

Functions with Variable-Length Arguments (`*args`)

The screenshot displays a learning interface with two main panels. The left panel, titled 'Exercise', contains the following text:

Python's built-in scope

Here you're going to check out Python's built-in scope, which is really just a built-in module called `builtins`. However, to query `builtins`, you'll need to `import builtins` because the name `builtins` is not itself built in...No, I'm serious! (*Learning Python, 5th edition*, Mark Lutz). After executing `import builtins` in the IPython Shell, execute `dir(builtins)` to print a list of all the names in the module `builtins`. Have a look and you'll see a bunch of names that you'll recognize! Which of the following names is NOT in the module `builtins`?

Instructions 50 XP

Possible answers

- ☒ `'sum'`
- ☐ `'range'`
- ☐ `'array'`
- ☐ `'tuple'`

Submit Answer

Take Hint (-15 XP)

The right panel is titled 'IPython Shell' and shows a prompt 'In [1]:' followed by a large dark area, likely representing the output of the shell.

Question:

Flexible arguments enable you to pass a variable number of arguments to a function.

In this exercise, you will practice defining a function that accepts a variable number of string arguments.

The function you will define is ``gibberish()'` which can accept a variable number of string values.

Its return value is a single string composed of all the string arguments concatenated together in the order they were passed to the function call. You will call the function with a single string argument and see how the output changes with another call using more than one string argument.

Recall from the previous video that, within the function definition, ``args'` is a tuple.

****Instructions:****

1. Complete the function header with the function name ``gibberish``. It accepts a single flexible argument ``*args``.
2. Initialize a variable ``hodgepodge`` to an empty string.
3. Concatenate the strings in ``args`` to ``hodgepodge`` using a ``for`` loop.
4. Return the variable ``hodgepodge`` at the end of the function body.
5. Call ``gibberish()`` with the single string ``"luke"``. Assign the result to ``one_word``.
6. Call ``gibberish()`` with five strings: ``"luke"`, `"leia"`, `"han"`, `"obi"`, `"darth"```. Assign the result to ``many_words``.
7. Print ``one_word`` and ``many_words``.

Answer:

```
# Define gibberish
def gibberish(*args):
    """Concatenate strings in *args together."""

    # Initialize an empty string: hodgepodge
    hodgepodge = ""

    # Concatenate the strings in args
    for word in args:
        hodgepodge += word

    # Return hodgepodge
    return hodgepodge

# Call gibberish() with one string: one_word
one_word = gibberish("luke")

# Call gibberish() with five strings: many_words
many_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one_word and many_words
print(one_word)
print(many_words)
```

Explanation:

1. `def gibberish(*args):` - Defines a function ``gibberish`` that accepts a variable number of arguments stored in the tuple ``args``.

2. `hodgepodge = ""` - Initializes an empty string ``hodgepodge`` to store the concatenated result.
3. `for word in args:` - Iterates over each element in ``args``, where each element is a string passed to the function.
4. `hodgepodge += word` - Concatenates each string in ``args`` to ``hodgepodge``.
5. `return hodgepodge` - Returns the final concatenated string stored in ``hodgepodge``.
6. `one_word = gibberish("luke")` - Calls ``gibberish`` with a single argument ``"luke"``. The returned value is ``"luke"``.
7. `many_words = gibberish("luke", "leia", "han", "obi", "darth")` - Calls ``gibberish`` with multiple arguments.
The returned value is ``"lukeleiahanobidarth"``.
8. `print(one_word)` - Prints the result of the single argument call, ``"luke"``.
9. `print(many_words)` - Prints the result of the multiple argument call, ``"lukeleiahanobidarth"``.