4. Operadores e Comandos Básicos

Neste capítulo você aprenderá a:

- Utilizar operadores unários e binários no *javascript*.
- Manipular operadores especiais no *javascript*.
- Utilizar comandos de entrada e saída de dados no *javascript*.

4.1 Operadores

Você já deve ter entendido que programação lida muito com matemática, certo? Se não, então saiba que sim! Operadores são exemplos claros disso. Praticamente todas as operações que vemos nas disciplinas de matemática existem em todas as linguagens de programação, como: soma (`+`), subtração (`-`), divisão (`/`), multiplicação ($\times$). E variações dessas operações. Veremos nesta seção dois tipos de operadores:

- Operadores binários: necessitam de dois dados, ou variáveis, para avaliar e retornar um valor.
- Operadores unários: necessitam de apenas dado, ou variável, para avaliar e retornar um valor;

Talvez ainda são esteja claro o que é o operador unário, embora o tenha utilizado várias vezes. Você alguma vez já fez uma operação como está: `-(-5)`? De colocar um sinal negativo na frente de um número. Este é um exemplo de operador que só precisa de um operando. Já quando precisarmos de dois operandos nosso operador será binário, exemplo, ao somar `5+9`. Sempre a operação será da esquerda para direita e sempre haverá um retorno para essa operação (Figura 4.1).

O ponto principal desta conversa inicial é que todo operando retorna um valor, caso não fosse dessa forma não teríamos como criar expressões como a do cálculo da raiz da equação de primeiro grau: `x = -b/a`. No cálculo de `x` é utilizado um operador unário, o menos (`-`), e o operador binário, o de divisão (`/`); o resultado dessas duas operações é então atribuído à variável `x` obtendo, assim, o valor desejado do cálculo.

Sem mais delongas vamos ver como trabalhar com os operadores unários e binários no

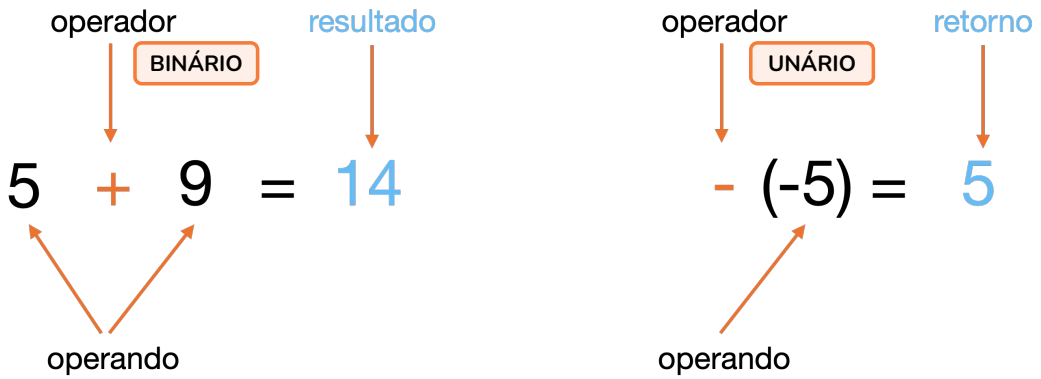

Figura 4.1: Exemplo de operador binário e de operador unário.

javascript. Este capítulo teve como base informações presentes no livro de Flanagan (2004) com algumas atualizações.

4.1.1 Operadores aritméticos

Os operadores aritméticos efetuam operações aritméticas ou outras manipulações numéricas em seus operandos. Os operadores aritméticos básicos são: `**` (potência), `*` (multiplicação), `/` (divisão), `%` (módulo: resto de uma divisão), `+` (adição) e `-` (subtração). A ordem apresentada exibe a ordem de precedência da operação. Caso queira forçar que uma operação seja realizada isoladamente utilize parênteses `()`.

Vejamos alguns exemplos no *javascript*, execute esses comandos no `console`:

```
1 3**2;      //potência, resultado 9
2 3*2;       //multiplicação, resultado 6
3 3/2;       //divisão, resultado 1.5
4 3%2;       //módulo, resultado 1
5 3+2;       //soma, resultado 5
6 3-2;       //subtração, resultado 1
7 2*2**3;    //primeiro potência (2**3 = 8), depois multiplicação (2*8),
    resultado 16
8 (2*2)**3; //primeiro a operação entre parênteses (2*2=4), depois a potê
    ncia (4**3), resultado 64
```

A notação destacada em vermelho representa um *comentário* – uma forma de você falar sobre o código sem que o texto seja processado pelo programa. É como se sua existência não fosse percebida pelo programa. Um comentário no *javascript* é definido com as duas barras juntas (`//`), tudo que estiver à direita delas será desconsiderado pelo programa.

A operação de `%` (módulo) é bastante útil para verificar a divisibilidade entre dois números. Por exemplo, se você quiser saber se um número `x` é par ou ímpar basta verifica o valor de `x%2`. Se o resultado for `0` é porque `x` é par (resto da divisão), se for `1` o número `x` é ímpar.

> ! Caso faça uso dos parênteses lembre-se de ao abir um parênteses, `"("`, não esquecer de fechar, `")"`. Caso isso não aconteça um erro será exibido.

O operador de soma também pode ser utilizado para somar `string`, no caso teremos uma concatenação. Caso seja realizada uma soma de uma `string` e um `number` o resultado será uma `string`. Vejamos alguns exemplos:

```
1  "1"+ "2";                //resultado: "12"
2  "1" + 2.3;               //resultado: "12.3"
3  "Deu três? " + 1 + 2;    //resultado: "Deu três? 12"
4  1 + 2 + " Deu três?";    //resultado: "3 Deu três?"
```

Observa-se neste exemplo que a ordem em que os operadores ocorrem é sempre da esquerda para direita. Por este motivo na linha 4 houve uma soma (1+2) antes da concatenação.

4.1.2 Operadores aritméticos unários

Os operadores unários modificam o valor de um único operando para produzir um novo valor. Em *javascript*, todos os operadores unários têm precedência alta. Todos operadores aritméticos unários descritos nesta seção: `+` (mais unário), `-` (menos unário), `++` (incremento) e `--` (decremento); convertem seu único operando em um número, se necessário. Note que os operadores `+` e `-` são usados tanto operadores unários como binários.

- `+`: este operador converte seu operando em um número e retorna esse valor convertido (utilize no caso da `string` conter um número ou no caso de um `boolean` (retorno 0, `false`, ou 1,`true`)).
- `-`: quando usado como operador unário, ele converte seu operando em um número, se necessário, e depois troca o sinal do resultado.
- `++`: este operador soma um (`1`) ao seu único operando. O operador também converte seu operando em um número antes de proceder com a operação de incremento. O valor de retorno do operador `++` depende de sua posição relativa ao operando.
  Quando usado antes do operando, onde é conhecido como operador de pré-incremento, ele incrementa o operando e retorna o valor incrementado desse operando.
  Quando usado após o operando, onde é conhecido como operador de pós-incremento, ele incrementa seu operando, mas retorna o valor não incrementado desse operando.
  Pode parecer confuso, mas é bem simples, depois veremos exemplos.
- `--`: este operador espera um operando. Ele converte o valor do operando em um número, subtrai `1` e atribui o valor decrementado ao operando. Assim como o operador `++`, o valor de retorno de `--` depende de sua posição relativa ao operando. Quando usado antes do operando, ele decrementa e retorna o valor decrementado. Quando usado após o operando, ele decrementa o operando, mas retorna o valor não decrementado.

Vamos aos exemplos comentados:

```
1  var i = +"1";          //i é um número: 1
2  var i = +"-5";         //i é um número: -5
3  var i = +true;         //i é um número: 1
4  var i = +false;        //i é um número: 0
5  var i = -(5);          //i é um número: -5
6  var i = -true;         //i é um número: -1
7  var i = -false;        //i é um número: -0
8  var i = 1, j = ++i;    //i e j são ambos 2
9  var i = 1, j = i++;    //i é 2 e j é 1
10 var i = 1, j = --i;    //i e j são ambos 0
11 var i = 1, j = i--;    //i é 0 e j é 1
```

Nas linhas 8 a 11 pode-se observar que em uma mesma linha podem ser criadas mais de uma variável com os identificadores separados por vírgula sem colocar `var` novamente.

4.1.3 Operadores de igualdade e desigualdade

Trabalhamos com alguns operadores lógicos como o igual (`==`) e maior que (`>`) antes, agora veremos como utilizar no *javascript* outros:

- `==`: operador binário que verifica se dois dados são iguais. Retorna `true` caso os dados sejam iguais e `false` caso sejam diferentes.
- `!=`: operador binário que verifica se dois dados são diferentes. Retorna `true` caso os dados sejam diferentes e `false` caso sejam iguais.

Verificaremos isso com alguns exemplos:

```
1 var i = 50;
2 var j = "0k";
3 var k = 50;
4 i == j;          //retorna: false
5 i == k;          //retorna: true
6 i != j;          //retorna: true
7 i == j;          //retorna: false
8 i != k;          //retorna: false
```

4.1.4 Operadores de comparação

Os operadores de comparação também retornam valores booleanos, similar aos operadores de igualdade. Iremos trabalhar com os seguintes:

- `<`: retorna `true` se o primeiro operando é menor que o segundo operando; caso contrário, retorna `false`.
- `>`: retorna `true` se o primeiro operando é maior que o segundo operando; caso contrário, retorna `false`.
- `<=`: retorna `true` se o primeiro operando é menor ou igual que o segundo operando; caso contrário, retorna `false`.
- `>=`: retorna `true` se o primeiro operando é maior ou igual que o segundo operando; caso contrário, retorna `false`.

Vejamos alguns exemplos:

```
1 11 > 3;         //retorna: true
2 8 < 10;         //retorna: true
3 12 >= 71;       //retorna: false
4 -13 <= -9;      //retorna: true
5 9 > 9;          //retorna: false
6 55 <= 8;        //retorna: false
```

Bem simples, né!? Bora lá que tem mais...

4.1.5 Operadores de lógicos

Os operadores lógicos `&&` (E lógico), `||` (OU lógico) e `!` (NÃO lógico) efetuam álgebra booleana e são frequentemente usados em conjunto com os operadores relacionais para combinar duas expressões relacionais em outra mais complexa. Antes de vermos esses operadores em ação apresento na Figura 4.2 as tabela-verdade de cada um deles.

Veremos um exemplo prático disso. Suponha que uma proposição seja $5 > 10$ (`A`) e outra proposição seja $9 == 9$ (`B`). Sabemos que `A` é falso e `B` é verdadeiro, certo? Logo, `A` E `B` será falso e `A` OU `B` será verdadeiro. Esses operadores serão usados principalmente se quisermos realizar expressões lógicas como saber se um número esta entre dois valores ($120 \geq x \geq 10$).

Vamos ver mais alguns exemplos com *javascript*:

```
1 var x = 50;
2 x >= 10;                    //retorna: true
3 x <= 120;                   //retorna: true
4 (x >= 10) && (x <= 120);    //retorna: true
5 (x >= 80) && (x <= 120);    //retorna: false
```

```
6  (x >= 80) || (x <= 120);     //retorna: true
7  !(x <= 120);                 //retorna: false
8  !(x > 120);                  //retorna: true
9  !(x < 50);                   //retorna: true
```

observe que para o `&&` retornar `true` as duas proposições (operadores) devem ser `true`. Por isso, na linha 4 temos o retorno `true`, é o caso da verificação $120 \geq 50 \geq 10$ (50 é maior ou igual a 10 E 50 é menor ou igual a 120); e na linha 5 temos o retorno `false`, $120 \geq 50 \geq 80$ (50 é maior ou igual que 80 E 50 é menor ou igual a 120). Na linha 6 temos um retorno `true`, pois, precisamos que pelo menos uma das proposições seja verdadeira. E na linha 7 temos uma negação de algo `true` que retorna `false`.

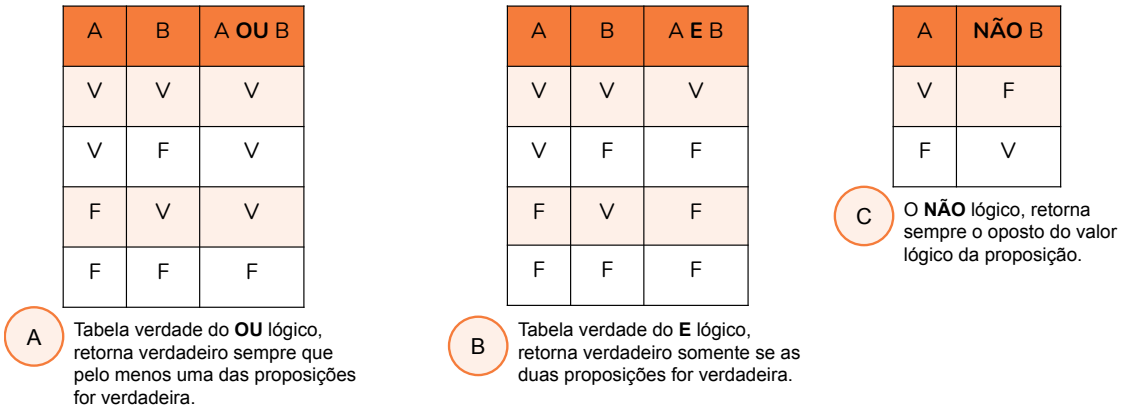

Figura 4.2: Tabela-verdade envolvendo os operadores de E lógico, OU lógico e NÃO lógico. Cada tabela exibe todas as combinações possíveis das proposições `A` e `B`.

4.1.6 Operadores de atribuição

Além do operador de atribuição normal (=), o *javascript* aceita vários outros operadores de atribuição que fornecem atalhos por combinar atribuição com alguma outra operação. Por exemplo, o operador `-=` efetua subtração e atribuição. A expressão a seguir:

```
total -= impostos
```

é equivalente a esta:

```
total = total - impostos.
```

Na Tabela 4.1 mostro uma lista das operações de atribuição possíveis, sua equivalência e exemplos. O uso desses operadores servem para simplificarmos mais o código.

Os operadores de atribuição só podem ser utilizados com variáveis.

Não se preocupe se você não entendeu tudo. Este material servirá de referência para quando precisar. Ao longo dos exemplos não utilizaremos nem 50% desses operadores. Faremos mais alguns exemplos com expressões compostas após vermos os comandos de entrada e saída de dados.

Operador	Exemplo	Equivalente	Estudo de Caso

+=	a+=b	a = a + b	`var a = 5;`
			`var b = 10;`
			`a += b;  //a é 15 e b é 10`

-=	a-=b	a = a - b	`var a = 15;`
			`var b = 10;`
			`a -= b;  //a é 5 e b é 10`

*=	a*=b	a = a * b	`var a = 3;`
			`var b = 5;`
			`a *= b;  //a é 15 e b é 5`

**=	a**=b	a = a ** b	`var a = 5;`
			`var b = 2;`
			`a **= b;  //a é 25 e b é 2`

/=	a/=b	a = a / b	`var a = 5;`
			`var b = 10;`
			`a /= b;  //a é 0.5 e b é 10`

%=	a%=b	a = a % b	`var a = 10;`
			`var b = 7;`
			`a %= b;  //a é 3 e b é 7`

Tabela 4.1: Principais operadores de atribuição.

4.2 Entrada e Saída de Dados

Vimos na Seção 2.2 que é importante termos comandos para entrada e saída de dados (informações), pois fazem parte do objetivo de todo programa: resolver problemas. Todas as representações de algoritmo utilizam esses comandos para obtermos a informação, na entrada (*input*), e após o processamento do programa escrevemos na saída (*output*) o resultado dos cálculos realizados ou mesmo um texto para notificar o usuário da nossa aplicação do que está ocorrendo internamente no programa.

Como comecei a falar de programa vamos começar a trabalhar com outra área do Replit, a área onde colocaremos o código-fonte (Figura 3.2E). Os comandos que falarmos nesta seção serão colocados nessa área.

4.2.1 Entrada de Dados no Javascript

A entrada dos dados no *javascript* será realizada por meio do comando `prompt()`. Opcionalmente, você pode adicionar uma `string` entre os parênteses do comando para informar ao usuário o que você quer. Por exemplo: `prompt("Informe seu nome");` será exibido no `console` o texto `Informe seu nome>` e o `console` aguarda que você escreva algo. Ao usar o `prompt()` em nosso programa teremos que atribuir sua execução a uma variável,

somente assim realmente iremos capturar a informação dada pelo usuário e posteriormente realizar operações com este dado.

Um detalhe importante relacionado ao `prompt()` é que independentemente do valor informado ele sempre irá convertê-lo para uma `string`, logo se quisermos capturar um `number` deveremos converter a informação capturada pelo `prompt()` de forma apropriada. Vejamos algumas conversões de tipo de dados, também chamadas de *casting*, que serão bastante úteis:

- `String(x)` ou `(x).toString()`: converte o dado `x` para uma `string`. Se `x` for um `boolean` a `string` será seu valor em forma de texto, `"true"` ou `"false"`.
- `Number(x)`: converte um dado `x` para o `number`; caso `x` seja uma `string` que possua algo além de números a conversão retorna o valor `NaN`, representa um valor inválido. Caso `x` seja um `boolean`, o valor retornado será `1` caso seja `true` e `0` caso seja `false`.
- `parseInt(x)`: este comando deve ser usado no caso em que você queira garantir que `x` seja transformado em um número inteiro.
- `parseFloat(x)`: este comando deve ser usado no caso em que você queira garantir que `x` seja transformado em um número real.

Irei limitar nossos estudos a esses comandos de conversão, pois serão suficientes para fazermos nossas práticas e exercícios, porém, existem outros tipos de conversão[1]. Você deve estar se perguntando, cadê os exemplos? Vamos ver os exemplos após mostrar os comandos de saída, aí as coisas irão fluir melhor.

4.2.2 Saída de Dados no Javascript

A saída de dados, como falamos na Seção 2.2, é a impressão de uma informação. Isso no *javascript* será realizado por meio do comando `console.log(x)`, onde `x` é a informação que queremos que apareça em nosso `console`. Este comando é bastante versátil, pois em `x` você pode usar qualquer tipo de dado, e caso queira imprimir informações diferentes você pode concatená-las ou utilizar a vírgula. Vejamos o Código 4.2:

Código 4.1 Saída de Dados Simples[2]

```
1 var a = 10;
2 var b = 7;
3 var soma_a_b = a+b;
4 console.log(a + "+" + b + "=" + soma_a_b)  //será impresso: 10+7=17
```

■

Para executar o Código 4.2, e os demais códigos que iremos ver a partir de agora, iremos colocar este comando na área de código-fonte e clicar no botão de executar (Figura 3.2B), o resultado aparecerá no `console`. Na Figura 4.3 mostro o passo a passo.

No Código 4.2 irei mostrar outras formas de utilizar o `console.log()` que produzirão o mesmo resultado.

Código 4.2 Saída de Dados Alternativas[3]

```
1 var a = 10;
2 var b = 7;
3 var soma_a_b = a+b;
4 console.log(a,"+",b,"=",soma_a_b)          //será impresso: 10 + 7 = 17
5 console.log(`${a} + ${b} = ${soma_a_b}`)   //será impresso: 10 + 7 = 17
```

[1] <https://www.w3schools.com/js/js_type_conversion.asp>
[2] <https://replit.com/@DaniloBorges/SaidaDeDados-Ex1>
[3] <https://replit.com/@DaniloBorges/SaidaDeDados-Ex2>

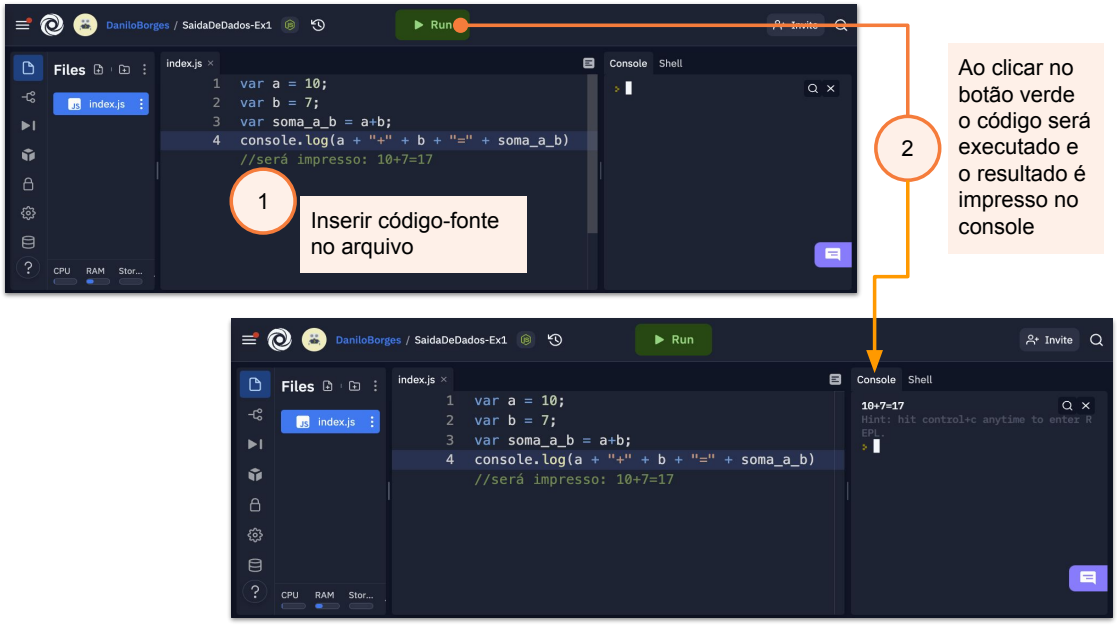

Figura 4.3: Executando o código. 1) O código-fonte fica nesta área. 2) O processo de executar o código é realizado botão verde (*Run*), o seu programa irá processar da linha 1 até a linha 4, e se houverem comandos de entrada e saída de dados esses valores aparecerão no `console`.

Na linha 4 temos cada termo que queremos exibir separado por vírgulas; na saída o comando retorna uma `string`, onde cada termo é convertido em uma `string` e a vírgula é substituída por um espaço, na concatenação. Na linha 5 temos um exemplo do poder da definição de uma `string` utilizando crase. Podemos escrever uma `string` normalmente e no local onde queremos que o valor da variável apareça a colocamos entre chaves com um cifrão junto a abertura da chave, (`${<variável>}`). ■

4.3 Combinando Comandos de Entrada e Saída de Dados no Javascript

Para ilustrar o uso dos comandos de entrada e saída de dados, antes de cada código apresentado falarei o que será realizado.

■ **Código 4.3** Entrada e Saída de Dados para Soma[4]. Neste código são obtidas duas informações pelo console que serão convertidas para `number` e somadas. Posteriormente é exibido cada operando e o resultado da soma.

```
1 console.log("== Programa que soma dois números == ");
2 var a = Number(prompt("Informe o primeiro número"));
3 var b = parseFloat(prompt("Informe o segundo número"));
4 var soma_a_b = a+b;
5 console.log("Resultado:")
6 console.log(`${a} + ${b} = ${soma_a_b}`)
```

Na Figura 4.4 mostro o passo a passo da execução deste código. No exemplo o usuário coloca dois números `80` e `78` e o programa imprime a soma dos mesmos: `158`. ■

[4] <https://replit.com/@DaniloBorges/EntradaSaidaDeDados-Ex1>

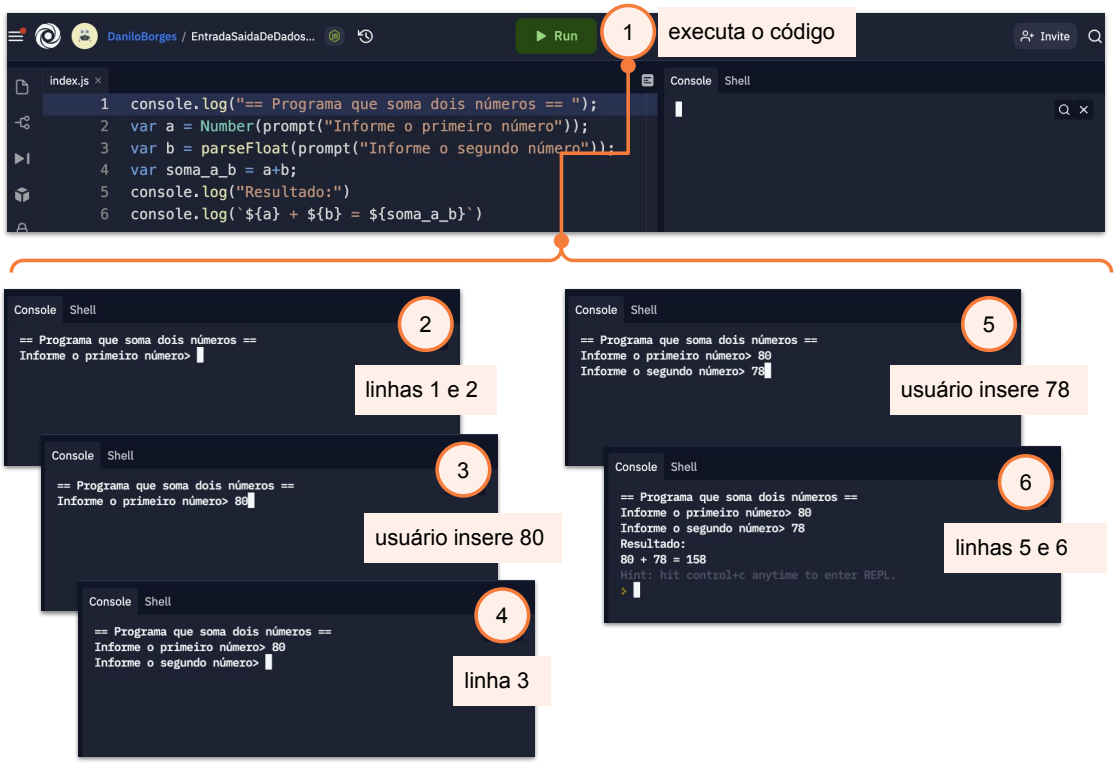

Figura 4.4: Execução do Código 4.5. A sequência indica como cada resultado irá aparecer no console, primeiramente deve-se executar o código. As linhas indicadas no texto se referem as linhas que são processadas no código-fonte para produzir a saída no console.

Os próximos dois exemplos não serão ilustrados, pratique e veja os resultados no `console`.

■ **Código 4.4** Entrada e Saída de Dados para Apresentação[5]. Neste código é obtido o nome do usuário e sua idade, e é posteriormente impresso essa informação no `console`.

```
1 console.log("== Programa apresentação == ");
2 var nome = prompt("Informe seu nome");
3 var idade = parseInt(prompt("Informe a sua idade"));
4 console.log(`Olá, ${nome}! Você tem ${idade} anos.`);
```

■

■ **Código 4.5** Entrada e Saída de Dados para Ano Bissexto[6]. Neste código é obtido um ano (2020, 2022, etc.), e posteriormente impresso no `console` qual seria a possibilidade do ano informado ser bissexto.

```
1 console.log("== Programa verifica bissexto == ");
2 var ano = parseInt(prompt("Informe ao ano"));
3 var verifica = ano%4;
4 console.log(`Para o ano ser bissexto ${ano} módulo 4 deve ser igual a
      zero.`);
5 console.log("O resultado foi: ",verifica);
```

■

[5] <https://replit.com/@DaniloBorges/EntradaSaidaDeDados-Ex2>
[6] <https://replit.com/@DaniloBorges/EntradaSaidaDeDados-Ex3>

Legal, começamos a desenvolver uma parte essencial para os nossos programas: uso de operadores e de comandos de entrada e saída. A próxima etapa será conhecer e utilizar as estruturas de alteração de fluxo na Parte III.

4.4 Exercícios

Exercício 4.1 Marque quais comandos de conversão que podemos utilizar para converter de `string` para `number`:

a) `Number()`
b) `console.log()`
c) `parseInt()`
d) `parseFloat()`

■

Exercício 4.2 Quais dos operadores abaixo podem ser tanto unários quanto binários:

a) `+`
b) `-`
c) `**`
d) `%`
e) `/`

■

Exercício 4.3 Quais dos operadores abaixo retornam um valor lógico, `boolean`:

a) `-`
b) `&&`
c) `==`
d) `!`
e) `>=`

■

Exercício 4.4 Utilize o `console` do javascript para realizar as seguintes operações:

a) $3-5^2$. *Resposta esperada:* `-22`
b) $\frac{25}{7}-\sqrt{25}$. Dica: na raiz quadrada você deve elevar o número para `0.5`. *Resposta esperada:* `-1.4285714285714284`
c) $\frac{3}{7}\times 5-3\times(-2+5^{-1})$. *Resposta esperada:* `7.542857142857143`
d) $8\%3 == 5$. *Resposta esperada:* `false`
e) $x>2$ `E` $x\leq 5$, onde $x=7$. *Resposta esperada:* `false`
f) $(x>2)$ `OU` $(x>5$ `E` $x\leq 6)$, onde $x=3$. *Resposta esperada:* `true`

■

Exercício 4.5 Faça um código que realiza a multiplicação de dois números e imprima o resultado. O usuário irá informar os dois números. Dica: utilize como base o Código 4.5.. ■

Exercício 4.6 Faça um código que recebe três números (`a`, `b` e `x`) e imprima o resultado da expressão `a*x+b`. O usuário deverá informar os três números. ■