Visual description, image, or sketch:

Recursion

When a function calls another function

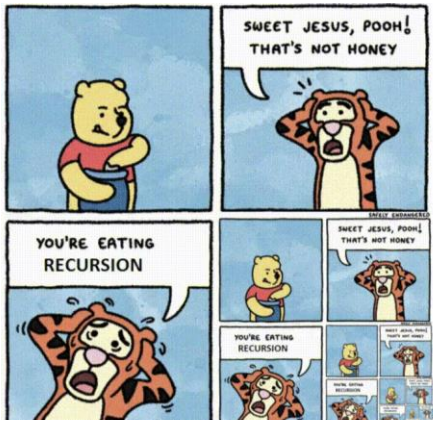
2 headlines appear

Each line fades in aligned to VO

Image appears aligned to VO

Objectives

1. evaluate the suitability of recursion
2. construct the base case of a recursive function
3. diagram the behavior of the call stack



Voiceover (VO) and/or other audio:

Welcome to this short video on Recursion. Recursion is an approach we can take to solving a problem.

You will learn to evaluate whether recursion is a suitable approach for a given problem. You will also learn to construct a base case for a recursive function as well as see a diagram for the behavior of a call stack

First, let's take a look at what recursion means.

Recursion is when a function calls another function.

Here's a cartoon that depicts one understanding of recursion.

Interaction, branching, etc.:

Notes:

Choose a picture that describes what recursion is. It will be especially helpful for visual learners.

Visual description, image, or sketch:

Iteration vs Recursion

Iteration is preferred when:

- keep repeating until a task is finished (a loop)

- longer and bigger in size

Recursion is preferred when:

- solving a large and complex problem and you want to break it into chunks

- shorter and result in cleaner code

- easier to do traversal within data structures such as trees and graphs

Headline appears

Each line fades in aligned to VO

Voiceover (VO) and/or other audio:

Depending on the situation, you will decide which is more suitable to use – iteration or a recursion.

Iteration is preferred when you want to keep repeating until a task is finished, such as when a loop counter reaches its limit. Iterative solutions can be much longer and bigger in size.

Recursion is preferred when you are solving a very large and complex problem and you want to break it into smaller chunks to help you solve the larger problem.

Recursive solutions are often shorter and result cleaner code. It is especially good for working on problems that have many possible branches. It is also easier to do traversal within data structures such as trees and graphs.

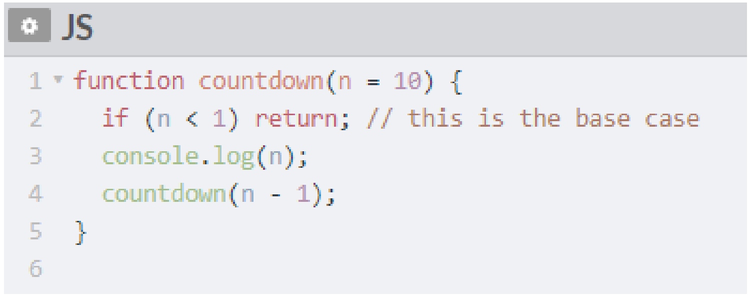
Interaction, branching, etc.:

Notes:

Visual description, image, or sketch:

Recursive Function

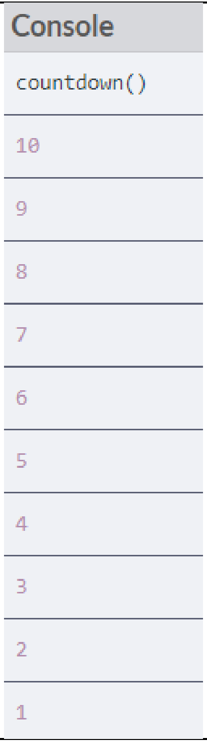
In recursion, we invoke the same function with a different input until we reach the base case.



JS

```
function countdown(n = 10) {
  if (n < 1) return; // this is the base case
  console.log(n);
  countdown(n - 1);
}

```



Console

```
countdown()
10
9
8
7
6
5
4
3
2
1
```

Headline appears

Fade-in each detail aligned to VO

Each image appears aligned to VO

Voiceover (VO) and/or other audio:

In recursion, we invoke the same function with a different input until we reach the base case.

In this example, we have a simple recursive function that does a countdown from 10 to 1.

In the function, we will check if a condition n < 1 is true. If it is, we end the function and return.

If it is not true, we print n and call the countdown function again, each time subtracting 1.

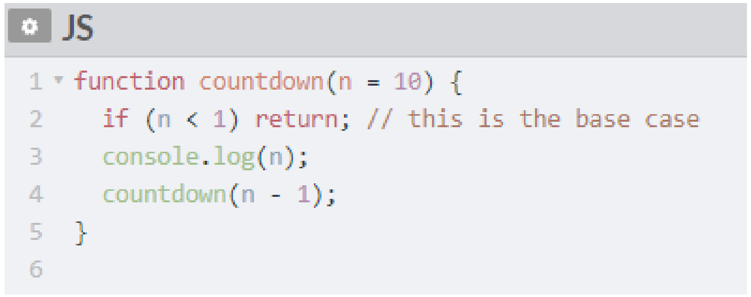
In the console, we see a countdown from 10 to 1.

Interaction, branching, etc.:

Notes:

Visual description, image, or sketch:

Base Case



```
JS
1 function countdown(n = 10) {
2   if (n < 1) return; // this is the base case
3   console.log(n);
4   countdown(n - 1);
5 }
6
```

- Every Recursive Function needs a Base Case or it can result in an error
- The base case lets the function know when to stop running
- Once the condition is met, the execution terminates

Headline appears

Image appears aligned to VO

Fade-in each detail aligned to VO

Voiceover (VO) and/or other audio:

In this example, let’s look at line 2. This is the base case.

This is important to have to let the function know when to stop running once the base case condition has been met. In our case, the condition is if the number is less than 1 (n < 1). Once the condition is met, the execution terminates.

- Every Recursive Function needs a Base Case or it can result in an error
- The base case lets the function know when to stop running
- Once the condition is met, the execution terminates

Interaction, branching, etc.:

Notes:

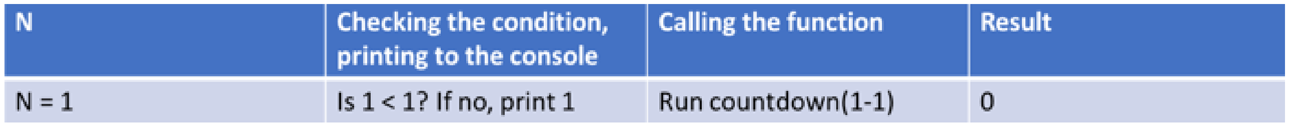
Visual description, image, or sketch:

What is a Call Stack

- A list of function invocations that are waiting but not done yet

"Last In, First Out"

The most recently invoked function is on the top of the stack
The bottom of the stack is the first function invoked
The stack is processed from top to bottom



N	Checking the condition, printing to the console	Calling the function	Result
N = 1	Is 1 < 1? If no, print 1	Run countdown(1-1)	0

Headline appears

Image appears aligned to VO

Fade-in each detail aligned to VO

Voiceover (VO) and/or other audio:

A call stack is a list of function invocations that are waiting but not done yet. It is a data structure created when a function is executed.

Any function you run has a call stack. It defines the variable, checks the variable, and prints the variables, and shows the output.

The call stack operates on a "Last In, First Out" meaning that the most recently invoked function will be on the top of the stack. The bottom of the stack is the first function invoked. The stack is processed from top to bottom.

Here is a row from a diagram of a call stack.

This is what is happening behind the scenes.

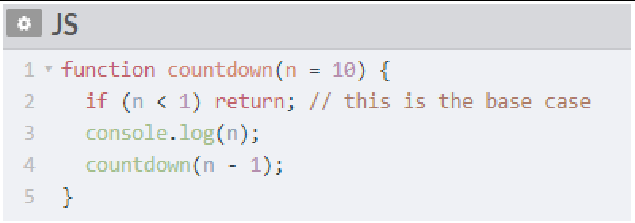
With recursion, we wait for the result or return values coming from other functions and these contexts are higher up in the stack

Interaction, branching, etc.:

Notes:

Visual description, image, or sketch:

Diagram of a Call Stack



```
function countdown(n = 10) {
  if (n < 1) return; // this is the base case
  console.log(n);
  countdown(n - 1);
}
```

Headline appears

Images appear aligned to VO

N	Checking the condition, printing to the console	Calling the function	Result
N = 1	Is 1 < 1? If no, print 1	Run countdown(1-1)	0
N = 2	Is 2 < 1? If no, print 2	Run countdown(2-1)	1
N = 3	Is 3 < 1? If no, print 3	Run countdown(3-1)	2
N = 4	Is 4 < 1? If no, print 4	Run countdown(4-1)	3
N = 5	Is 5 < 1? If no, print 5	Run countdown(5-1)	4
N = 6	Is 6 < 1? If no, print 6	Run countdown(6-1)	5
N = 7	Is 7 < 1? If no, print 7	Run countdown(7-1)	6
N = 8	Is 8 < 1? If no, print 8	Run countdown(8-1)	7
N = 9	Is 9 < 1? If no, print 9	Run countdown(9-1)	8
N = 10	Is 10 < 1? If no, print 10	Run countdown(10-1)	9

Voiceover (VO) and/or other audio:

This is an example of what a call stack might look like.

Let's go back to the example we looked at before, but now from the lens of what is happening behind the scenes in the call stack. The line at the bottom is the very first countdown(n=10) function that ran. The next countdown(n=9) function ran and is placed on top. And so forth.

Each time the function runs, there is a new n or number. It will result in the countdown function running again and the number being subtracted by 1. The output will be new each time the function runs until it meets the condition of n being less than 1.

Knowing how a call stack works is also helpful for debugging.

Interaction, branching, etc.:

Notes:

Visual description, image, or sketch:

Another example of Recursion

```
function building_a_house() {
    console.log(“step 1 - foundation is laid");
    framing();
    console.log("step 5 - painting the house");
}

framing(){
    console.log("step 2 - frame the house");
    wood();
    console.log("step 4 - electrical ");
}

wood(){
    console.log("step 3 - roofing")
}
```

Result: building_a_house()
step 1 - foundation is laid, step 2 - frame the house, step 3 - roofing, step 4 - electrical, step 5 - painting the house

- Fly-in animation for the headline
- Bring in the function with VO

Voiceover (VO) and/or other audio:

Here is another example of recursion.

What happens when this function runs:

1) building_a_house() runs
2) prints “step 1 - foundation is laid”, stops execution
3) calls framing() to run
4) prints “step 2 - frame the house”, stops execution
5) call wood() to run
6) prints “step 3 – roofing”, finishes running
7) jumps back to framing()
8) prints “step 4 - electrical“, finishes running
9) jumps back to building_a_house()
10)prints “step 5 - painting the house”, finishes execution

Interaction, branching, etc.:

Notes:

Visual description, image, or sketch:

Recap

- determine if recursion would a suitable approach to solving a problem
- create the base case for a recursive function
- know the behavior of call stack

Voiceover (VO) and/or other audio:

Now you know how to determine if recursion would be suitable approach to solving a problem

You also have knowledge of how to create a base case for a recursive function

You also saw what the behavior of a call stack might look like

Interaction, branching, etc.:

Notes: