

Function purpose:

Encapsulate duplicate code to facilitate code reuse and maintenance.

Parameters:

The input information that needs to be passed in when calling a function.

Return value:

The result returned to the caller after the function has finished processing.

Advantages of functions:**Code reuse:**

Avoid writing similar code repeatedly and reduce the chance of bugs.

Parameterization:

The same function can handle different inputs, which makes the code more flexible.

Abstraction:

Hide the details of the code and make the program structure clearer.

Maintainability:

If the code needs to be changed, we only need to modify it inside the function. This makes teamwork and future maintenance easier.

Convenient testing:

Functions can be tested individually to check whether each part works correctly.

Data Structures**Definition of data structure:**

A data structure is a way to organize and store data. Unlike a single variable, a data structure can contain multiple data items and have a more complex organization.

Dictionary:

A dictionary is a commonly used data structure in Python. It is similar to a real dictionary. We use a key to access its corresponding value. One key maps to one value, but different keys can map to the same value.

Key:

A unique identifier, usually a string.

Value:

The data associated with a key. It can be any data type, including numbers, strings, lists, or even another dictionary.

Difference Between Dictionaries and Lists

A list accesses elements by index, which is a number. The order of the elements matters.

A dictionary accesses elements by key. This is more intuitive and does not depend on position.

A dictionary is suitable for storing data with strong relationships, such as student IDs and grades, or task names and deadlines.

Web Server and HTML**Web server:**

A server that stores website data and code. It is usually online or in the cloud.

HTML:

HTML is the markup language used to create web pages. The server sends webpage content to the browser as HTML code. The browser then parses and renders it as a visible webpage.

HTML files usually use the .html file extension. The operating system recognizes this extension and usually opens the file in a browser by default.

HTML is made of **tags**. Tags are surrounded by angle brackets.

Start tag:

```
<tagname>
```

End tag:

```
</tagname>
```

Links

Links use the <a> tag. The href attribute specifies the link address.

Format:

```
<a href="http://www.google.com">Go to Google</a>
```

Tables

Tables are defined using the <table> tag.

Rows are defined using the <tr> tag.

Cells are defined using the <td> tag.

The first row is usually used as the table header. The first cell can be left empty to keep the columns aligned.

Example table structure:

```
<table>
<tr>
  <td></td> <td>Monday</td> <td>Tuesday</td>
</tr>
<tr>
  <td>9:00</td> <td></td> <td>Comp 101</td>
</tr>
</table>
```

Forms

Forms are used for user input and data submission. They are an important part of webpage interaction.

All form elements are wrapped inside the <form> tag.

A submit button uses:

```
<input type="submit">
```

The button text can be changed using the value attribute.

Common input types:

text: text box

password: password box, where the input is hidden

month: month picker

color: color picker

Radio Buttons

Radio buttons use:

```
<input type="radio">
```

The name attribute is used to group radio buttons together. This ensures that only one option can be selected from the same group.

Each radio button should have a label tag. The label should use the for attribute to connect to the radio button's id. This improves accessibility.

Labels

A label provides descriptive text for a form control.

When a label is connected to a form control, clicking the label text will focus or select the related input. This improves user experience and supports assistive technology.

Other controls, such as checkboxes and multiple-selection menus, are not explained in detail here. They can be studied further when needed.

Flask

Flask is a lightweight Python web framework. It wraps the core functions of a web server, so developers can quickly build web applications using Python.

Use the following line to import Flask:

```
from flask import Flask
```

A Flask server instance is usually created like this:

```
app = Flask(__name__)
```

Here, `__name__` represents the current module name. It helps Flask locate resources.

pyhtml Library

pyhtml is a Python library used to construct HTML elements in a structured way using Python code. It helps avoid errors and confusion caused by manually joining HTML strings.

Advantages of Code Structure

Clear indentation and hierarchy can reflect the DOM tree structure of HTML.

This makes the code easier to understand and debug.

Returning an HTML String

The object generated by pyhtml needs to be converted into a string using `str()` before it can be returned to the browser.

If you forget to return a string, it may cause an internal server error.

Multiple pages can be defined using multiple `@app.get('/path')` decorators, such as `/morningroutine`, `/favoritefoods`, and `/nothinginteresting`.

Each route corresponds to a view function, and each view function returns the content of the corresponding page.

Questions and Answers

Q1: Why use the POST method?

A1: POST is suitable for handling requests that submit data. It can prevent users from directly accessing the result page and causing errors.

Q2: How do we handle empty input or invalid input?

A2: Use `request.form.get()` with a default value to prevent the program from crashing. If the expression cannot be parsed, return a helpful message.

Q3: Why does the current version not support negative numbers?

A3: The current parsing method is simple. The negative sign may be mistakenly treated as a subtraction operator. This can be improved later.

Q4: Can the input box support decimals?

A4: The current implementation supports floating-point numbers. Division results are especially returned as floating-point numbers.

Q5: Why do we need to design multiple interface options?

A5: This helps avoid design bias, collect feedback, and choose the best user experience.

Q6: How should we handle code conflicts?

A6: Avoid conflicts with Python keywords. Use a module prefix or rename the imported object.

Cookies

Definition and purpose of cookies:

Cookies are small pieces of data sent between the browser and the server during requests and responses.

They are used to save user state information across different requests, such as shopping cart contents or login information.

Cookies are sent during interactions between a specific browser and a specific server.

Users can clear cookies in the browser, but they usually cannot directly modify the cookie data.

Cookies allow user-specific data to be stored without requiring the user to log in.

For example, shopping cart contents can still be saved even after the browser is closed.

In the video example, cookies are used to save the calculation history in a calculator application.

Cookies and Session in Flask

Flask uses the session object to manage cookies. The session object is basically a dictionary.

A user's calculation history can be stored as a list in:

```
session['calculations']
```

A secret key must be set for session to encrypt cookie data and keep it secure.

Cookie data must be encrypted to prevent users from changing it themselves.

The server uses the secret key to encrypt and decrypt cookie data.

If the cookie data is modified, the server will detect that it does not match after decryption. It will reject the data and may require the user to log in again.

The secret key should be a hard-to-guess string. Different applications should use different secret keys.

Limitations of Cookie Data Storage

Users can delete cookies at any time, so the data is not fully persistent.

Cookies can only access data for one user. They cannot combine data across different users. This limits data analysis and recommendation systems.

Cookies are suitable for storing short-term, individual user information.

Advantages of File Storage

File storage can be used not only in web applications, but also in non-web Python programs.

Data can be stored on the user's computer or on the server. In web applications, server-side storage is commonly used.

File storage is useful for long-term data saving and data access across programs.

JSON Format

JSON stands for **JavaScript Object Notation**. It is a widely used standardized data format.

Its structure is clear and can be easily mapped to Python data structures such as lists and dictionaries.

JSON supports complex data structures, such as lists containing dictionaries, or dictionaries containing lists.

It is useful for data serialization and deserialization.

Helper functions can simplify operations, such as:

```
write_json_file(filename, data)
```

```
read_json_file(filename)
```

The function:

```
create_if_not_exist(filename, base_structure)
```

is used to create a file with an empty base structure if the file does not exist. This helps avoid reading errors.

More Questions and Answers

Q: Why can't we use cookies to store all data?

A: Cookies can be deleted by users, and they only store data for one user. They cannot support cross-user data collection or long-term storage.

Q: Why is JSON suitable for Python data storage?

A: JSON matches Python dictionaries and lists very well. It supports complex nested data and is easy to serialize and deserialize.

Q: Why should we close a file after reading or writing?

A: Closing a file releases resources, avoids conflicts when multiple processes write to the file, and helps ensure data integrity.

Q: How can we avoid the "file not found" error when reading or writing files?

A: Use the create_if_not_exist function to create an empty file first, so the file exists before reading.

Q: Where is a file stored by default?

A: By default, it is stored in the current working directory. Before running the program, we need to check which folder the Python terminal is in, or change the directory using the integrated terminal.

Common Errors

Typical errors include request method mismatch, such as **Method Not Allowed**, and incorrect access to form data.

We should use:

```
request.form.get()
```

instead of:

```
request.get()
```

Two Main Ways to Save Data

Cookies:

Used to save user data on the web client side. Suitable for lightweight and short-term data storage.

JSON files:

Used to save structured data on the server side or locally. Suitable for persistent storage.

Prohibited Usage

Only using global variables outside functions to save data is not appropriate.

If the server crashes, the data will be lost. In exam projects, this may also cause marks to be deducted.

Library Function Overview

The library can search Wikipedia entries, get summaries, and find related articles.

Usage process:

Import the library:

```
import wikipedia
```

Use `wikipedia.search()` to search by keyword. It returns a list of possible matching entries.

After choosing an entry, use `wikipedia.summary()` to get the article summary.

Core Concepts and Key Terms

Cookies / Session:

A mechanism for browsers to save user state, helping maintain user identity across requests.

JSON file:

A lightweight data storage format. It is useful for persistently saving user information and application data.

User authentication:

The process of checking whether the username and password match, in order to implement secure access.

Form submission, POST / GET:

User actions submit data to the server through forms. The server responds based on the request method.

Redirection:

After an operation is completed, the user is sent to another page so the latest state can be displayed.

Error message passing:

Using session variables to carry error messages and improve user experience.

Data structure design:

Carefully separating user authentication data and user business data makes the program easier to maintain and extend.

List Methods

`append`: add an element to the end of a list.

`clear`: clear all elements in a list.

`insert(index, element)`: insert an element at a specific index.

`sort()`: sort a list.

Insert an element:

```
cafes.insert(index, coffee_cart)
```

Sort in ascending order:

cafes.sort()

Sort in descending order:

cafes.sort(reverse=True)

Dictionary Methods

dict.keys() returns a view object containing all keys in the dictionary.

dict.values() returns a view object containing all values in the dictionary.

dict.items() returns a view object containing all key-value pairs in the dictionary.

dict.pop(key) removes the item with the specified key and returns its value.

dict.clear() removes all items from the dictionary.

dict.copy() returns a shallow copy of the dictionary.

Python Terms

List:

A Python data structure used to store an ordered collection of elements. Elements can be repeated, and the list can be changed dynamically.

Element:

A single data item in a list.

Index:

The position number of an element in a list. It starts from 0.

Method:

A function that belongs to an object, such as append() and clear().

Loop / Iteration:

The process of visiting each element in a list one by one.

IndexError:

An error raised when trying to access an index that does not exist.

Assertion / assert:

assert is used to check whether a condition is true. If the condition is false, it raises an error. This helps with debugging.

Dictionary:

A collection of key-value pairs, used to store structured data.

List:

An ordered collection of elements.

Feature list:

A collection of properties of a house or restaurant, used for filtering conditions.

None type:

A special type in Python with only one value: None. It represents “no value” or “empty”.

Assert Usage

When the condition in an assert statement is True, there is no output.

When the condition is False, an error is raised and the program stops.

This helps quickly locate bugs in the code.

Function Framework

Use pass as a placeholder to make sure the function structure is complete and does not cause an error.

This makes it easier to fill in the logic step by step later.

Traversing a List of Houses

Use:

for house in houses:

to go through each dictionary in the list.

This allows us to extract data from each item and check conditions.

Extracting Fields into Variables

Extracting fields into variables can make the code easier to understand and debug.

Example:

```
address = house['address']
```

This improves code readability.

Handling None

None is used to represent an empty or invalid value, especially in problems such as finding the maximum value of an empty list.

It helps avoid type comparison errors.

Maximum Value Function Design

Set the initial value to None.

When the first number is found, assign it to the variable.

Then compare later numbers and update the maximum value.

This method works for negative numbers and empty lists.

Web Terms

Web browser:

Client-side software used to access and render webpages, such as Chrome, Firefox, or Safari.

Web server:

A server that stores webpage files and programs. It responds to browser requests and generates webpage content.

HTML:

HyperText Markup Language. It is the basic language of webpages.

Tag:

The basic unit of HTML code. It is surrounded by angle brackets, such as <p>, and is used to build webpage structure.

Element:

A complete structure made of a start tag, content, and an end tag.

Comment:

Text that is not displayed on the webpage. The format is:

```
<!-- comment content -->
```

Attribute:

Extra information inside a tag, written as a key-value pair, such as:

```
src="image.png"
```

Form:

A structure used to collect user input. It contains different input controls.

Input:

A data input element in a form, such as a text box, radio button, or submit button.

Radio button:

Allows the user to choose one option from a group. The name attribute is needed to group options.

Label control:

Adds descriptive text to a form control. It improves accessibility and user experience.

Flask:

A lightweight Python web framework used to quickly build web servers and applications.

Route:

The mapping between a URL path and its corresponding handler function. It decides which page the user visits.

View function:

A function that handles a request and returns HTML content. It is a core part of a Flask application.

pyhtml library:

A library that uses Python functions to construct HTML elements. It improves the readability and structure of HTML code.

Development server:

A server environment used only for development and debugging. It is not suitable for production deployment.

HTML string:

HTML code in string form that is returned to the browser to display webpage content.

Function stub:

An empty function or a function returning a placeholder during early development. It is used to build the program framework.

Flask Basic Introduction

Import Flask, create an app instance, start the development server, and handle simple routes.

Returning HTML Strings

A view function can return a simple string. The browser will render it as basic HTML.

Manually Joining HTML Strings

When writing more complex HTML by manually joining strings, it is easy to make tag errors, and the code becomes difficult to read.

Encapsulating HTML Tag Functions

Define functions to generate HTML tags.

This reduces spelling mistakes and improves code readability.

Introducing the pyhtml Library

Use pyhtml to generate structured HTML.

It can be used to demonstrate the construction of complex elements such as lists.

Multiple Route Example

Create multiple page routes to show the design of a multi-page web application.

Web Application Design Process

Design the interface, create function stubs, design data structures, and then write the code step by step.

Variable Meanings**question_text:**

String. The complete expression entered by the user.

answer:

Integer or floating-point number. The calculation result.

question_parts:

List. The number parts of the expression after splitting.

request.form:

Dictionary. The form data submitted in a Flask request.

Flask route:

Defines the webpage path and the corresponding processing function.

HTTP GET method:

Used to request a page.

HTTP POST method:

Used to submit data.

HTML form:

A user input interface element. It submits data to the server through a form.

request.form:

A Flask object that contains form data submitted in a POST request.

Minimum Viable Product, MVP:

The simplest version of a product that contains only the core functionality. It is useful for fast iteration and testing.

Cookie:

A small piece of data stored between the browser and server. It is automatically sent with requests and is used to save user sessions or preferences.

Session:

A dictionary object in Flask used to manage cookies. It supports storing user state information.

Secret Key:

A key used to encrypt cookie content. It protects data security and prevents tampering.

Encryption mechanism:

A process that converts plain text into encrypted text to prevent users from directly modifying it. The server decrypts and verifies whether the data is valid.

session.modified:

A flag showing that the Flask session data has been modified. It ensures that the update is sent to the client.

JSON Helper Functions**write_json_file:**

Writes Python data into a JSON file.

Input: file name and data.

The data can be a list or a dictionary.

read_json_file:

Reads data from a JSON file and returns a Python data structure.

Input: file name.

If the file does not exist, an error will occur.

create_if_not_exist:

If the specified file does not exist, this function creates it and writes an empty structure into it.

Input: file name and base data structure, such as a dictionary or a list.

This prevents the program from crashing when reading a file that does not exist.

Form Data Example

Get form data:

```
username = request.form.get('nameTB', '')
```

If the username is empty, redirect the user back to the game page.

Generate a random score:

```
score = random.randint(0, 100)
```

Check whether a file exists and create it if needed:

```
create_if_not_exist('highscores.json')
```

This ensures the file exists before reading or writing.

Read a JSON file:

```
high_scores = read_json_file('highscores.json')
```

Find and delete the lowest score:

Go through the dictionary to find the lowest score and the corresponding username.

Use:

```
pop(min_name)
```

to delete it.

Update the high score dictionary:

If the user already exists, update the score.

Otherwise, add a new item.

Write to a JSON file:

```
write_json_file('highscores.json', high_scores)
```

GET and POST in Routes

Different routes, such as the home page, game page, and high score page, can handle GET and POST requests separately.

Use redirection to avoid errors and display the latest result.

Dictionary Sorting

Use:

```
sorted(high_scores.items(), key=lambda x: x[1])
```

to sort dictionary items by value and get an ordered dictionary-like result.

More Web and API Terms

Cookies:

A technology that stores a small amount of data on the browser side. It is used to save user sessions or preferences.

JSON file:

A text file that stores data in JSON format. It is useful for storing and exchanging structured data.

PyPI:

Python Package Index. It is the official repository for third-party Python libraries.

API:

Application Programming Interface. It provides data or functionality through the network.

RapidAPI:

A platform that collects many APIs, making it easier for developers to find and subscribe to APIs.

requests library:

A third-party Python library used to send HTTP requests.

Wikipedia library:

A Python library that makes it easier to call the Wikipedia API and get article information.

User Data Files**users.json:**

Stores all registered users' usernames and passwords.

Format:

```
{username: password}
```

user_data.json:

Stores users' personal data, such as lists of favorite movies.

Format:

```
{username: [movie1, movie2, ...]}
```

Session / Cookies:

Used to store the username of the currently logged-in user. This keeps the login state and supports session management.

Login, Logout, and Reading User Data

Save the username into the session after successful login:

```
session['username'] = username
```

Remove the username when logging out:

```
session.pop('username', None)
```

Read user data example:

1. Function

Q1: What is the purpose of using functions in a program?

A: Functions are used to group repeated code into one reusable block. They help reduce duplicate code and make the program easier to read, test, and maintain. If we need to change the logic, we only need to modify the function once instead of changing the same code in many places.

Q2: Explain the difference between parameters and return values.

A: Parameters are the input values passed into a function when it is called. They allow the function to work with different data. A return value is the result sent back to the caller after the function finishes its task.

Q3: Why do functions make testing easier?

A: Functions make testing easier because each function usually performs one specific task. We can test each function separately with different inputs and check whether the output is correct. This helps us find bugs more easily.

2. Python List / Dictionary

Q4: What is a data structure?

A: A data structure is a way to organize and store data in a program. It can store multiple data items and allows the program to access, update, and manage the data efficiently.

Q5: Compare a Python list and a Python dictionary.

A: A list stores data in an ordered sequence and each element is accessed by its index. The index starts from 0. A dictionary stores data as key-value pairs, and each value is accessed by its key. Lists are useful when the order of data matters. Dictionaries are useful when data has a clear relationship, such as student IDs and grades.

Q6: When should we use a dictionary instead of a list?

A: We should use a dictionary when each piece of data has a clear label or identifier. For example, if we want to store student names and their scores, a dictionary is better because we can use the student name or ID as the key and the score as the value.

Q7: Why is a dictionary useful for storing user data?

A: A dictionary is useful for storing user data because each user can be identified by a unique key, such as a username. The value can store that user's information, such as a password, score, or a list of favourite movies. This makes it easy to find and update data for a specific user.

Q8: Explain what an IndexError is.

A: An IndexError happens when a program tries to access a list index that does not exist. For example, if a list has three elements, the valid indexes are 0, 1, and 2. Accessing index 3 will cause an IndexError.

3. HTML / Web 基础类

Q9: What is HTML used for?

A: HTML is used to create the structure and content of a webpage. A web server sends HTML code to a browser, and the browser renders it as a visible webpage for the user.

Q10: What is the difference between a web browser and a web server?

A: A web browser is client-side software used by the user to view webpages, such as Chrome or Safari. A web server stores website files and program code. It responds to browser requests and sends back webpage content.

Q11: What is the purpose of an HTML form?

A: An HTML form is used to collect user input and submit it to the server. It can contain input elements such as text boxes, password fields, radio buttons, and submit buttons.

Q12: Why should radio buttons in the same group use the same name attribute?

A: Radio buttons in the same group should use the same `name` attribute so that the user can only select one option from the group. Without the same name, the browser will treat them as separate options.

Q13: Why is the label tag useful in a form?

A: The `label` tag gives a description to an input control. If the label is connected to the input using the `for` attribute and the input `id`, users can click the label text to select or focus the input. This improves user experience and accessibility.

4. Flask / Route / pyhtml

Q14: What is Flask?

A: Flask is a lightweight Python web framework. It helps developers build web applications using Python. Flask provides tools for defining routes, handling requests, and returning webpage content to the browser.

Q15: What is a route in Flask?

A: A route is a connection between a URL path and a Python function. When the user visits a specific URL, Flask runs the corresponding view function and returns the result to the browser.

Q16: What is a view function?

A: A view function is a Python function that handles a web request and returns a response. The response is often an HTML string that the browser can display as a webpage.

Q17: Why do we need to convert pyhtml objects to strings?

A: `pyhtml` creates Python objects that represent HTML elements. Before returning them to the browser, we need to convert them into an HTML string using `str()`. If we do not return a proper string, Flask may produce an internal server error.

Q18: Why is pyhtml useful compared with manually writing HTML strings?

A: `pyhtml` is useful because it allows us to build HTML using structured Python code. It makes the code easier to read and reduces mistakes from manually joining strings or missing HTML tags.

5. GET / POST / request.form

Q19: Compare GET and POST methods.

A: GET is used to request data from the server, usually to display a page. POST is used to submit data to the server, such as form input. POST is better for actions that change data, such as logging in, registering, or adding an item.

Q20: Why should form submission usually use POST instead of GET?

A: Form submission usually uses POST because the user is sending data to the server. POST also helps avoid users directly accessing the result URL without submitting the form properly. It is more suitable for actions that process or change data.

Q21: What is `request.form` in Flask?

A: `request.form` is a Flask object that stores the form data submitted by a POST request. It works like a dictionary, so we can access input values using their form field names.

Q22: Why is `request.form.get()` safer than directly using `request.form[...]`?

A: `request.form.get()` is safer because it can return a default value if the field does not exist. This helps prevent the program from crashing. Directly using `request.form[...]` may cause an error if the key is missing.

6. Cookie / Session

Q23: What are cookies used for?

A: Cookies are small pieces of data stored in the browser and sent between the browser and the server. They are used to save user state across different requests, such as login information, shopping cart contents, or calculation history.

Q24: What is a session in Flask?

A: In Flask, ``session`` is used to store user state information. It behaves like a dictionary. For example, we can store the logged-in username in ``session['username']`` so the server knows which user is currently logged in.

Q25: Why does Flask session need a secret key?

A: Flask session needs a secret key to protect cookie data. The secret key is used to sign or encrypt the session data, so the server can detect whether the user has changed the cookie. This helps prevent tampering.

Q26: Why should the secret key be hard to guess?

A: The secret key should be hard to guess because it protects the session data. If someone guesses the secret key, they may be able to create or modify valid cookies, which can cause security problems.

Q27: What are the limitations of using cookies to store data?

A: Cookies have several limitations. Users can delete them at any time, so the data is not fully persistent. Cookies are also stored for one browser and one user, so they are not suitable for combining data across users. They are better for short-term and user-specific data.

7. JSON / File Storage

Q28: What is JSON?

A: JSON stands for JavaScript Object Notation. It is a common data format used to store and exchange structured data. JSON works well with Python dictionaries and lists, so it is useful for saving application data.

Q29: Why is JSON suitable for Python data storage?

A: JSON is suitable for Python data storage because its structure is similar to Python lists and dictionaries. It can store nested data, such as a dictionary containing lists. It is also easy to read from a file and write back to a file.

Q30: Compare cookies and JSON files for data storage.

A: Cookies store small amounts of data in the browser. They are useful for short-term, user-specific data, such as login state or preferences. JSON files store data on the server side or locally. They are better for long-term and structured data storage, such as user accounts, scores, or application data.

Q31: Why should we not only use global variables to store application data?

A: Global variables are not reliable for storing application data because the data will be lost if the server restarts or crashes. They are also not suitable for long-term storage. In a web application, important data should be stored in files or databases instead.

Q32: Why do we need to check whether a JSON file exists before reading it?

A: If the program tries to read a file that does not exist, it will cause an error. Checking or creating the file first can prevent the program from crashing. A helper function such as ``create_if_not_exist()`` can create an empty base file before reading.

Q33: Why should a file be closed after reading or writing?

A: A file should be closed after reading or writing to release system resources. Closing the file also helps ensure that all data has been written correctly and reduces the chance of file conflicts.

8. API / Library / requests

Q34: What is an API?

A: An API, or Application Programming Interface, is a way for one program to access data or functionality from another program or service. In web applications, APIs often allow us to send requests over the internet and receive data as a response.

Q35: Compare an API and a Python list.

A: A Python list is a local data structure used to store multiple values inside a program. An API is not a data structure. It is an interface that allows a program to get data or use services from another system, often through the internet. A list stores data directly, while an API provides a way to access external data or functions.

Q36: Why might a web application use an API?

A: A web application might use an API to get data from an external service instead of storing all data by itself. For example, it can use a weather API to get weather information, or a Wikipedia API to get article summaries. This makes the application more useful and dynamic.

Q37: What is the purpose of the `requests` library?

A: The `requests` library is used in Python to send HTTP requests to websites or APIs. It allows a Python program to get data from external web services.

Q38: What is the Wikipedia library used for?

A: The Wikipedia library is used to search Wikipedia articles and get article summaries. It makes it easier to use Wikipedia data in a Python program without manually handling all API details.

9. Authentication / Login / User Data

Q39: What is user authentication?

A: User authentication is the process of checking whether a username and password are correct. It is used to make sure that only valid users can access certain parts of an application.

Q40: Why do we store the username in session after login?

A: We store the username in session after login so the application can remember which user is currently logged in. This allows the user to move between pages without logging in again each time.

Q41: Why do we remove the username from session when logging out?

A: We remove the username from session when logging out so the application no longer treats the user as logged in. This protects the user's account and ends the current session.

Q42: Why might we use two JSON files, such as `users.json` and `user_data.json`?

A: We might use two JSON files to separate authentication data from user application data. For example, `users.json` can store usernames and passwords, while `user_data.json` can store each user's personal data, such as favourite movies. This makes the program easier to organize and maintain.

10. Design / MVP / Error Handling

Q43: What is an MVP?

A: MVP stands for Minimum Viable Product. It is the simplest version of a product that includes only the core features. It allows developers to test the main idea quickly and improve the product later based on feedback.

Q44: Why is it useful to create function stubs before writing full code?

A: Function stubs are useful because they help build the basic structure of the program first. They allow developers to plan the program and fill in the logic step by step. This also makes large programs easier to manage.

Q45: Why is error handling important in a web application?

A: Error handling is important because users may submit empty, incorrect, or unexpected input. Good error handling prevents the program from crashing and gives users helpful messages. This improves the user experience.

Q46: Why should we redirect after processing a POST request?

A: Redirecting after a POST request helps avoid repeated form submission when the user refreshes the page. It also sends the user to the correct updated page after the data has been processed.

11. Higher-Level Comparison Questions 综合对比题

Q47: Compare cookies, sessions, and JSON files.

A: Cookies are small pieces of data stored in the browser. They are useful for saving short-term user-specific information. Sessions in Flask use cookies to store user state, such as the current logged-in user. JSON files store structured data on the server side or locally. JSON files are better for long-term storage, while cookies and sessions are better for remembering temporary user state.

Q48: Compare Flask and HTML.

A: HTML is a markup language used to structure webpage content. Flask is a Python web framework used to build web applications and servers. HTML controls what the webpage looks like structurally, while Flask handles requests, routes, and server-side logic.

Q49: Compare pyhtml and normal HTML.

A: Normal HTML is written directly using HTML tags. `pyhtml` allows us to create HTML elements using Python code. `pyhtml` can make the page structure clearer inside a Python program and reduce errors caused by manually writing long HTML strings.

Q50: Compare server-side storage and client-side storage.

A: Client-side storage saves data in the user's browser, such as cookies. It is useful for small and temporary user-specific data. Server-side storage saves data on the server, such as JSON files or databases. It is better for long-term data, shared data, and data that should not be controlled by the user.