

More Python for Videogames

Videogames Technology
Asignatura transversal

Departamento de Automática

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

I. Path

- Definition
- Paths in Linux
- Paths in Windows
- The `__file__` variable

2. Reading and writing files

- Introduction
- Opening files
- Reading files
- Reading files: Example
- Writing files
- Writing files: Example
- Random access
- With

3. Pathlib

- Introduction
- Creating paths

- Common operations

- Example

4. Serialization

- The `pickle` module
- The `pickle` module: Examples
- The JSON module
- The JSON module: JSON format
- The JSON module: Examples

5. Exceptions in I/O

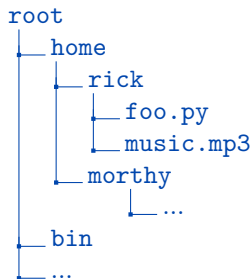
- Motivation
- Common I/O exceptions
- Examples

6. Sound in Arcade

- Introduction
- Loading sounds
- Playing sounds
- Built-in sounds
- Arcade sound API

Path

Definition



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**

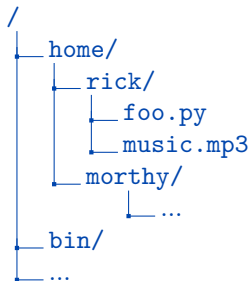
Working directory: Folder where your program is running from

The path separator is operating system dependent

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

Path

Paths in Linux



On Linux, the absolute path is:

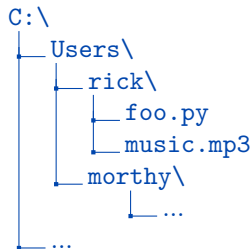
```
path = '/home/rick/music.mp3'
```

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

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

Path

Paths in Windows



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

```
1 import os
2
3 path = os.path.join('data', '
4     file.txt')
5 print(path)
```

Not recommended

```
1 import os
2
3 path = 'data' + os.sep + 'file.
4     txt'
5 print(path)
```

Use with caution

```
1 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(filename[, mode])`

Return: An object file

- `filename`: 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

```
1 # Read mode (default)
2 file = open('data.txt', 'r')
3 # or simply:
4 file = open('data.txt')
5
6 # Write mode - overwrites
7 file = open('output.txt', 'w')
8
9 # Append mode - adds to the end
10 file = open('log.txt', 'a')
11
12 # Read and write mode
13 file = open('data.txt', 'r+')
```

Binary mode

```
1 # Read binary file
2 file = open('image.png', 'rb')
3
4 # Write binary file
5 file = open('output.dat', 'wb')
```

Reading and writing files

Reading files (I)

The `read()` function

```
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 r: 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`

```
1 characters = 0
2 lines = 0
3
4 file = open('example.txt', 'r')
5
6 for line in file:
7     characters += 1
8     lines += len(line)
9
10 file.close()
11
12 print(f"Characters: {characters}, number of lines: {lines}")
```

Reading and writing files

Writing files (I)

The write() function

```
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

```
1 import os
2
3 name = 'Pepe'
4 age = 25
5 city = 'Alcalá de Henares'
6
7 path = 'data' + os.sep + 'person.txt'
8
9 file = open(path, 'w', encoding='utf-8')
10
11 file.write('Name: ' + name + '\n')
12 file.write('Age: ' + str(age) + '\n')
13 file.write('City: ' + city + '\n')
14
15 file.close()
```

Reading and writing files

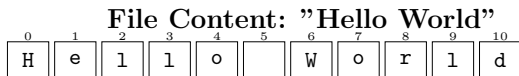
Random access (I)

METHOD	DESCRIPTION
<code>f.tell()</code>	Returns the pointer's position
<code>f.seek(n)</code>	Moves the pointer to position <code>n</code>

```
>>> 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)



1. **open('file.txt', 'r')**
File Pointer at position 0

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)

File Pointer at position 5

Pointer at position 11 (EOF)

Pointer repositioned to 6

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()
```

⇒

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

Reading and writing files

With (II)

Hello, world

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

Reading and writing files

With (III)

Reading a line each time

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



names.txt

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

```

1 import os
2
3 path = os.path.join("data",
4                     "file.txt")
5
6 file = open(path)
```

pathlib

```

1 from pathlib import Path
2
3 path = Path('data') / "file.txt"
4 # path is a Path object, not a
5   string!
6 file = open(str(path))
```


Pathlib

Creating 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

```
1 from pathlib import Path
2
3 # Create directory structure
4 project = Path('my_project')
5 data_dir = project / 'data'
6 data_dir.mkdir(parents=True, exist_ok=True)
7
8 # Create and write file
9 file_path = data_dir / 'output.txt'
10 file_path.write_text('Hello from pathlib!\n', encoding='utf-8')
11
12 # Check if file exists
13 if file_path.exists():
14     print(f"File created: {file_path}")
15     print(f"Size: {file_path.stat().st_size} bytes")
16
17 # Read file contents
18 content = file_path.read_text(encoding='utf-8')
19 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

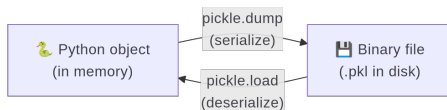
We can easily store and even transmit sequences of bytes ...

- ... 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

```
1 import pickle
2
3 numbers = [2, 5, 7, 8]
4
5 f = open('list.pkl', 'wb')
6
7 pickle.dump(numbers, f)
8
9 f.close()
```

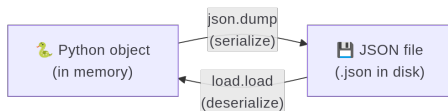
Load a list from a file

```
1 import pickle
2
3 f = open('list.pkl', 'rb')
4
5 my_numbers = pickle.load(f)
6
7 print(my_numbers)
8
9 f.close()
```



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`
- `x` may be a dictionary but **not an object**

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

Tile Map Editor generates JSON files!

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

```
1 import json
2
3 mylist = [ "John" , 42 , "Smith" ]
4
5 myfile = open( "myfile.json" , "w" )
6
7 json.dump(mylist , myfile , indent = 4)
```

Load a list from a file

```
1 import json
2
3 mylist = json.load( open( 'myfile.json' ) )
4
5 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 along a collection of built-in resources

- Sounds, music, sprites, ...
- Good for testing

Identified by

`" :resources:<path>"`

- (Link)

Use this in your scripts with caution!

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 )
```

Sound in Arcade

Example

```
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)
arcade.play_sound(music)

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

arcade.run()
```