```python
import os

path = os.path.join('data', 'file.txt')
print(path)
```

Not recommended

```python
import os

path = 'data' + os.sep + 'file.txt'
print(path)
```

Use with caution

```python
path = "data/file.txt"
```

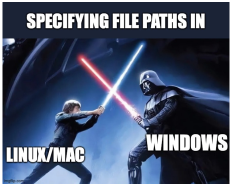
Pathlib provides a modern (i.e. object-oriented) solution

Path

The `__file__` variable

Python defines the variable `__file__`

- Contains the absolute path to the file
- Not defined if there is no file
- Useful to locate our project location



Reading and writing files

Introduction

File operation overview

1. Open the file in a specific mode
2. Perform operations on the file (read/write, among others)
3. Close the file

All file operations are performed through a *file object*

- First: call the `open()` function
- It returns the file object
- Always close the file, even in the event of failure

Reading and writing files

Opening files (I)

`open()`

`open(path[, mode])`

Return: An object file

- *path*: Path string
- *mode*: Characters describing how the file will be used
  - r: Read mode, *w*: Write mode (overwrites file)
  - r+: r/w mode, no truncation; *w+*: r/w mode, truncation
  - *a*: Write, appending mode
  - *b*: Binary mode, text mode by default

Always, always, always close the file: `f.close()` (unless in a `with` clause)

Reading and writing files

Opening files (II)

Text mode

```python
# Read mode (default)
file = open('data.txt', 'r')
# or simply:
file = open('data.txt')

# Write mode - overwrites
file = open('output.txt', 'w')

# Append mode - adds to the end
file = open('log.txt', 'a')

# Read and write mode
file = open('data.txt', 'r+')
```

Binary mode

```python
# Read binary file
file = open('image.png', 'rb')

# Write binary file
file = open('output.dat', 'wb')
```

Reading and writing files

Reading files (I)

The `read()` method

`f.read([size])`

Return: the specified number of bytes

- *size*: The number of bytes to be read from the file. Default reads the whole file

Option 1: Read the entire file (`f.read()`)

```
>>> f = open("/tmp/file", 'r')
>>> f.read()
'This is the entire file.\\n'
>>> f.read()
''
>>> f.close()
```

Reading and writing files

Reading files (II)

Option 2: Read a single line (`f.readline()`)

```
>>> f = open("/tmp/file2", 'r')
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'This is the second line of the file\n'
>>> f.readline()
''
>>> f.close()
```

Reading and writing files

Reading files (III)

Option 3: Read lines as list (`f.readlines()`)

```
>>> f = open("/tmp/file2", 'r')
>>> f.readlines()
['This is the first line of the file.\n',
'This is the second line of the file\n']
>>> f.close()
```

Option 4: Read in a loop

```
f = open("/tmp/file2", 'r')
for line in f:
    print(line, end='')
f.close()
```

Reading and writing files

Example

Number of lines and characters in file `example.txt`

```python
characters = 0
lines= 0

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

for line in file:
  lines+= 1
  characters += len(line)

file.close()

print(f"Characters: {characters}, number of lines: {lines}")
```

Reading and writing files

Writing files (I)

The `write()` method

`f.write(string)`

Return: Number of written bytes

- *string*: String to write

Example 1: Write a line

```
>>> f = open("/tmp/file", 'w+')
>>> f.write('This is a test\n')
15
>>> f.read()
''
>>> f.close()
```

Reading and writing files

Writing files (II)

Example 2: Write a number

```
>>> f = open("/tmp/file", 'w+')
>>> f.write(str(42))
2
>>> f.close()
```

Reading and writing files

Writing files: Example

```python
import os

name = 'Pepe'
age = 25
city = 'Alcalá de Henares'

path = 'data' + os.sep + 'person.txt'

file = open(path, 'w', encoding='utf-8')

file.write('Name: ' + name + '\n')
file.write('Age: ' + str(age) + '\n')
file.write('City: ' + city + '\n')

file.close()
```

Reading and writing files

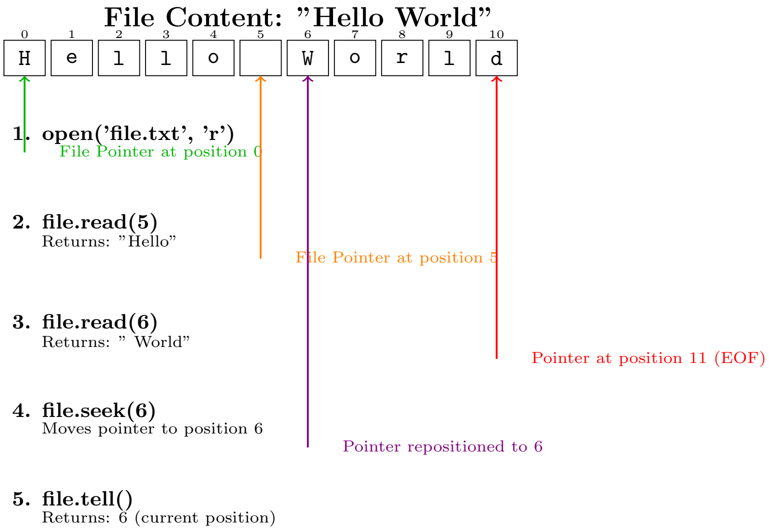
Random access (I)

Method	Description
`f.tell()`	Returns the pointer's position
`f.seek(n)`	Moves the pointer to position `n`

```
>>> f = open("/tmp/file", 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)
5
>>> f.read(1)
b'5'
>>> f.close()
```

Reading and writing files

Random access (II)



File Content: "Hello World"
0
1
2
3
4
5
6
7
8
9
10
H
e
l
l
o
W
o
r
l
d
File Pointer at position 0
File Pointer at position 5
Pointer at position 11 (EOF)
Pointer repositioned to 6

1. **open('file.txt', 'r')**
2. **file.read(5)**
   Returns: "Hello"
3. **file.read(6)**
   Returns: " World"
4. **file.seek(6)**
   Moves pointer to position 6
5. **file.tell()**
   Returns: 6 (current position)

Reading and writing files

`With` (I)

The `with` clause

```
with open(path, mode) as file:
    ...
```

It simplifies file operations

- No need to close files
- Better exception handling

```
f = open('file')
print(f.read())
f.close()
```

$\Rightarrow$

```
with open('file') as file:
    print(file.read())
```

Reading and writing files

With (II)

Hello, world

```
with open("file.txt", "w") as file:
    file.write("Hello, world.\n")
    file.write("This is another file.\n")
```

Reading and writing files

With (III)

Reading a line each time

```python
count_line = 0
with open('nombres.txt') as arch_names:
    for line in arch_names:
        count_line += 1
        print(f'{count_line}: {line.rstrip()}')
```

⇑ ⇓

names.txt

```
Juan
Laura
Pablo
Enrique
Javier
```

Output

```
1: Juan
2: Laura
3: Pablo
4: Enrique
5: Javier
```

Pathlib

Introduction

Pathlib is a module for working with paths

- Built-in module from Python 3.4
- Object-oriented
- Intuitive path operations
- Methods for common operations
- Supported by Arcade 3.x
- (Pathlib reference documentation)

os.path

```python
import os

path = os.path.join("data",
        "file.txt")

file = open(path)
```

pathlib

```python
from pathlib import Path

path = Path('data') / "file.txt"
# path is a Path object, not a
    string!
file = open(str(path))
```

Pathlib

Creatings paths

Basic Path

```
path = Path('folder/subfolder/file.txt')
```

Using / operator

```
path = Path('folder') / 'subfolder' / 'file.txt'
```

Current directory

```
path = Path.cwd()
```

Home directory

```
path = Path.home()
```

Pathlib

Common operations

Check if path exists

`path.exists()`

Check if path is a file

`path.is_file()`

Check if path is a directory

`path.is_dir()`

Create directory

`path.mkdir()`

Write to file

`path.write_text()`

Read file content

`path.read_text()`

Reading and writing files

Example

```python
from pathlib import Path

# Create directory structure
project = Path('my_project')
data_dir = project / 'data'
data_dir.mkdir(parents=True, exist_ok=True)

# Create and write file
file_path = data_dir / 'output.txt'
file_path.write_text('Hello from pathlib!\n', encoding='utf-8')

# Check if file exists
if file_path.exists():
    print(f"File created: {file_path}")
    print(f"Size: {file_path.stat().st_size} bytes")

# Read file contents
content = file_path.read_text(encoding='utf-8')
print(f"Content: {content}")
```

Serialization

Introduction

What happens if we need to store complex data structures?

- Think about lists, dictionaries or even objects ...

What happens if we need to transmit complex data structures?

Serialization

Converting a data object into a sequence of bytes

Deserialization

Converting a sequence of bytes into a data object

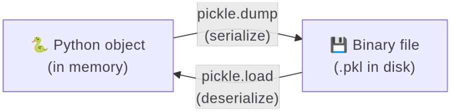
We can easily store and even transmit sequences of bytes ...

- ... and also reconstruct our original data

There are several serialization technologies: Pickles, JSON, XML, YAML, ...

Serialization

The `pickle` module



Given an object `x` and a file object `f` ...

- `pickle.dump(x, f)`: Serializes object x and writes it to file f
- `pickle.load(f)`: Reads and deserializes object from file f
- `x` may be a dictionary, list or even an object

Pickle uses a binary format

Serialization

The `pickle` module: Examples

Save a list to a file

```python
import pickle

numbers = [2, 5, 7, 8]

f = open('list.pkl', 'wb')

pickle.dump(numbers, f)

f.close()
```

Load a list from a file

```python
import pickle

f = open('list.pkl', 'rb')

my_numbers = pickle.load(f)

print(my_numbers)

f.close()
```

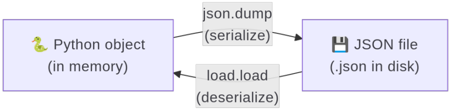
JP
@punsnparticles

Python users: “Python is a serious coding language to be taken seriously”

Also Python users: [typing] “import pickle”

9:35 AM · Feb 7, 2022 · Twitter for iPhone

Serialization
The JSON module



Given an object `x` and a file object `f` ...

- `json.dump(x, f)`: Serializes object x and writes it to file f
- `json.load(f)`: Reads and deserializes object from file f
- More limited than pickles

JSON is a text format

Serialization

The JSON module: JSON format

JSON: *JavaScript Object Notation*

- Data format for hierarchical data
- Created in 2001 for stateless client-server communication
- Text-based
- Interoperable (pickles only for Python)
- Complex data structures

filename.json

```
{
  "firstName": "John",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
  },
  "phoneNumbers": [ "111", "333" ]
}
```

Serialization

The JSON module: Examples

Save a list to a file

```
import json

mylist = ["John", 42, "Smith"]

myfile = open("myfile.json", "w")

json.dump(mylist, myfile, indent = 4)
```

Load a list from a file

```
import json

mylist = json.load(open('myfile.json'))

print(mylist)
```

Exceptions in I/O

Motivation

Errors happen ... more often in I/O operations

- File does not exist
- No permission to read/write
- Disk full
- Incorrect type of file

⇒ We need tools to handle errors: Exceptions

Exceptions in I/O

Common I/O exceptions

I/O operations may raise the following exceptions

- `FileNotFoundError`
  File does not exist
- `PermissionError`
  No permission to read/write
- `IOError`
  Disk full and other I/O errors
- `IsADirectoryError` and `NotADirectoryError`
  Incorrect type of file

Exceptions in I/O

Examples (I)

try-except

```
try:
    file = open('data.txt', 'r')
    content = file.read()
    print(content)
    file.close()
except FileNotFoundError:
    print("Error: The file does not exist")
except PermissionError:
    print("Error: You don't have permission")
except IOError as e:
    print("Error: I/O error occurred: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Exceptions in I/O

Examples (II)

try-except statement

```
file = None
try:
    file = open('data.txt', 'r')
    content = file.read()
    print(content)
except FileNotFoundError:
    print("Error: File 'data.txt' not found")
except PermissionError:
    print("Error: Permission denied")
except IOError as e:
    print(f"I/O error occurred: {e}")
finally:
    # This always executes, even if there's an error
    if file is not None:
        file.close()
        print("File closed successfully")
```

Exceptions in I/O

Examples (III)

try-except statement

```
# The file closes automatically, even if an exception occurs
try:
    with open('data.txt', 'r') as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("Error: File not found")
except PermissionError:
    print("Error: Permission denied")
except IOError as e:
    print("Error: I/O error occurred: {e}")
except Exception as e:
    print(f"Error: {e}")
```

Sound in Arcade

Introduction

Two steps:

1. Load the sound
2. Play the sound

Two ways to provide a path

- String
  ```
  path = 'laser.wav'
  ```
- Path
  ```
  path = Path('laser.wav')
  ```

Sound in Arcade

Loading sounds

Arcade supports two APIs

- Functional API:
  `laser_sound = arcade.load_sound("laser.wav")`
- Object oriented API:
  `laser_sound = Sound("laser.wav")`

  Both return a `Sound` object

Boolean `streaming` argument

- True: Streams from disk. Long files
- False: Loads the whole file. Short files

Sound in Arcade

Playing sounds

Two ways to play a sound:

- The `arcade.play_sound()` function
  `arcade.play_sound(laser_sound)`
- The `play()` method (object oriented)

Both return a `Player` object

- It controls the playback

Sound in Arcade

Built-in sounds

Arcade comes with a collection of built-in resources

- Sounds, music, sprites, ...
- Good for testing
- (Link)

Built-in resources

`":resources:<path>"`

Terminal

```
>> import arcade

>> sound = arcade.load_sound(
       ":resources:/sounds/coin2.wav")
>> arcade.play_sound(sound)

>> music = arcade.load_sound(
       ":resources:/music/1918.mp3",
       streaming=True)
>> arcade.play_sound(music)
```

⇑ Does not work as script!

Sound in Arcade

Example

```python
import arcade

WIDTH = 800
HEIGHT = 600

arcade.open_window(WIDTH, HEIGHT, "Example of sound in Arcade")

music = arcade.load_sound(
        ":resources:/music/1918.mp3",
        streaming=True)
# music = Sound(":resources:/music/1918.mp3", streaming=True)
arcade.play_sound(music)
# music.play() # Object-oriented style

arcade.start_render()
arcade.draw_text("Enjoy!", 350, 300, arcade.color.WHITE)
arcade.finish_render()

arcade.run()
```