Resolución de problemas

Curso de Fundamentos

Introducción

Antes de comenzar a profundizar en algunos aspectos bastante ingeniosos del código JavaScript, debemos comenzar a hablar sobre *la resolución de problemas* : la habilidad más importante que necesita un desarrollador.

La resolución de problemas es la actividad principal de los desarrolladores de software. Los lenguajes de programación y las herramientas que utilizan son secundarios a esta habilidad fundamental.

En su libro *"Piense como un programador"* , V. Anton Spraul define la resolución de problemas en programación como:

> *"La resolución de problemas consiste en escribir un programa original que realice un conjunto particular de tareas y cumpla con todas las restricciones establecidas."*

El conjunto de tareas puede abarcar desde la resolución de pequeños ejercicios de codificación hasta la creación de un sitio de redes sociales como Facebook o un motor de búsqueda como Google. Cada problema tiene su propio conjunto de limitaciones; por ejemplo, el alto rendimiento y la escalabilidad pueden no ser tan importantes en un ejercicio de codificación, pero serán vitales en aplicaciones como Google, que necesitan atender miles de millones de consultas de búsqueda cada día.

A menudo, los nuevos programadores descubren que la resolución de problemas es la habilidad más difícil de desarrollar. No es raro que los programadores principiantes aprendan rápidamente la sintaxis y los conceptos de programación, pero cuando intentan codificar algo por su cuenta, se quedan mirando fijamente su editor de texto sin saber por dónde empezar.

La mejor manera de mejorar tu capacidad para resolver problemas es adquirir experiencia creando muchos programas. Cuanto más practiques, mejor preparado estarás para resolver problemas del mundo real.

En esta lección repasaremos algunas técnicas que pueden usarse para ayudar con el proceso de resolución de problemas.

Resumen de la lección

Esta sección contiene una descripción general de los temas que aprenderá en esta lección.

- Explique los tres pasos del proceso de resolución de problemas.
- Explicar qué es el pseudocódigo y poder utilizarlo para resolver problemas.
- Ser capaz de dividir un problema en subproblemas.

Entender el problema

El primer paso para resolver un problema es comprender exactamente cuál es el problema. Si no comprende el problema, no sabrá cuándo lo ha resuelto con éxito y puede perder mucho tiempo con una solución incorrecta .

Para obtener claridad y comprensión del problema, escríbalo en un papel, reformúlelo en un lenguaje sencillo hasta que tenga sentido para usted y dibuje diagramas si eso le resulta de ayuda. Cuando puede explicar el problema a otra persona en un lenguaje sencillo, lo entiende.

Plan

Ahora que ya sabes qué es lo que quieres resolver, no te lances a codificar todavía. Es hora de planificar cómo lo vas a resolver primero. Algunas de las preguntas que debes responder en esta etapa del proceso:

- ¿Su programa tiene una interfaz de usuario? ¿Qué aspecto tendrá? ¿Qué funcionalidad tendrá la interfaz? Dibuje esto en un papel.
- ¿Qué entradas tendrá tu programa? ¿El usuario ingresará datos o recibirás información de algún otro lugar?
- ¿Cuál es el resultado deseado?
- Teniendo en cuenta sus datos, ¿cuáles son los pasos necesarios para obtener el resultado deseado?

The last question is where you will write out an algorithm to solve the problem. You can think of an algorithm as a recipe for solving a particular problem. It defines the steps that need to be taken by the computer to solve a problem in pseudocode.

Pseudocode

Pseudocode is writing out the logic for your program in natural language instead of code. It helps you slow down and think through the steps your program will have to go through to solve the problem.

Here's an example of what the pseudocode for a program that prints all numbers up to an inputted number might look like:

```
When the user inputs a number
Initialize a counter variable and set its value to zero
While counter is smaller than user inputted number increment t
Print the value of the counter variable
```

This is a basic program to demonstrate how pseudocode looks. There will be more examples of pseudocode included in the assignments.

Divide and conquer

From your planning, you should have identified some subproblems of the big problem you're solving. Each of the steps in the algorithm we wrote out in the last section are subproblems. Pick the smallest or simplest one and start there with coding.

It's important to remember that you might not know all the steps that you might need up front, so your algorithm may be incomplete -— this is fine. Getting started with and

solving one of the subproblems you have identified in the planning stage often reveals the next subproblem you can work on. Or, if you already know the next subproblem, it's often simpler with the first subproblem solved.

Many beginners try to solve the big problem in one go. **Don't do this**. If the problem is sufficiently complex, you'll get yourself tied in knots and make life a lot harder for yourself. Decomposing problems into smaller and easier to solve subproblems is a much better approach. Decomposition is the main way to deal with complexity, making problems easier and more approachable to solve and understand.

In short, break the big problem down and solve each of the smaller problems until you've solved the big problem.

Solving Fizz Buzz

To demonstrate this workflow in action, let's solve Fizz Buzz

Understanding the problem

> *"Write a program that takes a user's input and prints the numbers from one to the number the user entered. However, for multiples of three print* `Fizz` *instead of the number and for the multiples of five print* `Buzz` *. For numbers which are multiples of both three and five print* `FizzBuzz` *."*

This is the big picture problem we will be solving. But we can always make it clearer by rewording it.

Write a program that allows the user to enter a number, print each number between one and the number the user entered, but for numbers that divide by 3 without a remainder print `Fizz` instead. For numbers that divide by 5 without a remainder print `Buzz` and finally for numbers that divide by both 3 and 5 without a remainder print `FizzBuzz` .

Planning

Does your program have an interface? What will it look like? Our FizzBuzz solution will be a browser console program, so we don't need an interface. The only user interaction will be allowing users to enter a number.

What inputs will your program have? Will the user enter data or will you get input from somewhere else? The user will enter a number from a prompt (popup box).

What's the desired output? The desired output is a list of numbers from 1 to the number the user entered. But each number that is divisible by 3 will output `Fizz`, each number that is divisible by 5 will output `Buzz` and each number that is divisible by both 3 and 5 will output `FizzBuzz`.

Writing the pseudocode

What are the steps necessary to return the desired output? Here is an algorithm in pseudocode for this problem:

```
When a user inputs a number
Loop from 1 to the entered number
If the current number is divisible by 3 then print "Fizz"
If the current number is divisible by 5 then print "Buzz"
If the current number is divisible by 3 and 5 then print "Fizz
Otherwise print the current number
```

Dividing and conquering

As we can see from the algorithm we developed, the first subproblem we can solve is getting input from the user. So let's start there and verify it works by printing the entered number.

With JavaScript, we'll use the "prompt" method.

```
let answer = parseInt(prompt("Please enter the number you woul
```

The above code should create a little popup box that asks the user for a number. The input we get back will be stored in our variable `answer`.

> We wrapped the prompt call in a `parseInt` function so that a number is returned from the user's input.

With that done, let's move on to the next subproblem: "Loop from 1 to the entered number". There are many ways to do this in JavaScript. One of the common ways - that you actually see in many other languages like Java, C++, and Ruby - is with the for loop:

```
let answer = parseInt(prompt("Please enter the number you woul

for (let i = 1; i <= answer; i++) {
  console.log(i);
}
```

If you haven't seen this before and it looks strange, it's actually straightforward. We declare a variable `i` and assign it 1: the initial value of the variable `i` in our loop. The second clause, `i <= answer` is our condition. We want to loop until `i` is greater than `answer`. The third clause, `i++`, tells our loop to increment `i` by 1 every iteration. As a result, if the user inputs 10, this loop would print numbers 1 - 10 to the console.

> Most of the time, programmers find themselves looping from 0. Due to the needs of our program, we're starting from 1

With that working, let's move on to the next problem: If the current number is divisible by 3, then print `Fizz`.

```
let answer = parseInt(prompt("Please enter the number you woul

for (let i = 1; i <= answer; i++) {
  if (i % 3 === 0) {
    console.log("Fizz");
  } else {
    console.log(i);
  }
}
```

We are using the modulus operator (`%`) here to divide the current number by three. If you recall from a previous lesson, the modulus operator returns the remainder of a

division. So if a remainder of 0 is returned from the division, it means the `current` number is divisible by 3.

After this change the program will now output this when you run it and the user inputs 10:

```
1
2
Fizz
4
5
Fizz
7
8
Fizz
10
```

El programa está empezando a tomar forma. Los últimos subproblemas deberían ser fáciles de resolver, ya que la estructura básica ya está establecida y son solo variaciones diferentes de la condición que ya tenemos establecida. Abordemos el siguiente: si el número actual es divisible por 5, entonces imprima `Buzz` .

```
let answer = parseInt(prompt("Please enter the number you woul

for (let i = 1; i <= answer; i++) {
  if (i % 3 === 0) {
    console.log("Fizz");
  } else if (i % 5 === 0) {
    console.log("Buzz");
  } else {
    console.log(i);
  }
}
```

Cuando ejecute el programa ahora, debería ver este resultado si el usuario ingresa 10:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
```

Tenemos un subproblema más que resolver para completar el programa: Si el número actual es divisible por 3 y 5 entonces imprimir `FizzBuzz`.

```
let answer = parseInt(prompt("Please enter the number you woul

for (let i = 1; i <= answer; i++) {
  if (i % 3 === 0 && i % 5 === 0) {
    console.log("FizzBuzz");
  } else if (i % 3 === 0) {
    console.log("Fizz");
  } else if (i % 5 === 0) {
    console.log("Buzz");
  } else {
    console.log(i);
  }
}
```

Tuvimos que mover un poco las condiciones para que funcione. La primera condición ahora verifica si `i` es divisible por 3 y 5 en lugar de verificar si `i` es solo divisible por 3. Tuvimos que hacer esto porque si lo mantuviéramos como estaba, ejecutaría la primera condición `if (i % 3 === 0)`, de modo que si `i` fuera divisible por 3, se imprimiría `Fizz` y luego pasaría al siguiente número en la iteración, incluso si `i` también fuera divisible por 5.

Con la condición `if (i % 3 === 0 && i % 5 === 0)` en primer lugar, verificamos que `i` sea divisible por 3 y por 5 antes de pasar a verificar si es divisible por 3 o por 5 individualmente en las `else if` condiciones.

¡El programa ya está completo! Si lo ejecutas ahora, deberías obtener este resultado cuando el usuario ingrese 20:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
```

Asignación

1. Lea Cómo pensar como un programador: lecciones de resolución de problemas de Richard Reis.
2. Mira Cómo empezar a pensar como programador de Coding Tech. Tiene una duración de una hora, pero está repleto de información y definitivamente vale la pena verlo.

3. Lea este artículo Pseudocódigo: qué es y cómo escribirlo de Built In.

Comprobación de conocimientos

Las siguientes preguntas son una oportunidad para reflexionar sobre temas clave de esta lección. Si no puede responder una pregunta, haga clic en ella para revisar el material, pero tenga en cuenta que no se espera que memorice o domine este conocimiento.

- ¿Cuáles son las tres etapas del proceso de resolución de problemas?
- ¿Por qué es importante entender claramente el problema primero?
- ¿Qué puedes hacer para ayudar a obtener una comprensión más clara del problema?
- ¿Cuáles son algunas de las cosas que debes hacer en la etapa de planificación del proceso de resolución de problemas?
- ¿Qué es un algoritmo?
- ¿Qué es el pseudocódigo?
- ¿Cuáles son las ventajas de dividir un problema en partes y resolver los problemas más pequeños?

Recursos adicionales

Esta sección contiene enlaces útiles a contenido relacionado. No es obligatoria, por lo que se la puede considerar complementaria.

- Lee el primer capítulo de Think Like a Programmer: An Introduction to Creative Problem Solving (*no es gratuito*). Los ejemplos de este libro están en C++, pero entenderás todo, ya que la idea principal del libro es enseñar a los programadores a resolver mejor los problemas. Es un libro increíble y vale cada centavo. Te convertirá en un mejor programador.
- Mira este vídeo sobre técnicas de programación repetitiva .
- Vea a Jonathan Blow resolver problemas difíciles , donde brinda sabios consejos sobre cómo abordar la resolución de problemas en proyectos de software.

Mejorar en GitHub Informar un problema Ver registro de cambios de la lección

Ver curso

Marcar como completado

Próxima lección

¡Apóyanos!

El Proyecto Odin está financiado por la comunidad. ¡Únase a nosotros para ayudar a estudiantes de todo el mundo apoyando el Proyecto Odin!

Más información Dona ahora

THE ODIN PROJECT

Educación en codificación de alta calidad mantenida por una comunidad de código abierto.

Sobre nosotros

Acerca de

Equipo

Guías

Guías de la comunidad

Guías de instalación

Blog

Casos de éxito

Apoyo

Preguntas frecuentes

Contribuir

Contáctenos

Legal

Términos

Privacidad