

Sumário

1. Introdução
 1. Objetivo
 2. Onde tirar dúvidas
 3. Como funciona o livro?
 4. Onde acessar o layout do projeto?
2. Dia 1 - Onde tudo começa
 1. Exercício 00 - Prepare o ambiente
 2. Exercício 01 - Inicie projeto
 3. Exercício 02 - Crie o HTML do cabeçalho
 4. Exercício 03 - Estruture o HTML
 5. Exercício 04 - Projeto no GitHub
 6. Desafios top top top
3. Dia 2 - Dando vida ao projeto
 1. Exercício 01 - Criar HTML da seção
 2. Exercício 02 - Base do CSS
 3. Exercício 03 - Primeiro componente
 4. Exercício 04 - Componente simple-action
 5. Exercício 05 - Componente primary-button
 6. Exercício 06 - Primeiro container
 7. Exercício 07 - Container main-menu
 8. Desafios top top top

Introdução

Salve (um oi, da periferia de São Paulo)! Tudo bom? Prazer, sou Marco Bruno. Esse livro é free (grátis) e você pode piratear ele à vontade, só não tenta ganhar uns trocados (dinheiro) em cima dele, certo?

Esse livro é um apoio para as aulas free e ao vivo do Mini curso de HTML, CSS e JavaScript que fiz nos dias 9, 10, 11, 12 e 13 de Dezembro pela twitch.tv/marcobrunobr.

Objetivo

Quero trocar o conhecimento com você sobre **HTML**, **CSS** e **JavaScript** para você começar seus estudos para o objetivo de ser Developer FrontEnd (pessoa desenvolvedora frontend).

Onde tirar dúvidas

Todas as suas dúvidas podem ser tiradas nos canais a seguir. Criei um grupo no facebook como uma tentativa de ajudar a galera dos países da África que falam português (Angola, Cabo Verde, Guiné-Bissau, Moçambique, São Tomé e Príncipe e Guiné Equatorial) assim como nós brasileiros e por favor, não envie imagens, vídeos ou links externos porque a galera da África não tem um acesso à internet bom e o que tem é muito caro. É sério! Se alguém está com dúvida no grupo do facebook, por favor responda em texto e direto no facebook, ou simplesmente não responda, deixe para quem tem um bom coração.

- [Discord](#)
- [Facebook](#)
- [Telegram](#)

Como funciona o livro?

Cada capítulo é um dia de estudo, recomendo você não fazer mais de um capítulo por dia pois é importante deixar a mente respirar para você absorver o conhecimento de forma mais eficiente e feliz. Esse livro é totalmente prático, é composto só com exercícios. A **primeira parte** do exercício é chamado de **Tarefas**, que é apenas uma descrição do que deve ser feito. Se está assistindo as vídeo aulas ao vivo pela twitch.tv/marcobrunobr, recomendo você tentar fazer o exercício apenas com as informações passadas na **Tarefas**. Se após uns 5 minutos não conseguir executar o exercício ou em algum momento do exercício travar, você poderá consultar a terceira parte do exercício. A **segunda parte** é chamada de **Passo a passo**, aqui mostro uma das formas de você chegar no que foi solicitado na **Tarefas**. Se você finalizou o exercício seguindo apenas as instruções da **Tarefas** é provável que seu código estará diferente do apresentado no **Passo a passo** e essa é uma das belezas da programação, não existe um único caminho para solucionar um problema. Seja livre e crítico de forma construtiva com você e todos, não considere que o meu código é melhor que o seu, diferente não é pior. No final de toda capítulo tem os Desafios top top top, esses desafios é para você validar como está absorvendo o conhecimento das aulas.

Pronto! Agora vai para o seu Dia 1 e se ficar com alguma dúvida fique à vontade para entrar no [Discord](#), [Facebook](#) e [Telegram](#).

Onde acessar o layout do projeto?

O layout utilizado nesse projeto não foi criado pela comunidade CollabCode, infelizmente não consegui achar o dono dele, caso você consiga por favor não deixe de nos avisar para lhe dar os devidos créditos. Você pode acessar o layout no link a seguir do Figma, é necessário você realizar um cadastro no Figma para ter acesso completo ao layout:

https://www.figma.com/file/CywyY7njNGl6SfxbDnBOiW/404_error_page?node-id=0%3A25

Dia 1 - Onde tudo começa

Não precisa sofrer pra saber o que é melhor pra você

Parte da música [Não Existe Amor Em SP](#) de Criolo

Exercício 00 - Prepare o ambiente

Nesse exercício instalaremos tudo que precisamos para começar nosso estudos em **HTML**, **CSS** e **JavaScript**. Será instalado um editor de texto (*Visual Studio Code* - Estúdio visual de código), um browser (navegador) que será o Firefox Developer (navegador criado pela Mozilla para pessoas desenvolvedoras) e o Git (sistema de versionamento de código). O *Visual Studio Code* também conhecido como *VSCode* é o editor de texto mais usado no mercado de trabalho, Firefox Developer é um dos navegadores mais usados pelas pessoas desenvolvedoras e o Git é de longe o sistema de versionamento mais utilizado no mercado de trabalho.

Tarefas

Entre no sites a seguir e instale o VSCode, o Firefox Developer e o Git:

1. <https://code.visualstudio.com>
2. <https://www.mozilla.org/en-US/firefox/developer>
3. <https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Instalando-o-Git>

Passo a passo

1. Instalar o VSCode

Entre no site <https://code.visualstudio.com> faça o download (baixar um arquivo) para o SO (Sistema Operacional) que você está usando e em seguida instale-o, se tiver qualquer problema com a instalação não deixe de pedir ajuda em um dos canais da comunidade CollabCode:

- [Discord](#)
- [Facebook](#)
- [Telegram](#)

2. Instalar o Firefox Developer

Acesse o site <https://www.mozilla.org/en-US/firefox/developer> realize o download para o SO que você está usando, em seguida faça a instalação, tendo qualquer dúvida fique à vontade mandar suas dúvidas em um dos [canais da CollabCode](#).

3. Instalar Git

Acesso o eBook do Git no site <https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Instalando-o-Git> que lá terá todo processo de download e instalação para o SO que você estiver usando. Ficou com alguma dúvida não deixe de perguntar.

Exercício 01 - Inicie projeto

Tarefas

1. Pelo terminal, crie uma pasta com o nome **jokempo** (sem acentuação mesmo) no Desktop (área de trabalho) do seu SO ou em outro lugar que você lembrará que ela estará lá;
2. Entre na pasta do seu projeto usando o terminal e inicie o Git dentro dela;
3. Dentro da pasta **jokempo** crie um pasta **src** (abreviação de source que significa fonte).

Passo a passo

1. Crie pasta do projeto

Abra o terminal do seu SO, se estiver no Windows use o Power Shell, Git BASH ou similares para que o comando do Linux funcione. Se estiver no Linux ou Mac, ambos têm comandos muitos similares. Agora que você já está com o seu terminal aberto vamos executar os seguintes passos:

Entre no Desktop com o comando a seguir:

```
cd ~/Desktop
```

Criar uma pasta chamada **jokempo** dentro do Desktop:

```
mkdir jokempo
```

Acessar a pasta **jokempo**:

```
cd jokempo
```

Caso você tenha criado uma pasta com o nome errado você pode usar o comando `rm -r nomeDaPasta` para excluir a pasta e aí só precisa criá-la novamente ou ainda o comando `mv nomeDaPastaErrado nomeDaPastaCerto` para renomear.

2. Iniciar Git

No terminal e dentro da pasta **jokempo** execute o comando a seguir, esse comando criará uma pasta oculta no seu projeto chamada **.git** (toda pasta e arquivos que começam com um ponto são ocultas). Dentro desta pasta ficará tudo sobre o Git que é o sistema de versionamento de arquivos que vamos utilizar durante todo o projeto. Tenha calma se você não sabe o que é Git, teremos um dia para entrar mais a fundo no conhecimento sobre ele.

```
git init
```

Se tudo deu certo você receberá uma saída similar no seu terminal:

```
Initialized empty Git repository in /home/marcobruno/Desktop/jokempo/.git/
```

Está na dúvida se a pasta **.git** foi criada? Digite o comando a seguir que ele listará todos os arquivos e pasta que tem lá dentro:

```
ls -la
```

Terá uma resposta similar a esta no terminal:

```
total 12
drwxrwxr-x  3 marcobruno marcobruno 4096 Dec  9 06:38 .
drwxr-xr-x 10 marcobruno marcobruno 4096 Dec  9 06:38 ..
drwxrwxr-x  7 marcobruno marcobruno 4096 Dec  9 06:38 .git
```

3. Crie a pasta src:

No terminal e dentro da pasta **jokempo** crie a pasta **src**, nela ficarão todos códigos do nosso projeto:

```
mkdir src
```

Rode o comando a seguir e confirme se a pasta **src** foi criada com sucesso:

```
ls -la
```

Agora você terá a seguinte resposta no seu terminal:

```
total 16
drwxrwxr-x  4 marcobruno marcobruno 4096 Dec  9 06:48 .
drwxr-xr-x 10 marcobruno marcobruno 4096 Dec  9 06:38 ..
drwxrwxr-x  7 marcobruno marcobruno 4096 Dec  9 06:48 .git
drwxrwxr-x  2 marcobruno marcobruno 4096 Dec  9 06:48 src
```

Dúvidas, não deixe de perguntar em um dos [canais da CollabCode](#).

Exercício 02 - Crie o HTML do cabeçalho

Nesse exercício vamos entender o que é **HTML**, quais suas responsabilidades e o que é uma tag.

Tarefas

1. Abrir o VSCode pelo terminal na pasta do projeto;
2. Crie um arquivo **404.html** dentro da pasta **src** que criamos a partir da **jokempo**;
3. Instale a extensão Live Server no VSCode;
4. Dentro do arquivo **404.html** crie o HTML do cabeçalho.

Acesse o layout do projeto em:

https://www.figma.com/file/CywyY7njNGI6SfxbDnBOiW/404_error_page?node-id=0%3A25

Passo a passo

1. Acesse o layout

Primeiro acesse o layout do projeto em:

https://www.figma.com/file/CywyY7njNGI6SfxbDnBOiW/404_error_page?node-id=0%3A25. Para ter acesso completo ao layout é necessário você realizar um cadastro no Figma.

Figma é uma ferramenta para você criar o layout de um site, aplicativo mobile ou sistema web. É umas das ferramentas mais utilizadas pelos Designers, as outras são Sketch e Adobe XD.

2. Abra o VSCode

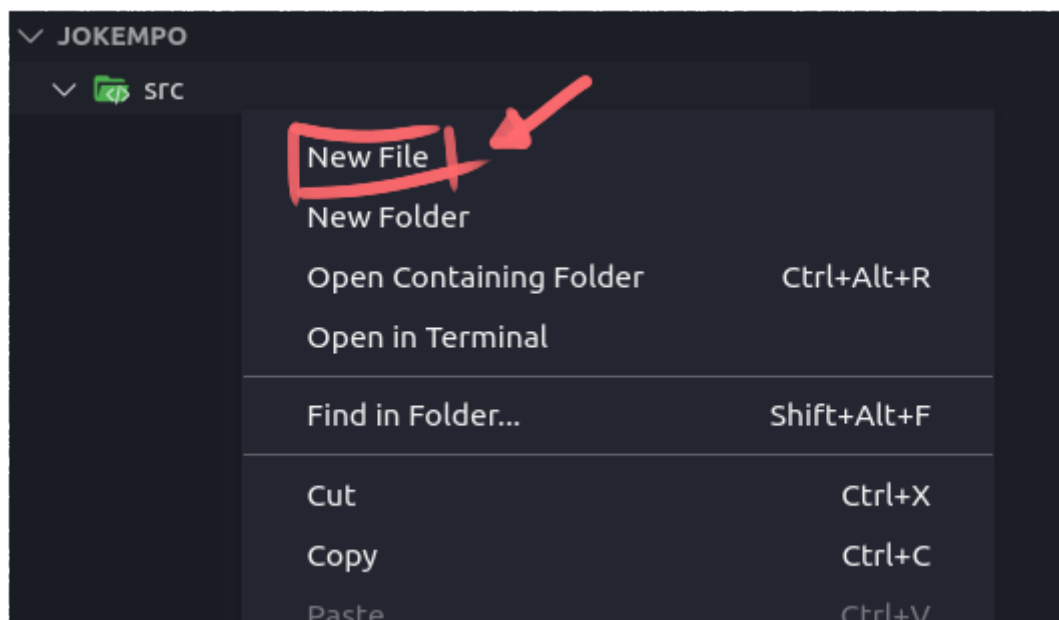
No terminal já dentro da pasta **jokempo** abra o VSCode com o comando a seguir, esse comando também já nos deixará dentro da pasta do nosso projeto:

```
code .
```

O comando **code** abrirá o VSCode e a seguir dele colamos um ponto, isso significa que queremos abrir o VSCode a partir da pasta que estamos no terminal, no nosso caso a **jokempo**.

3. Crie o arquivo 404.html

Dentro da pasta **src** que está localizada em **jokempo** crie um arquivo **404.html**, faça isso usando o VSCode. Dentro dele no lado esquerdo tem a árvore de pastas e arquivos do nosso projeto (Explorer), clique com o botão direito do mouse sobre a pasta **src**, depois clique em *New File* (Novo Arquivo).



4. Faça HTML do cabeçalho

Entre no [layout do projeto](#) e implemente o HTML do cabeçalho do projeto.

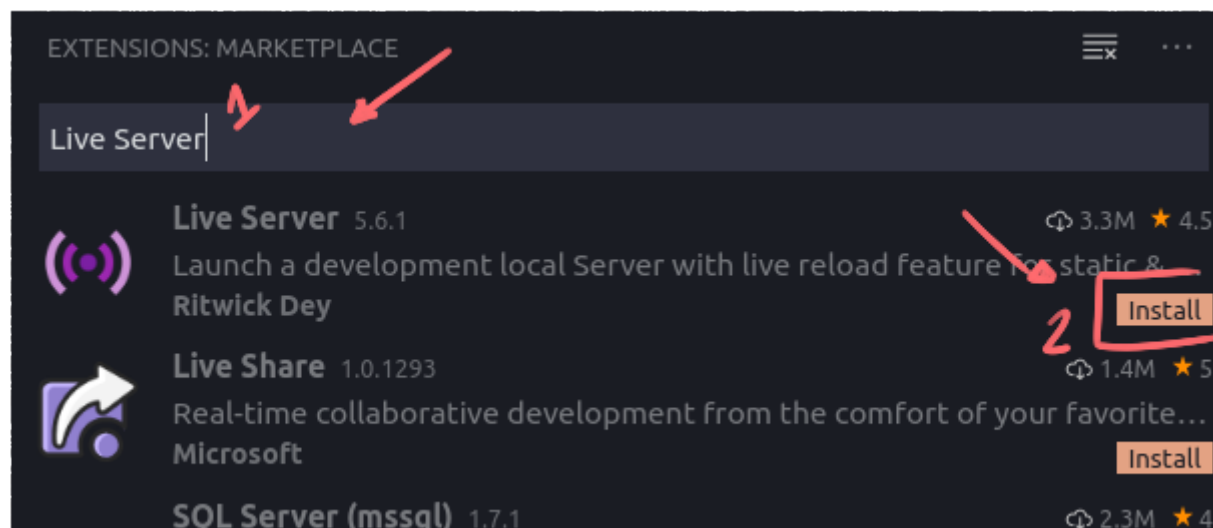


O HTML é uma linguagem de marcação que tem a responsabilidade de cuidar do conteúdo para o usuário, portanto todo texto, imagem, áudio, vídeo ou qualquer outra representação de conteúdo deverá ficar sobre a responsabilidade do HTML.

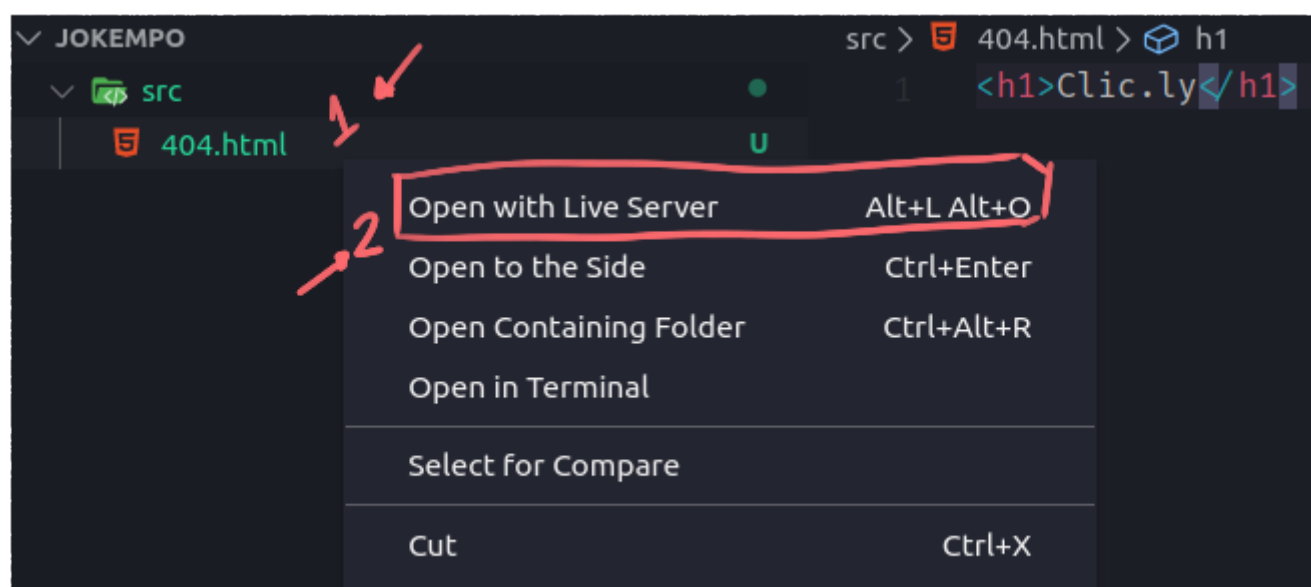
Abra o arquivo **404.html** e dentro dele crie o título **Clic.ly** usando a tag de título de primeira importância **<h1>**:

```
<h1>Clic.ly</h1>
```

Agora precisamos ver o resultado do que acabamos de criar no navegador, para facilitar esse processo vamos instalar uma extensão no VSCode chamada **Live Server** (Servidor ao vivo). Para instalar esta extensão, aperte as teclas *ctrl*, *shift*, *X* juntas (**ctrl + shift + X**), no lugar da nossa árvore de arquivo aparecerá a loja de extensões do VSCode (é loja mas não vamos pagar nada), busque por **Live Server**, depois clique no botão *install* (instalar). Não é necessário reiniciar o VSCode para que o Live Server funcione.



Para usar o Live Server, volte para o Explorer (árvore dos arquivos) e clique com o botão direito do mouse sobre o arquivo **404.html** e depois na opção *Open with Live Server* (Abrir com o Live Server):



Após alguns segundos, automaticamente será aberto o navegador que você definiu como padrão no seu SO. Além disso, toda vez que você salvar o arquivo **404.html** com uma alteração o Live Server também vai atualizar automaticamente o seu navegador. Se tudo deu certo você terá um resultado similar ao a seguir no seu navegador:

Clic.ly

Na linha a seguir do nosso título **Clic.ly** vamos implementar o nosso menu de navegação principal que tem 4 itens: Home (página inicial); Pricing (precificação); About Us (sobre nós) e Log in (entrar). Cada item ficará dentro de uma tag `<a>` que utilizamos para representar um link de navegação interna ou externa. Feito o código não esqueça de salvar.

`<a>` é abreviação de anchor (âncora), é estranho para momento atual, mas no começo da internet usamos o termo **navegar na web** quando estamos utilizando ela.

```
<h1>Clic.ly</h1>

<a href="">Home</a>
<a href="">Pricing</a>
<a href="">About Us</a>
<a href="">Log in</a>
```

Visite seu navegador e verá que o Live Server já atualizou e desta forma você já pode ver o resultado visual do seu código:

Clic.ly

[Home](#) [Pricing](#) [About Us](#) [Log in](#)

Agora precisamos informar para o navegador que os 4 links que acabamos de colocar, juntos são a navegação do site, para isso utilize a tag `<nav>` (*navitagion* - navegação) envolvendo os 4 links:

```
<h1>Clic.ly</h1>

<nav>
  <a href="">Home</a>
  <a href="">Pricing</a>
  <a href="">About Us</a>
  <a href="">Log in</a>
</nav>
```

Se for até o navegador verá que nada foi alterado visualmente, mas a tag `<nav>` está lá. Essa é uma tag de estrutura, ela não tem impacto visual, ela ajuda a deixar o seu site mais semântico e dessa forma ajuda para acessibilidade e SEO (*Search Engine Optimization* - Otimização para motores de busca).

Criar um HTML semântico, significa que você está usando cada tag com o propósito pelo qual ela foi criada. Dessa forma você torna o seu site mais acessível e te ajudará com o SEO.

Quando o seu site tem um bom SEO (*Search Engine Optimization* - Otimização para motores de busca), significa que você trabalhou para deixar ele otimizado para buscadores como o Google. Para aparecer em primeiro na busca orgânica do Google não basta somente ter um HTML semântico mas isso ajuda bastante.

Acessibilidade é obrigatório nos sites, tem leis sobre isso e se não seguir os padrões você poderá ser processado, recomendo seguir o [checklist de acessibilidade para desenvolvedores](#) criado pelo governo brasileiro, se não tiver no Brasil procure ver as leis do seu país.

Bora melhorar ainda mais a semântica do nosso cabeçalho usando a tag `<header>` (cabeçalho) que deve envolver o nosso título e a navegação:

```
<header>
  <h1>Clic.ly</h1>

  <nav>
    <a href="">Home</a>
    <a href="">Pricing</a>
    <a href="">About Us</a>
    <a href="">Log in</a>
  </nav>
</header>
```

Se visitar novamente o navegador, verá que não teve muito impacto visual, mas novamente deixamos o nosso site mais acessível e com um SEO mais feliz, trabalhando uma melhor semântica no HTML.

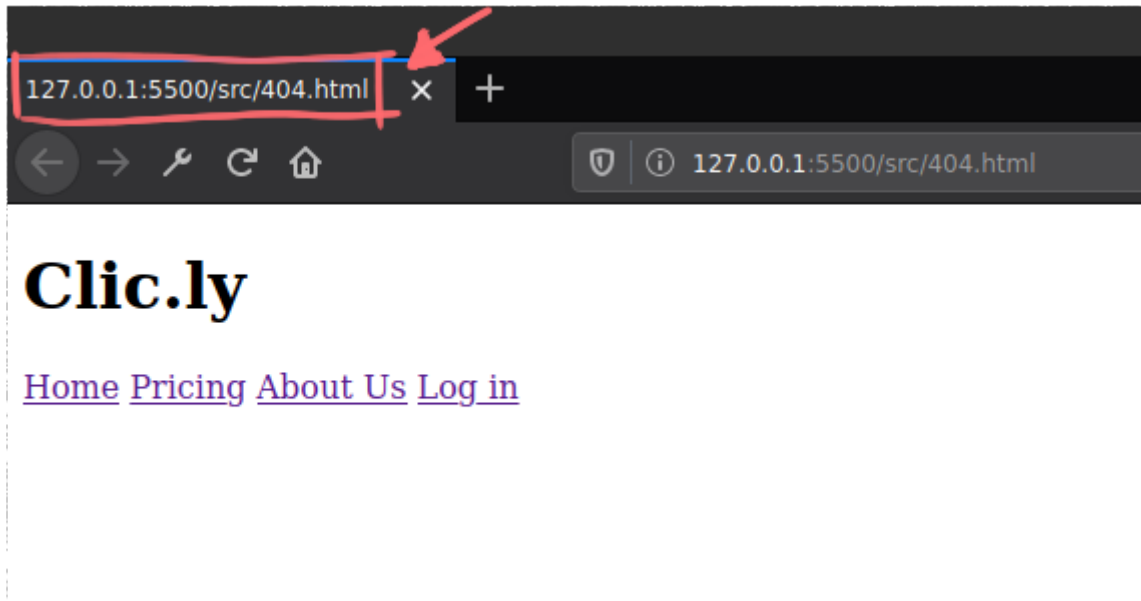
Pronto! Entregamos o HTML do nosso cabeçalho. Se ficou alguma dúvida, só tirar em algum dos [canais da CollabCode](#).

Exercício 03 - Estruture o HTML

Nossa página já está funcionando, mas podemos deixá-la mais organizada com algumas tags de estrutura e configuração.

Tarefas

1. Alterar o título da aba do navegador para *Clic.ly - Not found page* (Clic.ly - página não existe)



2. Fazer com que letras com ê, ç e ã funcione em todos os navegador;
3. Separar o que é configuração do que é conteúdo;
4. Informar a versão do HTML para o navegador.

Passo a passo

1. Mude conteúdo da aba do navegador

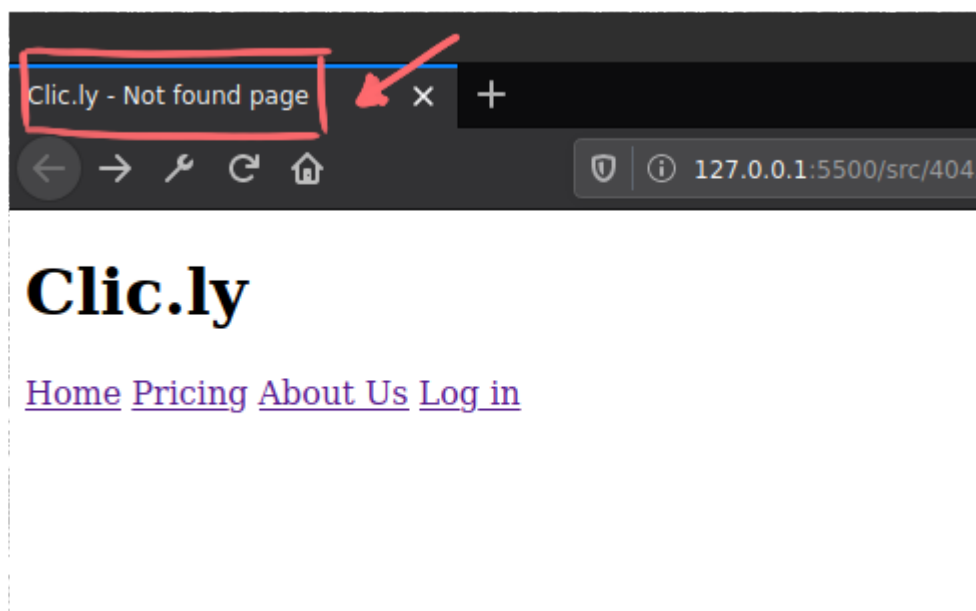
Para adicionar o conteúdo *Clic.ly - Not found page* como conteúdo da aba do navegador, use a tag `<title>` antes do tag `<header>`:

```
<title>Clic.ly - Not found page</title>

<header>
  <h1>Clic.ly</h1>

  <nav>
    <a href="">Home</a>
    <a href="">Pricing</a>
    <a href="">About Us</a>
    <a href="">Log in</a>
  </nav>
</header>
```

Salve e visite o navegador que terá um resultado semelhante a este:



2. Faça funcionar o ç e letras com acentuação

Para as letras ê, ç e ã funcionarem, use a tag `<meta>` com o atributo `charset` e passe para ele o valor `utf-8`. Essa tag ficará antes da tag `<title>`:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<header>
  <h1>Clic.ly</h1>

  <nav>
    <a href="">Home</a>
    <a href="">Pricing</a>
    <a href="">About Us</a>
    <a href="">Log in</a>
  </nav>
</header>
```

`utf-8` é uma tabela de caracteres que suporte a maior parte dos idiomas do mundo, então caso você precise fazer um site que funcione em português e japonês ela dará conta.

A tag `<meta>` é bem genérica, você usará ela em outros contextos mas agora utilizamos para definir uma tabela de caracteres com o atributo `charset` (conjunto de caracteres).

3. Separe a configuração do conteúdo

Tem duas grandes categorias de tags no código, uma que é a configuração e o navegador fica com o controle dela e a segunda é o conteúdo e você tem mais controle. As tags de configuração (`<meta>` e

`<title>`) ficarão dentro da tag `<head>` (cabeça) e a de conteúdo (`<header>` e tudo que está dentro dela) ficarão dentro da tag `<body>` (corpo):

```
<head>
  <meta charset="utf-8">
  <title>Clic.ly - Not found page</title>
</head>
<body>
  <header>
    <h1>Clic.ly</h1>

    <nav>
      <a href="">Home</a>
      <a href="">Pricing</a>
      <a href="">About Us</a>
      <a href="">Log in</a>
    </nav>
  </header>
</body>
```

Daqui para frente vamos respeitar essas duas grandes categorias do HTML, tudo que for conteúdo para o usuário deverá ficar dentro da tag `<body>` e o que for configuração deverá ficar dentro da tag `<head>`.

Outra tag de estrutura é a tag `<html>` que envolve tanto a tag `<head>` quando a tag `<body>`, cole ela também no seu código:

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Clic.ly - Not found page</title>
  </head>
  <body>
    <header>
      <h1>Clic.ly</h1>

      <nav>
        <a href="">Home</a>
        <a href="">Pricing</a>
        <a href="">About Us</a>
        <a href="">Log in</a>
      </nav>
    </header>
  </body>
</html>
```

4. Informe a versão do HTML

Tivemos algumas versão do HTML, as três últimas foram **HTML4**, depois **XHTML** e hoje estamos na **HTML5**. Para pedir ao navegador para utilizar o **HTML5**, coloque na primeira linha do **404.html** `<!DOCTYPE html>`:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Clic.ly - Not found page</title>
  </head>
  <body>
    <header>
      <h1>Clic.ly</h1>

      <nav>
        <a href="">Home</a>
        <a href="">Pricing</a>
        <a href="">About Us</a>
        <a href="">Log in</a>
      </nav>
    </header>
  </body>
</html>
```

`<!DOCTYPE html>` não significa apenas que é pro navegador usar o HTML5, na verdade estamos falando para ele usar a última versão estável do HTML, se amanhã for o HTML6 não precisaremos mais mudar o `<!DOCTYPE html>` o próprio navegador dará conta do recado.

Exercício 04 - Projeto no GitHub

Se você quiser compartilhar o código do seu projeto com outros desenvolvedores ou guardar o código por segurança, é recomendado salvá-lo em algum lugar que não seja sua máquina. GitHub é o maior serviço de hospedagem de códigos do mundo que usa o Git como sistema de versionamento. Projetos *opensource* (código aberto) como React, Svelte, Angular, Vue e muitos outros estão no GitHub além da maior parte das empresas usarem ele, por esses motivos vamos usá-lo também. Tem muitas outras vantagens em usar o Git e GitHub, mas vamos conhecendo outras com o decorrer do projeto.

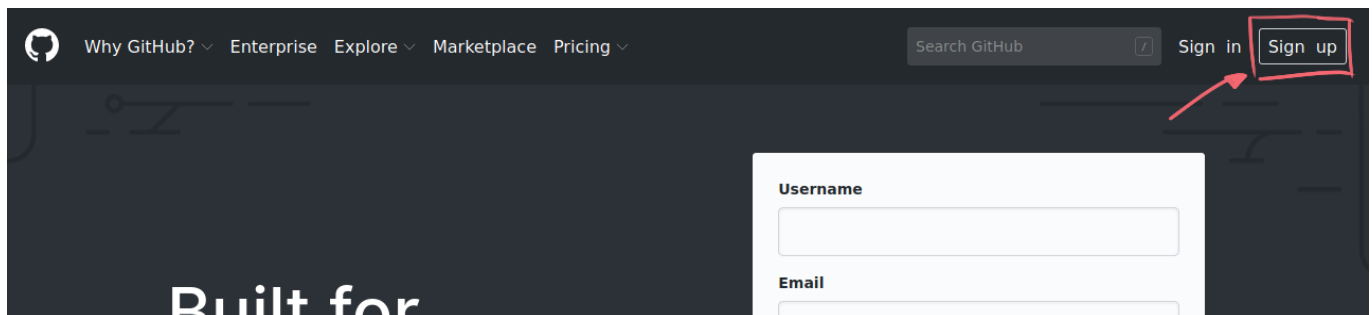
Tarefas

1. Criar uma conta no GitHub;
2. Criar um repositório do projeto;
3. Criar e adicionar uma SSH key no GitHub;
4. Associar projeto local com o remoto do GitHub;
5. Configurar o git localmente;
6. Adicionar arquivos para serem enviados para o repositório;
7. Guardar estado do código e enviar para o GitHub;
8. Enviar o código para o repositório remoto do GitHub.

Passo a passo

1. Crie conta no GitHub

Entre no site <https://github.com> clique no link *Sign up* (inscrever-se):



Preencha os campos *Username* (nome do usuário), *Email address* (Endereço de email), *Password* (senha), resolva o puzzle (enigma) e por último clique no botão azul com o texto *Next: Select a plan*:

Join GitHub

Create your account

Username *

marcobruno-exemplo



1

Email address *

marcobrunobra@gmail.com



2

Password *

.....

3

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Email preferences☐ Send me occasional product updates, announcements, and offers.**Verify your account**

Por favor, resolva este puzzle para
que possamos saber que é uma
pessoa real

[Verificar](#)

4

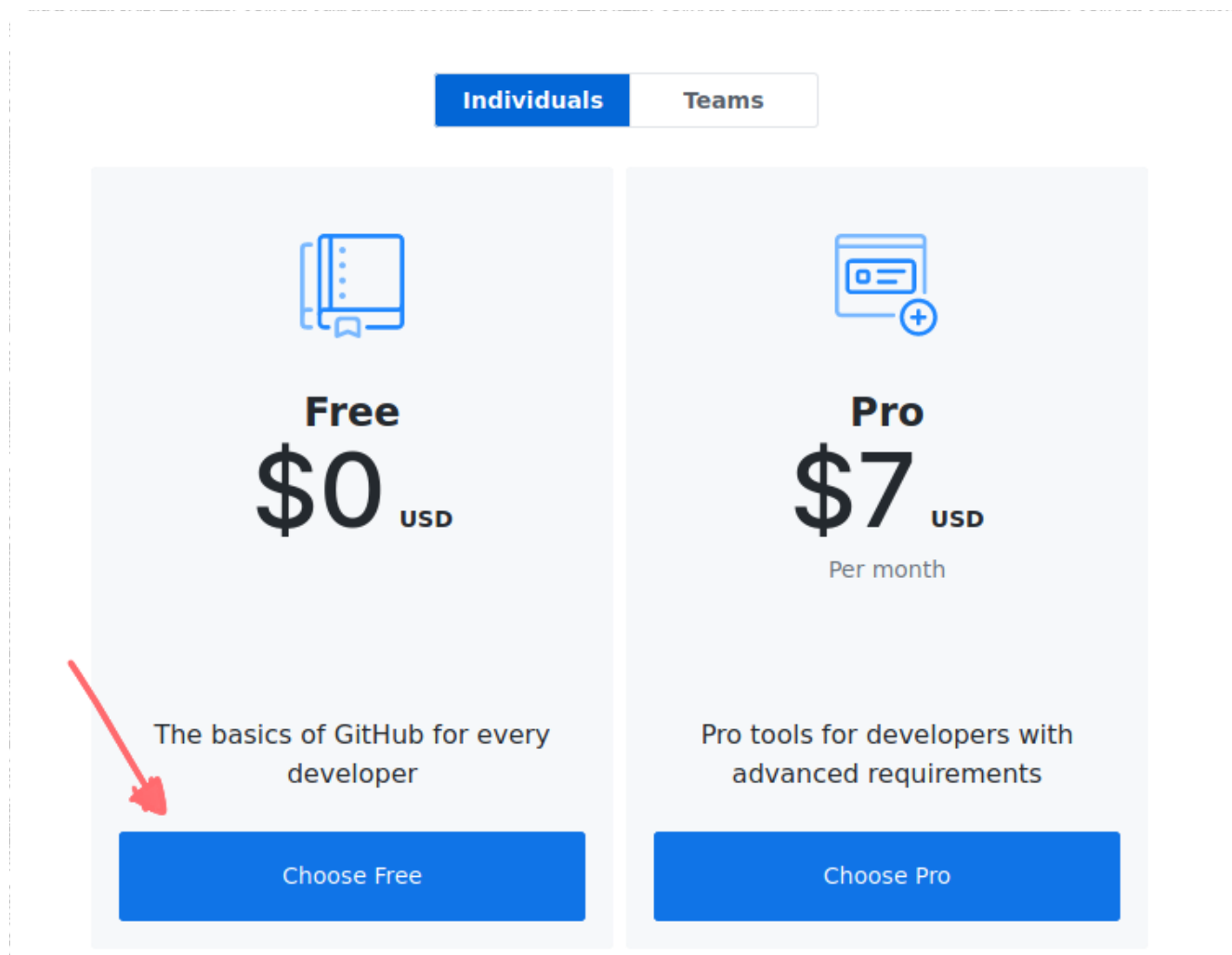
[Next: Select a plan](#)

5

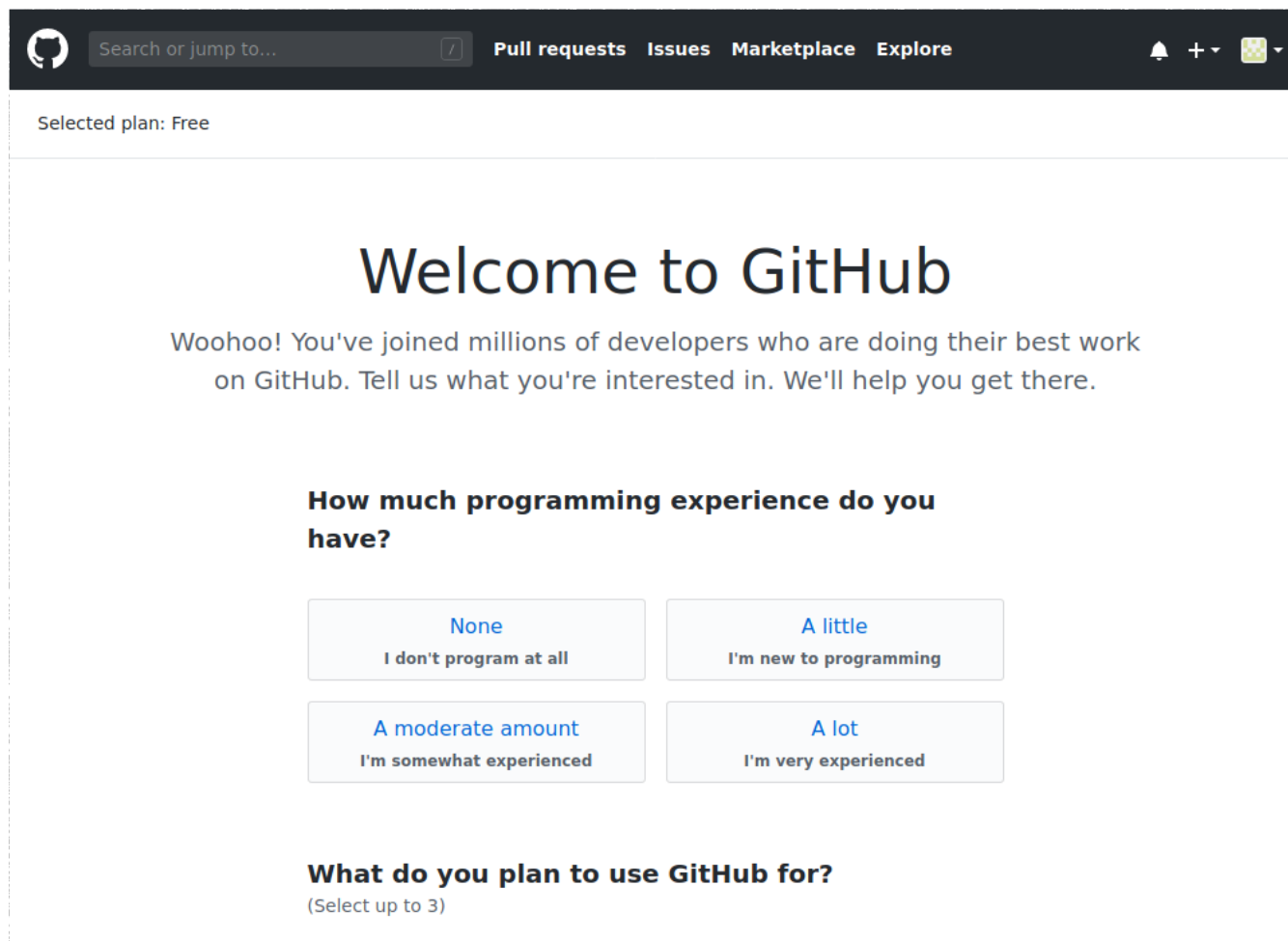
By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

Depois você terá que escolher o plano do GitHub, não é necessário contratar nenhum plano no começo dos estudos, então escolha a opção free.

Obs. Se não aparecer a tela dos planos, só clique no logo do GitHub que fica no topo do lado direito para sair da tela que você está, fique tranquilo se entrou nessa tela o próprio GitHub definiu que sua conta é a versão free.



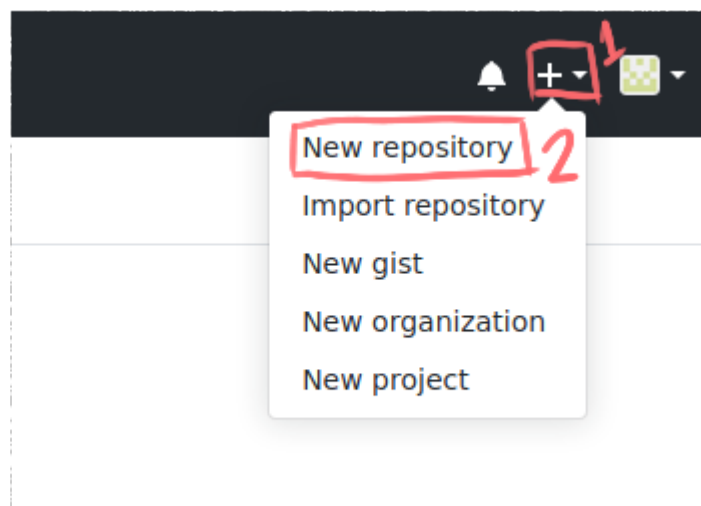
Se tudo der certo você verá a tela de boas-vindas do GitHub:



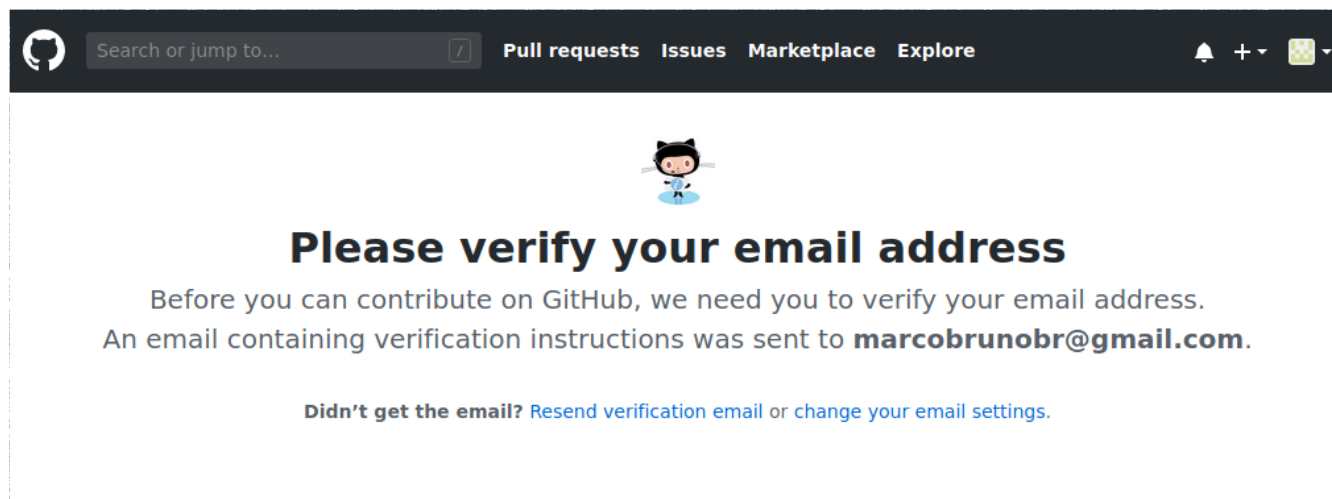
The screenshot shows the GitHub welcome page. At the top, there's a dark header with the GitHub logo, a search bar, and navigation links: Pull requests, Issues, Marketplace, and Explore. Below the header, it says "Selected plan: Free". The main content area has a large heading "Welcome to GitHub" followed by a message: "Woohoo! You've joined millions of developers who are doing their best work on GitHub. Tell us what you're interested in. We'll help you get there." Below this is a section titled "How much programming experience do you have?" with four buttons: "None" (I don't program at all), "A little" (I'm new to programming), "A moderate amount" (I'm somewhat experienced), and "A lot" (I'm very experienced). At the bottom, there's a section titled "What do you plan to use GitHub for?" with a note "(Select up to 3)".

2. Crie um repositório

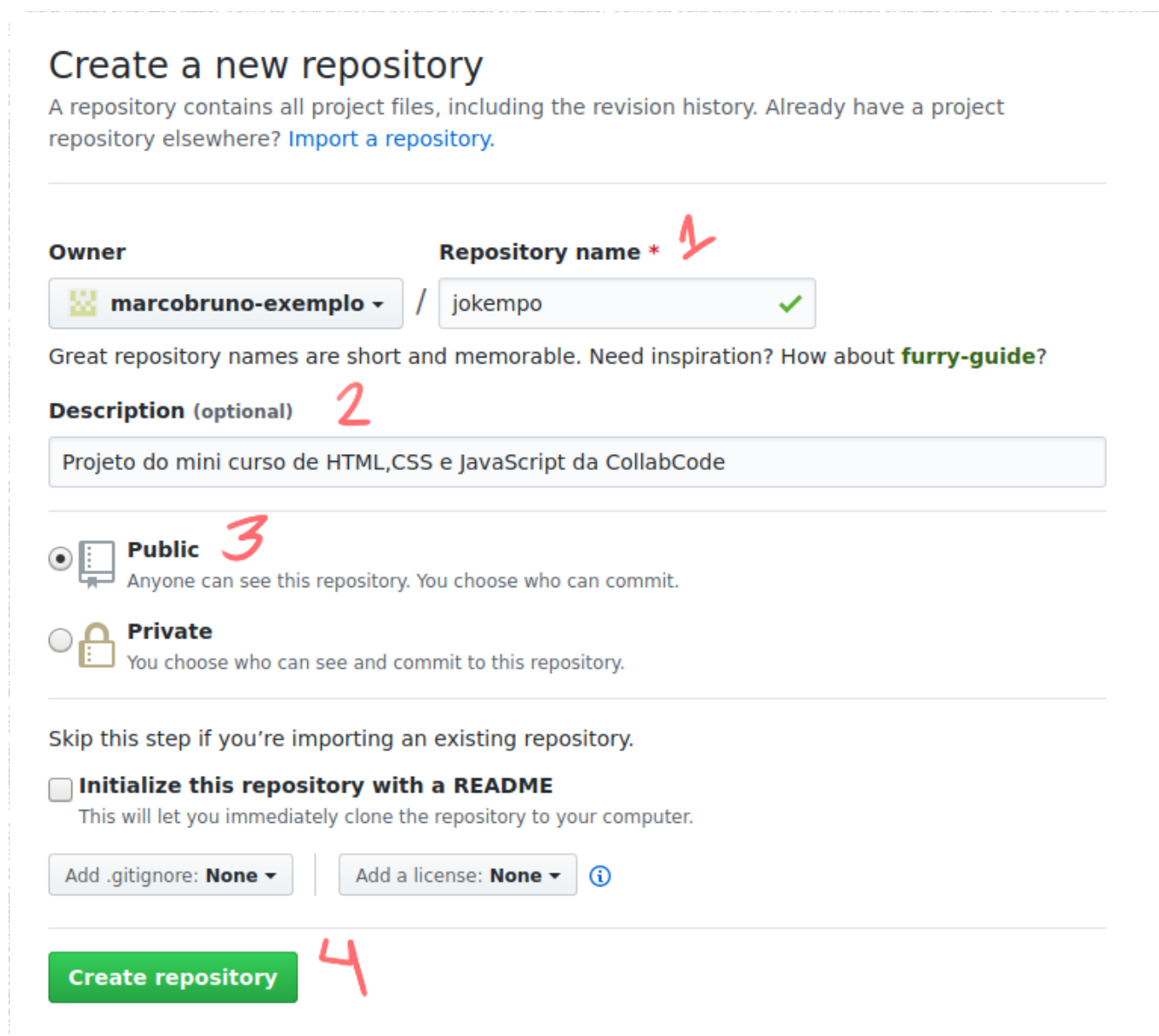
No lado direito e no topo da tela de boas-vindas tem um sinal de **mais** (+), clique nele e depois na opção *New repository* (Novo repositório):



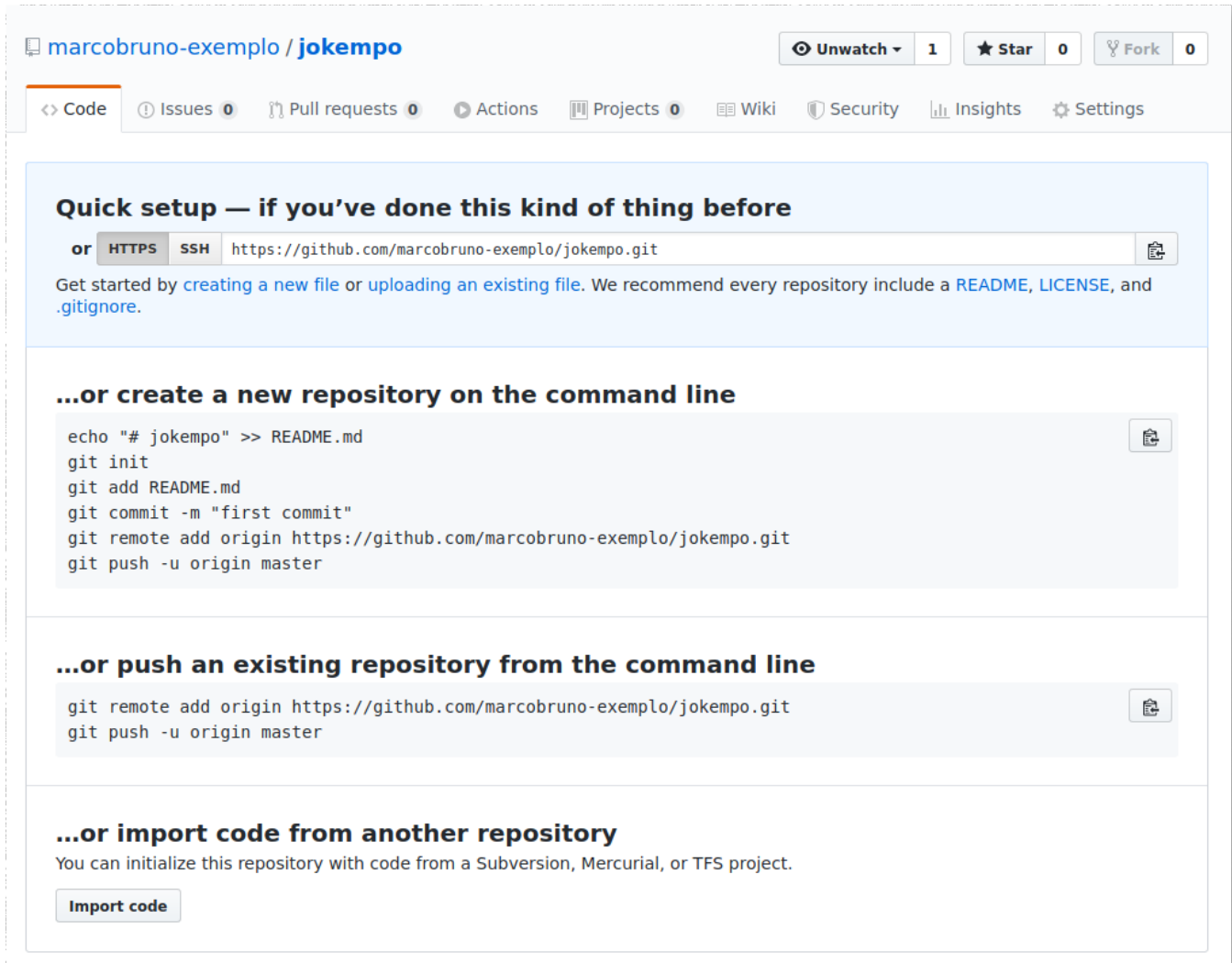
Se você cair na tela a seguir, primeiro precisa confirmar seu email e só depois poderá criar um repositório:



Agora que seu email já está confirmado, precisa informar *Repository name* (Nome do repositório), *Description* (Descrição), definir com *Public* (Público) e depois clicar no botão verde com o texto *Create repository* (Criar repositório):



Se chegou na tela a seguir, significa que conseguiu criar um repositório remoto para o seu projeto:



The screenshot shows the GitHub repository page for 'marco-bruno-exemplo / jokempo'. At the top, there are buttons for 'Unwatch', 'Star' (1), and 'Fork' (0). Below the repository name, there are tabs for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The main content area has a light blue header with the text 'Quick setup — if you've done this kind of thing before'. Below this, there are two tabs: 'HTTPS' and 'SSH'. The 'SSH' tab is selected, and the URL 'https://github.com/marco-bruno-exemplo/jokempo.git' is displayed. Below the URL, there is a text box with the text 'Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).' Below this, there is a section titled '...or create a new repository on the command line' with a code block containing the following commands:

```
echo "# jokempo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/marco-bruno-exemplo/jokempo.git
git push -u origin master
```

 Below this, there is a section titled '...or push an existing repository from the command line' with a code block containing the following commands:

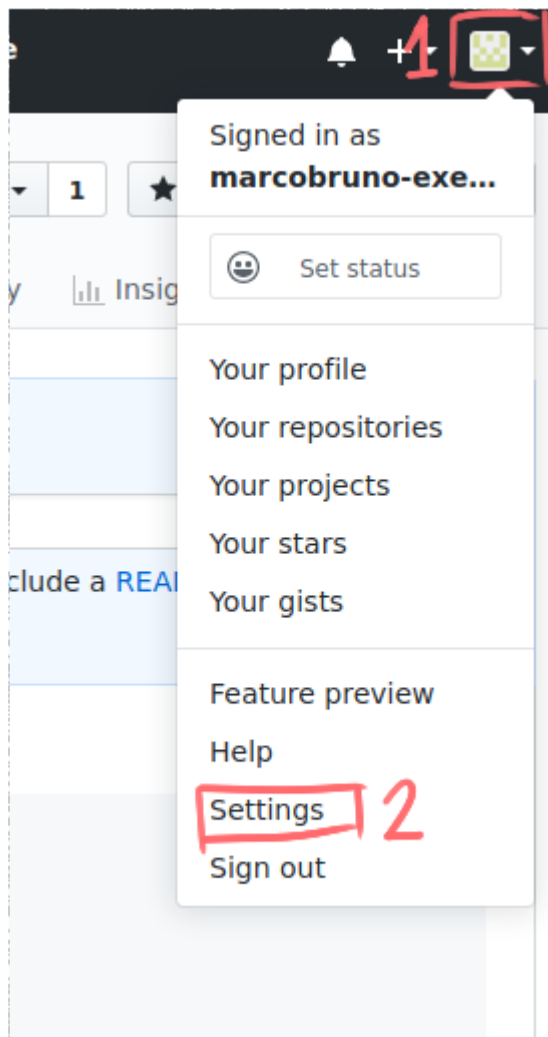
```
git remote add origin https://github.com/marco-bruno-exemplo/jokempo.git
git push -u origin master
```

 Below this, there is a section titled '...or import code from another repository' with the text 'You can initialize this repository with code from a Subversion, Mercurial, or TFS project.' and a button labeled 'Import code'.

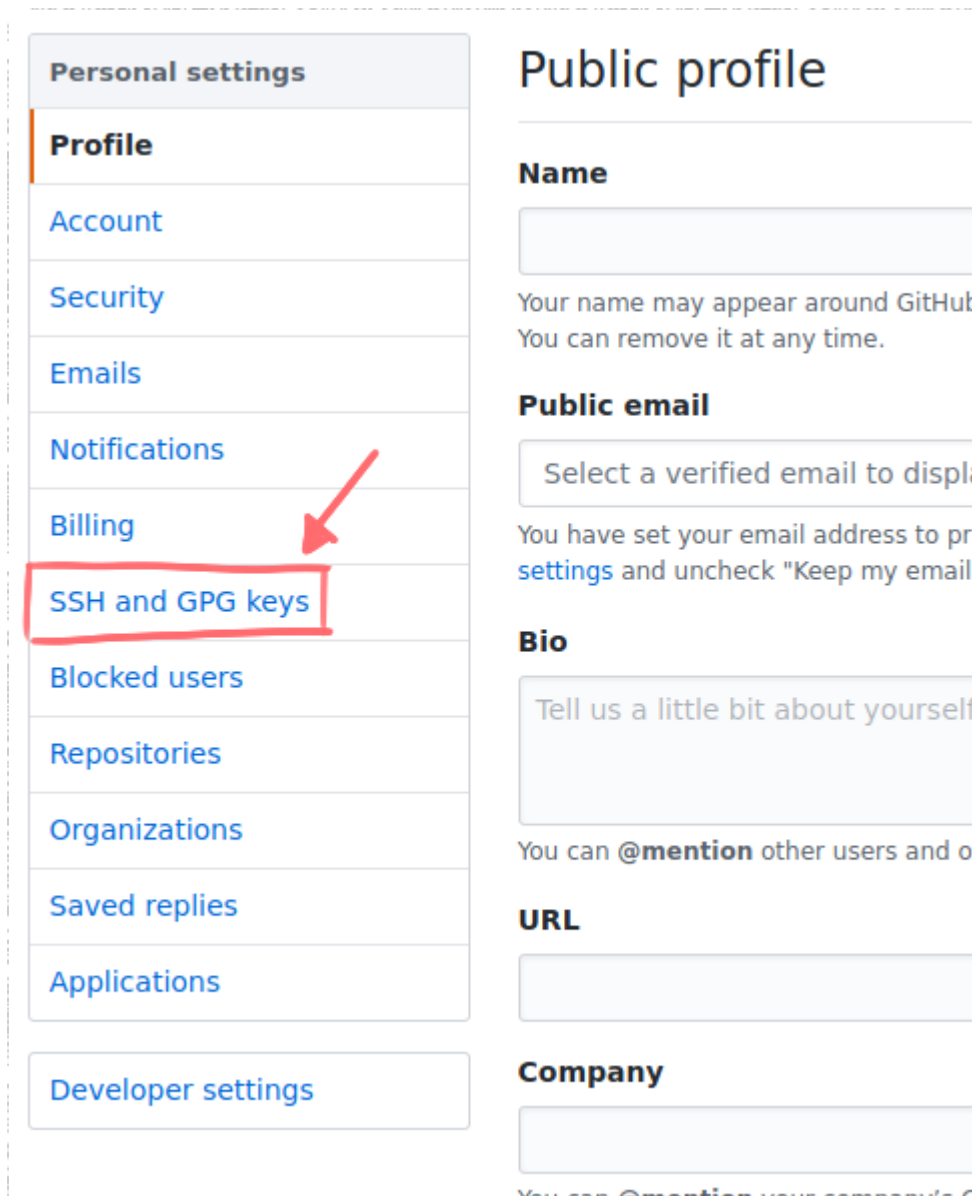
3. Crie e adicione a chave SSH no GitHub

Agora é possível associar o repositório local (pasta do nosso projeto que rodamos o comando `git init`) com o repositório remoto (o que criamos agora no GitHub). Tem duas formas de criar essa associação, **https** e **ssh**, a mais usada é a segunda por ser mais segura e também pela facilidade no dia a dia já que com o **ssh** não é necessário ficar digitando usuário e senha toda vez que quiser atualizar o repositório remoto, em vez da senha o ssh envia para o GitHub um *SSH key* (chave que você cria e é segura e criptografada).

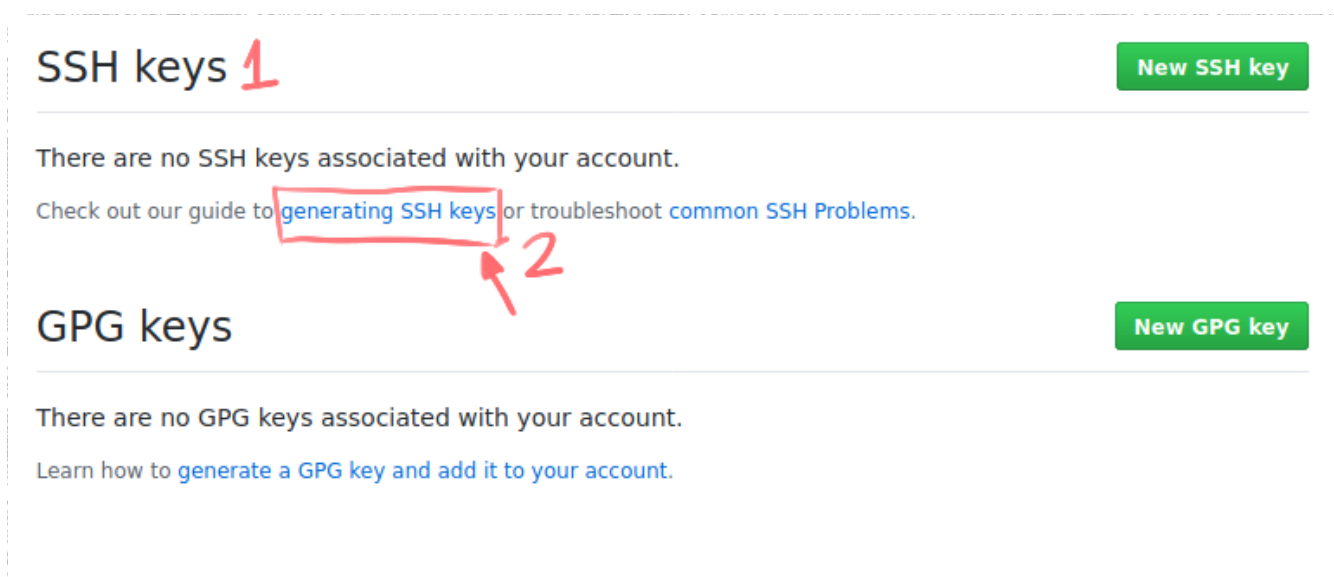
É exatamente a chave que você criará agora então volte no GitHub. Na parte superior direita da tela, ao lado direito do sinal de **mais** tem um emoji não muito bonito (que é onde deveria aparecer a sua foto), clique nele e depois em *Settings* (configurações):



Agora você está nas configurações do seu usuário no GitHub, no lado esquerdo tem algumas opções, clique na *SSH and GPG keys* (chaves SSH e GPG):



Ao lado direito do menu de navegação apareceu duas seções, a primeira se chama *SSH keys*, nela tem um link com o texto *generating SSH keys* (gerando chaves SSH) clique nele:



Você será redirecionado para uma nova página e nela terá o link com o texto *Generating a new SSH key and adding it to the ssh-agent* (Gerando uma nova chave SSH e adicionando-a ao ssh-agent), clique nele:

[GitHub.com](#) / [Authentication](#) / [Connecting to GitHub with SSH](#)

Connecting to GitHub with SSH

You can connect to GitHub using SSH.

About SSH

Using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to GitHub without supplying your username or password at each visit.

Checking for existing SSH keys

Before you generate an SSH key, you can check to see if you have any existing SSH keys.

Generating a new SSH key and adding it to the ssh-agent

After you've checked for existing SSH keys, you can generate a new SSH key to use for authentication, then add it to the ssh-agent.

Adding a new SSH key to your GitHub account

To configure your GitHub account to use your new (or existing) SSH key, you'll also need to add it to your GitHub account.

Na nova página terá o processo para você criar a sua chave SSH pelo terminal, primeiro escolha o SO que você está e depois siga o processo, se tiver dificuldade com inglês como eu, abra o translate e não deixe de aprender. Faça apenas o processo da seção *Generating a new SSH key* (Gerando uma nova chave SSH).

Mac

Windows

Linux

All

1

If you don't already have an SSH key, you must [generate a new SSH key](#). If you're unsure whether you already have an SSH key, check for [existing keys](#).

If you don't want to reenter your passphrase every time you use your SSH key, you can [add your key to the SSH agent](#), which manages your SSH keys and remembers your passphrase.

Generating a new SSH key

- 1 Open Terminal.
- 2 Paste the text below, substituting in your GitHub email address.

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private rsa key pair.
```

- 3 When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

```
> Enter a file in which to save the key (/home/you/.ssh/id_rsa): [Press enter]
```

- 4 At the prompt, type a secure passphrase. For more information, see "[Working with SSH key passphrases](#)".

```
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

Você acabou de gerar a chave pública e privada para o GitHub, no terminal rode o comando a seguir que exibirá o conteúdo da chave pública no terminal:

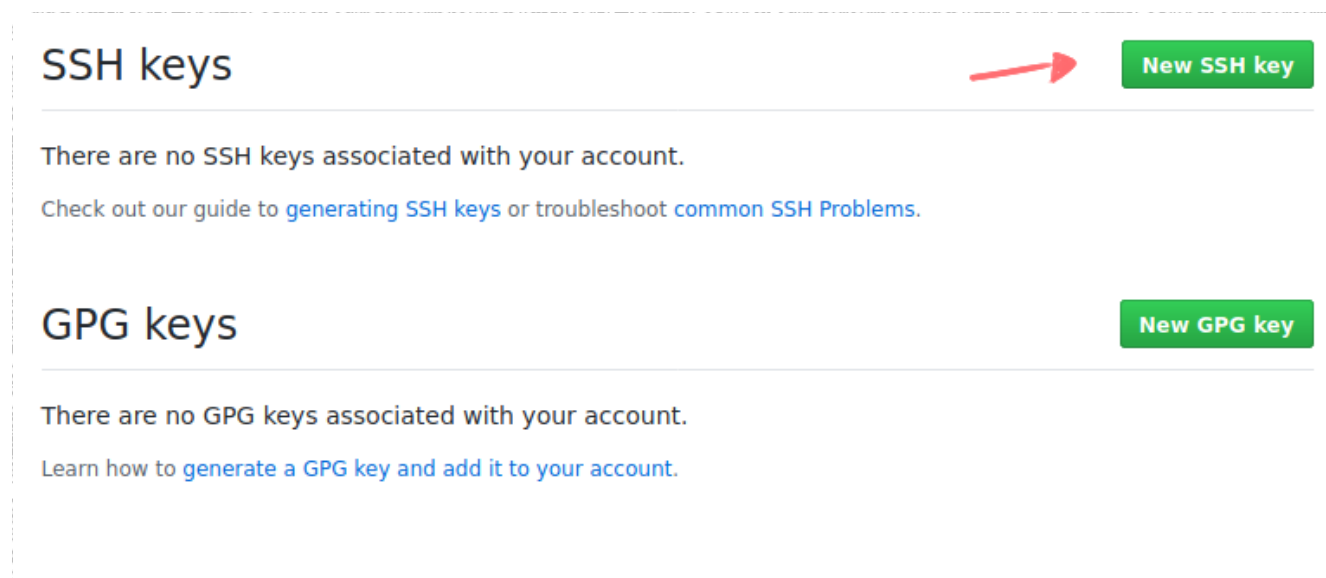
```
cat ~/.ssh/id_rsa.pub
```

Copie a saída que apareceu no terminal que será similar a que está a seguir:

```
ssh-rsa
KSAJDFHAKSJHuHfBaNCsa/zd10SFAR90qInetBapQF3oVeInJdpGk/n8zqHHR2HJKYfpxiMKPLV
D9sicL5IwLIUhuMWLDjDhFIH0ls9z2ofdxt6vQgrHz+teLraltn4yKQhe6vdiLVKar90+dklfaj
hf1kjheuna8dsiufnbwebasdMTMzQFeeJb0cQejmMIEvsKa0mgKFJ/sadhfaishfisaufnueee
eakfj;lda2gNtAyY9jy48Hoba6mtyf3SGSRs6AmKYluIAe4+xAV7Iqf2z/wGBTGDL+HGnj8IXpe
D8LAGc4Ye0Uh1yaw0qan9S9lm6Ci8+g3/
XR08m+KBoEzofl43q0ezEi/IbaLSpVAjadlc5w8KpZS/Hr4dK+1MXD/AWB1TE67R+sJ+eDk4msA
X0yK0fE12a/BgBMt1G/+hGf6s3DSgsw== marco.bruno.br@gmail.com
```

Importante: Você deve copiar o conteúdo inteiro que começa com **ssh-rsa** e termina com **seu e-mail**!

Volte nas configurações do GitHub na parte do *SSH Keys* e clique no botão verde que está no lado direito com o texto *New SSH key* (Nova chave SSH):



Preencha o *Title* (Título) da sua chave SSH, depois cole-a no campo *Key* (Chave) e para finalizar clique no botão verde com o texto *Add SSH Keys* (Adicionar chave SSH):

SSH keys / Add new

Title

Notebook 1

Key 2


```
ssh-rsa KSAIDFHAKSIHuHtBdNESz/zh1CSRAq9yqlaeeBmpQF3mVhIgjJpGk
/n8zqHHR2HJKYfpxiMKPLVD9sicL5IwLIUhuMWLDjDhFIHOls9z2ofdxt6vQgrHz+teLraltn4yKQhe6vdiLVKa
r9O+dklfajhflkjheuna8dsiufnbwebasdMTMzQFeejBocQejmMIEvsKa0mgKFJ
/sadhfaisuhfisaufnueeeekfj;lda2gNtAyY9jy48Hoba6mtyf3SGSRs6AmKYlulAe4+xAV7Iqf2z
/wGBTGDL+HGnj8IXpeD8LAGc4YeOUh1yawOqan9S9lm6Ci8+g3/
XRO8m+KBoEzofl43qOezEi/lbaLSpVAjadic5w8KpZS/Hr4dK+1MXD
/AWB1TE67R+sJ+eDk4msAX0yKOfEl2a/BqBMtIG/+hGf6s3DSgsw== marco.bruno.br@gmail.com
```

Add SSH key 3

Se tudo deu certo, você verá uma tela similar a essa:

SSH keys New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.


SSH

Note
5f:3d:76:9e:2e:ff:08:15:82:03:b5:f9:12:d1:d1:25
Added on 9 Dec 2019
Never used — Read/write

Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

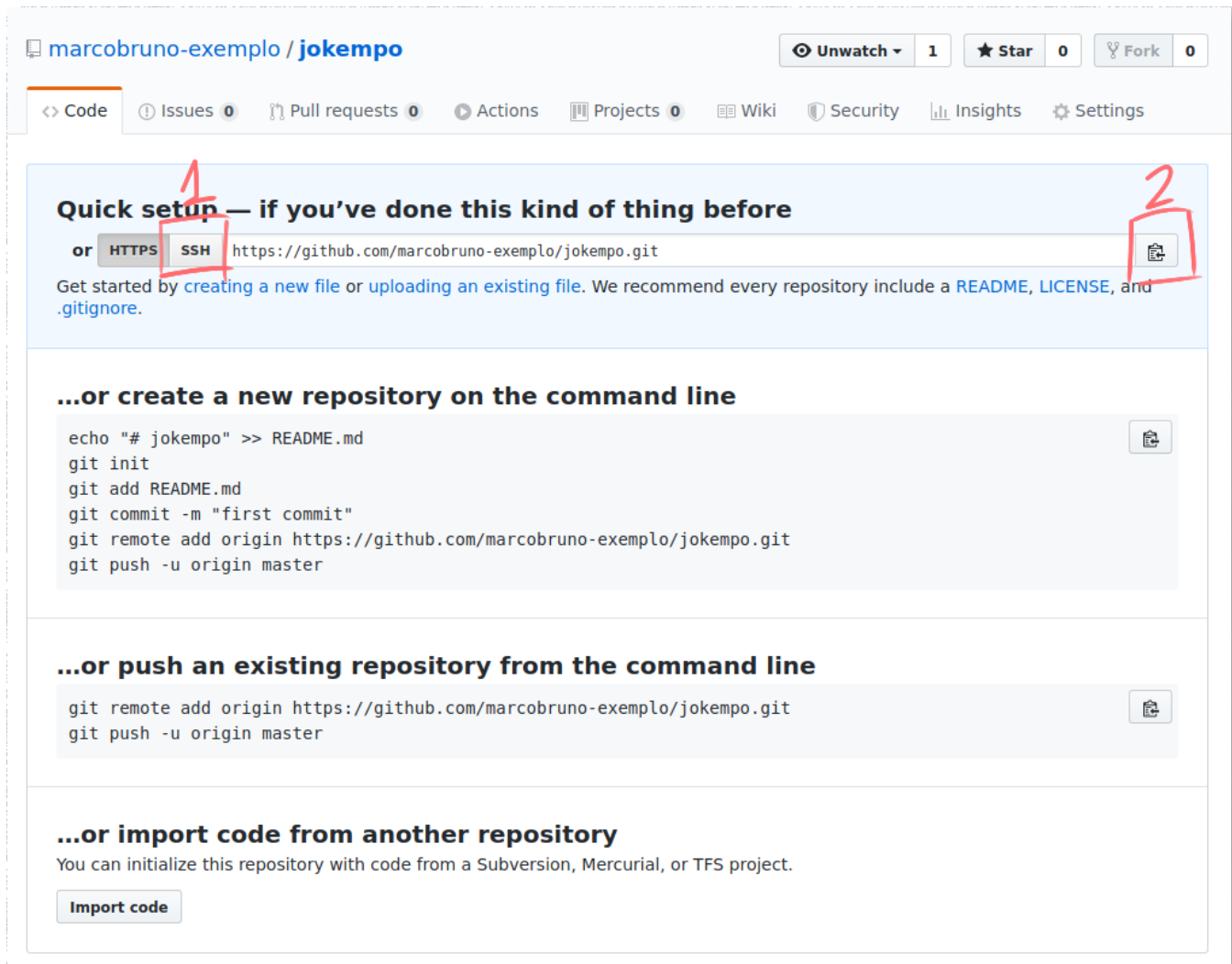
GPG keys New GPG key

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

4. Associe projeto local com repositório remoto do GitHub

Volte na tela do repositório que criamos no começo do exercício, nela terá um botão com o nome de SSH, clique nele depois em um ícone que copiará a URL SSH do repositório:



marco-bruno-exemplo / jokempo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Quick setup — if you've done this kind of thing before

or **SSH** `https://github.com/marco-bruno-exemplo/jokempo.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# jokempo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/marco-bruno-exemplo/jokempo.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/marco-bruno-exemplo/jokempo.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

Abra o terminal, dentro da pasta do projeto **jokempo** execute o comando a seguir que associará o nosso projeto local com o repositório remoto do GitHub:

```
git remote add origin https://github.com/marco-bruno-exemplo/jokempo.git
```

5. Configurar o git localmente

Rode os comandos a seguir para configurar o nome (**user.name**) e o e-mail (**user.email**) que devem aparecer toda vez que você for guardar o estado do seu código:

```
git config --global user.name "Seu nome"
git config --global user.email "seu.email@exemplo.com"
```

Para ver que tudo foi configurado corretamente, rode o comando abaixo:

```
git config -l
```

Se tudo ocorreu bem você verá uma saída similar a esta:

```
user.name=Marco Bruno
user.email=marco.bruno.br@gmail.com
```

6. Adicione os arquivos que você alterou

Execute o comando a seguir para adicionar o arquivo que foi alterado na lista de arquivos que devem ser enviados para o repositório remoto do GitHub:

```
git add src/404.html
```

Para verificar se foi adicionado, rode o comando abaixo:

```
git status
```

Aparecerá uma saída parecida com esta:

```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   src/404.html
```

7. Guarde o estado do código

Rode o comando a seguir, ele guardará o estado do código como está agora, isso dará a você o poder de máquina do tempo para o seu código:

```
git commit -m "Crie o HTML do cabeçalho"
```

O comando `commit` com o parâmetro `-m` possibilita você adicionar uma mensagem para esse momento do código, dessa forma se precisar voltar nesse momento do código ficará mais fácil para achar. O texto da mensagem de `commit` sempre fica no imperativo, se possível em inglês mas se você não sabe inglês não se preocupe com isso nesse momento, você está começando a aprender programação, mas o conhecimento de inglês é bem importante uma vez que você quer ser programador. Para saber mais sobre a mensagem de commit, [leia o post do Lucas Caton](#).

8. Envie o código para o repositório remoto do GitHub

Execute o comando a seguir, ele enviará o código para o repositório remoto que associamos com o comando `git remote add`, anteriormente nesse exercício:

```
git push
```

Como você está executando o comando `git push` pela primeira vez, é necessário associar a sua *branch* (ramificação) local com uma do repositório remoto, por isso receberá a seguinte mensagem de erro:

```
fatal: The current branch master has no upstream branch.  
To push the current branch and set the remote as upstream, use  
  
git push --set-upstream origin master
```

A parte legal é que o próprio git já te oferece a solução para o erro que é o comando `git push --set-upstream origin master`, mas você pode usar um comando menor para chegar no mesmo resultado que é:

```
git push -u origin master
```

Por favor, por enquanto não se preocupe o que é *branch*, mais pra frente te explico com mais calma. Nos próximos envios para o repositório remoto não será mais necessário usar o comando `git push -u origin master`, só o comando `git push` será o suficiente.

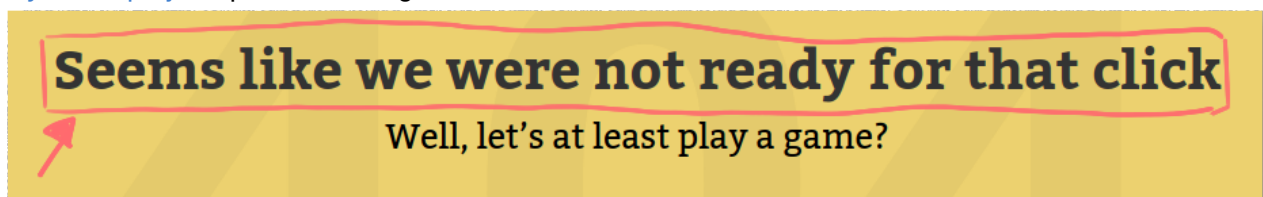
Desafios top top top

Os desafios top top top são para você tentar implementar junto com seu grupo de estudo ou só você. Recomendo você convidar uma pessoa para estudar junto com você, ao menos isso me ajuda bastante para ter outro ponto de vista e me motiva bastante ter uma pessoa comigo no mesmo caminho. Se não tiver não deixe de fazer por sua conta. Esses desafios são para você validar como foi a sua absorção do conhecimento passado na aula e também tentar evoluir um pouco sem ajuda.

Básico

Desafio básico é sempre algo que expliquei na aula e é considerado de fácil implementação:

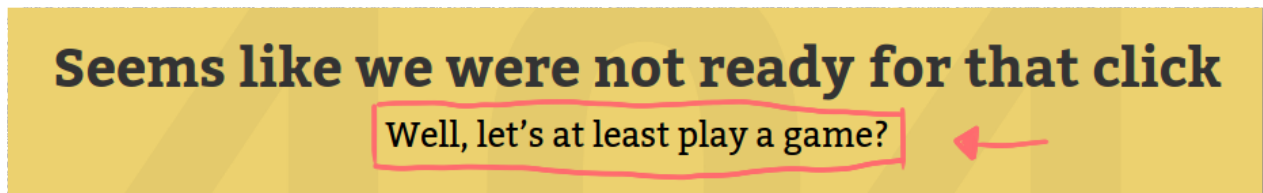
- Crie o HTML do título do 404, conforme a imagem a seguir. Lembre-se que você pode ver tudo no [layout do projeto](#) que está no Figma.



Intermediário

O desafio intermediário é um pouco mais complicado, mas com o conhecimento que passei na aula é suficiente para implementar, onde apenas mudo um pouco o contexto. Depois do dia 2 esse desafio pode ter conteúdo de várias aulas.

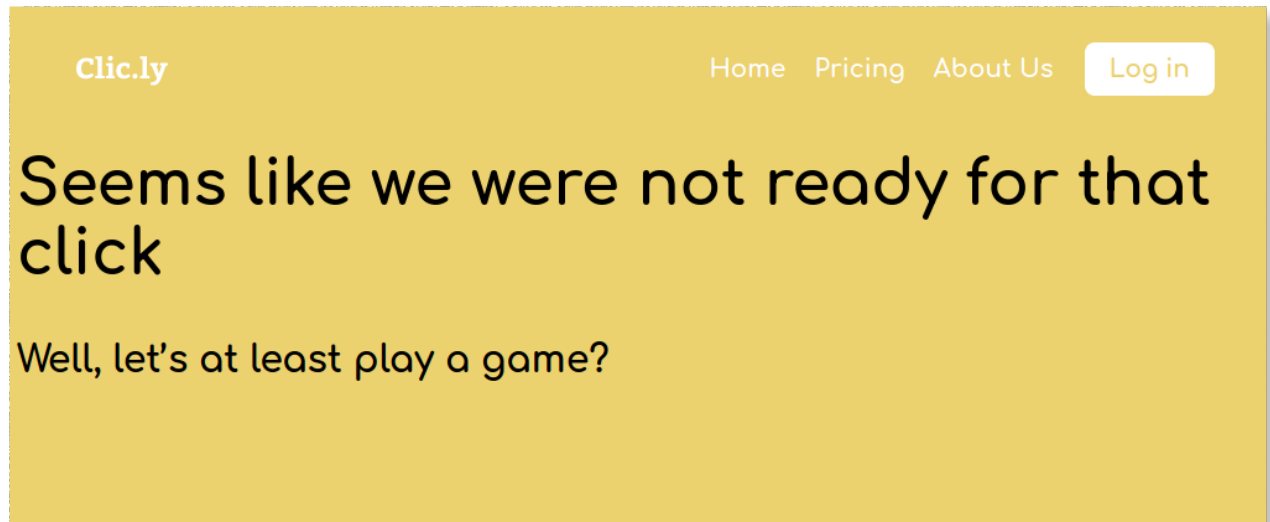
- Crie o subtítulo do 404, conforme está na imagem a seguir.



Avançado

É algo que não expliquei na aula e você precisará implementar por conta e também correr atrás do conhecimento. Mas tenha calma se não conseguir executar um dos desafios porque no dia seguinte implementarei todos os desafios.

- Faça o CSS do cabeçalho e também deixe a pagina toda amarela, conforme na imagem a seguir.



Dia 2 - Dando vida ao projeto

A pressa é a inimiga da vitória. O fraco não tem espaço e o covarde morre sem tentar!

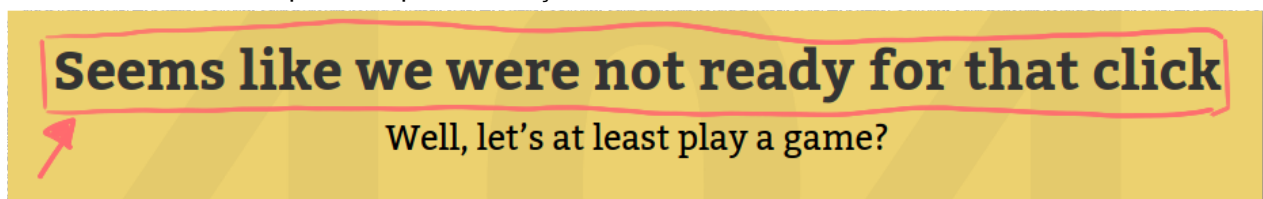
Racionais MC's

Exercício 01 - Criar HTML da seção

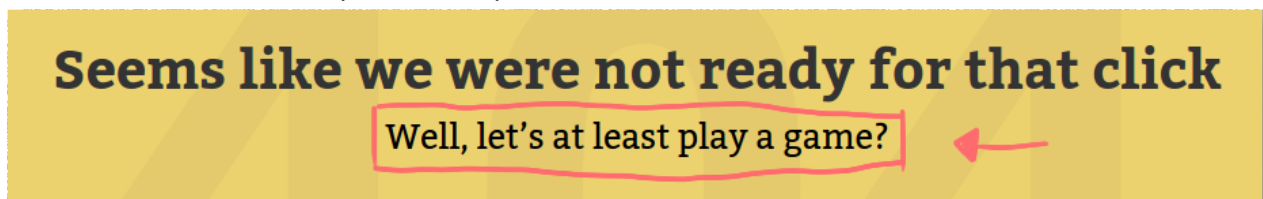
Aqui é uma ótima oportunidade para você validar seu conhecimento sobre HTML que aprendeu nos exercícios do dia 1. Criaremos o título e subtítulo que fica a seguir do nosso cabeçalho.

Tarefas

1. Crie o HTML do título que fica após o cabeçalho



2. Crie o HTML do subtítulo após o título que acabamos de criar



Passo a passo

1. Crie HTML do título

Após a tag `<header>` crie uma tag de título de primeira importância (`<h1>`) e dentro dela coloque o conteúdo *Seems like we were not ready for that click* (Parece que não estávamos prontos para esse clique). O código que está **dentro da sua tag** `<body>` ficará da seguinte forma, com a adição do título:

```
<header>
  <h1>Clic.ly</h1>

  <nav>
    <a href="">Home</a>
    <a href="">Pricing</a>
    <a href="">About Us</a>
    <a href="">Log in</a>
  </nav>
</header>

<h1>Seems like we were not ready for that click</h1>
```

2. Crie HTML do subtitle

A seguir do título que acabamos de construir, adicione uma tag que represente um título de segunda importância, também conhecido como subtítulo (`<h2>`) e dentro dela coloque este texto: *Well, let's at least play a game?* (Bem, pelo menos vamos jogar um jogo?). Seu código que está dentro do `<body>` ficará da seguinte forma:

```
<header>
  <h1>Click.ly</h1>

  <nav>
    <a href="">Home</a>
    <a href="">Pricing</a>
    <a href="">About Us</a>
    <a href="">Log in</a>
  </nav>
</header>

<h1>Seems like we were not ready for that click</h1>
<h2>Well, let's at least play a game?</h2>
```

Exercício 02 - Base do CSS

Definiremos algumas bases para a parte visual do nosso site com CSS, as cores da nossa aplicação usando variáveis do CSS e utilizaremos uma destas cores aplicando-a no `<body>`.

Tarefas

1. Crie uma pasta **css** que ficará dentro da **src**;
2. Crie uma pasta **settings** que ficará dentro da **css**;
3. Crie um arquivo **color.css** na pasta **settings** com uma variável chamada `--color-primary` que terá o valor `#ecd16f`, não deixe de importar o arquivo no **404.html**;
4. Crie uma pasta **elements** dentro da **css**;
5. Crie um arquivo **base.css** e nele você deve atribuir a cor `--color-primary` como cor de fundo para o body.

Passo a passo

1. Crie pasta CSS

No terminal e dentro da pasta do projeto **jokempo**, entre na pasta **src**, usando o comando a seguir:

```
cd src
```

Crie a pasta **css**:

```
mkdir css
```

2. Crie pasta settings

Dentro da pasta **css** e no terminal crie a pasta **settings**:

```
mkdir settings
```

3. Crie arquivo color.css

Volte no VSCode e crie um arquivo chamado **color.css** que ficará dentro da pasta **settings** que acabamos de criar. Dentro deste arquivo crie o selector `:root` que representa a aplicação toda, dentro do selector crie a variável `--color-primary` e atribua o hexadecimal `#ecd16f` que representa a cor amarela do nosso site. O código do arquivo **settings.css** ficará assim:

```
:root {  
  --color-primary: #ecd16f;  
}
```

Entre no arquivo **404.html** e adicione a tag `<link>` para relacioná-la com o arquivo **color.css**, o código dentro da tag `<head>`, ficará da seguinte forma:

```
<meta charset="utf-8">  
<title>Clic.ly - Not found page</title>  
  
<link rel="stylesheet" href="css/settings/color.css">
```

O seletor `:root` é usado para criar recursos que podem ser acessados por qualquer outro seletor da nossa aplicação. Por isso que usamos ele para guardar nossas variáveis de cor, já que ela será usada pela aplicação toda em muitos contextos diferentes.

Toda variável no CSS parece uma propriedade do CSS, por esse motivo é obrigatório que ela começar com dois hifens (`--`), elas também são chamada de custom properties (propriedades customizadas). Você pode ler mais sobre na MDN (Mozilla developer network):

https://developer.mozilla.org/en-US/docs/Web/CSS/--*

Dentro da pasta **settings** ficarão todas as variáveis que disponibilizaremos para nossa aplicação. A estrutura de pasta que estamos usando é muito similar a uma padrão conhecido com o ITCSS (Inverted Triangle CSS). Recomendo você ler o post do Willian Justen para se aprofundar mais nesse tema: <https://willianjusten.com.br/organizando-seu-css-com-itcss/>

4. Crie pasta

Volte ao terminal e crie a pasta **elements** dentro da pasta **css**. Estando dentro da pasta **css** rode o comando a seguir:

```
mkdir elements
```

5. Crie arquivo base.css

No VSCode crie um arquivo **base.css** dentro da pasta **elements**, nele crie um seletor de tag para a **body**, no seletor defina a cor de fundo usando a propriedade `background-color` com o valor da variável `--color-primary`. O código do arquivo **base.css**, ficará dessa forma:

```
body {  
  background-color: var(--color-primary);  
}
```

Para usar um variável do CSS precisamos envolvê-la com `var(--nome-da-variavel)`.

Importe o arquivo **base.css** no arquivo **404.html** usando a tag `<link>` dentro do `<head>` e após o importe do arquivo **color.css**. Seu código dentro da tag `<head>` ficará da seguinte forma:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">
```

Se visitar o seu navegador a sua página estará similar a esta:

Clic.ly

[Home](#) [Pricing](#) [About Us](#) [Log in](#)

Seems like we were not ready for that click

Well, let's at least play a game?

Exercício 03 - Primeiro componente

Não vou te ensinar a criar um CSS complicado de dar manutenção, se você criar um site, sistema ou aplicação que tem um custo alto de manutenção isso te trará muita frustração a curto prazo e poderá fazer com que o seu projeto venha a falhar. Um produto com custo alto se torna menos competitivo no mercado de trabalho. Então criaremos o nosso primeiro componente e tudo que precisarmos para deixar nosso CSS organizado e de fácil manutenção.

Tarefas

1. Crie uma pasta **components** dentro da pasta **css**;
2. Adicione a **class main-logo** no **<h1>** que está dentro da tag **<header>**
3. Crie um arquivo **main-logo.css** dentro da pasta **components** e faça o CSS para ele ficar da seguinte forma (lembre-se do **<link>** para o arquivo **main-logo.css** no **404.html**):



Passo a passo

1. Crie pasta components

Agora não precisa mais usar o terminal para criar as pastas, acho que já pegou a ideia dele, pode criar a pasta **components** usando o próprio VSCode, essa pasta ficará dentro da pasta **css**.

2. Adicione o main-logo

Adicione o atributo **class** com o valor **main-logo** na tag **<h1>** que está dentro da tag **<header>** no nosso arquivo **404.html**:

```
<h1 class="main-logo">Clic.ly</h1>
```

3. CSS do componente main-logo

Dentro do arquivo **color.css** adicione uma nova cor chamada **--color-second** e atribua a ela o valor de **#fff** que é o hexadecimal da cor branca:

```
:root {  
  --color-primary: #ecd16f;  
  --color-second: #fff;  
}
```

Dentro da pasta **components** crie um arquivo **main-logo.css**, nesse arquivo crie um seletor de classe chamado **main-logo**:

```
.main-logo {  
  
}
```

Seletores de classes começam com um ponto (**.main-logo**), não esqueça do ponto porque mesmo sem isso ele é um seletor válido para o CSS, só que sem o ponto no começo ele se torna um seletor de tag

Dentro do seletor **.main-logo** adicione a cor branca para o texto usando a propriedade CSS **color** com o valor **var(--color-second)**:

```
.main-logo {  
  color: var(--color-second);  
}
```

Faça o importe do arquivo **main-logo.css** no arquivo **404.html** usando a tag **<link>**, ela ficará depois do relacionamento do arquivo **base.css**. O código que fica **dentro da sua tag <head>** ficará assim:

```
<meta charset="utf-8">  
<title>Clic.ly - Not found page</title>  
  
<link rel="stylesheet" href="css/settings/color.css">  
<link rel="stylesheet" href="css/elements/base.css">  
  
<link rel="stylesheet" href="css/components/main-logo.css">
```

Precisamos importar nossa família de fontes do nosso **.main-logo** que é a *Bitter*, usaremos o *Google Fonts* para baixar esta família no nosso arquivo **404.html**. Para isso também vamos usar uma tag **<link>** mas ela não será direcionada a um arquivo CSS e sim ao link da fonte *Bitter* no *Google Fonts* e a tag **<link>** dela ficará antes do arquivo **color.css** como está demonstrado a seguir:


```
<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
```

Link para a fonte Bitter no Google Fonts: <https://fonts.google.com/specimen/Bitter>

Abra o arquivo **main-logo.css**, aplique a fonte *Bitter* no seletor `.main-logo` usando a propriedade `font-family` com o valor `'Bitter'`, `serif`:

```
.main-logo {
  color: var(--color-second);
  font-family: 'Bitter', serif;
}
```

A propriedade `font-family` aceita mais de um valor de fontes, ela dá prioridade para fontes que estão à direita e caso a fonte não exista ou não funcione por algum motivo ela tenta carregar a próxima fonte. No final é recomendado usar `serif` (qualquer fonte serifada) ou `sans-serif` (qualquer fonte não serifada), caso nenhuma das fontes funcionem, o SO (sistema operacional) recomendará uma fonte para o navegador.

Nos componentes evitamos usar propriedades que definam altura, largura, espaços (respiros), posicionamento ou tamanho. Algumas propriedades que evitamos nos componentes são: `width`, `height`, `margin`, `padding`, `display`, `position` e `float`. Isso garantirá uma maior flexibilidade do uso dos componentes e portanto uma manutenção mais feliz.

Exercício 04 - Componente simple-action

Criaremos o componente dos links que ficam na navegação principal do nosso site. Ele não usa a mesma família de fontes do nosso logo e é uma ótima oportunidade para praticar o conceito de componentes.

Tarefas

1. Adicione a classe `simple-action` nos três primeiros itens da navegação principal do site;
2. Crie o arquivo `simple-action.css` dentro da pasta `components`;
3. Implemente o CSS que deixará o componente como na imagem a seguir, dentro do arquivo `simple-action`:



Passo a passo

1. Adicione a classe simple-action

Dentro do arquivo `404.html` adicione o atributo `class` com o valor `simple-action` nas três primeiras tags `<a>` que estão dentro da tag `<nav>`. O código da sua `<nav>` ficará da seguinte forma:

```
<nav>
  <a class="simple-action" href="">Home</a>
  <a class="simple-action" href="">Pricing</a>
  <a class="simple-action" href="">About Us</a>
  <a href="">Log in</a>
</nav>
```

2. Crie arquivo simple-action.css

Dentro da pasta `css`, temos a pasta `components`. Dentro da pasta `components` adicione um novo arquivo `simple-action.css`, faça isso dentro do VSCode.

Abra o arquivo `404.html` e adicione a tag `<link>` que fará o importe do arquivo que acabamos de criar, essa tag ficará depois do importe do componente `main-logo`. O código que fica dentro do seu `<head>` ficará dessa forma:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
<link rel="stylesheet" href="css/components/simple-action.css">
```

3. CSS do simple-action

Crie o seletor de classes `.simple-action` dentro do arquivo **simple-action.css**:

```
.simple-action {

}
```

Defina a cor branca para o texto do seletor `.simple-action` usando a propriedade `color` passando para ela a variável `--color-second`. O seletor ficará dessa forma:

```
.simple-action {
  color: var(--color-second);
}
```

No navegador terá o seguinte resultado:



Home Pricing About Us Log in

Remova o traço que fica na base do texto, usando a propriedade `text-decoration` com o valor `none`:

```
.simple-action {
  color: var(--color-second);
  text-decoration: none;
}
```

Esse é o resultado no navegador:

Home Pricing About Us

Abra o arquivo **404.html** e adicione a tag `<link>` a seguir da tag que realiza o importe da fonte *Bitter*. Essa nova tag importará a família de fonte *Comfortaa*. O código que está **dentro do seu `<head>`** ficará assim:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?
family=Comfortaa:300,400,500,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
<link rel="stylesheet" href="css/components/simple-action.css">
```

Importe realizado, volte ao arquivo **simple-action.css** e aplique a fonte *Comfortaa* usando a propriedade `font-family` com o valor `sans-serif`:

```
.simple-action {
  color: var(--color-second);
  text-decoration: none;
  font-family: 'Comfortaa', sans-serif;
}
```

Você terá o seguinte resultado visual no navegador:

Home Pricing About Us

Defina a fonte como negrito usando a propriedade `text-weight` com o valor `bold`:

```
.simple-action {
  color: var(--color-second);
  text-decoration: none;
  font-family: 'Comfortaa', sans-serif;
  font-weight: bold;
}
```

O resultado no navegador será esse:



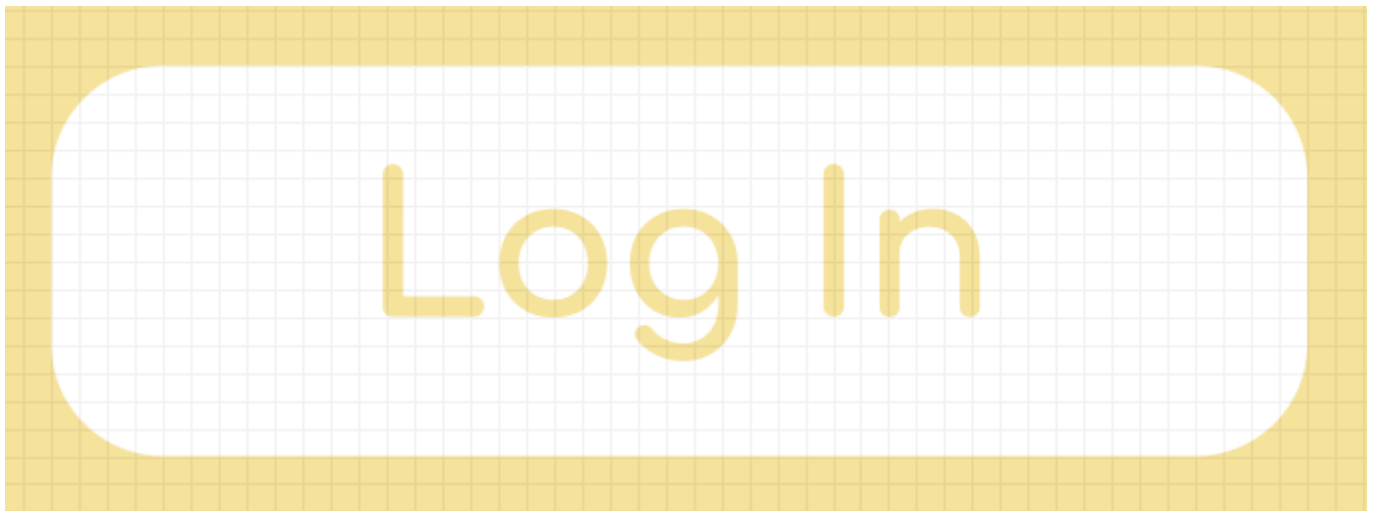
Home Pricing About Us

Exercício 05 - Componente primary-button

O componente primary-button terá um pouco mais de propriedades comparado com os outros dois que acabamos de criar e ele é um tipo de componente usado em muitos contextos dentro de um site afinal botões estão espalhados em sites da internet, não é mesmo!

Tarefas

1. Coloque a classe `primary-button` no quarto link da navegação principal do nosso site;
2. Adicione um arquivo novo na pasta **components** chamado **primary-button.css**;
3. Construa todo o CSS necessário no arquivo **primary-button.css** para o componente `primary-button` ficar similar a imagem a seguir, lembre-se das regras de um componente:



Passo a passo

1. Adicione classe primary-button

Abra o arquivo **404.html** e adicione o atributo `class` no quarto link do menu principal com o valor `primary-button`. A sua `<nav>` ficará assim:

```
<nav>
  <a class="simple-action" href="">Home</a>
  <a class="simple-action" href="">Pricing</a>
  <a class="simple-action" href="">About Us</a>
  <a class="primary-button" href="">Log in</a>
</nav>
```

2. Crie o arquivo primary-button.css

Pelo VSCode crie o arquivo **primary-button.css** dentro da pasta **components** que está localizada na pasta **css**.

Abra o arquivo **404.html** e adicione uma tag `<link>` que importará o arquivo que acabamos de criar, essa tag ficará após a tag de importe do componente **simple-action.css**, dentro do seu `<head>` você terá o

seguinte código:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?
family=Comfortaa:300,400,500,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
<link rel="stylesheet" href="css/components/simple-action.css">
<link rel="stylesheet" href="css/components/primary-button.css">
```

3. CSS do primary-button

Dentro do arquivo **primary-button.css** crie um seletor de classe para o componente **primary-button**:

```
.primary-button {

}
```

Atribua a cor de fundo branca para o componente usando a propriedade **background-color** com o valor **var(--color-second)**:

```
.primary-button {
  background-color: var(--color-second);
}
```

Veja no seu navegador:



Remova o traço sob o texto usando a propriedade **text-decoration** com o valor **none**:

```
.primary-button {
  background-color: var(--color-second);
  text-decoration: none;
}
```

No navegador terá o seguinte resultado:



Defina a cor de texto para amarelo usando a propriedade `color` com o valor `var(--color-primary)`:

```
.primary-button {  
  background-color: var(--color-second);  
  text-decoration: none;  
  color: var(--color-primary);  
}
```



Para adicionar bordas arredondadas utilize a propriedade `border-radius` com o valor `4px`:

```
.primary-button {  
  background-color: var(--color-second);  
  text-decoration: none;  
  color: var(--color-primary);  
  border-radius: 4px;  
}
```



Mude a família de fonte para a Comfortaa usando a propriedade `font-family` com o valor `'Comfortaa', sans-serif`:

```
.primary-button {  
  background-color: var(--color-second);  
  text-decoration: none;  
  color: var(--color-primary);  
  border-radius: 4px;  
  font-family: 'Comfortaa', sans-serif;  
}
```



Exercício 06 - Primeiro container

Praticamos bastante o conceito de componentes, agora chegou a hora de estudar o conceito de containers. Nosso primeiro container será aplicado na tag `<header>`, um container tem a responsabilidade de juntar os componentes além de posicionar e definir o tamanho, espaçamentos e posicionamento deles no contexto do container. Bora praticar que ficará mais claro tudo que tentei dizer.

Tarefas

- Crie uma pasta **containers** dentro da pasta **css**;
- Crie um arquivo **main-header.css** dentro da pasta **containers**;
- Adicione a classe **main-header** para a tag `<header>` que temos dentro do `<body>`;
- Crie o CSS necessário para deixar todos os componentes e o próprio container dessa forma:

The image shows a header design for a website. On the left is the logo 'Clic.ly' in a yellow box. To the right of the logo are three links: 'Home', 'Pricing', and 'About Us'. Further to the right is a 'Log In' button inside a rounded rectangle.

Home

Pricing

About Us

Log In

Passo a passo

1. Crie pasta containers

Cria a pasta **containers** dentro da pasta **css**.

2. Crie arquivo main-header.css

Crie o arquivo **main-header.css** dentro da pasta **containers**.

Abra o arquivo **404.html** e adicione a tag `<link>` que irá relacionar com o arquivo **main-header.css**, ela ficará depois de todos os `<link>`s dos componentes:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?
family=Comfortaa:300,400,500,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
<link rel="stylesheet" href="css/components/simple-action.css">
<link rel="stylesheet" href="css/components/primary-button.css">

<link rel="stylesheet" href="css/containers/main-header.css">
```

3. Adicione a classe main-header

Dentro do arquivo **404.htm** tem uma tag `<header>`, nela adicione a propriedade `class` com o valor `main-header`, sue `<header>` ficará assim:

```
<header class="main-header">
  <h1 class="main-logo">Clic.ly</h1>

  <nav>
    <a class="simple-action" href="">Home</a>
    <a class="simple-action" href="">Pricing</a>
    <a class="simple-action" href="">About Us</a>
    <a class="primary-button" href="">Log in</a>
  </nav>
</header>
```

4. CSS do container

Crie o seletor de class do container `main-header` dentro do arquivo **main-header.css**:

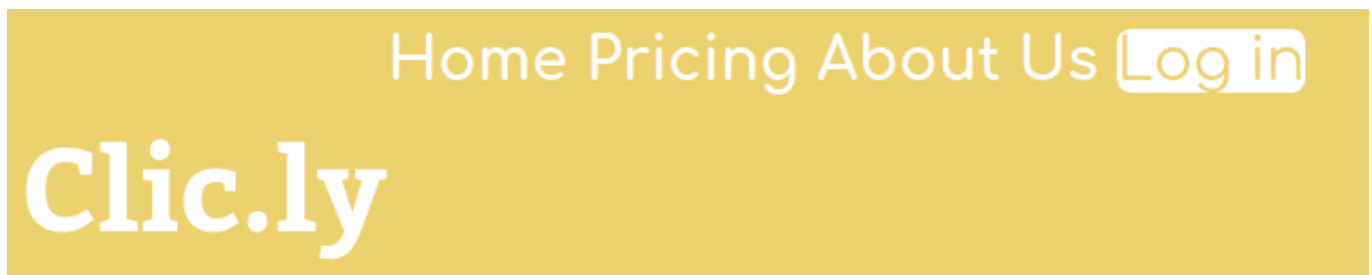
```
.main-header {

}
```

Adicione a propriedade `display` com o valor `flex` para deixar o menu ao lado do logo:

```
.main-header {
  display: flex;
}
```

No navegador terá um resultado similar:



Use a propriedade `align-items` com o valor `center` para alinhar o logo e a navegação ao centro:

```
.main-header {  
  display: flex;  
  align-items: center;  
}
```

Esse é o resultado no navegador:



Use a propriedade `justify-content` com o valor `space-between` para mover o logo para esquerda e o menu para a direita:

```
.main-header {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
}
```

Resultado no navegador:



Crie o seletor que pegará apenas o componente `main-logo` quando for um filho direto do container `main-header`, adicione esse seletor após o do `.main-header`:

```
.main-header {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
}  
  
.main-header > .main-logo {  
  
}
```

Dentro do seletor que acabamos de criar, defina o tamanho de **20px** para o **main-logo** usando a propriedade **main-logo**:

```
.main-header {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
}  
  
.main-header > .main-logo {  
  font-size: 20px;  
}
```

Você terá o seguinte resultado visual no navegador:

cllc.ly

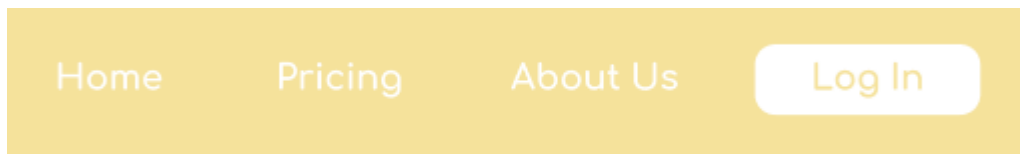
Home Pricing About Us [Log in](#)

Exercício 07 - Container main-menu

Para finalizar o nosso header, é necessário criar o container **main-menu** que posicionará e definir o tamanho dos itens do menu de navegação principal.

Tarefas

1. Crie o arquivo **main-menu.css** dentro da pasta **containers**;
2. Crie o arquivo **gap.css** dentro da pasta **settings**;
3. Defina em variáveis os respiros da nossa aplicação no arquivo **gap.css**;
4. Adicione a classe **main-menu** na tag **<nav>** que está dentro do **<header>**;
5. Crie o CSS necessário para deixar o container **main-menu** dessa maneira:



Passo a passo

1. Crie arquivo main-menu.css

Dentro da pasta **containers** crie o arquivo **main-menu.css**.

Abra o arquivo **404.html** e dentro dele use a tag **<link>** para relacionar o arquivo **main-menu.css**, essa tag ficará antes do container **main-header.css**, o código que está **dentro do seu <head>** ficará da seguinte forma:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?
family=Comfortaa:300,400,500,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
<link rel="stylesheet" href="css/components/simple-action.css">
<link rel="stylesheet" href="css/components/primary-button.css">

<link rel="stylesheet" href="css/containers/main-menu.css">
<link rel="stylesheet" href="css/containers/main-header.css">
```

2. Crie arquivo gap.css

Crie um arquivo **gap.css** dentro da pasta **settings**, esse arquivo terá variáveis com os espaços (respiros, termo mais usado pelos designers) entre os elementos do nosso layout.

Abra o arquivo **404.html** e adicione a tag `<link>` que importará o arquivo **gap.css**. Ela ficará após o `<link>` do arquivo **color.css** e o conteúdo da sua tag `<head>` ficará assim:

```
<meta charset="utf-8">
<title>Clic.ly - Not found page</title>

<link href="https://fonts.googleapis.com/css?
family=Bitter:400,400i,700&display=swap" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?
family=Comfortaa:300,400,500,700&display=swap" rel="stylesheet">

<link rel="stylesheet" href="css/settings/color.css">
<link rel="stylesheet" href="css/settings/gap.css">
<link rel="stylesheet" href="css/elements/base.css">

<link rel="stylesheet" href="css/components/main-logo.css">
<link rel="stylesheet" href="css/components/simple-action.css">
<link rel="stylesheet" href="css/components/primary-button.css">

<link rel="stylesheet" href="css/containers/main-menu.css">
<link rel="stylesheet" href="css/containers/main-header.css">
```

3. Crie variável de respiro

Dentro do arquivo **gap.css** crie o seletor `:root`:

```
:root {

}
```

No seletor `:root`, adicione a variável `--gap-small` com o valor de `22px`:

```
:root {
  --gap-small: 22px;
}
```

4. Adicione classe main-menu

No arquivo **404.html** adicione o atributo `class` com o valor `main-menu` na tag `<nav>` que está dentro da tag `<header>`. Sua tag `<nav>` ficará assim:

```
<nav class="main-menu">
  <a class="simple-action" href="">Home</a>
  <a class="simple-action" href="">Pricing</a>
  <a class="simple-action" href="">About Us</a>
  <a class="primary-button" href="">Log in</a>
</nav>
```

5. CSS do container main-menu

Crie o seletor que pegará os filhos diretos do container `main-menu` que têm a classe `simple-action`. Repare que tem que ser apenas os **filhos diretos**, portanto será necessário usar o seletor avançado de filho direto que é representado pelo sinal de maior (`>`):

```
.main-menu > .simple-action {

}
```

Utilize a propriedade `margin-right` com o valor `var(--gap-small)` dentro do seletor `.main-menu > .simple-action` para adicionar um *gap* (respiro) externo do lado direito de cada um dos componentes:

```
.main-menu > .simple-action {
  margin-right: var(--gap-small);
}
```

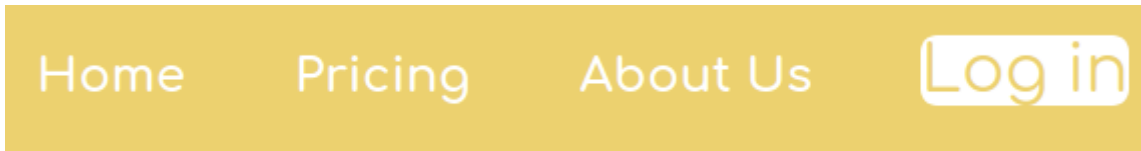
O resultado no navegador será:



Defina o tamanho da fonte como `12px` usando a propriedade `font-size` dentro do seletor `.main-menu > .simple-action`:

```
.main-menu > .simple-action {
  margin-right: var(--gap-small);
  font-size: 12px;
}
```

Você verá o seguinte resultado visual no navegador:



Crie um seletor para pegar o filho direito do container `main-menu` que tem a classe `primary-button`, esse seletor ficará após o seletor `.main-menu > .simple-action`:

```
.main-menu > .primary-button {  
  
}
```

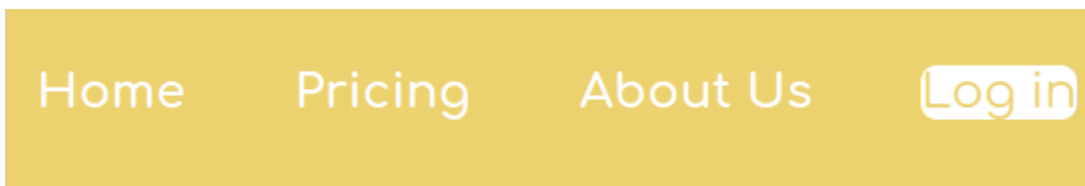
Defina a propriedade `display` com o valor `inline-block` para que as propriedades `width` e `height` do elemento funcionem e ele continue ao lado dos demais elementos do container `main-menu`:

```
.main-menu > .primary-button {  
  display: inline-block;  
}
```

Coloque a fonte com o tamanho de `12px` utilizando a propriedade `font-size` para o componente `primary-button` que é filho direto do container `main-menu`:

```
.main-menu > .primary-button {  
  display: inline-block;  
  font-size: 12px;  
}
```

O resultado no navegador será:



Adicione a largura de `60px` utilizando a propriedade `width` para o `primary-button` que está dentro do `main-menu`:

```
.main-menu > .primary-button {  
  display: inline-block;  
  font-size: 12px;  
  width: 60px;  
}
```

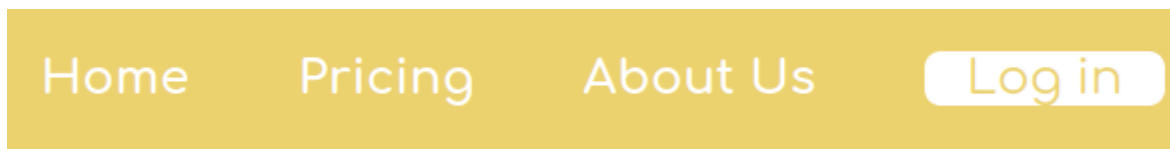

O resultado no navegador será esse:



Para centralizar o texto dentro do botão utilizaremos a propriedade `text-align` com o valor `center`, dentro do seletor `.main-menu > .primary-button`:

```
.main-menu > .primary-button {  
  display: inline-block;  
  font-size: 12px;  
  width: 60px;  
  text-align: center;  
}
```

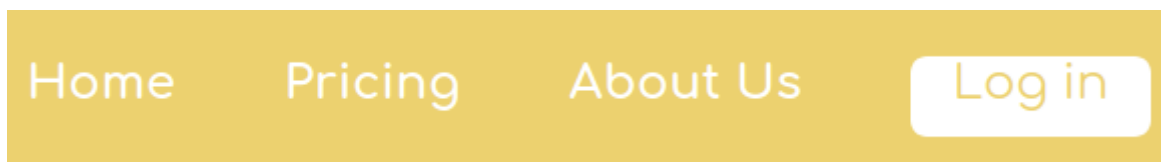
Esse é o resultado no seu navegador:



Use a propriedade `height` para definir a altura de `20px` para o componente `primary-button` que é filho direto do container `main-menu`:

```
.main-menu > .primary-button {  
  display: inline-block;  
  font-size: 12px;  
  width: 60px;  
  text-align: center;  
  height: 20px;  
}
```

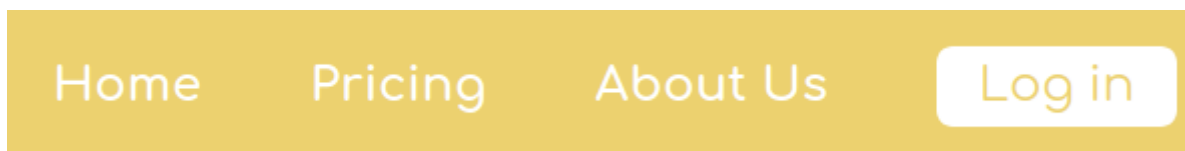
Vá no seu navegador e você verá isso:



Para o texto do componente `primary-button` também ficar centralizado no eixo vertical deixaremos a altura da linha com a mesma altura da caixa do componente. Use a propriedade `line-height` com o valor de `20px` que é o mesmo valor que definimos na propriedade `height`:

```
.main-menu > .primary-button {  
  display: inline-block;  
  font-size: 12px;  
  width: 60px;  
  text-align: center;  
  height: 20px;  
  line-height: 20px;  
}
```

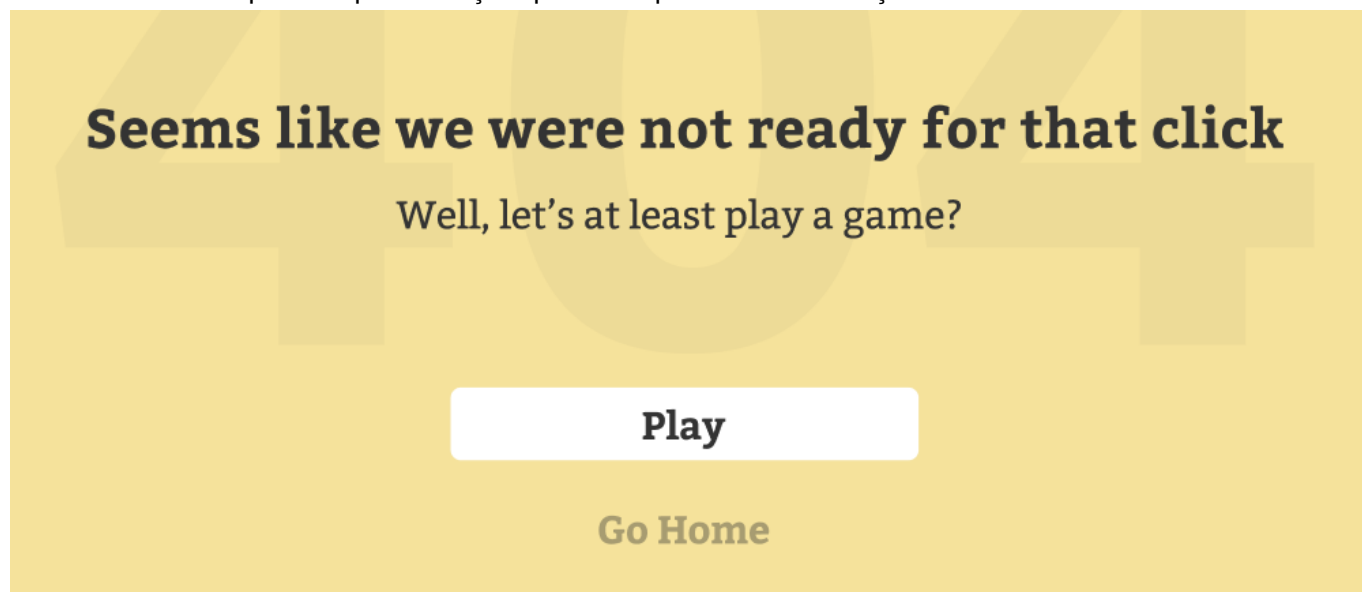
O layout do container `main-menu` estará assim no seu navegador:



Desafios top top top

Básico

Crie todo o HTML que falta para a seção que está após o nosso cabeçalho:



Intermediário

Crie todo o CSS necessário para que a seção que vem depois do header fique exatamente como está na imagem do desafio básico. Lembre-se de criar os *components* e *containers* necessários para deixar o nosso código simples e feliz para manutenções futuras. Sistema que é fácil de dar manutenção tem um custo mais baixo e é um produto mais competitivo no mercado de trabalho.

Avançado

Remova ao máximo o CSS que o navegador coloca nas tags, cada navegador escolhe um valor diferente para as propriedades e isso fará com que o nosso layout não funcione igual em navegadores diferentes. Essa técnica é chamada de *reset CSS* (recompôr o CSS).