RETURN VALUES

Functions can **return multiple values**

```
def concatenator(*words):
    sentence = ''
    for word in words:
        sentence += word + ' '
    last_word = words[-1]
    return sentence.rstrip(), last_word
```

The values to return must be separated by commas

```
concatenator('Hello', 'world!', 'How', 'are', 'you?')
```

```
('Hello world! How are you?', 'you?')
```

This returns a tuple of the specified values

```
sentence, last_word = concatenator('Hello', 'world!', 'How', 'are', 'you?')

print(sentence)
print(last_word)
```

You can unpack the tuple into variables during the function call

```
Hello world! How are you?
you?
```

The variable 'sentence' is assigned to the first element returned in the tuple, so if the order was switched, it would store 'you?'

RETURN VALUES

Functions can return multiple values as **other types of iterables** as well

```
def concatenator(*words):
    sentence = ''
    for word in words:
        sentence += word + ' '
    last_word = words[-1]
    return [sentence.rstrip(), last_word]
```

```
concatenator('Hello', 'world!', 'How', 'are', 'you?')
```

```
['Hello world! How are you?', 'you?']
```

Wrap the comma-separated return values in square brackets to return them inside a list

```
def concatenator(*words):
    sentence = ''
    for word in words:
        sentence += word + ' '
    last_word = words[-1]
    return {sentence.rstrip(): last_word}

concatenator('Hello', 'world!', 'How', 'are', 'you?')
```

```
{'Hello world! How are you?': 'you?'}
```

Or use dictionary notation to create a dictionary (this could be useful as input for another function!)