Importance of Return

What is `return`?

- The `return` statement **sends a value back** to the part of the program that called the function. This value can be used later in the program.
- If `return` is not used, the function will return `None` by default.

When to use `return`?

1. **When you need a result**: If your function performs a calculation or process that needs to be used later, you should use `return`. For example:

   ```python
   def add(a, b):
       return a + b  # Returns the sum of a and b
   result = add(5, 3)  # Now 'result' stores the value 8
   ```

2. **When a function provides a value**: If the function is expected to output something, like a calculated value or a transformed object, use `return` to provide that value:

   ```python
   def build_profile(first, last, **user_info):
       user_info['first_name'] = first
       user_info['last_name'] = last
       return user_info  # Returns the user profile dictionary
   ```

When you don't need `return`?

- If the function is only performing an action (like printing something or modifying a value in-place), you don't need to use `return`. For example, if you're simply printing a message:

  ```python
  def greet(name):
      print(f"Hello, {name}!")  # No need for 'return' here
  ```

How to remember to use `return`?

1. **Understand the purpose**: Always think about what you want your function to do. If the function needs to **return a result**, make sure to include `return` to send it back.
2. **Plan with comments or pseudocode**: Writing comments can help plan the logic. For example:

   ```python
   # Function should return the full name
   def full_name(first, last):
       return f"{first} {last}"  # Return is clearly planned here
   ```

3. **Identify outputs**: Ask yourself, "What do I need from this function elsewhere in the program?" If you need the result elsewhere, return it.
4. **Use templates**: Adopt a habit of structuring functions with a clear return statement:

   ```python
   def my_function():
       result = do_something()
       return result
   ```

5. **Test with print**: While developing, use print statements to verify the function's behavior. This can help catch situations where you might forget to return a value.
6. **Take advantage of IDE warnings**: Many code editors provide warnings when a function doesn't return anything. Use these as reminders.
7. **Type hints**: Use Python's type hints to remind yourself when a function should return a specific type of value. For example:

   ```python
   def add(a: int, b: int) -> int:
       return a + b  # Type hint reminds that an int should be returned
   ```

Refactor with comments

Here's an example of how commenting and planning can help ensure you remember to return the right value:

```python
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    user_info['first_name'] = first  # Add first name
    user_info['last_name'] = last  # Add last name
    # We want to send back the modified user_info dictionary, so we return it
    return user_info  # Don't forget to return the final result

# Example of using the function
profile = build_profile('Alice', 'Johnson', location='NYC', job='Engineer')
print(profile)  # Outputs the returned profile
```

Summary

- Use `return` to send data back from a function.
- If a function processes or calculates something important, return the result so it can be used later.
- Use strategies like commenting, planning, and testing to avoid forgetting `return`.

By adopting these practices, you'll develop the habit of using `return` properly, making your functions more reliable and effective.