MODULE 3
INTRODUCTION TO PYTHON PROGRAMMING

Syllabus: Manipulating Strings: Working with Strings, Useful String Methods, Project: Password Locker, Project: Adding Bullets to Wiki Markup Reading and Writing Files: Files and File Paths, The os.path Module, The File Reading/Writing Process, Saving Variables with the shelve Module,Saving Variables with the print.format() Function, Project: Generating Random Quiz Files, Project: Multi Clipboard, Textbook 1: Chapters 6 , 8

DEFINITION: In programming, a **string** is a sequence of characters enclosed in quotation marks. These characters can be letters, numbers, symbols, or whitespace.

Types of Strings:

1. **Single-quoted strings**:
   - Created using single quotes (' ')
   - Example: 'Hello'
2. **Double-quoted strings**:
   - Created using double quotes (" ")
   - Example: "World"
3. **Raw strings**:
   - Created by prefixing the string with r or R (e.g., r'...' or R'...').
   - These strings treat backslashes as literal characters and not escape characters.
   - Example: r'C:\Users\Al'
4. **Multiline strings**:
   - Created using triple quotes (''' or """).
   - Used for strings that span multiple lines.

Example:

```
'''This is a
multiline string.'''
```

String Example:

```
name = "Alice"  # This is a string
```

Strings are widely used for text processing, storing data, and more in programming.

1. WORKING WITH STRINGS

Python provides several ways to create, manipulate, and display strings. Let's break it down step by step:

String Literals:A string in Python is a sequence of characters enclosed in quotes. There are two types of string quotes:

AALIYA WASEEM, AIML, JNNCE

1. Single Quotes: 'This is a string'
2. Double Quotes: "This is a string"

Using Double Quotes for Strings: If you want to include a single quote inside a string, you can use double quotes:

spam = "That's Alice's cat."

In this case, the double quotes tell Python where the string starts and ends, and the single quote inside the string is not considered as the end of the string.

Escape Characters: Sometimes, you need special characters inside a string, such as quotes, backslashes, or new lines. This is where **escape characters** come in.

What is an Escape Character?

An escape character is a backslash (\) followed by a character that modifies the string. Even though it consists of two characters, it is considered a single escape character.

Table 6-1: Escape Characters

Escape character	Prints as
\'	Single quote
\"	Double quote
\t	Tab
\n	Newline (line break)
\\	Backslash

Example of Escape Characters

Single Quote Inside a String:
spam = 'Say hi to Bob\'s mother.'

Double Quote Inside a String:
spam = "He said, \"Hi!\""

Using Escape Characters in Strings

Here are some practical examples:

```
# Single quote inside single quotes
spam = 'That\'s Alice\'s cat.'
# Double quote inside double quotes
spam = "He said, \"Hello!\""
# Backslash
```


AALIYA WASEEM,DEPT.AIML,JNNCE

```
path = 'C:\\Users\\Alice'
# New line
poem = 'Roses are red,\nViolets are blue.'
# Tab
tabbed_text = 'Name\tAge\tLocation'
```

Conclusion

- **Single and Double Quotes**: Used to create strings.
- **Escape Characters**: Allow special characters like quotes, backslashes, and new lines inside strings.

This makes it easy to include complex text in Python strings without errors.

2. RAW STRINGS MULTILINE STRINGS

Raw Strings: A raw string is a string where Python ignores all escape characters. This means that backslashes are treated as part of the string, not as escape characters.

Why use raw strings? When working with strings that contain many backslashes, such as file paths or regular expressions.

Example of a Raw String:

```
print(r'C:\Users\Al\Desktop')
```

Output: C:\Users\Al\Desktop

In this case, the \n, \t, and \ characters are treated as part of the string rather than special escape characters.

Multiline Strings: Sometimes you need a string that spans multiple lines. Instead of using multiple \n characters, you can use triple quotes.

- Begins and ends with three single quotes ''' or three double quotes """.
- All lines within the triple quotes are considered part of the string.

Example of a Multiline String:

```
print('''Hello,
This is a multiline string example.
You can include newlines, single quotes, and double quotes.''')
```

Output:
Hello,
This is a multiline string example.
You can include newlines, single quotes, and double quotes.

AALIYA WASEEM, AIML, JNNCE

You can also escape quotes inside multiline strings, but it's optional:

```
print('''Dear Alice,
Eve's cat has been arrested for catnapping, cat burglary, and extortion.
Sincerely,
Bob''')
```

Output:

```
Dear Alice,
Eve's cat has been arrested for catnapping, cat burglary, and extortion.
Sincerely,
Bob
```

Using Triple Quotes for Comments: Multiline strings can also be used for comments that span multiple lines.

Example:

```
# Regular comment
print('Hello!')  # This is a single-line comment

# Multiline comment using triple quotes
"""This is a test Python program.
This is useful when you need to write long explanations or document a program in a more detailed way"""
```

AALIYA WASEEM, AIML, JNNCE

3. INDEXING AND SLICING STRINGS

Think of a string as a line of letters, numbers, or symbols. For example:
"Hello, world!"

Each character in the string has a position number called an index. These start at 0 for the first character, then go 1, 2, 3, and so on.

Here's how the string "Hello, world!" looks with its index numbers:

```
H e l l o ,   w o r l d !
0 1 2 3 4 5 6 7 8 9 10 11 12
```

Accessing Single Characters

If you want to grab just one character, you can use its **index** like this:

```
spam = "Hello, world!"

print(spam[0])  # Gets 'H', the character at position 0

print(spam[4])  # Gets 'o', the character at position 4

print(spam[-1]) # Gets '!', the last character
```

Slicing (Getting Parts of the String)

You can also take **parts of the string** by slicing it. To slice, you give a **starting index** and an **ending index** like this:

```
print(spam[0:5])  # Gets characters from position 0 to 4: 'Hello'
```

- The starting index is **included**.
- The ending index is **not included**.

So spam[0:5] means:
Start at position 0 (H), go up to (but not include) position 5.

Shortcuts for Slicing

If you don't give a starting or ending index, Python assumes some defaults:

- spam[:5] means "start at the beginning, stop at position 5."
- spam[7:] means "start at position 7, go to the end."

Example: print(spam[:5]) # 'Hello' (start from the beginning)

```
print(spam[7:])  # 'world!' (start at position 7)
```

Original String Stays the Same

When you slice a string, Python gives you a **copy** of the part you asked for. The original string doesn't change:

```
spam = "Hello, world!"
fizz = spam[0:5]
print(fizz)  # 'Hello' (just the part you sliced)
print(spam)  # 'Hello, world!' (the original string is still the same)
```

Key Points:

1. Strings are like lists of characters, and you can use numbers (indexes) to pick parts of them.
2. Indexes start at 0 for the first character.
3. Slicing lets you pick parts of the string, from one position to another.

AALIYA WASEEM, AIML, JNNCE

4. The original string doesn't change when you slice it.

4. THE IN AND NOT IN OPERATORS WITH STRINGS

The in and not in operators let you check if one string is inside another string. These checks give you a simple True or False answer.

Here's how it works, step by step:

The in Operator

The in operator checks if a smaller string (a substring) is inside a bigger string.
If the smaller string is found, the result is True. If it's not found, the result is False.

Examples:

print('Hello' in 'Hello, World') # True, because 'Hello' is inside 'Hello, World'

print('HELLO' in 'Hello, World') # False, because Python is case-sensitive ('HELLO' ≠ 'Hello')

print('' in 'spam') # True, because an empty string is technically everywhere

The not in Operator

The not in operator checks if a smaller string is not inside a bigger string.
If the smaller string is not found, the result is True. If it's found, the result is False.

Examples:

print('cats' not in 'cats and dogs') # False, because 'cats' *is* inside 'cats and dogs'

print('fish' not in 'cats and dogs') # True, because 'fish' is not in 'cats and dogs'

Key Things to Remember:

1. Case matters: 'Hello' and 'HELLO' are not the same.
2. Spaces count: 'Hello ' in 'Hello, World' would be False because of the extra space.
3. The result is always True or False (Boolean).

Think of it as asking:

- in: "Does this piece exist in that string?"
- not in: "Is this piece missing from that string?

5. PUTTING STRINGS INSIDE OTHER STRINGS

Putting strings inside other strings is a common task in Python. Here are three simple ways to do it:

1. Using + (Concatenation)

You join strings using the + operator, but you must convert numbers to strings with str():

AALIYA WASEEM , AIML, JNNCE

```
name = 'Al'

age = 4000

print('Hello, my name is ' + name + '. I am ' + str(age) + ' years old.')

# Output: Hello, my name is Al. I am 4000 years old.
```

2. Using %s (String Interpolation)

The %s acts as a placeholder for values. Put the values in a tuple ( ) after the string:

```
name = 'Al'

age = 4000

print('My name is %s. I am %s years old.' % (name, age))

# Output: My name is Al. I am 4000 years old.
```

3. Using f-strings (Best and Easiest)

Start the string with f and use {} to insert variables or expressions directly:

```
name = 'Al'

age = 4000

print(f'My name is {name}. Next year I will be {age + 1}.')

# Output: My name is Al. Next year I will be 4001.
```

Key Point:

- Use f-strings for simplicity and readability. Don't forget the f before the string

6. USEFUL STRING METHODS

Here's an easy guide to some helpful string methods, with examples to make things clear!

1. Changing Case

- upper(): Converts all letters to uppercase.
- lower(): Converts all letters to lowercase.

Examples:

```
text = "Hello, World!"

print(text.upper())  # 'HELLO, WORLD!'

print(text.lower())  # 'hello, world!'
```

AALIYA WASEEM, AIML, JNNCE

Note: These methods don't change the original string. If you want to save the result, assign it to a variable:

```
text = text.upper()  # Saves the uppercase version in the same variable
```

2. Checking Case

- isupper(): Checks if all letters are uppercase.
- islower(): Checks if all letters are lowercase.

Examples:

```
print("HELLO".isupper())  # True

print("hello".islower())  # True

print("Hello123".islower())  # False (not all letters are lowercase)
```

3. String Validation Methods (isX)

These methods check what a string is made of. They return True or False.

- isalpha(): Only letters, no spaces or numbers.
- isalnum(): Letters and numbers only, no spaces.
- isdecimal(): Only numbers.
- isspace(): Only spaces, tabs, or newlines.
- istitle(): Each word starts with an uppercase letter.

Examples:

```
print("hello".isalpha())  # True
print("hello123".isalpha())  # False (contains numbers)
print("hello123".isalnum())  # True
print("12345".isdecimal())  # True
print("   ".isspace())  # True
print("This Is Title Case".istitle())  # True
```

4. Checking Start and End

- startswith(): Checks if a string starts with a specific substring.
- endswith(): Checks if a string ends with a specific substring.

Examples:

```
text = "Hello, world!"
print(text.startswith("Hello"))  # True
print(text.endswith("world!"))   # True
print(text.endswith("Hello"))    # False
```

5. Combining Methods

You can chain methods together:

AALIYA WASEEM, AIML, JNNCE

```
text = "Hello"

print(text.upper().lower().isupper())  # False (ends up as lowercase)
```

6. Real-Life Examples

Example 1: Case-Insensitive Comparisons

```
print("How are you?")
feeling = input()  # User types: "GREat"
if feeling.lower() == "great":
    print("I feel great too!")
else:
    print("I hope your day gets better!")
```

Example 2: Validating Input

```
while True:
    age = input("Enter your age: ")
    if age.isdecimal():  # Ensures input is only numbers
        break
    print("Please enter a valid number.")
while True:
    password = input("Enter a password (letters and numbers only): ")
    if password.isalnum():  # Ensures no special characters or spaces
        break

    print("Passwords can only have letters and numbers.")
```

EM, AIML, JNNCE

Method	Description	Example	Output
`upper()`	Converts all letters to uppercase	`"Hello".upper()`	`'HELLO'`
`lower()`	Converts all letters to lowercase	`"HELLO".lower()`	`'hello'`
`isupper()`	Checks if all letters are uppercase	`"HELLO".isupper()`	`True`
`islower()`	Checks if all letters are lowercase	`"hello".islower()`	`True`
`isalpha()`	Checks if only letters (no numbers/spaces)	`"abc".isalpha()`	`True`
`isalnum()`	Checks if letters and numbers only	`"abc123".isalnum()`	`True`
`isdecimal()`	Checks if only numbers	`"123".isdecimal()`	`True`
`isspace()`	Checks if only spaces/tabs/newlines	`" ".isspace()`	`True`
`istitle()`	Checks if title case (each word capitalized)	`"Hello World".istitle()`	`True`
`startswith()`	Checks if string starts with a substring	`"Hello".startswith("He")`	`True`
`endswith()`	Checks if string ends with a substring	`"Hello".endswith("lo")`	`True`

1. The join() Method

The **join()** method combines a list of strings into a single string, inserting a specified string between each item.

How It Works:

- **Called on a string**: The string specifies what goes **between** each list item.
- **Pass a list**: Provide the list of strings you want to combine.

Examples:

```
# Join with a comma and space

print(', '.join(['cats', 'rats', 'bats']))  # Output: 'cats, rats, bats'

# Join with a space

print(' '.join(['My', 'name', 'is', 'Simon']))  # Output: 'My name is Simon'

# Join with 'ABC'

print('ABC'.join(['My', 'name', 'is', 'Simon']))  # Output: 'MyABCnameABCisABCSimon'
```

Key Points:

- The string (', ', ' ', 'ABC') is **inserted between** each list item.
- The **list must only contain strings**, or you'll get an error.

2. The split() Method

The **split()** method breaks a string into a list of smaller strings. By default, it splits wherever there's **whitespace** (spaces, tabs, newlines).

How It Works:

- **Called on a string**: The string is the one you want to split.
- **Returns a list**: Breaks the string into parts.

Examples:

```
# Split by spaces (default)

print('My name is Simon'.split())  # Output: ['My', 'name', 'is', 'Simon']

# Split by a custom string ('ABC')

print('MyABC name ABCABC Simon'.split('ABC'))  # Output: ['My', 'name', 'is', 'Simon']

# Split by the letter 'm'

print('My name is Simon'.split('m'))  # Output: ['My na', 'e is Si', 'on']
```

Key Points:

AALIYA WASEEM, AIML, JNNCE

- By default, it splits wherever there's **whitespace**.
- You can specify a **delimiter** (like 'ABC' or 'm') to split the string in a specific way.

3. Splitting Multiline Strings

The **split('\n')** method splits a string into a list of lines. Each line ends where there's a newline (\n).

Example:

```
spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment."
Please do not drink it.
Sincerely,
Bob'''
# Split by newline characters
print(spam.split('\n'))
```

Output:

```
[
 'Dear Alice,',
 'How have you been? I am fine.',
 'There is a container in the fridge',
 'that is labeled "Milk Experiment."',
 '',
 'Please do not drink it.',
 'Sincerely,',
 'Bob'
]
```

Each line becomes an item in the list. Empty lines are included as empty strings ('').

Method	Action	Example	Output
`join()`	Combines a list into a single string	`', '.join(['cats', 'rats'])`	`'cats, rats'`
`split()` (default)	Splits a string by whitespace	`'My name is Simon'.split()`	`['My', 'name', 'is', 'Simon']`
`split(delimiter)`	Splits a string by a specific substring	`'MyABCname'.split('ABC')`	`['My', 'name']`
`split('\n')`	Splits a string into lines	`'Line1\nLine2'.split('\n')`	`['Line1', 'Line2']`

The partition() Method

The **partition()** method splits a string into three parts based on a specific **separator** string you provide. It creates a tuple with three parts:

WASEEM, AIML, JNNCE

1. **Before** the separator.
2. The **separator** itself.
3. **After** the separator.

How It Works:

1. **Call it on a string** and provide the separator as an argument.
2. The method looks for the first occurrence of the separator.
3. Returns a tuple with:
   - Text **before** the separator.
   - The **separator**.
   - Text **after** the separator.

Examples:

```
# Split using 'w' as the separator

print('Hello, world!'.partition('w'))

# Output: ('Hello, ', 'w', 'orld!')

# Split using 'world' as the separator

print('Hello, world!'.partition('world'))

# Output: ('Hello, ', 'world', '!')
```

If the Separator Appears Multiple Times: The method only splits at the **first occurrence** of the separator.

```
# Split using 'o' as the separator

print('Hello, world!'.partition('o'))

# Output: ('Hell', 'o', ', world!')
```

If the Separator Is Not Found: The whole string is returned as the **first part**, and the second and third parts are empty strings.

```
# Separator 'XYZ' not found

print('Hello, world!'.partition('XYZ'))

# Output: ('Hello, world!', '', '')
```

Assigning the Results to Variables: You can use **multiple assignment** to save the three parts in separate variables.

```
# Split using a space (' ') as the separator

before, sep, after = 'Hello, world!'.partition(' ')
```

AALIYA WASEEM, AIML, JNNCE

```
# Access the parts

print(before)  # Output: 'Hello,'

print(sep)     # Output: ' '

print(after)   # Output: 'world!'
```

Why Use Partition()?

- Quickly divide a string into **before**, **separator**, and **after** parts.
- Useful for tasks like splitting a sentence into its components or isolating specific sections of a string.

In short, **partition()** makes it easy to break a string into three useful pieces based on a specific keyword or character

Justifying Text with rjust(), ljust(), and center()

These methods help you align text neatly by adding **extra spaces** (or other characters) to the left, right, or both sides of your string. This is super useful when formatting output like tables or lists. Let's break it down:

rjust(): Right-Justify

- Adds **spaces** to the **left** of the string so the text is pushed to the **right**.
- The number you provide is the **total length** of the new string.

Example:

```
print('Hello'.rjust(10))

# Output: '     Hello'  (5 spaces added on the left to make it 10 characters long)
```

You can also add **other characters** instead of spaces:

```
print('Hello'.rjust(10, '*'))

# Output: '*****Hello'
```

ljust(): Left-Justify

- Adds **spaces** to the **right** of the string so the text stays on the **left**.
- Again, the number you provide is the **total length** of the new string.

Example:

```
print('Hello'.ljust(10))

# Output: 'Hello     '  (5 spaces added on the right)
```

Use custom characters too:

AALIYA WASEEM, AIML, JNNCE

```
print('Hello'.ljust(10, '-'))

# Output: 'Hello-----'
```

center(): Center the Text

- Adds **spaces** (or custom characters) to **both sides** of the string to center it.
- The total length of the resulting string is determined by the number you provide.

Example:

```
print('Hello'.center(10))

# Output: '  Hello   ' (2 spaces on each side for a total of 10 characters)
```

Use custom characters:

```
print('Hello'.center(10, '='))

# Output: '==Hello==='
```

7. PROJECT: ADDING BULLETS TO WIKI MARKUP

This project teaches you how to create a Python program (bulletPointAdder.py) that automates adding bullet points (*) to the start of each line of text. Here's how it works in simple terms:

What the Program Does:

1. Take text from the clipboard (the text you copied).
2. Add a star (*) at the start of every line.
3. Put the modified text back onto the clipboard so you can paste it anywhere.

This is useful if you want to create a bulleted list for something like a Wikipedia article, but don't want to add * manually for each line.

How It Works:

Step 1: Copy and Paste from the Clipboard

Python has a library called pyperclip that lets you:

- Get text from the clipboard using pyperclip.paste().
- Put text back onto the clipboard using pyperclip.copy().

Basic setup for the program:

```
import pyperclip

text = pyperclip.paste()  # Get text from clipboard

# TODO: Modify the text to add bullets
```

AALIYA WASEEM, AIML, JNNCE

pyperclip.copy(text) # Put the modified text back on the clipboard

Step 2: Add Bullets to Each Line

The clipboard text is a single long string, but we want to:

1. Break it into separate lines.
2. Add a star (*) to the start of each line.

Use the split('\n') method to split the text into a list of lines:t

```
lines = text.split('\n')  # Break text into a list of lines
```

Loop through each line and add a star at the start:

```
for i in range(len(lines)):  # Loop through each line
    lines[i] = '* ' + lines[i]  # Add "* " at the start of each line
```

Step 3: Combine the Lines Back Together

Now that every line starts with a star, combine them back into a single string using '\n'.join(lines):

```
text = '\n'.join(lines)  # Combine the list into a single string with newlines
```

Step 4: Update the Clipboard

Finally, put the modified text (with bullets) back onto the clipboard so it's ready to paste:

```
pyperclip.copy(text)  # Copy the modified text back to the clipboard
```

Final Code:

Here's the complete program:

```
import pyperclip

# Get text from clipboard

text = pyperclip.paste()

# Add a star to the start of each line

lines = text.split('\n')  # Split text into lines

for i in range(len(lines)):  # Loop through each line

    lines[i] = '* ' + lines[i]  # Add a star to the start of the line

# Join the lines back into a single string

text = '\n'.join(lines)
```

AALIYA WASEEM, AIML, JNNCE

Copy the modified text back to the clipboard

pyperclip.copy(text)

Why This is Useful:

You can adapt this program to:

- Remove spaces.
- Add numbers or custom symbols.
- Change the format of your text for specific tasks.

8. PROJECT : PASSWORD LOCKER

1. Start

- The program begins its execution.

2. Check Command-Line Arguments

- The program checks if you have provided the account name (the account you want the password for) as an input.
- If no account name is provided:
  - The program displays a usage message like:
    "Usage: python pw.py [account_name]"
    and stops.
- If an account name is provided, it moves to the next step.

3. Look Up the Account

- The program checks whether the provided account name exists in the password dictionary (where all accounts and their passwords are stored).

4. Does the Account Exist?

- Yes: If the account exists:
  - The program copies the password to the clipboard using the pyperclip module.
  - It also prints a message like:
    "Password for [account_name] copied to clipboard!"
- No: If the account doesn't exist:
  - The program displays a message:
    "There is no account named [account_name]"

5. End

- After the above steps, the program ends.

AALIYA WASEEM, AIML, JNNCE

Why Is This Useful?

- You don't need to memorize complex passwords.
- Just run this script and paste the password where needed.
- Safe and efficient for managing multiple accounts!

Scenario: Using the Password Locker

You Have Multiple Accounts Imagine you have passwords for different accounts like:

- Email: email123password
- Facebook: fb_secure2025
- Twitter: tweet@secure!

You don't want to memorize all these passwords. Instead, you've written a Python program called pw.py to store and retrieve these passwords securely.

How It Works: The program has a dictionary where the account names (like "email", "facebook", "twitter") are the keys, and their passwords are the values.

```
PASSWORDS = {
  "email": "email123password",
  "facebook": "fb_secure2025",
  "twitter": "tweet@secure!"
}
```

Example: Retrieving a Password Let's say you need the password for your Facebook account.
Steps

Open the terminal and type:
python pw.py facebook

Here, "facebook" is the account name you want the password for.

What Happens Next?

- The program checks if you provided an account name. Since you typed "facebook," it moves forward.
- It looks in the dictionary for the "facebook" account.
- **Found it!** The program copies fb_secure2025 to your clipboard (using the pyperclip module).

The terminal prints:
Password for facebook copied to clipboard!

You Paste the Password
Go to the Facebook login page, click the password field, and press **Ctrl + V** to paste the password.

What If the Account Doesn't Exist? Let's say you type:
python pw.py instagram

The program will check for "instagram" in the dictionary but won't find it.

AALIYA WASEEM, AIML, JNNCE

The terminal will print:
There is no account named instagram

What If You Forget to Provide an Account Name? If you just type: python pw.py

The program will notice you didn't provide an account name and print:
Usage: python pw.py [account_name]

This reminds you to include the account name when running the program.

Summary

Command: python pw.py [account_name]

- If the account exists: Copies the password and prints a success message.
- If the account doesn't exist: Tells you the account isn't found.
- If no account name is provided: Shows usage instructions.

CHAPTER 8

1. READING AND WRITING FILES

Definition: A file path is the location of a file on your computer, like an address that tells your system where to find it. It consists of:

- **Filename**: The name of the file (e.g., document.txt).
- **Path**: The hierarchy of folders that leads to the file (e.g., C:\Users\Al\Documents\document.txt).

Concept

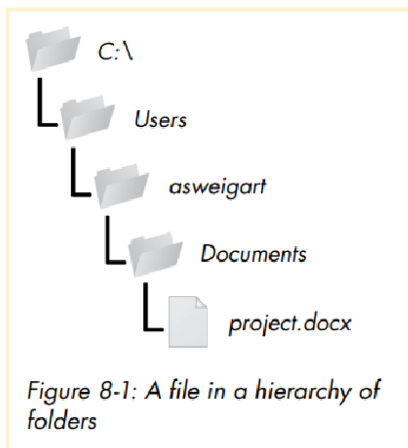

Figure 8-1: A file in a hierarchy of folders

1. **File Extensions**:
   - The part after the last dot in the filename (e.g., .txt, .docx).
   - Tells what type of file it is (e.g., .docx for Word documents, .png for images).
2. **Path Structure**:
   - **Root Folder**: The starting point of the path.
     - Windows: C:\
     - macOS/Linux: /

WASEEM, AIML, JNNCE

- **Folders (Directories)**: Containers for files and other folders.
    - Example: C:\Users\Al\Documents means:
        - C:\: Root folder (drive).
        - Users: Folder inside the root.
        - Al: Folder inside Users.
        - Documents: Folder inside Al.

3. **Separators**:
    - **Windows**: Backslash \ separates folders.
    - **macOS/Linux**: Forward slash / separates folders.
4. **Pathlib Module**:
    - A Python module to handle paths easily.
    - Automatically adjusts the separator based on the operating system.
    - Uses the / operator to join paths (even on Windows).

Examples

Basic Example of a File Path

- Windows: C:\Users\Al\Documents\project.docx
- macOS/Linux: /Users/Al/Documents/project.docx

Key Advantages of pathlib

1. **Cross-Platform**:
    - Works on Windows, macOS, and Linux without worrying about slashes.
2. **Readable Code**:
    - The / operator makes paths easy to join.
3. **Error Prevention**:
    - Avoids common bugs caused by manually adding slashes or using os.path.

1. Understanding Files and Paths

- **Filename**: The name of the file (e.g., projects.docx).
- **Path**: The location of the file on your computer, which includes the folder hierarchy. In Windows, a path might look like C:\Users\asweigart\Documents\projects.docx. In other operating systems like macOS or Linux, paths have a similar structure but may use different slashes (/).

2. Choosing a File Format

- Decide the format of the file. Common formats include:
    - **Text files** (.txt)
    - **Comma-separated values** files (.csv)
    - **Excel files** (.xlsx)
    - **JSON** files (.json)
    - **Binary files** (.bin)

3. Creating or Opening a File

Creating a New File: Use functions or methods to create a new file.

AALIYA WASEEM, AIML, JNNCE

Example (in Python):

```
with open('filename.txt', 'w') as file:
    file.write('Initial content')
```

Opening an Existing File:Use functions or methods to open an existing file and perform read or write operations.

Example (in Python):

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

4. Writing Data to a File: After opening a file in write mode ('w'), data can be written using functions like write() or writelines().

Example:

```
with open('filename.txt', 'w') as file:
    file.write('Hello, world!')
```

5. Appending to a File: Use 'a' mode to append data to an existing file.

Example:

```
with open('filename.txt', 'a') as file:
    file.write('\nAppended text')
```

6. Reading Data from a File: Use read modes ('r', 'rb' for binary) to read data from files.

Example:

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

7. Handling File Paths

- Ensure paths are correctly specified based on the operating system (Windows, macOS, Linux).
- Example:
    - Windows path: C:\Users\username\Documents\filename.txt
    - macOS/Linux path: /Users/username/Documents/filename.txt

8. Closing the File: Always close the file after reading or writing operations to free up system resources.

Example:

```
file.close()
```

9. Error Handling: Handle errors such as file not found or permissions issues.

Example:

```
try:
    with open('filename.txt', 'r') as file:
```

AALIYA WASEEM, AIML, JNNCE

```
    content = file.read()
except FileNotFoundError:
  print("File not found.")
```

By following these steps, data can be stored in files and accessed or modified as needed even after the program has stopped running.

2. THE CURRENT WORKING DIRECTORY

Current Working Directory (cwd): Current Working Directory (cwd): This is the location from which a program starts working. Any files or directories you refer to will be relative to this location unless a full path is given.

Examples:

os.getcwd() returns the current working directory.

```
import os

os.getcwd()
```

Changing the current working directory with os.chdir('path'):

```
os.chdir('C:\\Windows\\System32')
```

If a file is named project.docx and your cwd is C:\Python34, then project.docx refers to C:\Python34\project.docx.

Error Example: Trying to change to a non-existent directory will result in an error.

```
os.chdir('C:\\ThisFolderDoesNotExist')
```

Absolute vs. Relative Paths

1. **Absolute Path:**
   - Starts from the root folder (e.g., C:\Users\username\Documents\file.txt).
2. **Relative Path:**
   - Starts from the current working directory.
   - Example: project.docx if cwd is C:\Python34.

Dot (.) and Dot-dot (..)

1. Dot (.): Refers to "this directory" (the current directory).
   - Example: .\myfile.txt refers to C:\Python34\myfile.txt when cwd is C:\Python34.
2. Dot-dot (..): Refers to "the parent folder" (the directory one level above).
   - Example: ..\myfile.txt when cwd is C:\Python34 refers to C:\myfile.txt.

Summary

- Current Working Directory (cwd) is where Python looks for files by default.
- Absolute Path includes the full path (e.g., C:\path\to\file.txt).
- Relative Path starts from cwd and uses . or .. to navigate directories.

AALIYA WASEEM, AIML, JNNCE

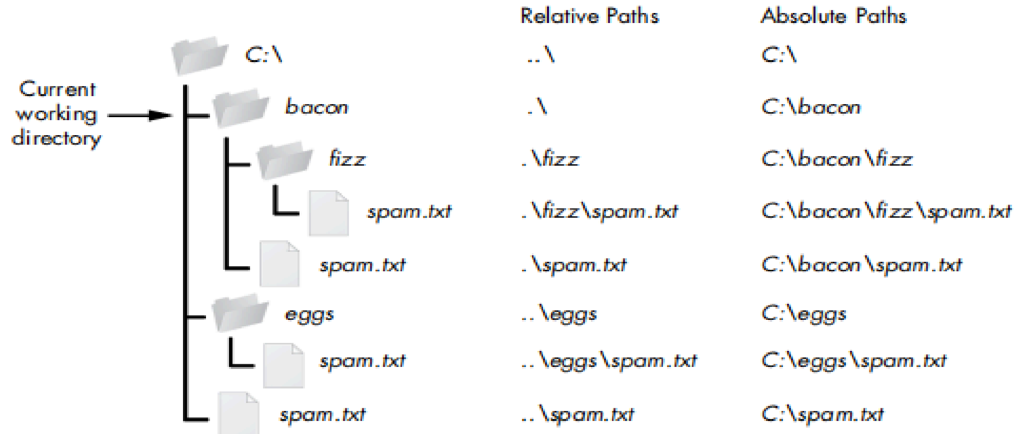

Figure 8-2: The relative paths for folders and files in the working directory C:\bacon

3. OS PATH MODULE

The os.path module in Python provides a variety of helpful functions for working with file paths. Let's break it down in an easy way with simple scenarios:

1. Working with Absolute and Relative Paths

- Absolute Path: A full path from the root directory (e.g., C:\Users\username\Documents\file.txt).
- Relative Path: A path relative to the current working directory (e.g., file.txt if your cwd is C:\Users\username\Documents).

Scenario: Suppose you have a Python project and your current working directory (cwd) is C:\Python34.

You want to get the absolute path of a folder named Scripts:

```
import os

print(os.path.abspath('.\\Scripts'))
```

Output: C:\Python34\Scripts

2. Checking if a Path is Absolute: Use os.path.isabs(path) to check if a path is absolute.

Scenario: You have a relative path ./Scripts and want to check if it's absolute:

```
import os

print(os.path.isabs('./Scripts'))
```

Output: False

Now, if you convert it to an absolute path:

```
print(os.path.isabs(os.path.abspath('./Scripts')))
```

AALIYA WASEEM, AIML,

Output: True

3. Relative Paths: Use os.path.relpath(path, start) to get a relative path from a starting point.

Scenario:

If you want a relative path from C:\ to C:\Windows:
print(os.path.relpath('C:\\Windows', 'C:\\'))

Output: 'Windows'

Now, if you want to go from C:\spam\eggs to C:\Windows:
print(os.path.relpath('C:\\Windows', 'C:\\spam\\eggs'))

Output: '..\\..\\Windows' (moving up two directories)

4. Getting Directory Name and Base Name

- os.path.dirname(path) gives you everything before the last slash.
- os.path.basename(path) gives you everything after the last slash.

Scenario:

You have a file path C:\Windows\System32\calc.exe:

```
import os
path = 'C:\\Windows\\System32\\calc.exe'
print(os.path.dirname(path))
print(os.path.basename(path))
```
Output:

    - C:\Windows\System32
    - calc.exe

5. Combining Directory and Base Name: Use os.path.split(path) to get a tuple of the directory and base name.

Scenario:

Using C:\Windows\System32\calc.exe:

```
import os
calcFilePath = 'C:\\Windows\\System32\\calc.exe'
print(os.path.split(calcFilePath))
```
- Output:
    - Directory: C:\Windows\System32
    - Base name: calc.exe

6. Splitting a Path: path.split(os.path.sep) splits a path into its components based on the system's file separator (os.sep).

Scenario:

On Windows:

print(calcFilePath.split(os.path.sep))

- Output: ['C:', 'Windows', 'System32', 'calc.exe']

On macOS or Linux:

print('/usr/bin'.split(os.path.sep))

- Output: ['', 'usr', 'bin']

Summary

- Absolute Paths: Full paths from the root directory.
- Relative Paths: Paths relative to the current working directory.
- os.path functions help handle paths easily across different systems, checking if paths are absolute, creating relative paths, and separating paths into directories and file names.

AALIYA WASEEM, AIML, JNNCE

4. THE FILE READING/WRITING PROCESS

In Python, working with files involves three main steps: opening the file, performing read or write operations, and then closing the file.

1. Opening a File: To open a file, use the open() function.

Syntax: file = open('filename.txt', mode)

mode can be:

- 'r' for read (default mode).
- 'w' for write (creates a new file or overwrites if the file already exists).
- 'a' for append (adds content to the end of the file).

Scenario: Opening a file in read mode:

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

2. Reading/Writing the File

Once the file is opened, use methods like read() or write() to interact with it.

Reading from a file:
read() reads the entire file content as a string.

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

Writing to a file:

write(data) writes data into the file.

```
file.write('Hello, world!')
```

Scenario: Reading from a file:

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

Writing to a file:

```
file = open('example.txt', 'w')
file.write('New content')
file.close()
```

3. Closing the File: Always close the file after you finish reading or writing to it to free up system resources.

```
file.close()
```

Scenario: After writing or reading:

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

Summary

To work with files in Python:

1. Open the file using open('filename', mode).
2. Read or Write data using read() or write(data).
3. Close the file with file.close().

This process handles plaintext files such as .txt or .py files, where content is simple text.

5. OPENING AND READING FILES IN PYTHON

In Python, you can open files using the open() function. Let's break it down step by step with simple examples.

1. Opening a File

To open a file:

- Use open('path_to_file') where path_to_file can be an absolute or relative path.
- For example, to open a file named hello.txt stored in your user home folder:
  - **Windows**: open('C:\\Users\\your_home_folder\\hello.txt')
  - **MacOS/Unix**: open('/Users/your_home_folder/hello.txt')

Example:

AALIYA WASEEM, AIML, JNNCE

- On Windows: helloFile = open('C:\\Users\\asweigart\\hello.txt')
- On MacOS: helloFile = open('/Users/asweigart/hello.txt')

2. Reading Files

Once you have a File object, you can use different methods to read its contents:

a) read() Method

- read() reads the entire file as a single string.

Example:

```
helloFile = open('hello.txt')
helloContent = helloFile.read()
print(helloContent)
```

Output:
Hello world!

b) readlines() Method

- readlines() reads the file line by line and returns a list of strings, where each string represents a line.

Example:

Creating sonnet29.txt with four lines:

When, in disgrace with fortune and men's eyes,
I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries,
And look upon myself and curse my fate,
Reading the file with readlines():

```
sonnetFile = open('sonnet29.txt')
sonnetLines = sonnetFile.readlines()
print(sonnetLines)
```

Output:

['When, in disgrace with fortune and men\'s eyes,\n', ' I all alone beweep my outcast state,\n', ' And trouble deaf heaven with my bootless cries,\n', ' And look upon myself and curse my fate,\n']

3. Closing the File: After you're done reading or writing to a file, always remember to close it to free up system resources:

```
helloFile.close()
```

Summary

- Use open('path') to open a file.

AALIYA WASEEM, AIML, JNNCE

- Use read() to get the entire content as a string.
- Use readlines() to get a list of lines from the file.
- Always close the file with close() after you're done

6. SAVING VARIABLES WITH THE SHELVE MODULE

The shelve module in Python makes it easy to save and load data from your program so it doesn't get lost when you close the program. It works like a magic storage box that saves data to your computer and lets you retrieve it later.

Here's how it works in simple terms:

Saving Data:

- You create a "shelf" (a special storage object) by calling shelve.open() and giving it a filename. This creates some hidden files on your computer to store the data.
- You can store your data in the shelf just like you'd store items in a dictionary. For example, you can save a list of cat names using a key like 'cats'.
- After saving your data, you must close the shelf to make sure everything is saved properly.

Example:

```
import shelve
shelfFile = shelve.open('mydata')  # Open a shelf file named 'mydata'
cats = ['Zophie', 'Pooka', 'Simon']  # List of cat names
shelfFile['cats'] = cats  # Save the list using the key 'cats'
shelfFile.close()  # Close the shelf
```

Loading Data:

- You can open the shelf again later and retrieve the data using the same key.
- The data you saved earlier will still be there, even after restarting your program or computer!

Example:

```
import shelve
shelfFile = shelve.open('mydata')  # Open the same shelf file
print(shelfFile['cats'])  # Get the list of cats
shelfFile.close()  # Close the shelf
```

Extra Features:

- The shelf acts like a dictionary. You can use .keys() to see all the keys or .values() to see all the saved data.
- If you want the keys or values as a real list, wrap them in list().

Example:

```
import shelve
shelfFile = shelve.open('mydata')  # Open the shelf
print(list(shelfFile.keys()))  # See all the keys
print(list(shelfFile.values()))  # See all the values
```


AALIYA WASEEM, AIML, JNNCE

shelfFile.close()

How It Works:

When you use shelve, it creates some hidden files (.bak, .dat, .dir on Windows or .db on Mac/Linux) to store your data. These files are handled automatically, so you don't need to worry about them.

Why Use a Shelve? It's perfect for saving data that your program might need later, like settings, game progress, or lists of items. It's like saving notes in a file but easier because Python takes care of the tricky parts for you!

Example for understanding purpose:

Imagine you're running a cooking club and want to keep track of your favorite recipes. You don't want to write them down every time you reopen your notebook. Instead, you want to save them in a "digital box" that remembers them for you.

Day 1: Adding Recipes You decide to save some of your favorite recipes:

- Spaghetti Carbonara
- Chocolate Cake
- Caesar Salad

You write these recipes down and store them in a special box (the shelve module). Once the recipes are safely saved, you lock the box and leave for the day.

Day 2: Checking the Recipes The next day, you want to see the recipes you saved. You unlock the box, look inside, and find your list exactly as you left it:

- Spaghetti Carbonara
- Chocolate Cake
- Caesar Salad

The box has remembered everything you stored the previous day!

Day 3: Adding a New Recipe You come across a new recipe for Banana Bread and decide to add it to your collection. You unlock the box, add "Banana Bread" to the list, and lock it again.

Now your collection looks like this:

- Spaghetti Carbonara
- Chocolate Cake
- Caesar Salad
- Banana Bread

Day 4: Showing the Recipes to a Friend Your friend asks about your recipes. You unlock the box, take out the list, and show them:

- Spaghetti Carbonara
- Chocolate Cake
- Caesar Salad

AALIYA WASEEM, AIML, JNNCE

- Banana Bread

Key Points in This Scenario

1. The Box = Shelve Module
   The box acts like a storage container for your data (recipes), saving it securely so you don't lose it when you stop working.
2. Unlocking and Locking = Opening and Closing the Shelf
   To access your saved data, you open the box. When you're done, you close it to ensure everything stays safe.
3. Persistence
   No matter how many days pass or how many times you open and close the box, the data you've stored remains unchanged unless you update it.
4. Flexibility
   You can add new recipes, view the existing ones, or even remove some if you no longer need them.

This scenario shows how the shelve module can be used to save and retrieve data in your programs, just like storing and managing items in a box.

7. SAVING VARIABLES WITH THE PPRINT.PFORMAT() FUNCTION

The pprint.pformat() function in Python helps save data in a clean, easy-to-read format as Python code. Here's how it works, step by step:

1. Pretty Formatting:
   - It turns your data (like a list or dictionary) into a string that looks neat and is valid Python code.
2. Saving as a Python File:
   - You can write this formatted string to a .py file. This creates a Python script that stores your data.
3. Using the Saved Data:
   - Later, you can import the .py file as a module to access the saved data, just like any other Python script.

Example in Simple Terms:

- You have a list of cats with names and descriptions.
- You save this list to a file called myCats.py using pprint.pformat().
- Now, whenever you need that data, you can import myCats and use the list in your programs.

Why Use This?

- Readable & Editable: The saved data is plain text, so you can open and edit it with any text editor.
- Limitations: This method works only for simple data types (e.g., numbers, strings, lists, dictionaries) but not for complex objects like open files.

For most cases, shelve is better for saving data, but this method is handy when you need human-readable and editable files.

AALIYA WASEEM, AIML, JNNCE

Example: Imagine you have a list of fruits and want to save it for later use in your programs.

Step 1: Save the List to a .py File

```
import pprint
# A list of fruits
fruits = ['apple', 'banana', 'cherry']
# Save the list to a Python file
with open('myFruits.py', 'w') as file:
    file.write('fruits = ' + pprint.pformat(fruits) + '\n')
```

This creates a file called myFruits.py, and its content will look like this:

```
fruits = ['apple', 'banana', 'cherry']
```

Step 2: Use the Saved List in Another Program

```
import myFruits
# Access the saved list of fruits
print(myFruits.fruits)  # Output: ['apple', 'banana', 'cherry']
# Use the list
for fruit in myFruits.fruits:
    print(f"I love {fruit}!")
```

Output:

```
['apple', 'banana', 'cherry']
I love apple!
I love banana!
I love cherry!
```

Why is This Useful?

- The list is saved in a file you can import and reuse in other programs.
- You can open myFruits.py in any text editor to view or edit the fruits list.

8. PROJECT: GENERATING RANDOM QUIZ FILES

Imagine you're a teacher and you want to create unique quiz files for your students so they can't copy each other's answers. Here's how you can do it step by step using Python:

Step 1: Set Up the Data

First, you create a dictionary that contains the states and their capitals:

```
capitals = {

    'Alabama': 'Montgomery',
    'Alaska': 'Juneau',
    'Arizona': 'Phoenix',
    # ... more states and capitals ...
}
```

This is the "question bank" for your quiz.

AALIYA WASEEM, AIML, JNNCE

Step 2: Prepare to Create Quizzes

Decide how many quizzes you want (e.g., 35 quizzes). For each quiz, you:

1. Open a file to save the quiz.
2. Open another file to save the answer key.

Step 3: Randomize the Questions

Use Python's random.shuffle() to randomize the order of states:

```
import random
states = list(capitals.keys())  # Get the list of states
random.shuffle(states)         # Shuffle them randomly
```

This ensures every quiz has questions in a different order.

Step 4: Create Questions and Options

For each question:

1. Pick the correct capital for the state.
2. Randomly select three wrong answers from the list of all capitals.
3. Combine the correct answer and wrong answers into a list.
4. Shuffle the list so the correct answer isn't always in the same position.

Example:

```
correct_answer = capitals[state]  # Correct answer
wrong_answers = random.sample([c for c in capitals.values() if c != correct_answer], 3)
answer_options = wrong_answers + [correct_answer]
random.shuffle(answer_options)
```

Step 5: Write the Quiz

Write each question to the quiz file:

```
quizFile.write(f"{question_num + 1}. What is the capital of {state}?\n")

for i, option in enumerate(answer_options):

    quizFile.write(f"    {'ABCD'[i]}. {option}\n")

quizFile.write("\n")
```

Step 6: Write the Answer Key

Save the correct answer to the answer key file:

```
correct_letter = 'ABCD'[answer_options.index(correct_answer)]

answerKeyFile.write(f"{question_num + 1}. {correct_letter}\n")
```


AALIYA WASEEM, AIML, JNNCE

Step 7: Repeat for All Questions and Quizzes

Repeat the above steps for all 50 states in each quiz and for all 35 quizzes.

Step 8: Run the Program

After running the program, you'll have:

- 35 quiz files with randomized questions and options (e.g., capitalsquiz1.txt, capitalsquiz2.txt).
- 35 answer key files with the correct answers (e.g., capitalsquiz_answers1.txt, capitalsquiz_answers2.txt).

Why Is This Useful?

- Unique Quizzes: Prevent cheating by randomizing questions and answers.
- Easy to Create: Python automates repetitive work.
- Reusable: You can update the data (e.g., new states or topics) and generate new quizzes easily.

This way, Python helps you create professional, randomized quizzes quickly

9. PROJECT: MULTI CLIPBOARD

1. What is a Multiclipboard?

A multiclipboard is a program that allows you to save multiple pieces of text with keywords and quickly retrieve them whenever needed. Think of it as a clipboard with superpowers, where you can store several pieces of text at once and retrieve any of them using specific names (keywords).

2. What Does This Program Do?

- Save the current clipboard content using a keyword.
- Retrieve clipboard content by entering the keyword.
- List all the saved keywords.

3. Tools and Concepts Used

- sys.argv: Reads command-line arguments you enter when running the program.
- pyperclip: Allows the program to interact with the clipboard (copy/paste).
- shelve: A Python module for saving data persistently, like a lightweight database.

4. How It Works (Step by Step)

Step 1: Setting Up the Program

We start by:

1. **Importing the necessary modules:**
    - sys for reading commands.
    - pyperclip for clipboard operations.
    - shelve for saving data.
2. **Opening a "shelf" file:**

AALIYA WASEEM, AIML, JNNCE

- A shelf file acts as a small database where we save keywords and their corresponding clipboard content.

Step 2: Saving Clipboard Content

When you type the command:

py mcb.pyw save <keyword>

The program:

1. Checks if the first argument is save and there's a second argument (<keyword>).
2. Copies the current clipboard content.
3. Saves it into the shelf file using <keyword> as the key.

For example:

- You copy "Hello, world!" to the clipboard.
- Run: py mcb.pyw save greeting
- Now "greeting" is saved as the keyword for "Hello, world!"

Step 3: Listing All Keywords

When you type the command:

py mcb.pyw list

The program:

1. Check if the first argument is list.
2. Copies a list of all saved keywords from the shelf file to the clipboard.
3. You can paste the list into a text editor to view the keywords.

For example: If you have saved greeting and farewell, running list will copy:

['greeting', 'farewell']

Step 4: Retrieving Clipboard Content

When you type the command:

py mcb.pyw <keyword>

The program:

1. Checks if the <keyword> exists in the shelf file.
2. Copies the text associated with that keyword to the clipboard.

For example:

- Run: py mcb.pyw greeting
- The text "Hello, world!" is copied back to your clipboard.

AALIYA WASEEM, AIML, JNNCE

5. How It All Fits Together

Here's a quick summary of what happens based on your input:

1. Save: Stores clipboard text under a keyword.
2. List: Copies all keywords to the clipboard.
3. Retrieve: Copies the saved text for a specific keyword to the clipboard.

6. Why is This Useful?

- Automates repetitive tasks, such as filling out forms.
- Saves time when working with frequently used text snippets.
- Makes it easy to organize and manage multiple pieces of clipboard content.

AALIYA WASEEM, AIML, JNNCE