期末项目 – 利率计算器

Skills Network

预计所需时间： 2 小时

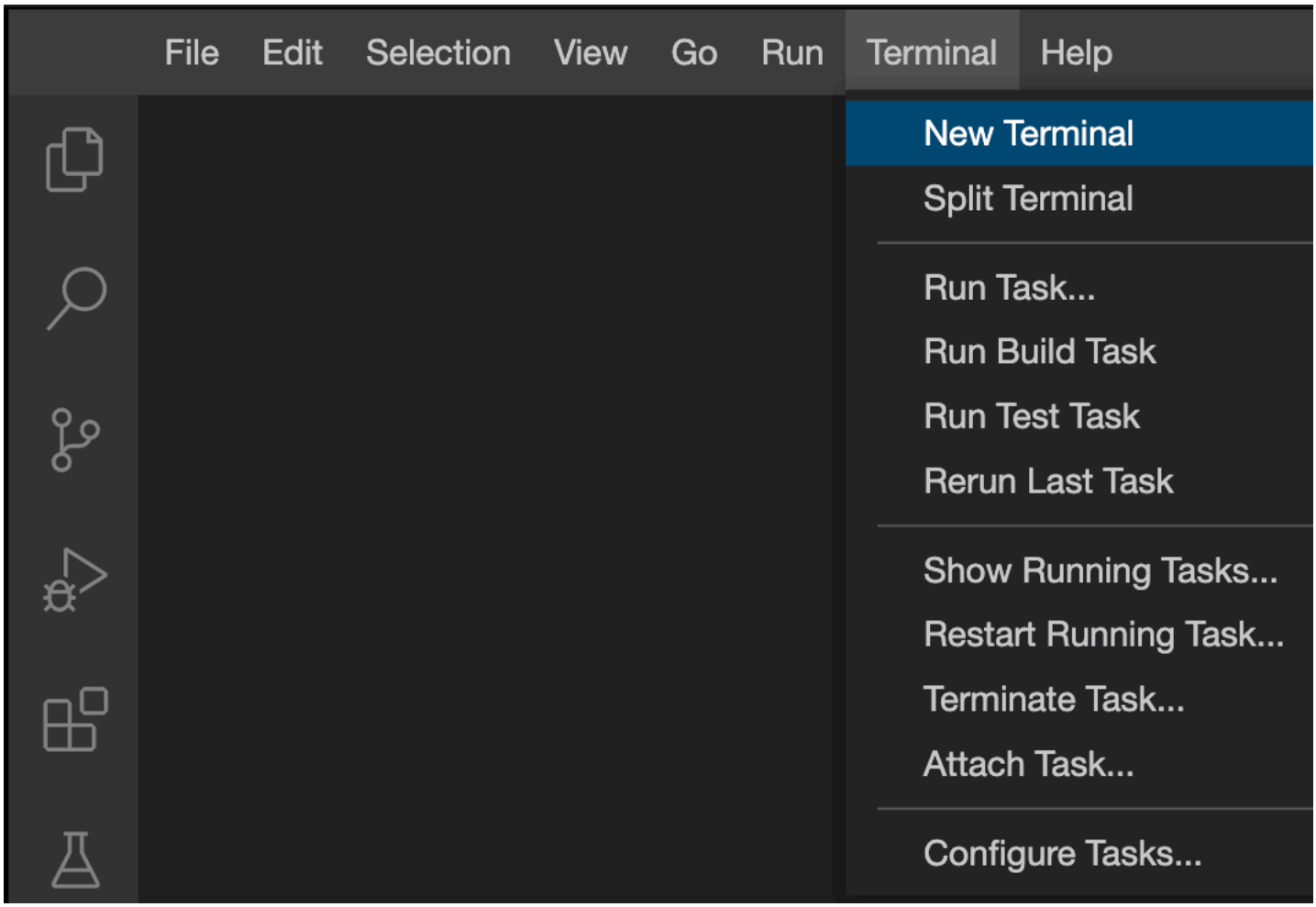
在这个期末项目中，您将致力于利率计算器，并利用您在本课程中学到的技能进行增强。

目标：

1. 学习如何在 Chrome 中调试，使用断点、观察和控制台中的错误
2. 使用 Jasmine 测试您的业务逻辑
3. 改善 SEO，以便在搜索引擎上获得更好的覆盖和可见性
4. 使用 Webpack 创建资产

设置：创建应用程序

1. 通过编辑器中的菜单打开终端窗口：**终端 > 新终端**。



2. 如果您当前不在项目文件夹中，请复制并粘贴以下代码以切换到您的项目文件夹。

```
cd /home/project
```

3. 如果 Git 仓库尚不存在，请运行以下命令以克隆包含此项目所需启动代码的 Git 仓库。

```
[ ! -d 'intermediate_webdev_finalproject' ] && git clone https://github.com/ibm-developer-skills-network/intermediate_webdev_finalpr
```

4. 切换到目录 **intermediate_webdev_finalproject** 开始实验。

```
cd intermediate_webdev_finalproject
```

5. 列出该目录的内容以查看本实验的工件。

```
ls
```

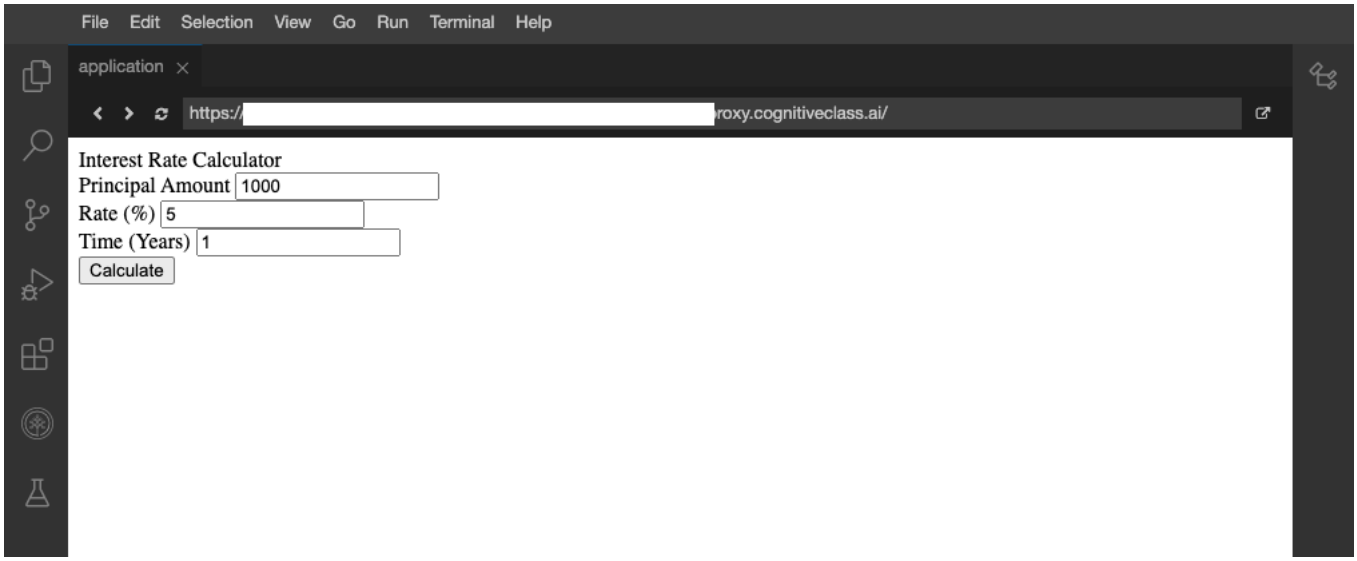
6. 在终端运行以下命令以托管您的网页。

```
python3 -m http.server
```

7. 要在浏览器中测试您的应用程序，请首先运行应用程序。

启动应用程序

8. 它的样子如下：



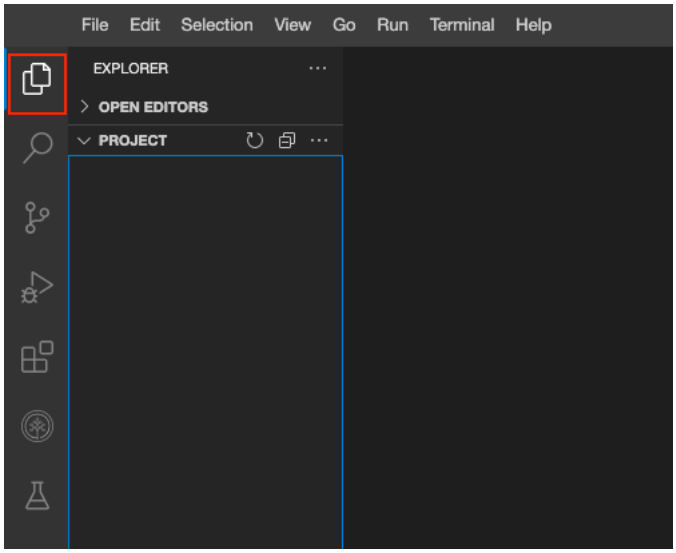
9. 您可以复制 URL 并粘贴到浏览器中，以查看您网页的开发者视图。

`⌘ Cmd + ⌥ Option + U` 在 Mac 上或 `CTRL + U` 在 Windows 上。

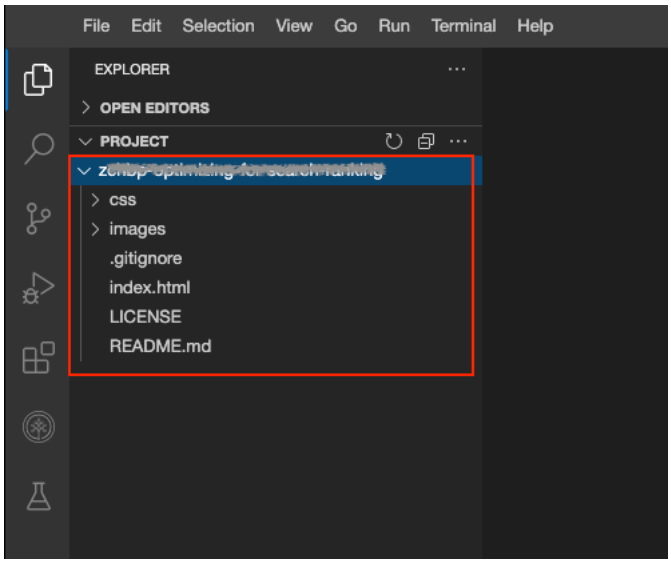
在云IDE中处理文件

如果您是云IDE的新手，本节将向您展示如何在云IDE中创建和编辑项目中的文件。

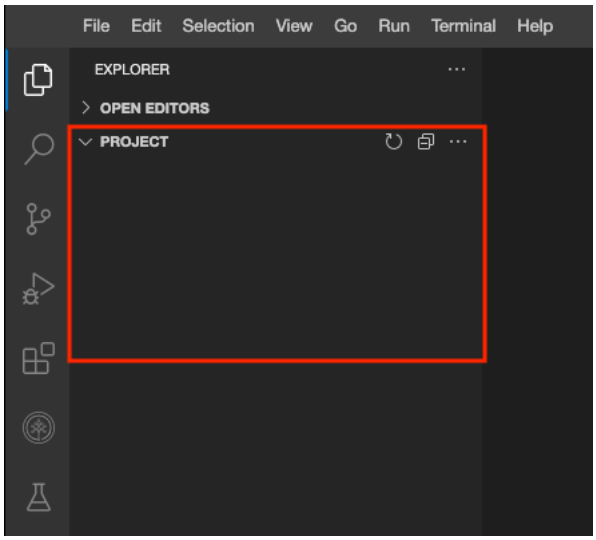
要查看云IDE中的文件和目录，请单击此文件图标以显示它。



如果您已克隆（使用 `git clone` 命令）模板/起始代码，则它将如下所示：

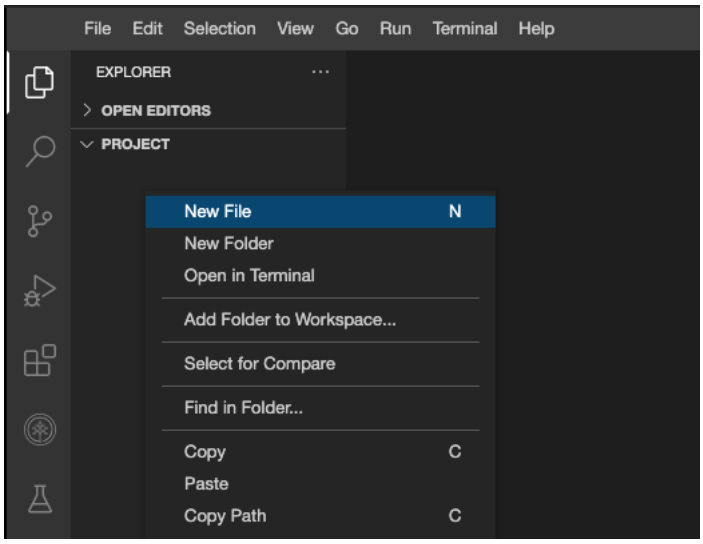


否则，空项目看起来是这样的：



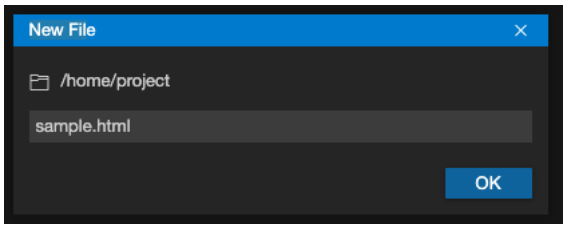
创建新文件

您可以右键单击并选择“新建文件”选项以在项目中创建文件。

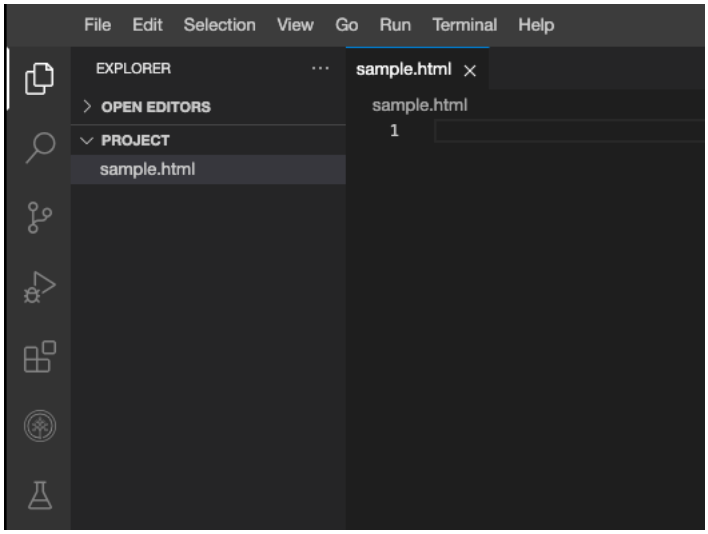


您也可以选择 文件 -> 新建文件 来执行相同操作。

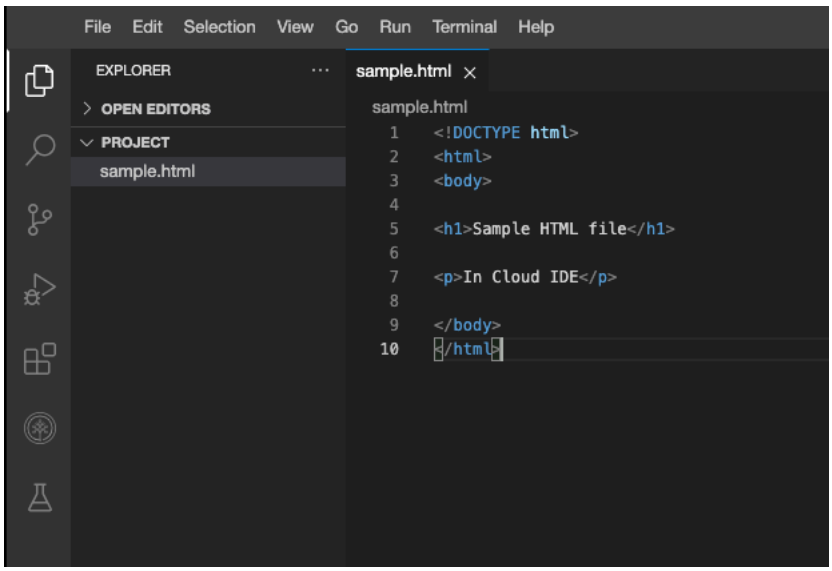
然后，它会提示您输入此新文件的名称。在下面的示例中，我们正在创建 `sample.html`。



单击目录结构中的文件名 `sample.html` 将在右侧窗格中打开该文件。您可以创建所有不同类型的文件；例如，JavaScript 文件可以命名为 `FILE_NAME.js`。

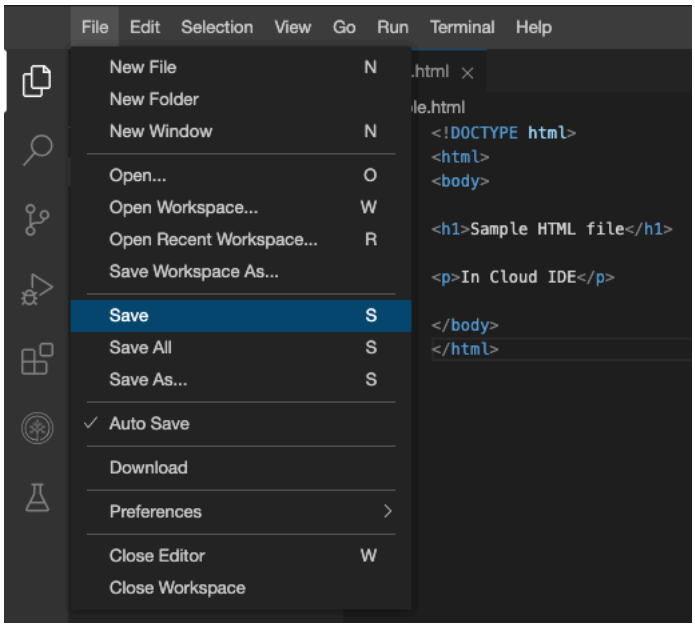


在示例中，我们只是粘贴了一些基本的 HTML 代码，然后保存了文件。



并通过以下方式保存它：

- 进入菜单。
- 在 Mac 上按 `⌘ Cmd + S` 或在 Windows 上按 `CTRL + S`。
- 或者，它也可以自动保存。



任务 1: favicon

我们首先注意到没有定义 favicon，并且在某些情况下，它也会抛出错误：

```
GET http://HOSTNAME:8000/favicon.ico 404 (未找到文件)
```

使用以下 favicon.ico 文件，并将其放在项目的根目录。默认情况下，它期望文件名为 `favicon.ico`。



在终端中运行以下命令，以在项目目录中下载图像。

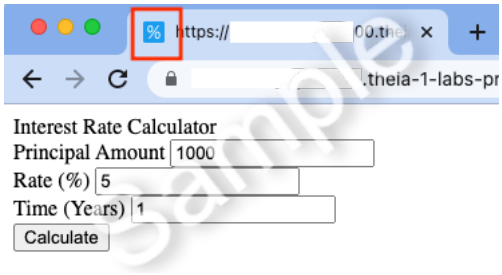
```
cd /home/project/intermediate_webdev_finalproject
curl -Ol https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-WD0231EN-SkillsNetwork/images/favicon.i
```

虽然不是必需的，但您也可以在 HTML 的 `head` 部分明确地定义 favicon。

▼ 提示

```
<link rel="icon" href="favicon.ico">
```

如果你刷新浏览器，你应该能在浏览器标签中看到我们的利率计算器的图标。



评估：截取错误的屏幕截图并将其保存为 `01-favicon-visible.png`。

任务 2：缺失的 CSS 引用

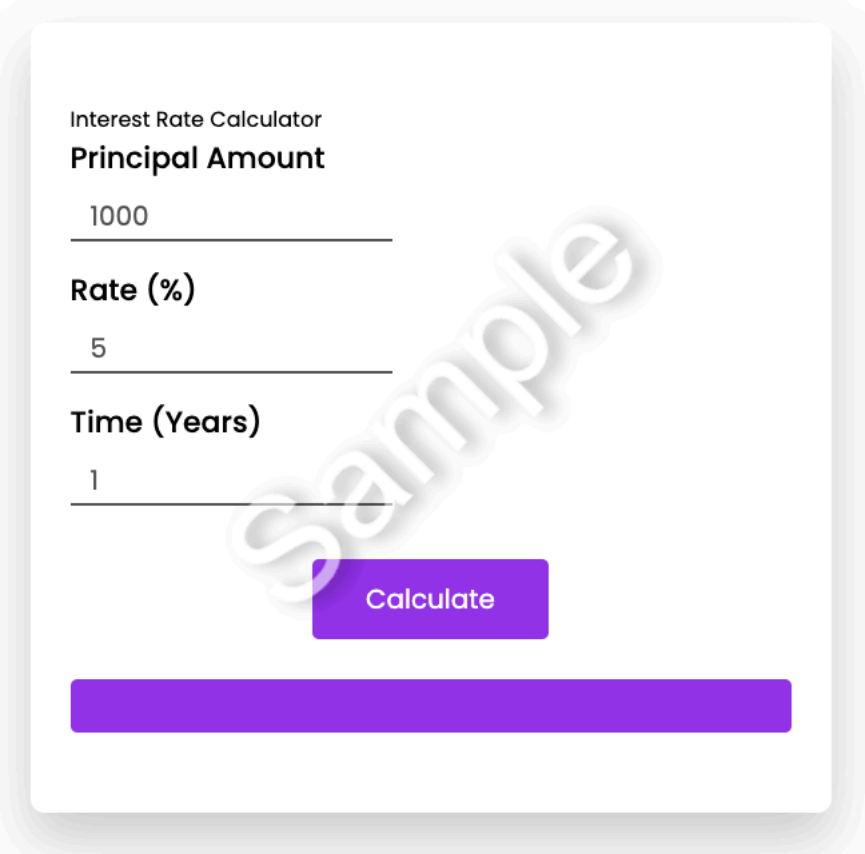
如果你检查页面，你会发现元素已经定义了 CSS，但 CSS 文件本身并没有在页面上被引用。

让我们在 `head` 部分添加 CSS 引用，并在浏览器中刷新页面。

▼ 提示

```
<link rel="stylesheet" href="style.css" />
```

如果您现在刷新浏览器，您应该会看到 CSS 被应用。



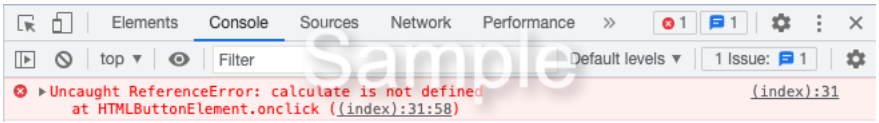
评估：截取输出的屏幕截图，显示导航栏的 CSS 工作情况，并将其保存为 `02-css-working.png`。

任务 3：缺失的 JavaScript 引用

当你按下 `Calculate` 按钮时会发生什么？什么也不发生！

但是在 Chrome DevTools 中查看并检查错误。

```
Uncaught ReferenceError: calculate is not defined
at HTMLButtonElement.onclick ((index):31:56)
```



评估：截取错误的屏幕截图，并将其保存为 `03-calculate-missing.png`。

之所以没有任何反应，是因为虽然你已经有了 `script.js` 文件，但它并没有在你的 HTML 页面中被引用（就在结束的 body 标签 `</body>` 之前）。而 `calculate` 函数是在那个 JavaScript 文件中定义的。

现在，将 JavaScript 引用添加到你的页面中。

▼ 提示

```
<script src="script.js"></script>
```

请尝试通过刷新浏览器并重新计算利息来再次尝试。

你现在看到什么错误?

▼ 提示

```
script.js:3 Uncaught TypeError: Cannot read properties of null (reading 'value')
    at calculate (script.js:3:47)
    at HTMLButtonElement.onclick ((index):31:58)
```

我们将在下一个任务中修复这些错误。

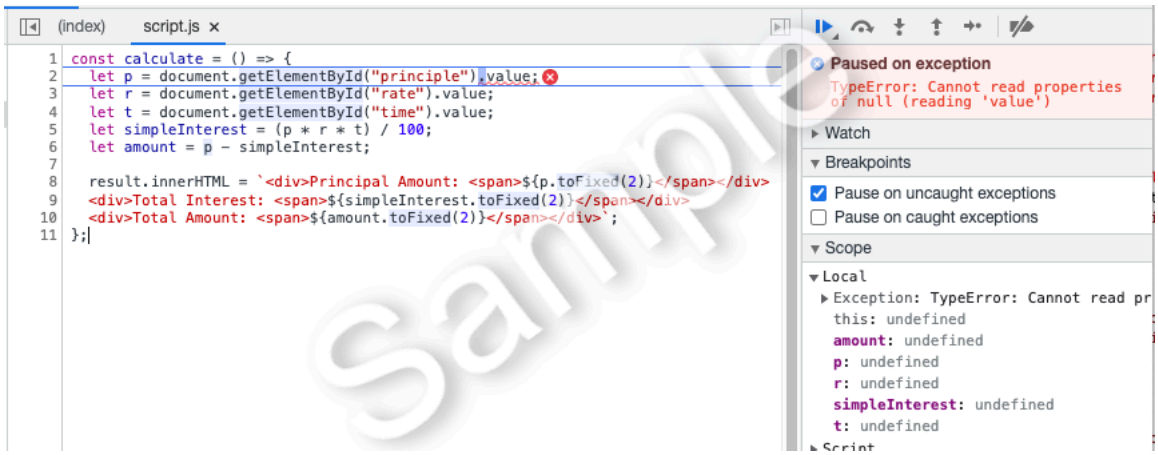
任务 4：逐步调试

让我们通过选择这个选项来暂停未捕获的异常。



现在再试一次计算操作。

它应该突出显示您有异常的行。



评估：截取调试器停止时的屏幕截图，并将其保存为 `04-pause-on-exception.png`。

错误 `TypeError: Cannot read properties of null (reading 'value')` 意味着我们的代码无法找到该元素，因此没有 `value` 属性。

查看页面的源代码并找到错误。

▼ 提示

元素名称是 `principal` 而不是 `principle`。

```
<input type="number" id="principal" value="1000" />
```

修正JavaScript文件中的名称，使其变为：

```
let p = document.getElementById("principal").value;
```

任务 5：使用正确的数据类型

再次计算利息，您将看到错误：

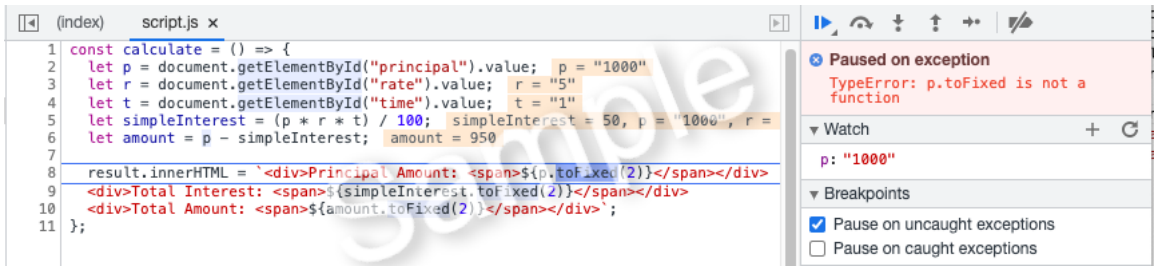
```
TypeError: p.toFixed is not a function
```

要理解这个错误，您需要了解这个函数 `toFixed`。

toFixed() 方法使用定点表示法格式化数字。

在 DevTools 中，对 `p` 变量添加观察。它看起来是什么数据类型?

▼ 提示
它是字符串类型，而不是数字。并显示值为 "1000"。



评估：截取一张显示 `p` 添加在观察中的屏幕截图，并将其保存为 `05-watch-on-p.png`。

让我们将您的输入的数据类型正确转换为数字，而不是字符串。

```
let p = Number(document.getElementById("principal").value);
```

```
  let r = Number(document.getElementById("rate").value);
  let t = Number(document.getElementById("time").value);
```

重新计算，你会看到输出。

任务 6：准备测试

目前，你的 `calculate` 函数没有产生正确的结果。在本金 1000，5% 利率，1 年的情况下；总金额应该是 1050，而不是 950。

$$\text{简单利息} = \frac{P \times R \times T}{100} \text{简单利息} = 100\text{P} \times \text{R} \times \text{T}$$

$$\text{总金额} = P + \text{简单利息} \text{总金额} = \text{P} + \text{简单利息}$$

从我们的例子：

$$\text{简单利息} = \frac{1000 \times 5 \times 1}{100} = 50 \text{简单利息} = \frac{1000 \times 5 \times 1}{100} = 50$$

$$\text{总金额} = 1000 + 50 = 1050 \text{总金额} = 1000 + 50 = 1050$$

为了使我们的代码可测试，我们需要将业务逻辑与 UI 逻辑分开。

你需要在 `script.js` 中创建两个独立的函数来执行：

1. `calculateSimpleInterest`：接受本金、利率和时间作为输入，并返回如前所示的计算结果。

▼ 提示

```
const calculateSimpleInterest = (principal, rate, time) => {
  return (principal * rate * time) / 100;
}
```

2. `calculateTotalPayableAmount`：一个相当简单的函数，它接受本金和利息金额作为输入，并通过将它们相加来返回最终值。然而，你会注意到，我们之前的开发者犯了一个错误，而是将这两者相减。我们暂时就这样保持。

▼ 提示

```
const calculateTotalPayableAmount = (principal, interestAmount) => {
  return principal - interestAmount;
}
```

让我们重构 calculate 函数以使用这两个新函数：

```
const calculate = () => {
  let p = Number(document.getElementById("principal").value);
  let r = Number(document.getElementById("rate").value);
  let t = Number(document.getElementById("time").value);
  let simpleInterest = calculateSimpleInterest(p, r, t);
  let amount = calculateTotalPayableAmount(p, simpleInterest);
  let result = document.getElementById("result");
  result.innerHTML = `<div>Principal Amount: <span>${p.toFixed(2)}</span></div>
  <div>Total Interest: <span>${simpleInterest.toFixed(2)}</span></div>
  <div>Total Amount: <span>${amount.toFixed(2)}</span></div>`;
};
```

在您那里，再进行一次更改，以便您可以使用 Jasmine 测试这段原生 JavaScript 代码。

```
if (typeof module !== 'undefined')
  module.exports = { calculateSimpleInterest, calculateTotalPayableAmount, calculate };
```

上述代码在 NodeJS 环境中简单地导出函数，并使其在 Jasmine 测试套件中可见。

任务 7：使用 Jasmine 进行测试

1. 安装 Jasmine

让我们将 Jasmine 作为开发依赖安装到我们的项目中，以便我们可以运行测试。然后也初始化它。

▼ 提示

```
npm install --save-dev jasmine
npx jasmine init
npx jasmine
```

您应该看到以下输出：

未找到规格



评估：截取终端的屏幕截图，显示未找到规格，并将其保存为 `06-no-spec.png`。

2. 编写测试

创建一个文件 `spec/scriptSpec.js`，并以以下语句开始，以引用您的两个计算函数：

```
const { calculateSimpleInterest, calculateTotalPayableAmount } = require('../script');
```

然后在你将要编写单元测试的地方添加 `describe` 占位符：

```
describe("Interest Rate Calculator Tests", () => {
})
```

您现在需要编写两个测试：

i. `calculateSimpleInterest` 的测试：

调用 `calculateSimpleInterest` 函数，参数为：

- 本金: `1000`
- 利率: `5`
- 时间: `1`

预期结果为 `50`

▼ 提示

```
it("Calculate Simple Interest", () => {
    var actual = calculateSimpleInterest(1000, 5, 1);
    expect(actual).toBe(50);
});
```

ii. 测试 `calculateTotalPayableAmount`

调用 `calculateTotalPayableAmount` 函数，参数为：

- principal: `1000`
- interestAmount: `50`

预期结果为 `1050`

▼ 提示

```
it("Calculate Total Interest", () => {
    var actual = calculateTotalPayableAmount(1000, 50);
    expect(actual).toBe(1050);
});
```

通过运行 `npx jasmine` 进行测试。你应该会看到类似 `Expected 950 to be 1050.` 的消息。

3. 修复错误并再次测试

让我们修复 `script.js` 中 `calculateTotalPayableAmount` 的计算错误。

```
const calculateTotalPayableAmount = (principal, interestAmount) => {
  return principal + interestAmount;
}
```

再次测试，并确保两个测试都通过。



评估：截取终端的屏幕截图，显示两个测试都通过，并将其保存为 `07-both-tests-passed.png`。

任务 8：使用 Webpack 打包你的网页项目

现在你将使用 Webpack 使你的利率计算器可分发。

首先安装依赖。

▼ 提示

```
npm install webpack webpack-cli html-webpack-plugin --save-dev
```

通过在终端中按 `CTRL + C` 停止你的网络服务器。

重构项目，使其具有 `src` 和 `dist` 目录。首先在项目根目录下创建这两个目录。

将 `index.html`、`script.js`、`favicon.ico` 和 `style.css` 移动到 `src` 目录中。

你现在已经将 `script.js` 移动到 `src` 目录中，因此你需要在 `scriptSpec.js` 中更新引用为：

```
const { calculateSimpleInterest, calculateTotalPayableAmount } = require('../src/script');
```

如果你现在运行 `npx webpack`，你会看到一个错误。这是因为它无法识别一个入口点，通常是 `index.js`。所以你现在需要添加 webpack 的配置文件。

创建 webpack.config.js

首先在项目根目录下创建 `webpack.config.js`。

你需要在其中定义以下属性：

- mode
- entry（将是 `'./src/script.js'`）
- plugins（仅 HtmlWebpackPlugin）
- output.clean（设置为 `true`）
- output.libraryTarget（设置为 `'window'`）
- module（仅一个用于 css 的规则）

▼ 提示

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    mode: "development",
    entry: "./src/script.js",
    plugins: [
        new HtmlWebpackPlugin({
            template: 'src/index.html',
            favicon: "./src/favicon.ico",
        })
    ],
    output: {
        clean: true,
        libraryTarget: 'window'
    },
    module: {
        rules: [
            {
                test: /\.(css)$/i,
                type: "asset/resource",
                generator: {
                    filename: "[name][ext]",
                },
            },
        ],
    },
};
```

您应该看到如下输出：

```
asset favicon.ico 15 KiB [emitted]
asset main.js 3.28 KiB [compared for emit] (name: main)
asset index.html 1.17 KiB [emitted]
./src/script.js 932 bytes [built] [code generated]
webpack 5.77.0 compiled successfully in 133 ms
```

从 `/dist` 目录运行

```
cd dist
python3 -m http.server
```

如果你浏览应用程序，你会注意到我们丢失了 CSS。

需要 CSS

将 CSS 引用更改为在 `/src/index.html` 中使用 `require`

▼ 提示

```
<link rel="stylesheet" href=<%=require('./style.css')%> type="text/css" media="all" />
```

参考 main.js 而不是 script.js

Webpack 将我们的 JavaScript 文件打包成 `main.js`。这将由 Webpack 添加，因此我们可以安全地从 `/src/index.html` 中删除对 `script.js` 的引用。

再次使用 Webpack

在项目根目录打开另一个终端并运行 `npx webpack`。然后在你的第一个终端（来自 `dist` 目录）运行 HTTP 服务器。

如果你现在刷新浏览器，一切应该按预期工作。



评估：从 Cloud IDE 中截取你的 dist 目录的屏幕截图，并将其保存为 `08-dist-directory.png`。

任务 9：改善 SEO

您现在将通过提供以下内容来改善 SEO：

- 将我们的计算器的 `title` 修改为 *利率计算器 – 快速计算贷款利息*

▼ 提示：标题

```
<title>Interest Rate Calculator – Quickly Calculate Interest on Loans</title>

# 利率计算器

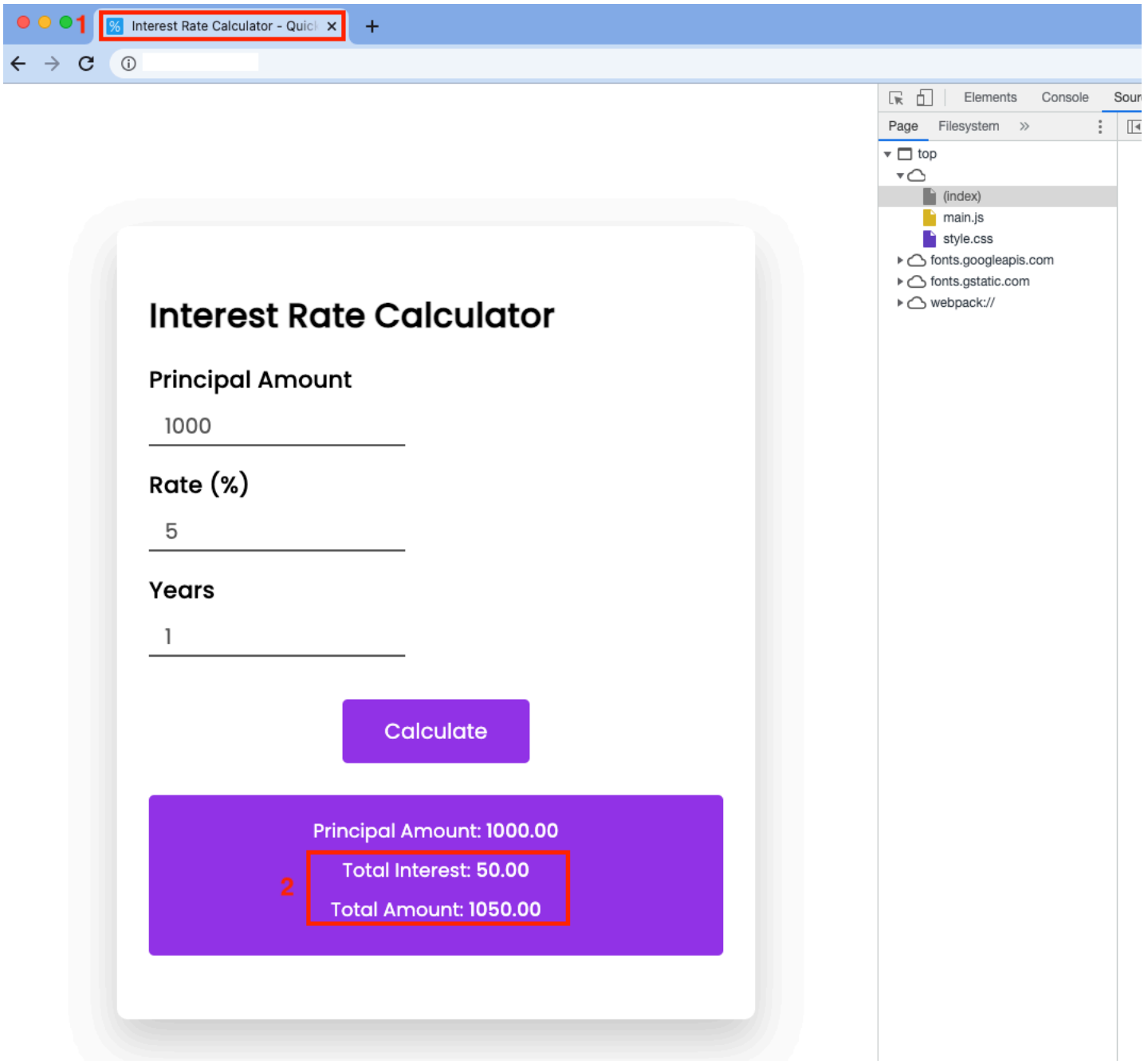
<h1>Interest Rate Calculator</h1>
```

- 添加 `meta` 标签用于描述，内容如下：

  > “快速准确地计算贷款利率，使用我们易于使用的利率计算器。确定您将支付的利息，并做出明智的财务决策。立即尝试我们的免费在线计算器。”

▼ 提示：meta

```
<meta name="description" content="Quickly and accurately calculate interest rates on loans with our easy-to-use interest rate calcul
```



评估：截取您的最终解决方案和源的屏幕截图，并将其保存为 `09-final-output.png`。

提交清单

1. **评估**：截取浏览器标签的屏幕截图并保存为 `01-favicon-visible.png`。
2. **评估**：截取显示 CSS 正常工作的计算器的屏幕截图（颜色和格式正确）并保存为 `02-css-working.png`。
3. **评估**：截取错误的屏幕截图并保存为 `03-calculate-missing.png`。
4. **评估**：截取调试器停止处的屏幕截图并保存为 `04-pause-on-exception.png`。
5. **评估**：截取在监视器中添加 `p` 的屏幕截图并保存为 `05-watch-on-p.png`。
6. **评估**：截取终端的屏幕截图，显示没有找到规范并保存为 `06-no-spec.png`。
7. **评估**：截取终端的屏幕截图，显示两个测试均已通过并保存为 `07-both-tests-passed.png`。
8. **评估**：截取 Cloud IDE 中的 dist 目录的屏幕截图并保存为 `08-dist-directory.png`。
9. **评估**：截取最终解决方案及其源代码的屏幕截图并保存为 `09-final-output.png`。

恭喜！到此为止。

Author(s)

Muhammad Yahya

(C) IBM Corporation. All rights reserved.