Python Lecture 6: File Handling and Exception Handling

September 24, 2025

1 Revision of Lecture 5: Functions Using Data Structures

To revise the data structures from Lecture 5, we will create five functions, each utilizing a different data structure: list, dictionary, tuple, array (from NumPy), and set. These functions will perform simple operations. Finally, a main program will call all five functions.

1.1 Function Using List: Sum of Elements

This function takes a list of numbers and returns their sum.

```python
def sum_list(numbers):
    total = 0
    for num in numbers:
        total += num
    return total
```

Example: $\mathrm{sum}_{l}ist([1, 2, 3])returns6$.

1.2 Function Using Dictionary: Get Value by Key

This function takes a dictionary and a key, returning the corresponding value or "Not Found".

```python
def get_dict_value(my_dict, key):
    return my_dict.get(key, "Not Found")
```

Example: $\mathrm{get}_{d}ict_{v}alue(”name” : ”Alice”, ”age” : 25, ”age”)returns25$.

1.3 Function Using Tuple: Unpack and Add

This function takes a tuple of three numbers, unpacks them, and returns their sum.

```python
def sum_tuple(tup):
    a, b, c = tup
    return a + b + c
```

Example: $\mathrm{sum}_{t}uple((4, 5, 6))returns15$.

1.4 Function Using Array (NumPy): Compute Mean

This function uses NumPy to compute the mean of an array.

```
import numpy as np

def compute_mean_array(arr):
    return np.mean(arr)
```

Example: compute$_{m}ean_{a}rray(np.array([10, 20, 30]))returns20.0$.

1.5 Additional NumPy Examples

This subsection provides a detailed explanation and examples of using NumPy arrays, including 1D and 2D arrays, stacking operations, and mathematical functions. These examples demonstrate NumPy's power for numerical computations.

Vertical stacking (np.vstack) combines arrays along the vertical axis (axis 0), effectively adding rows from one array below another. It requires arrays to have the same number of columns. Horizontal stacking (np.hstack) combines arrays along the horizontal axis (axis 1), adding columns from one array to the right of another. It requires arrays to have the same number of rows.

```
# Array (1D or 2D)
import numpy as np

arr = np.array([1, 2, 4, 10])
print(arr)
print(arr.shape)
print("Sum: ", arr.sum())
print("Mean = Average = ", arr.mean())

arr2D = np.array([[1, 2, 10], [2, 5, 7], [4, 6, 12], [5, 8, 9]])
print(arr2D)
print(arr2D.shape)
print("Sum: ", arr2D.sum())
print("Mean = Average = ", arr2D.mean())

arr1 = np.array([[1, 2, 3], [4, 5, 6]])
print("Array:\n", arr1)

arr2 = np.array([[10, 20, 30], [40, 50, 60]])
print("\nZeros:\n", arr2)
print(arr2.shape)

arr3 = np.array([[10, 20, 30, 40], [40, 50, 60, 40]])
print("\nZeros:\n", arr3)
print(arr3.shape)

stack_v = np.vstack((arr1, arr2))
print("\nVertical Stack:\n", stack_v)
print(stack_v.shape)

stack_v2 = np.vstack((arr1, arr3))
print("\nVertical Stack:\n", stack_v2)
print(stack_v2.shape)

stack_h = np.hstack((arr1, arr2))
```

```
print("\nHorizontal Stack:\n", stack_h)
print(stack_h.shape)

stack_h2 = np.hstack((arr1, arr3))
print("\nHorizontal Stack:\n", stack_h2)
print(stack_h2.shape)

# Mathematical Operations
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
print("Array:\n", arr1)

arr2 = np.array([[10, 20, 30], [40, 50, 60]])
print("\nZeros:\n", arr2)
print(arr2.shape)

print("\nAdd:\n", np.add(arr1, arr2))
print("Subtract:\n", np.subtract(arr1, arr2))
print("Multiply:\n", np.multiply(arr1, arr2))
print("Divide:\n", np.divide(arr1, arr2))
print("Dot product:\n", np.dot(arr1, arr2.T))

arr4 = np.array([1, 2, 3, 10, 4, 5, 6])
print("Array:\n", arr4)
print("\nSum:", np.sum(arr4))
print("Mean:", np.mean(arr4))
print("Median:", np.median(arr4))
print("Variance:", np.var(arr4))
print("Std Dev:", np.std(arr4))
print("Min:", np.min(arr4))
print("Max:", np.max(arr4))
print("Argmin (index of min):", np.argmin(arr4))
print("Argmax (index of max):", np.argmax(arr4))
```

Expected Output (partial, varies by environment): “‘ [1 2 4 10] (4,) Sum: 17 Mean = Average = 4.25 [[1 2 10] [2 5 7] [4 6 12] [5 8 9]] (4, 3) Sum: 71 Mean = Average = 5.916666666666667 ... “‘

The following table summarizes common NumPy array functions, their meanings, simple examples, and results. It provides a quick reference for array operations.

1.6 Function Using Set: Check Membership

This function takes a set and a value, checking if the value is in the set.

```
def check_in_set(my_set, value):
    if value in my_set:
        return "Present"
    else:
        return "Absent"
```

Example: check$_i n_s et(1, 2, 3, 2) returns ”Present”$.

1.7 Main Program Calling All Functions

The main program demonstrates calling all five functions.

```
import numpy as np

```

```
def main():
    # List example
    my_list = [1, 2, 3]
    print("Sum of list:", sum_list(my_list))

    # Dictionary example
    my_dict = {"name": "Alice", "age": 25}
    print("Value for 'name':", get_dict_value(my_dict, "name"))

    # Tuple example
    my_tuple = (4, 5, 6)
    print("Sum of tuple:", sum_tuple(my_tuple))

    # Array example
    my_array = np.array([10, 20, 30])
    print("Mean of array:", compute_mean_array(my_array))

    # Set example
    my_set = {1, 2, 3}
    print("Check 3 in set:", check_in_set(my_set, 3))

main()
```

Output: Sum of list: 6 Value for 'name': Alice Sum of tuple: 15 Mean of array: 20.0 Check 3 in set: Present

2 File Handling in Python

File handling allows Python programs to read from and write to files. Python provides built-in functions to create, read, update, and delete files.

2.1 Opening a File

Use the `open()` function to open a file. It takes the file path and mode as arguments. The following table summarizes all available modes for opening text and binary files, providing a comprehensive reference for file handling.

Modes: - 'r': Read (default) - 'w': Write (creates file if not exists, truncates if exists) - 'a': Append (creates if not exists) - 'b': Binary mode (e.g., 'rb' for reading binary) - 'x': Exclusive creation (fails if file exists) - '+': Read and write (e.g., 'r+')

Example: Opening a file for reading.

```
file = open("example.txt", "r")
print(file.read())
file.close()
```

2.2 Closing a File

Always close files using `close()` to free resources. Better to use `with` statement for automatic closing.

Example with `with`:

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
# File is automatically closed here
```

2.3 Reading from a File

- read(): Reads the entire file. - readline(): Reads one line. - readlines(): Reads all lines into a list.

Example: Reading entire file.

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

Example: Reading line by line.

```
with open("example.txt", "r") as file:
    line = file.readline()
    while line:
        print(line.strip())
        line = file.readline()
```

Example: Reading all lines.

```
with open("example.txt", "r") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())
```

2.4 Writing to a File

- write(): Writes a string. - writelines(): Writes a list of strings.

Example: Writing to a file (overwrites).

```
with open("output.txt", "w") as file:
    file.write("Hello, World!\n")
    file.write("This is a test.")
```

Example: Appending to a file.

```
with open("output.txt", "a") as file:
    file.write("\nAppended line.")
```

Example: Writelines.

```
lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
with open("output.txt", "w") as file:
    file.writelines(lines)
```

2.5 Binary Files

For images, videos, etc., use binary modes ('rb', 'wb').

Example: Reading a binary file (e.g., image).

```
with open("image.jpg", "rb") as file:
    data = file.read()
    # Process binary data
```

Example: Writing binary.

```
data = b'\x00\x01\x02'  # Example binary data
with open("binary.bin", "wb") as file:
    file.write(data)
```

2.6 File Methods and Attributes

- `tell()`: Current position. - `seek(offset, whence)`: Move position. - `name`: File name. - `mode`: File mode. - `closed`: If closed.

Example:

```
with open("example.txt", "r") as file:
    print("File name:", file.name)
    print("Mode:", file.mode)
    file.seek(5)  # Move to position 5
    print("Position:", file.tell())
    content = file.read()
    print(content)
```

3 Exception Handling in Python

Exception handling manages errors gracefully using try-except blocks.

3.1 Try-Except Block

Try code that may raise an exception, catch with except.

Example:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

Output: Cannot divide by zero!

3.2 Multiple Except Blocks

Handle different exceptions.

Example:

```
try:
    num = int("abc")
except ValueError:
    print("Invalid number!")
except TypeError:
    print("Type error!")
```

Output: Invalid number!

3.3 Else and Finally

- Else: Runs if no exception. - Finally: Always runs.

Example:

```python
try:
    result = 10 / 2
except ZeroDivisionError:
    print("Division error!")
else:
    print("Result:", result)
finally:
    print("Execution complete.")
```

Output: Result: 5.0 Execution complete.

3.4 Raising Exceptions

Use `raise` to throw exceptions.

Example:

```python
def check_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative!")
    return age

try:
    check_age(-1)
except ValueError as e:
    print(e)
```

Output: Age cannot be negative!

3.5 File Handling with Exceptions

Common in files: FileNotFoundError, IOError.

Example: Handling file not found.

```python
try:
    with open("nonexistent.txt", "r") as file:
        print(file.read())
except FileNotFoundError:
    print("File not found!")
except IOError:
    print("IO error occurred!")
finally:
    print("File operation attempted.")
```

Output: File not found! File operation attempted.

3.6 Common File-Related Exceptions

The table below summarizes common exceptions thrown during file handling in Python, their causes, and how to handle them with simple examples.

4 Summary of File Types

4.1 Comparison of Text and Binary Files

The following table compares text and binary files in terms of data representation, file modes, use cases, advantages, and disadvantages. It highlights how text files are suited for readable content while binary files handle raw data efficiently, providing a clear distinction to help choose the appropriate type based on the applications needs.

4.2 When to Use Text vs. Binary Files

This table outlines scenarios for using text versus binary files, including specific examples of file extensions. It serves as a guide for selecting file types based on the nature of the data and the required operations, emphasizing practicality in real-world applications.

4.3 How to Use Text and Binary Files

The table below explains the methods for handling text and binary files in Python, with concise code examples. It demonstrates the differences in opening modes and data handling, making it easier to implement file operations correctly.

5 Assignments

5.1 Assignment 1

Write a program that reads the content of a text file specified by the user. Include exception handling for FileNotFoundError and print an appropriate message if the file does not exist.

5.2 Assignment 2

Create a program that writes a list of strings to a binary file using ’wb’ mode and then reads the binary data back, printing it as bytes. Handle any IOError that may occur during the process.

5.3 Assignment 3

Develop a function that appends user input to an existing text file. Use exception handling to catch and manage any permission-related errors or other IOExceptions, ensuring the program doesn’t crash.

5.4 Assignment 4

Implement a script that opens a file, uses seek to move to a specific position (e.g., the middle of the file), reads from there, and handles exceptions like ValueError for invalid seek positions or FileNotFoundError if the file is missing.

Function	Meaning	Example	Result
np.array()	Creates an array from a list.	np.array([1, 2, 3])	[1 2 3]
np.zeros()	Creates an array of zeros.	np.zeros(3)	[0. 0. 0.]
np.ones()	Creates an array of ones.	np.ones(3)	[1. 1. 1.]
np.empty()	Creates an uninitialized array.	np.empty(3)	Random values
shape	Returns the shape (dimensions).	arr.shape for [1,2,3]	(3,)
ndim	Returns the number of dimensions.	arr.ndim for 2D array	2
size	Returns the total number of elements.	arr.size for 2x3 array	6
sum()	Computes the sum of elements.	np.sum([1,2,3])	6
mean()	Computes the mean.	np.mean([1,2,3])	2.0
median()	Computes the median.	np.median([1,2,3])	2.0
var()	Computes the variance.	np.var([1,2,3])	0.666...
std()	Computes the standard deviation.	np.std([1,2,3])	0.816...
min()	Finds the minimum value.	np.min([1,2,3])	1
max()	Finds the maximum value.	np.max([1,2,3])	3
argmin()	Index of minimum value.	np.argmin([1,2,3])	0
argmax()	Index of maximum value.	np.argmax([1,2,3])	2
add()	Element-wise addition.	np.add([1,2], [3,4])	[4 6]
subtract()	Element-wise subtraction.	np.subtract([1,2], [3,4])	[-2 -2]
multiply()	Element-wise multiplication.	np.multiply([1,2], [3,4])	[3 8]
divide()	Element-wise division.	np.divide([1,2], [3,4])	[0.333 0.5]
dot()	Dot product.	np.dot([1,2], [3,4])	11
reshape()	Reshapes the array.	np.reshape([1,2,3,4], (2,2))	[[1 2] [3 4]]
transpose()	Transposes the array.	np.transpose([[1,2],[3,4]])	[[1 3] [2 4]]
vstack()	Stacks arrays vertically.	np.vstack([[1,2],[3,4]])	[[1 2] [3 4]] (same if same shape)
hstack()	Stacks arrays horizontally.	np.hstack([[1,2],[3,4]])	[1 2 3 4] (for 1D)

Table 1: Common NumPy Array Functions with Examples and Results

Mode	Description	Example Use	File Existence Behavior
'r'	Read (default, text mode)	Reading a text file (e.g., "example.txt")	Must exist, else FileNotFoundError
'w'	Write (creates file, truncates if exists, text mode)	Writing a new text file	Creates if not exists, overwrites if exists
'a'	Append (creates if not exists, text mode)	Adding to a log file	Creates if not exists, appends if exists
'r+'	Read and write (text mode)	Modifying an existing text file	Must exist, else FileNotFoundError
'w+'	Write and read (creates, truncates, text mode)	Creating and reading a text file	Creates if not exists, overwrites if exists
'a+'	Append and read (creates if not exists, text mode)	Appending and reading a log	Creates if not exists, appends if exists
'rb'	Read (binary mode)	Reading an image (e.g., "image.jpg")	Must exist, else FileNotFoundError
'wb'	Write (creates, truncates, binary mode)	Writing a binary file (e.g., "data.bin")	Creates if not exists, overwrites if exists
'ab'	Append (creates if not exists, binary mode)	Appending binary data	Creates if not exists, appends if exists
'rb+'	Read and write (binary mode)	Modifying a binary file	Must exist, else FileNotFoundError
'wb+'	Write and read (creates, truncates, binary mode)	Creating and reading binary data	Creates if not exists, overwrites if exists
'ab+'	Append and read (creates if not exists, binary mode)	Appending and reading binary data	Creates if not exists, appends if exists
'x'	Exclusive creation (fails if file exists, text mode)	Creating a new text file only if it doesnt exist	Throws FileExistsError if exists, creates if not
'xb'	Exclusive creation (fails if file exists, binary mode)	Creating a new binary file only if it doesnt exist	Throws FileExistsError if exists, creates if not

Table 2: File Opening Modes in Python

Exception Type	When Thrown	How to Handle	Example Code

FileNotFoundError	When opening a non-existent file in ’r’ or ’r+’ mode.	Use try-except to catch and print a message or create the file.	`try:`
			`    open('nonexistent.txt', 'r')`
			`except FileNotFoundError:`
			`    print("File not found!")`
			Output: File not found!

PermissionError	When no permission to read/write (e.g., read-only file in ’w’ mode).	Catch and inform user, or change permissions if possible.	`try:`
			`    open('protected.txt', 'w')`
			`except PermissionError:`
			`    print("Permission denied!")`
			Output: Permission denied!

IsADirectoryError	When treating a directory as a file (e.g., open(’dir/’)).	Ensure path is a file, not directory.	`try:`
			`    open('directory/', 'r')`
			`except IsADirectoryError:`
			`    print("Path is a directory!")`
			Output: Path is a directory!

NotADirectoryError	When a non-directory path is expected as directory.	Verify path structure.	`import os`
			`try:`
			`    os.listdir('file.txt')`
			`except NotADirectoryError:`
			`    print("Not a directory!")`
			Output: Not a directory!

IOError/OSError	General IO errors (e.g., disk full, invalid path).	Catch broadly and log error.	`try:`
			`    open('/invalid/path', 'r')`
			`except IOError as e:`
			`    print(f"IO error: {e}")`
			Output: IO error: [Errno 2] No such file...

ValueError	Invalid mode or seek position.	Validate inputs before operations.	`try:`
			`    open('file.txt', 'invalid')`
			`except ValueError:`
			`    print("Invalid mode!")`
			Output: Invalid mode!

Table 3: Common File Handling Exceptions in Python

Aspect	Text Files	Binary Files
Data Representation	Human-readable characters (e.g., ASCII, UTF-8). Stored as strings.	Raw binary data (bytes). Not human-readable.
File Modes	’r’, ’w’, ’a’, ’r+’, etc.	’rb’, ’wb’, ’ab’, ’rb+’, etc.
Use Cases	Configuration files, logs, scripts, CSV/JSON data. When data needs to be edited manually.	Images (JPEG, PNG), videos (MP4), executables, serialized objects. When preserving exact byte structure is crucial.
Advantages	Easy to read/edit with text editors. Platform-independent for plain text.	Efficient for non-text data. No encoding issues.
Disadvantages	Inefficient for large non-text data. Encoding/decoding required.	Not readable without specialized tools. Platform-dependent sometimes (e.g., endianness).

Table 4: Differences Between Text and Binary Files

File Type	When to Use	Examples
Text Files	When data is string-based, needs human readability, or for data exchange (e.g., APIs, configs). Suitable for logging or simple data storage.	.txt (plain text notes), .csv (data tables), .json (structured data), .py (Python scripts).
Binary Files	When dealing with non-text media, serialized objects, or performance-critical data where byte-level precision is needed.	.jpg (images), .mp3 (audio), .exe (executables), .pickle (serialized Python objects).

Table 5: When to Use Text and Binary Files with Examples

File Type	How to Use in Python	Example Code

Text Files	Open with text modes, use read-/write for strings. Handle encoding if needed (default UTF-8).	`with open(’data.txt’, ’w’) as f:`
		`    f.write(’Hello’)`
		`with open(’data.txt’, ’r’) as f:`
		`    print(f.read())  # Hello`

Binary Files	Open with binary modes, use bytes for read/write. No automatic encoding.	`with open(’data.bin’, ’wb’) as f:`
		`    f.write(b’\x48\x65\x6C\x6C\x6F’)`
		`with open(’data.bin’, ’rb’) as f:`
		`    print(f.read())  # b’Hello’`

Table 6: How to Use Text and Binary Files with Examples