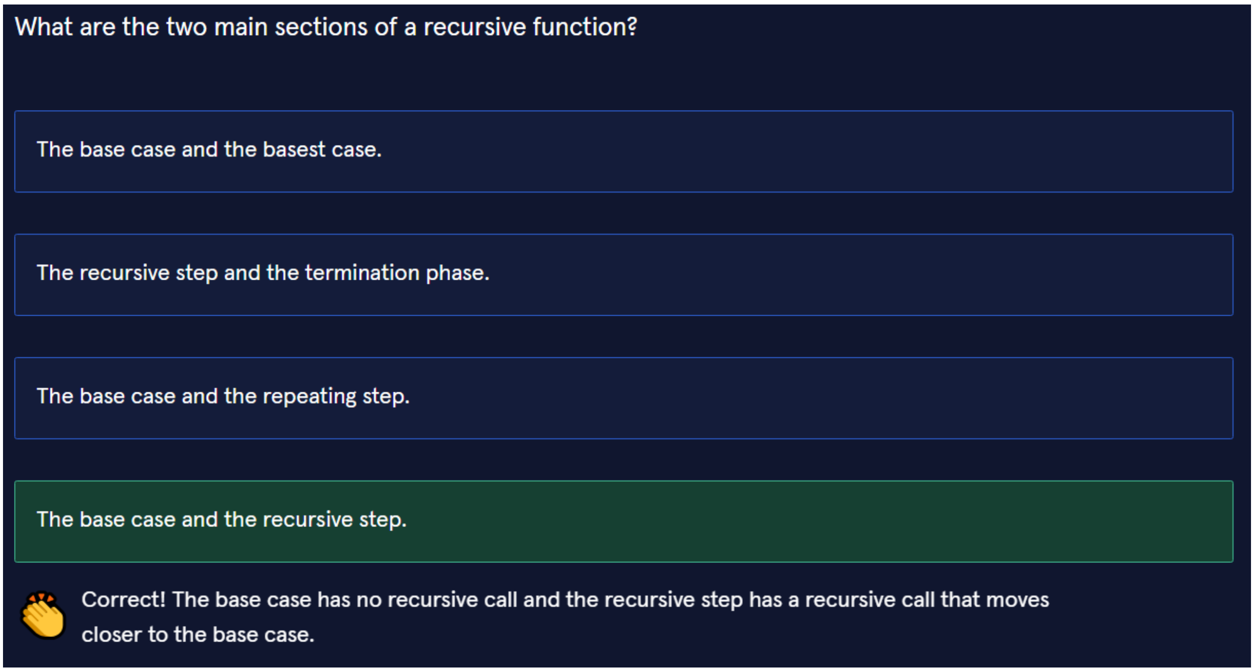
QUIZ



What are the two main sections of a recursive function?

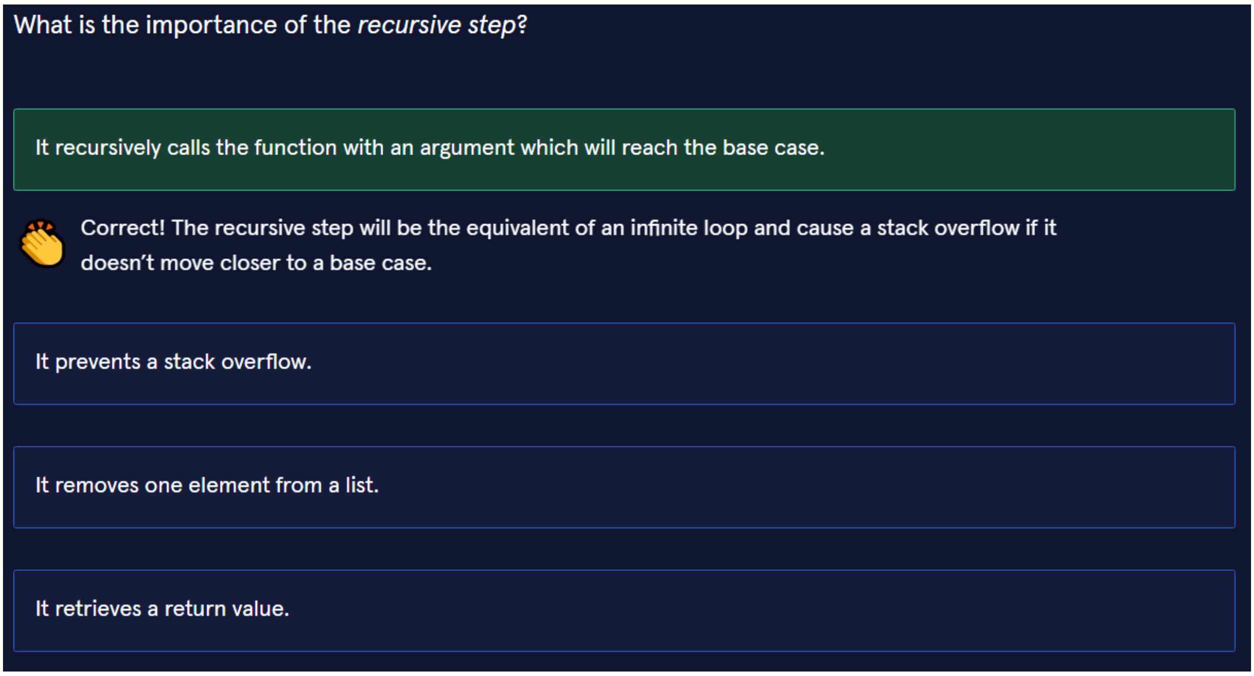
The base case and the basest case.

The recursive step and the termination phase.

The base case and the repeating step.

The base case and the recursive step.

👏 Correct! The base case has no recursive call and the recursive step has a recursive call that moves closer to the base case.



What is the importance of the *recursive step*?

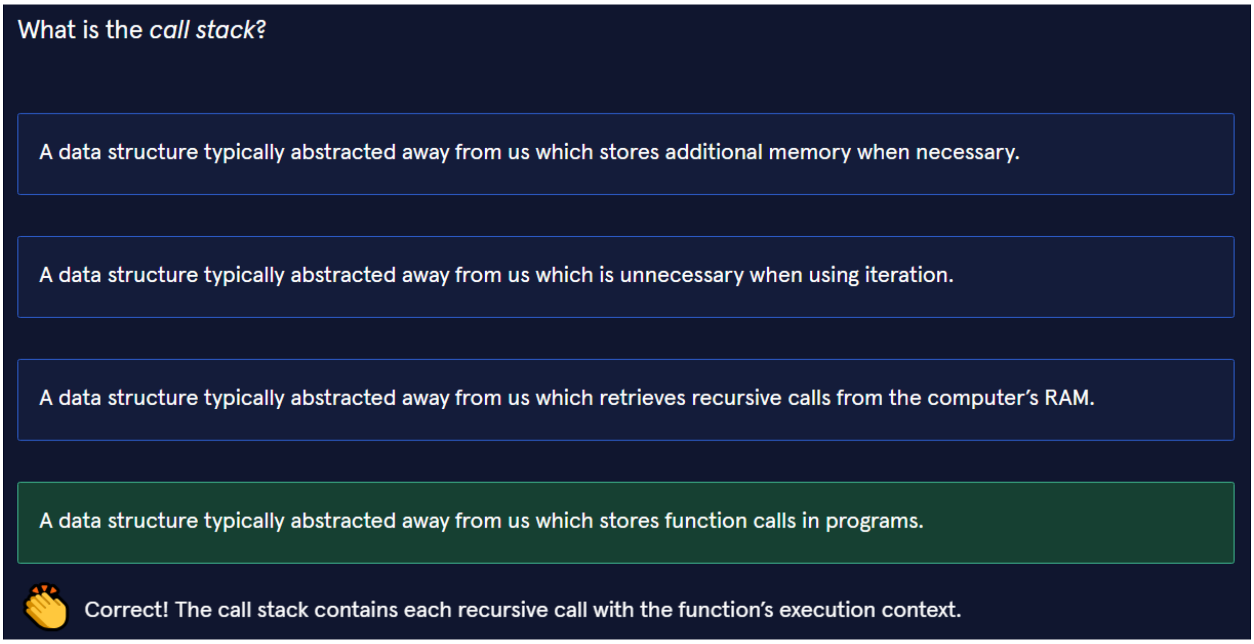
It recursively calls the function with an argument which will reach the base case.

👏 Correct! The recursive step will be the equivalent of an infinite loop and cause a stack overflow if it doesn't move closer to a base case.

It prevents a stack overflow.

It removes one element from a list.

It retrieves a return value.



What is the *call stack*?

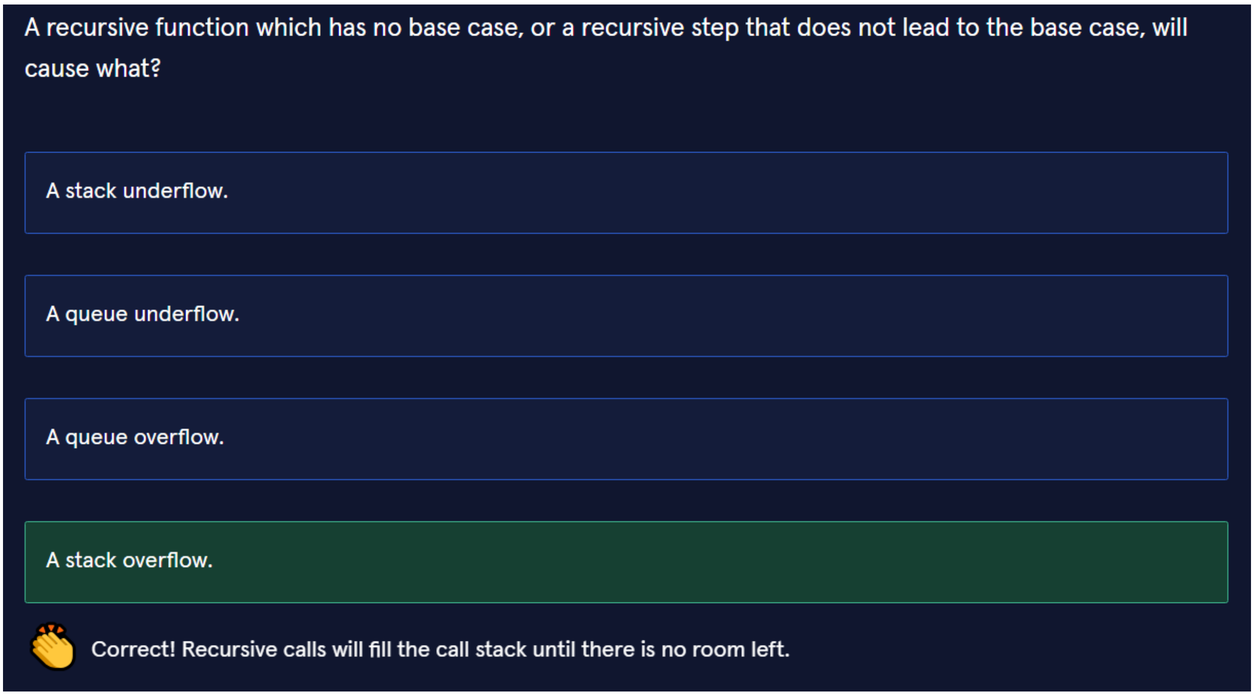
A data structure typically abstracted away from us which stores additional memory when necessary.

A data structure typically abstracted away from us which is unnecessary when using iteration.

A data structure typically abstracted away from us which retrieves recursive calls from the computer’s RAM.

[✓ A data structure typically abstracted away from us which stores function calls in programs.]

👏 Correct! The call stack contains each recursive call with the function’s execution context.



A recursive function which has no base case, or a recursive step that does not lead to the base case, will cause what?

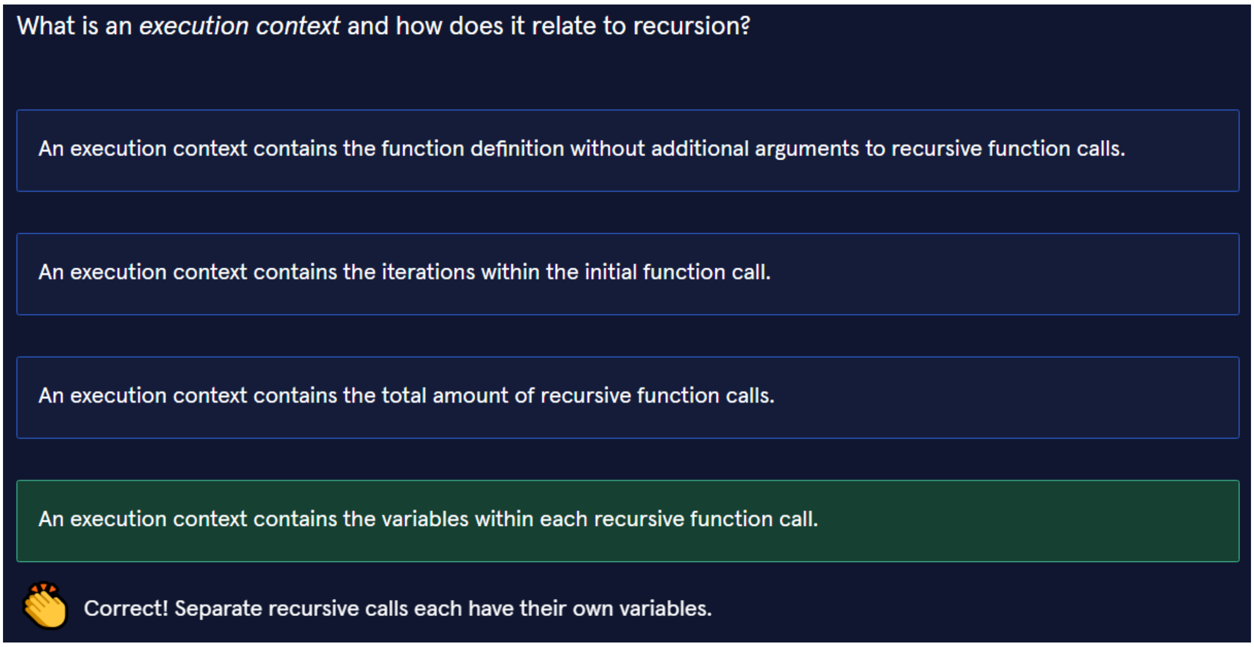
A stack underflow.

A queue underflow.

A queue overflow.

[✓ A stack overflow.]

👏 Correct! Recursive calls will fill the call stack until there is no room left.



What is an *execution context* and how does it relate to recursion?

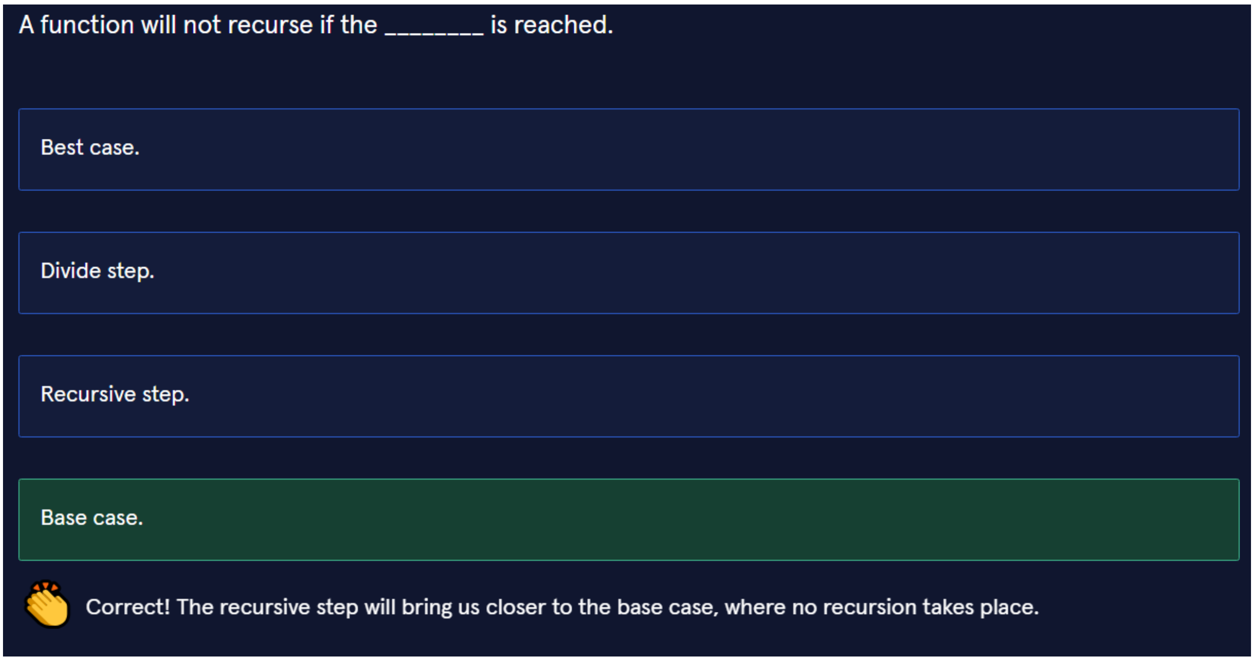
An execution context contains the function definition without additional arguments to recursive function calls.

An execution context contains the iterations within the initial function call.

An execution context contains the total amount of recursive function calls.

[✓ An execution context contains the variables within each recursive function call.]

👏 Correct! Separate recursive calls each have their own variables.



A function will not recurse if the ________ is reached.

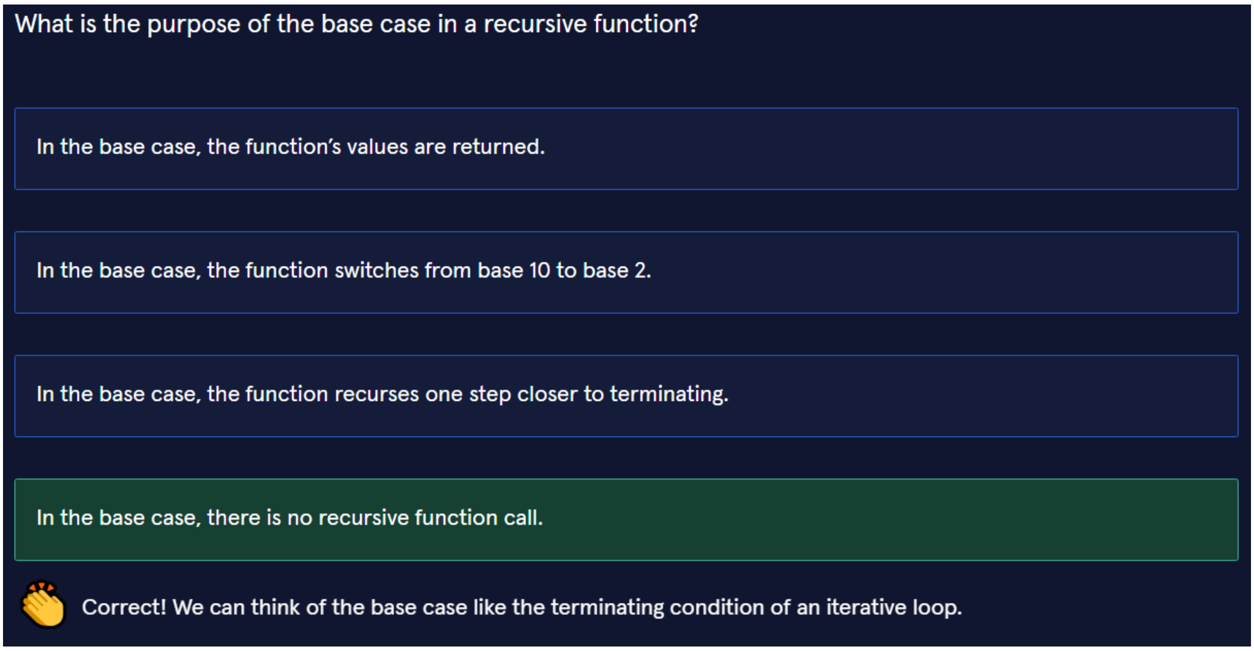
Best case.

Divide step.

Recursive step.

[✓ Base case.]

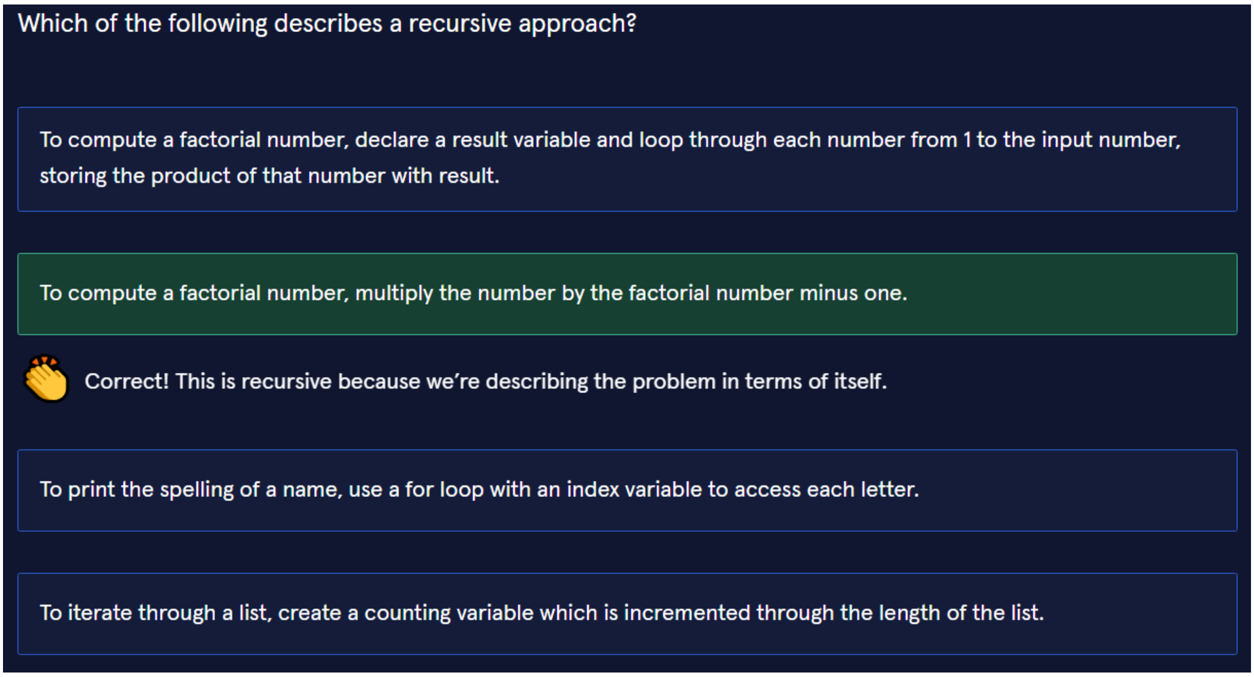
👏 Correct! The recursive step will bring us closer to the base case, where no recursion takes place.



What is the purpose of the base case in a recursive function?

- In the base case, the function’s values are returned.
- In the base case, the function switches from base 10 to base 2.
- In the base case, the function recurses one step closer to terminating.
- [✓ In the base case, there is no recursive function call.]

👏 Correct! We can think of the base case like the terminating condition of an iterative loop.

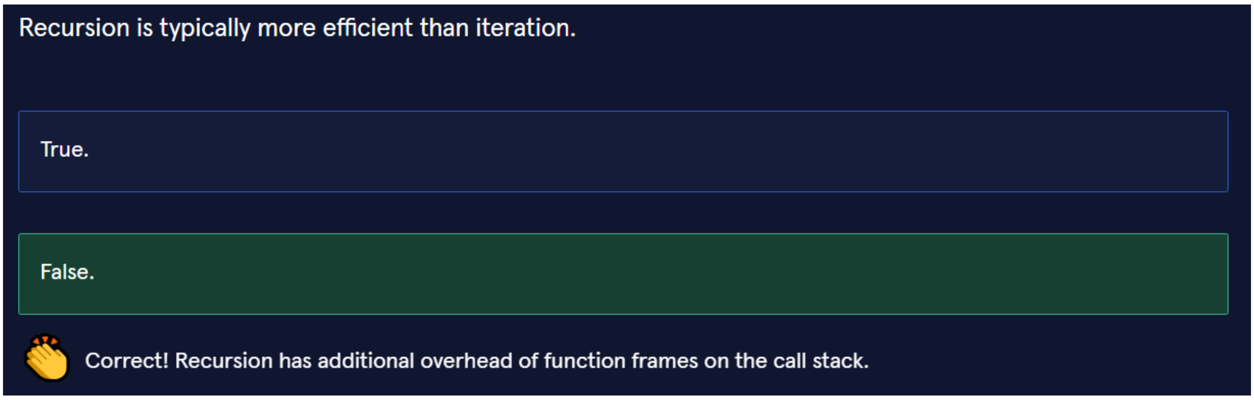


Which of the following describes a recursive approach?

- To compute a factorial number, declare a result variable and loop through each number from 1 to the input number, storing the product of that number with result.
- [✓ To compute a factorial number, multiply the number by the factorial number minus one.]

👏 Correct! This is recursive because we’re describing the problem in terms of itself.

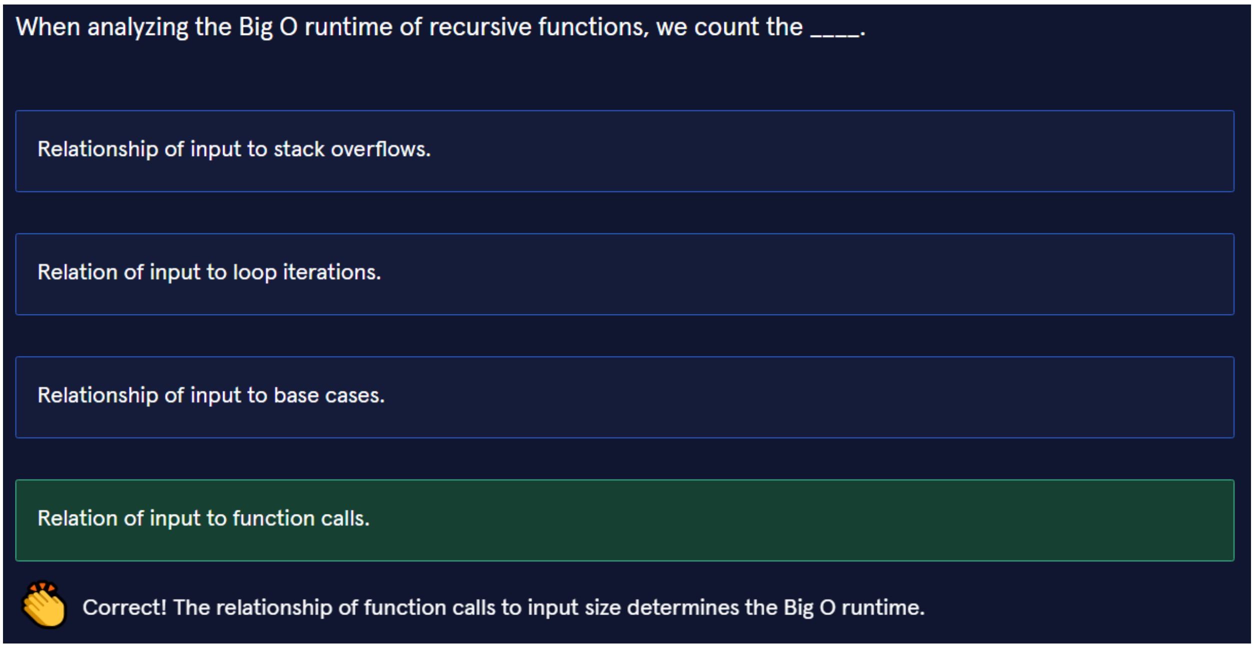
- To print the spelling of a name, use a for loop with an index variable to access each letter.
- To iterate through a list, create a counting variable which is incremented through the length of the list.



Recursion is typically more efficient than iteration.

- True.
- [✓ False.]

👏 Correct! Recursion has additional overhead of function frames on the call stack.



When analyzing the Big O runtime of recursive functions, we count the ____.

Relationship of input to stack overflows.

Relation of input to loop iterations.

Relationship of input to base cases.

[✓ Relation of input to function calls.]

👏 Correct! The relationship of function calls to input size determines the Big O runtime.