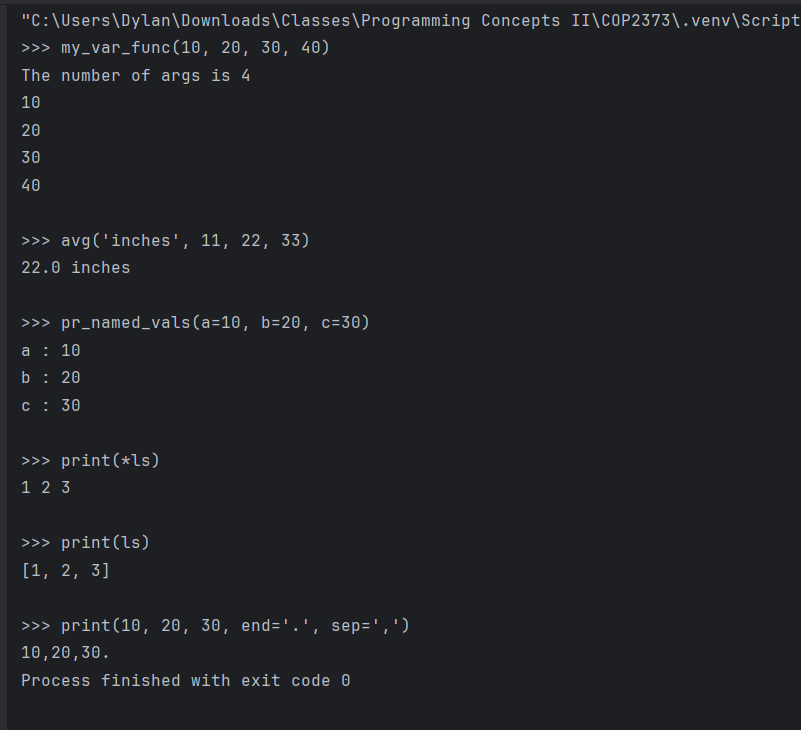
A screen shot of a computer program

AI-generated content may be incorrect.Screenshots:

A screen shot of a computer program

AI-generated content may be incorrect.



**Advantages of Using \*\*kwargs\*\***

**1. Flexible Function Parameters:**

\*\*kwargs allows you to handle a variable number of named arguments, which is particularly useful when the number and names of inputs are uncertain. This feature is especially handy for passing configuration options or building APIs.

**2. Cleaner Function Interfaces:**

Instead of specifying numerous default parameters, you can capture all optional keyword arguments with \*\*kwargs and manage them dynamically within your function. This approach keeps your function signatures neat and concise.

**3. Supports Forwarding Arguments:**

You can easily pass \*\*kwargs to other functions through argument unpacking. This capability simplifies the creation of wrapper functions or decorators without the need to hardcode every possible argument.

**Disadvantages of Using \*\*kwargs**

**1. Harder to Debug:**

If a user passes an incorrect keyword (due to a typo or an unexpected key), Python won’t raise an error unless you manually check for it. This can make bugs more challenging to identify.

**2. No Auto-Completion or Documentation Support:**

IDEs like PyCharm may struggle to predict which keys \*\*kwargs will accept, resulting in a lack of code hints, auto-completion, and built-in documentation. This can make functions more difficult to discover and use correctly.

**3. Reduces Function Transparency:**

When \*\*kwargs is used, other developers may have little understanding of what keys the function expects unless they are documented. This can lead to confusion or improper use of the function. misuse.