Bloomify

Quiz Complete

Quiz ended.

Your Results:

Level: Remember

Topic: Module 5: Data Structures - Dictionaries and Sets

Question: Explain the fundamental differences between dictionaries and sets in Python, highlighting their respective uses and the key operations performed on each data structure. Provide examples to illustrate their unique characteristics.

Your Answer:

```
In Python, **dictionaries** and **sets** are both built

A **dictionary** is a collection of **key-value pairs**

```python
person = {'name': 'Alice', 'age': 30}
print(person['name'])  # Output: Alice
```

Key operations for dictionaries include adding/updating

A **set** is an **unordered collection of unique elemen

```python
colors = {'red', 'blue', 'green'}
colors.add('yellow')
print('blue' in colors)  # Output: True
```

Key set operations include `add()`, `remove()`, and matl

Summary: Use dictionaries when you need to map keys
```

Result: Incorrect (69.2%)

Level: Remember

Topic: Module 6: File Handling

Question: Describe a practical scenario where you would need to use different file modes ('r', 'w', 'a', 'x', 'b', 't', '+') in Python for file handling, explaining the specific purpose of each mode chosen and the potential consequences of selecting the wrong mode.

Your Answer:

```
Here's a **practical scenario** to illustrate the use o

---

### 📝 **1. ``'r'`` – Read**

**Use case:** Load existing user configuration/settings

```python
with open('config.txt', 'r') as f:
    settings = f.read()
```

Why: You're only **reading** the file.
Wrong mode consequence: If you accidentally use ``'w

🆕 **2. ``'w'`` – Write (overwrite)**

Use case: Save new settings or overwrite existing l

```python
with open('session_log.txt', 'w') as f:
    f.write('Session started.\n')
```

Why: You want a **clean file** each time.
Wrong mode consequence: Using ``'r'`` would **raise a

➕ **3. ``'a'`` – Append**

Use case: Log each user action without erasing prev

```python
```
```

```
with open('session_log.txt', 'a') as f:
    f.write('User clicked button.\n')
```

Why: Keeps a **growing log** of activity.
Wrong mode consequence: Using `'w'` would **overwri

🚫 **4. `'x'` – Exclusive create**

Use case: Create a file for **first-time setup only

```python
with open('first_run.txt', 'x') as f:
    f.write('App initialized.')
```

Why: You want to ensure the file is **only created
Wrong mode consequence: Using `'w'` would silently

📦 **5. `'b'` – Binary**

Use case: Save or load **user profile pictures** or

```python
with open('profile.jpg', 'rb') as f:
    image_data = f.read()
```

Why: Binary mode is required for **images, audio, e
Wrong mode consequence: Using `'t'` or omitting `'b

📄 **6. `'t'` – Text (default)**

Use case: Read or write plain text files like notes

```python
with open('notes.txt', 'rt') as f:
    print(f.read())
```

Why: `'t'` ensures the file is handled as **text**
Wrong mode consequence: Using `'b'` for text files

```
---

### 🔁 **7. ``'+'`` – Read and write**

**Use case:** Update a config file by reading current s

```python
with open('config.txt', 'r+') as f:
    data = f.read()
    f.seek(0)
    f.write(data.replace('theme=light', 'theme=dark'))
```

Why: You need **both read and write** access in one
Wrong mode consequence: Without ``'+'``, you'd have t

⚠️ Summary of Potential Issues if You Choose the Wr

Wrong Mode Used	Problem Caused
``'r'`` on missing file	Raises `FileNotFoundError
``'w'`` instead of ``'a'``	Overwrites existing data
``'t'`` instead of ``'b'``	Corrupts binary data or m
``'b'`` instead of ``'t'``	Returns bytes instead of
``'x'`` on existing file	Raises `FileExistsError`
Missing ``'+'`` when needed	Can't read/write in the s

Each file mode is designed for a **specific behavior**,
```

Result: Incorrect (69.0%)

Level: Remember

Topic: Module 4: Functions and Modules

Question: Explain the concept of variable scope in Python, differentiating between local and global scope. Provide examples to illustrate how variable scope affects access and modification of variables within functions and modules.

Your Answer:

```
### 🔍 **Variable Scope in Python**

**Variable scope** determines **where in your code a va
```

```
---

### 🔹 **Global Scope**

A variable defined **outside any function or block** is

```python
x = 10  # Global variable

def show():
    print(x)  # Can access global variable

show()
print(x)  # Still accessible here
```

🔹 **Local Scope**

A variable defined **inside a function** is in the **lo

```python
def example():
    y = 5  # Local variable
    print(y)

example()
# print(y)  # ❌ Error: y is not defined outside the fu
```

🔄 **Modifying Global Variables Inside Functions**

To modify a global variable inside a function, use the

```python
count = 0

def increment():
    global count
    count += 1

increment()
print(count)  # Output: 1
```

Without `global`, Python would treat `count` as a new l
```

🧭 Scope Summary:

Scope Type	Where It's Defined	Where It's Acces
Global	Outside all functions	Everywhere in th
Local	Inside a function	Only within that

💡 Why It Matters

* Helps avoid **accidental changes** to global variable
* Prevents **naming conflicts** in larger programs.
* Encourages **modular, maintainable code**.

If you'd like, I can also explain **nonlocal** scope or

Result: Incorrect (66.8%)

Back to Dashboard