SuPyGirls Documentation

Release 2.0.0

Carlo E. T. Oliveira

Dec 07, 2020

Contents:

CHAPTER 1

Bem Vindos à Introdução à Computação

Aqui vamos ter uma introdução básica dos preparativos e primeiros passos para programar jogos para Web usando Python. Vamos usar a IDE PyCharm.

1.1 Selecione seu sistema operacional:

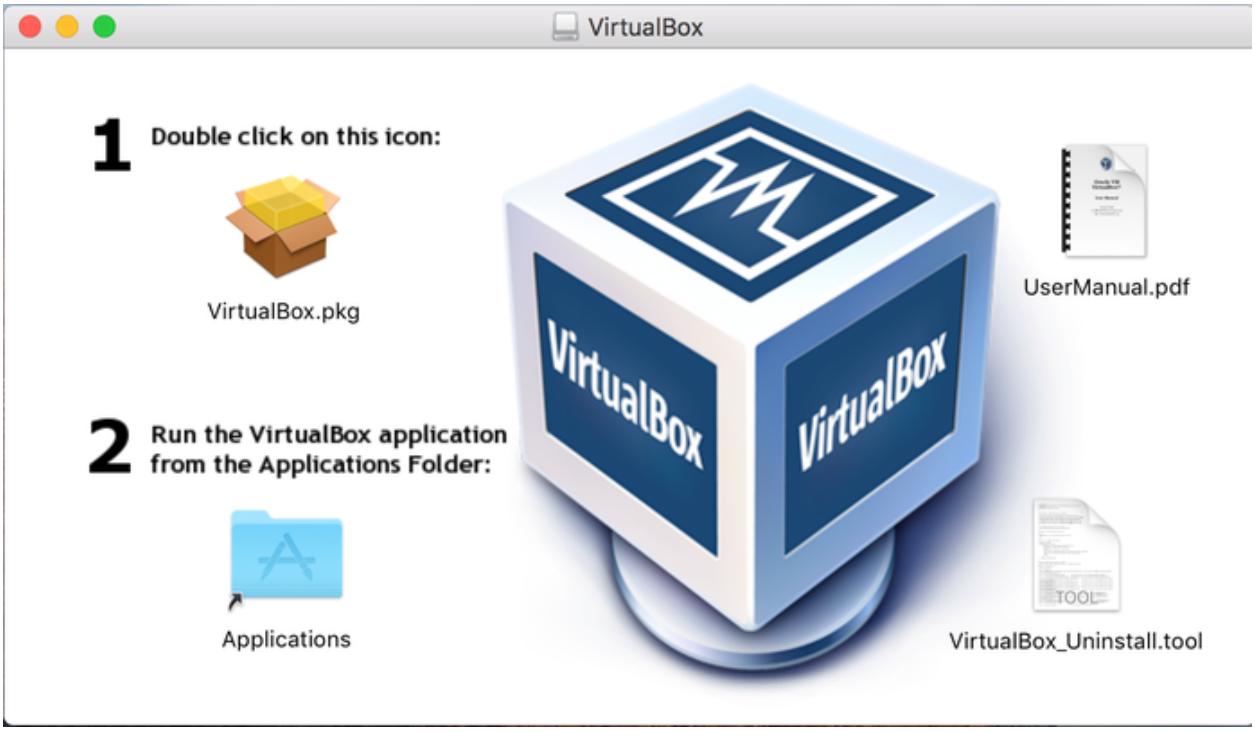
1.1.1 INSTALAÇÃO DE PROGRAMAS

INSTALAR ORACLE VM VIRTUALBOX

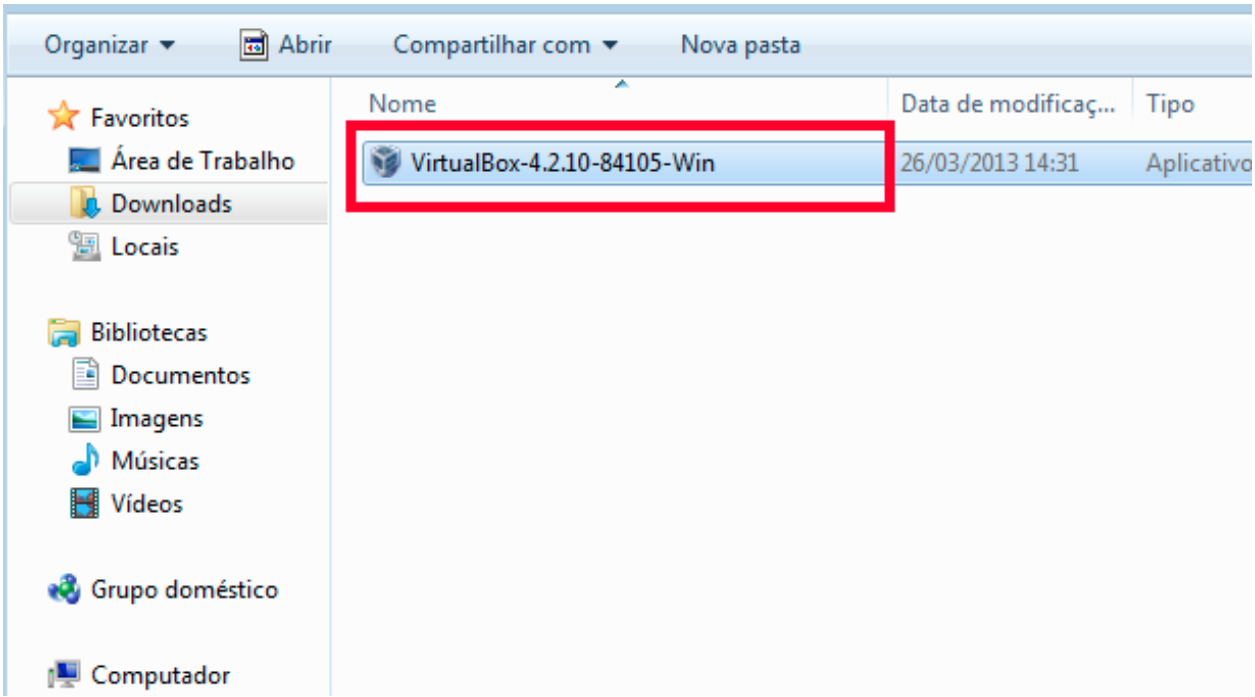
O VirtualBox é um sistema que possibilita criar e gerenciar maquinas virtuais no seu computador. O que isso significa? Significa que é possível rodar outro sistema operacinoal no seu computador sem desinstalar o atual. Pode-se rodar o Linux dentro do Windows, o Windows dentro do Mac e o Mac dentro do Windows, etc.

Passo a Passo

Baixe o programa no site oficial da Oracle.

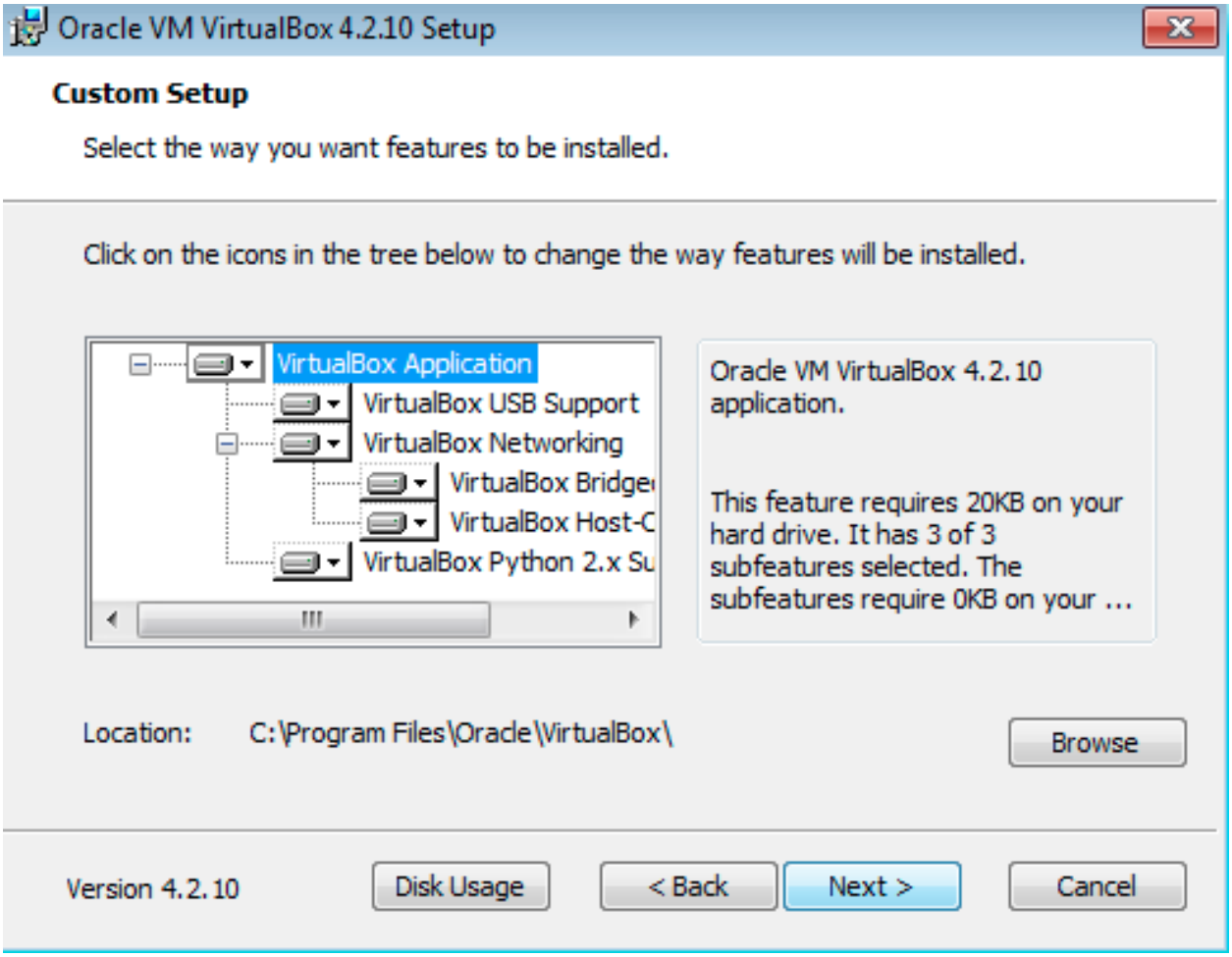


Clique no ícone do Virtualox na sua pasta de downloads.

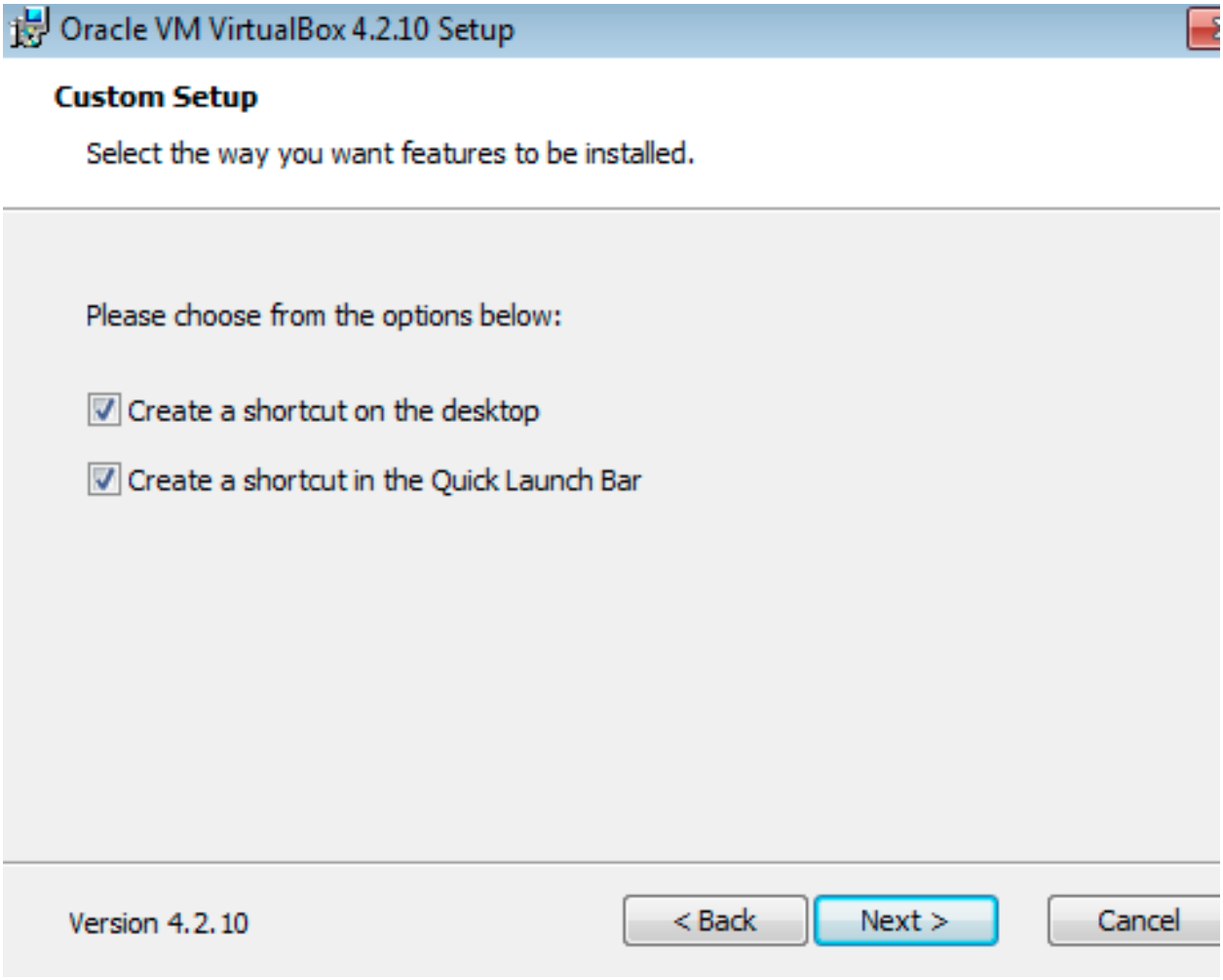


Abrirá uma aba que indica o início do seu processo de instalação. Clique em NEXT.

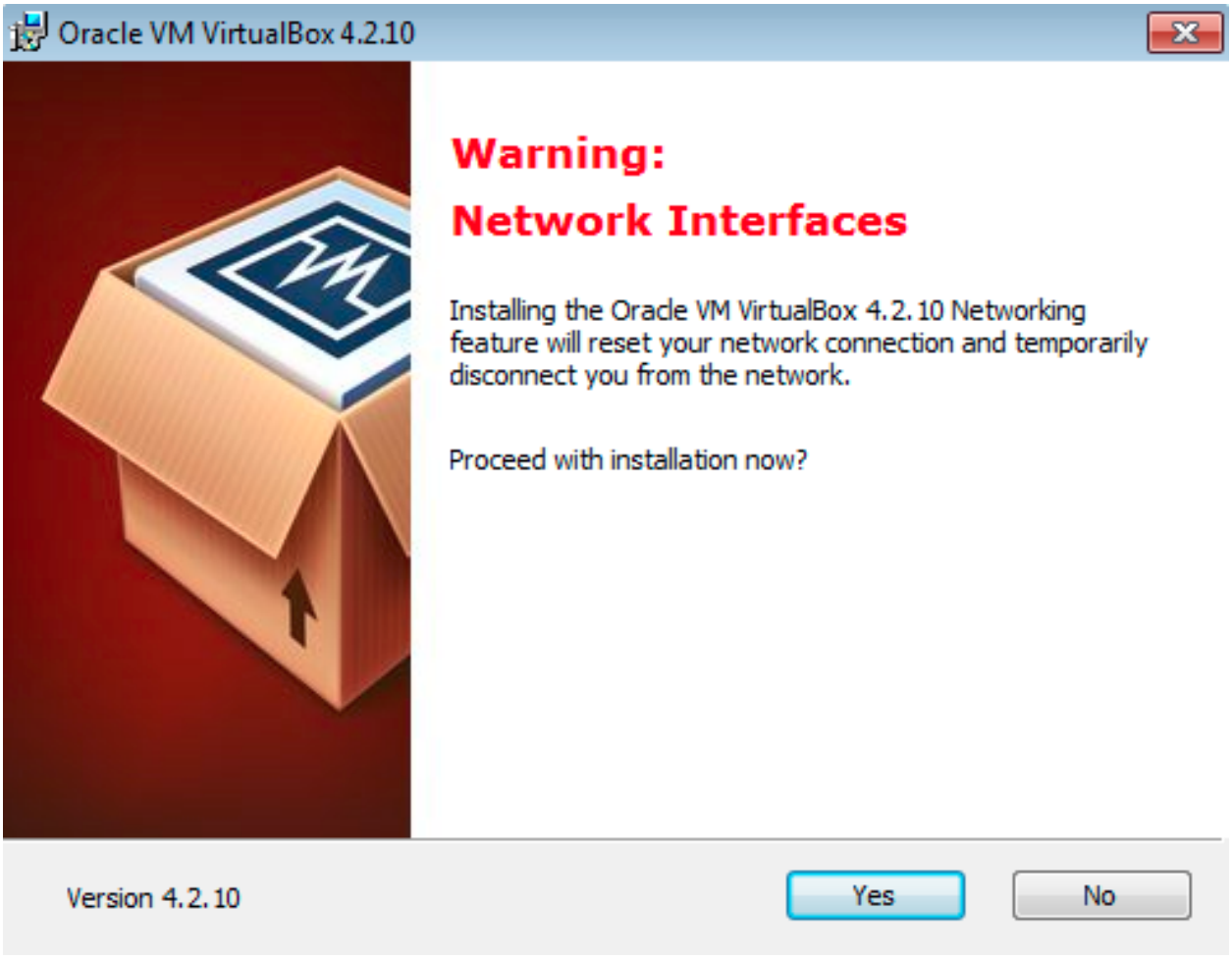
A próxima aba indica o local em que o programa será salvo em seu computador. Mantenha as informações selecionadas ou mude a seu gosto e clique em NEXT.



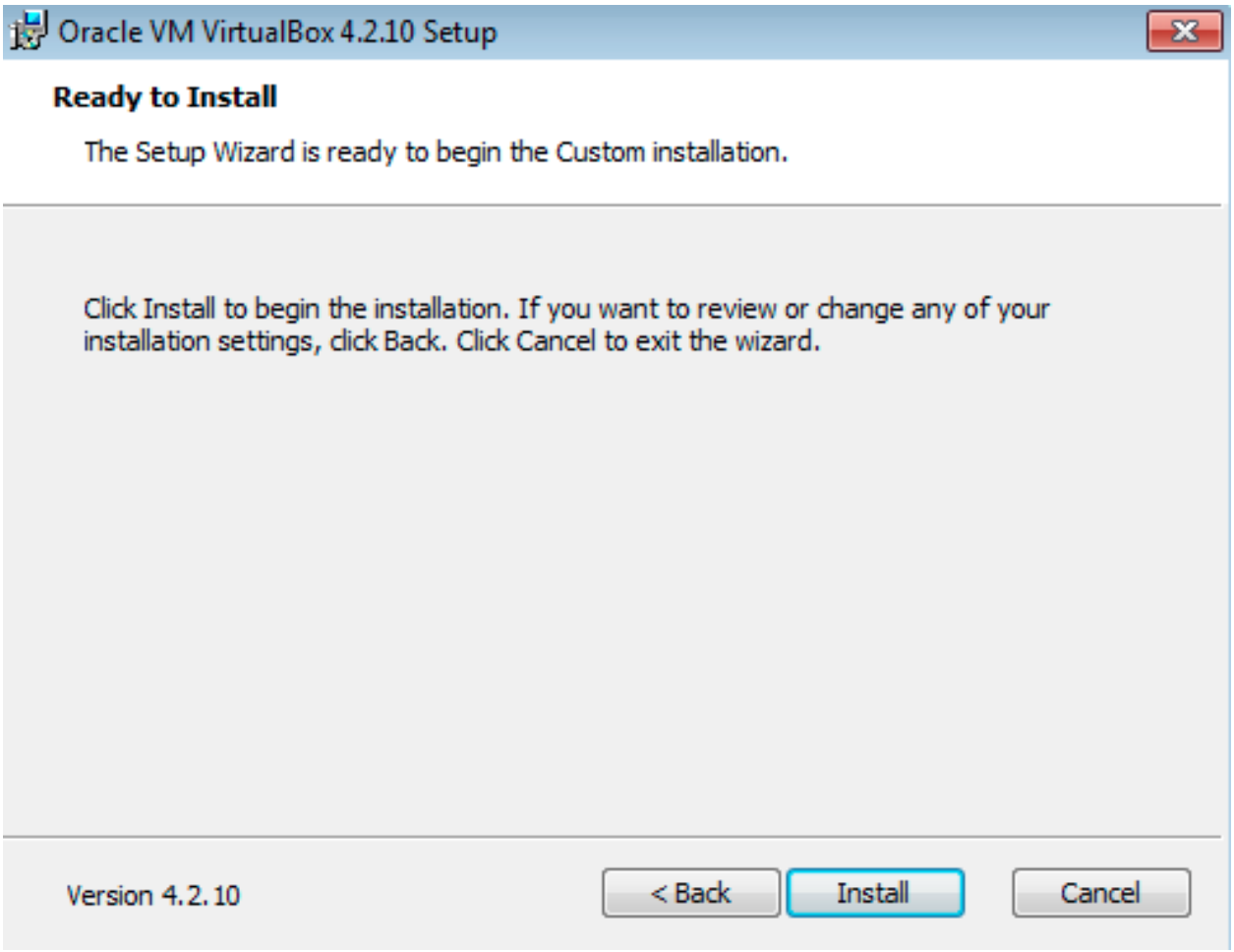
A aba abaixo indica se você quer um atalho para o seu Desktop (primeiro quadrado selecionado) e para sua barra de ferramentas (segundo quadrado selecionado). Faça suas escolhas e clique em NEXT.



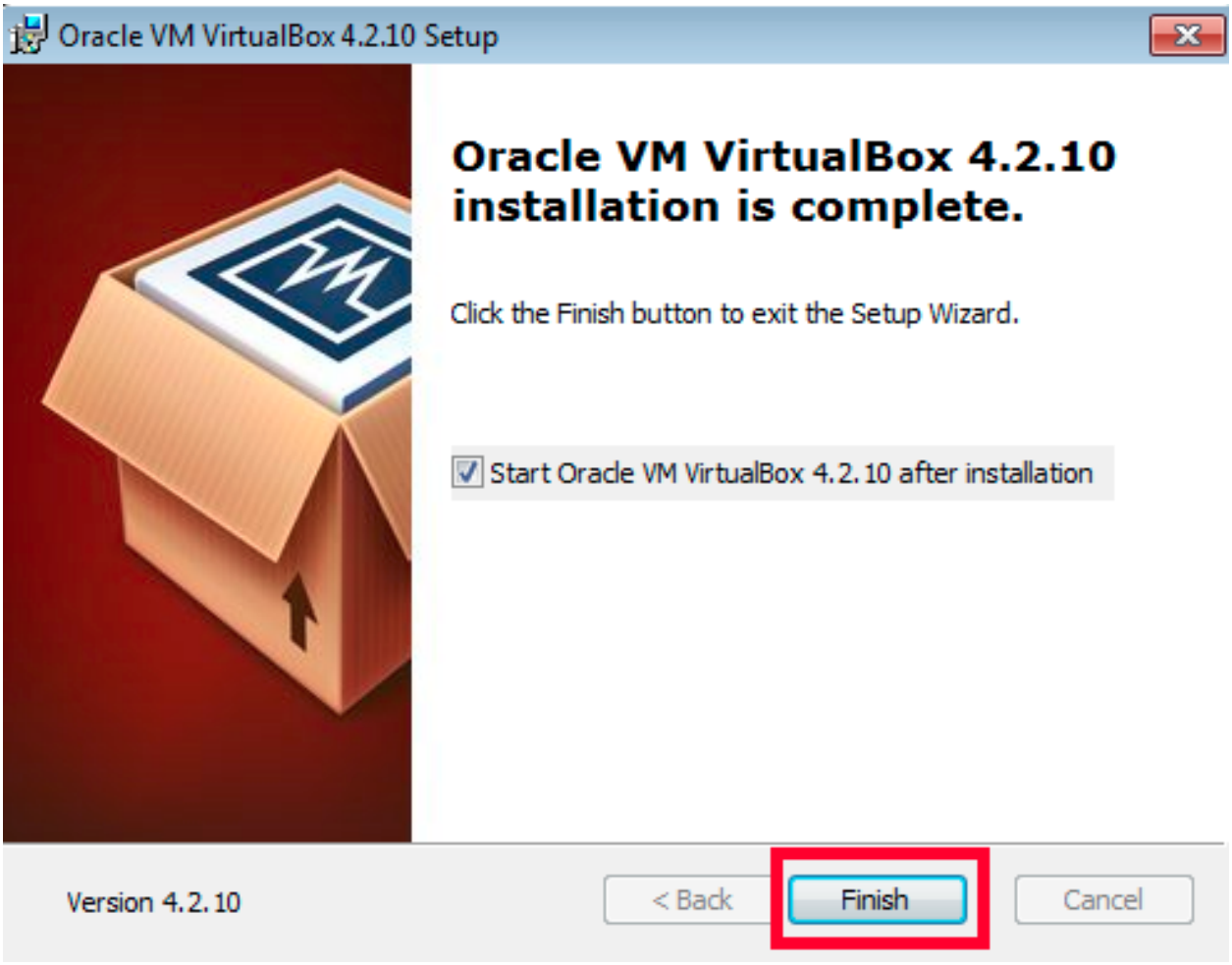
Essa é uma aba de atenção. Indica que para prosseguir com a instalção você será disconectado da internet. Salve o que for preciso e clique em NEXT.



Finalmente, estamos prontos para iniciar o real processo de instalção. Clique em NEXT.

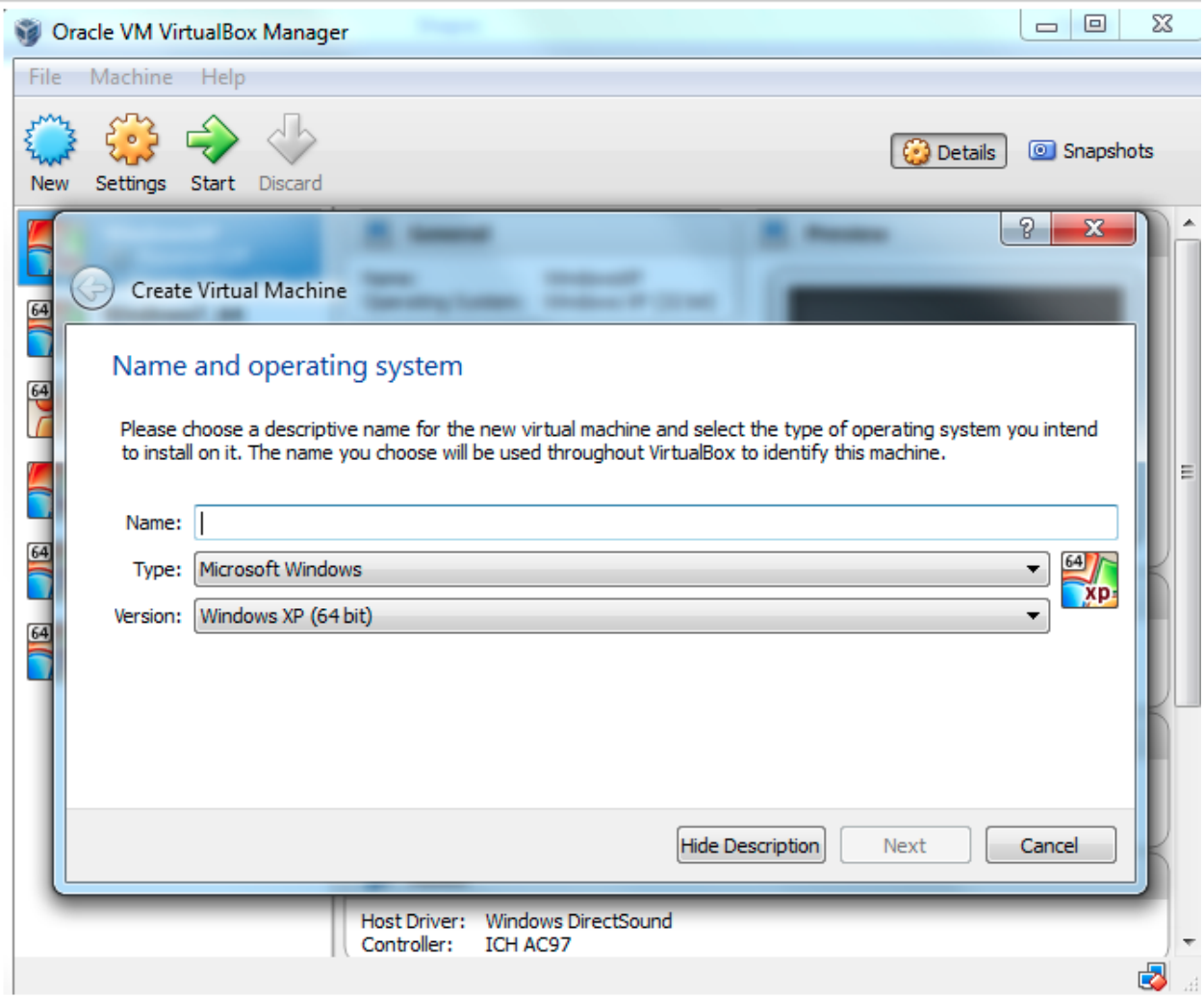


Sua intalção está completa. Clique em FINISH.



INSTALAR UBUNTU NO VIRTUALBOX

Abrir o Oracle VM VirtualBox Gerenciador e clicar em Creat Virtual Machine.

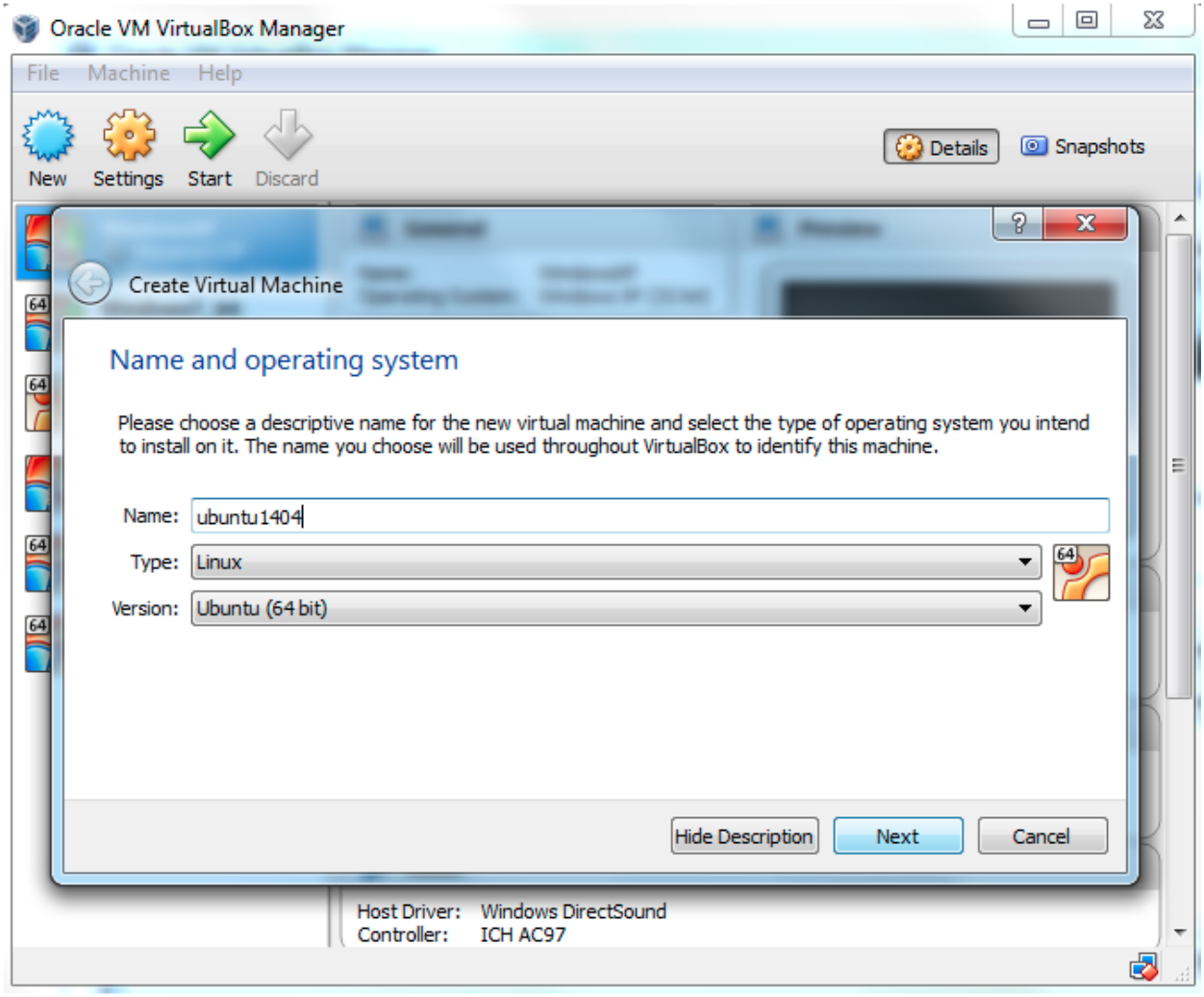


Criar:

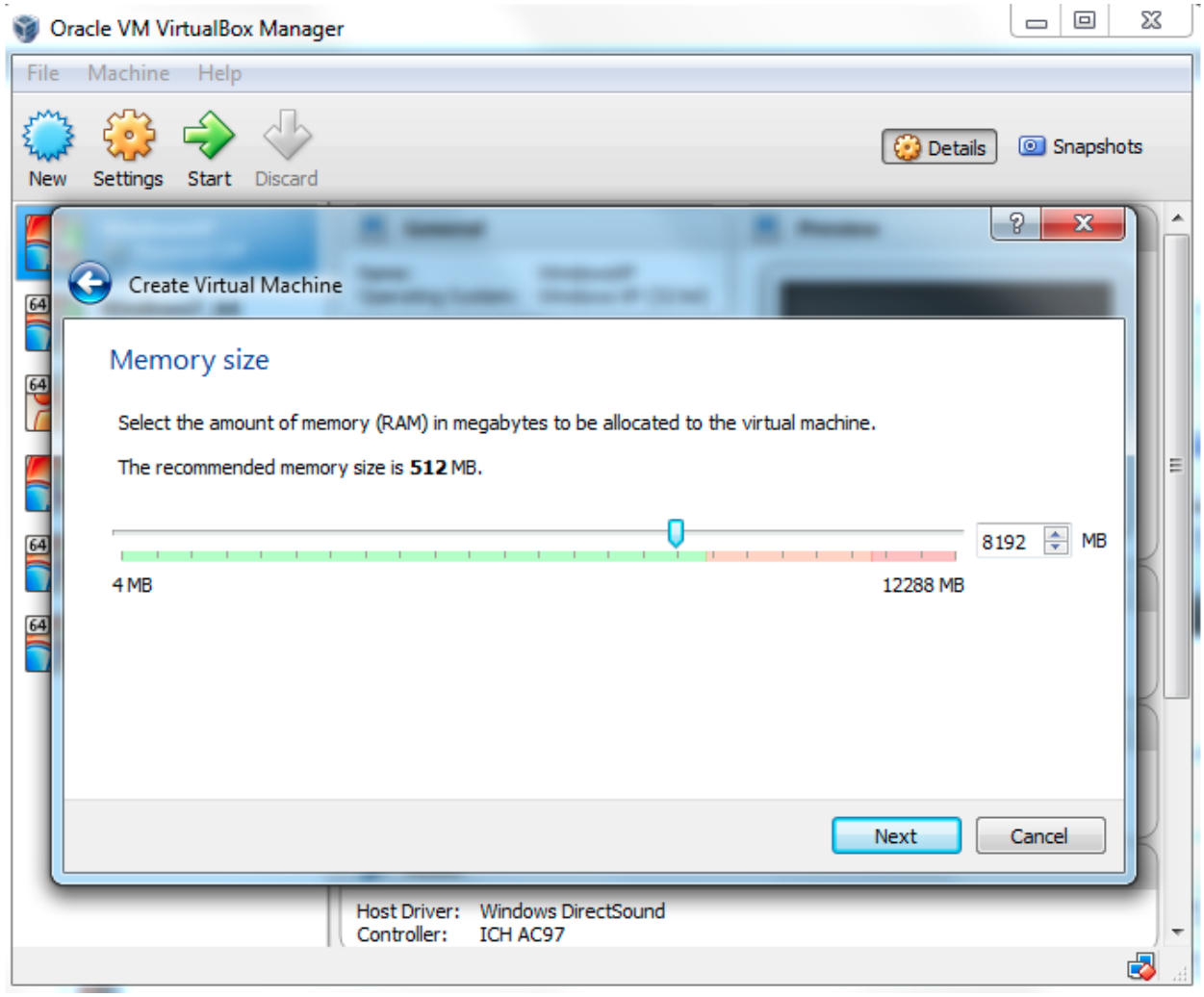
Nome:IntroComp

Tipo:Linux

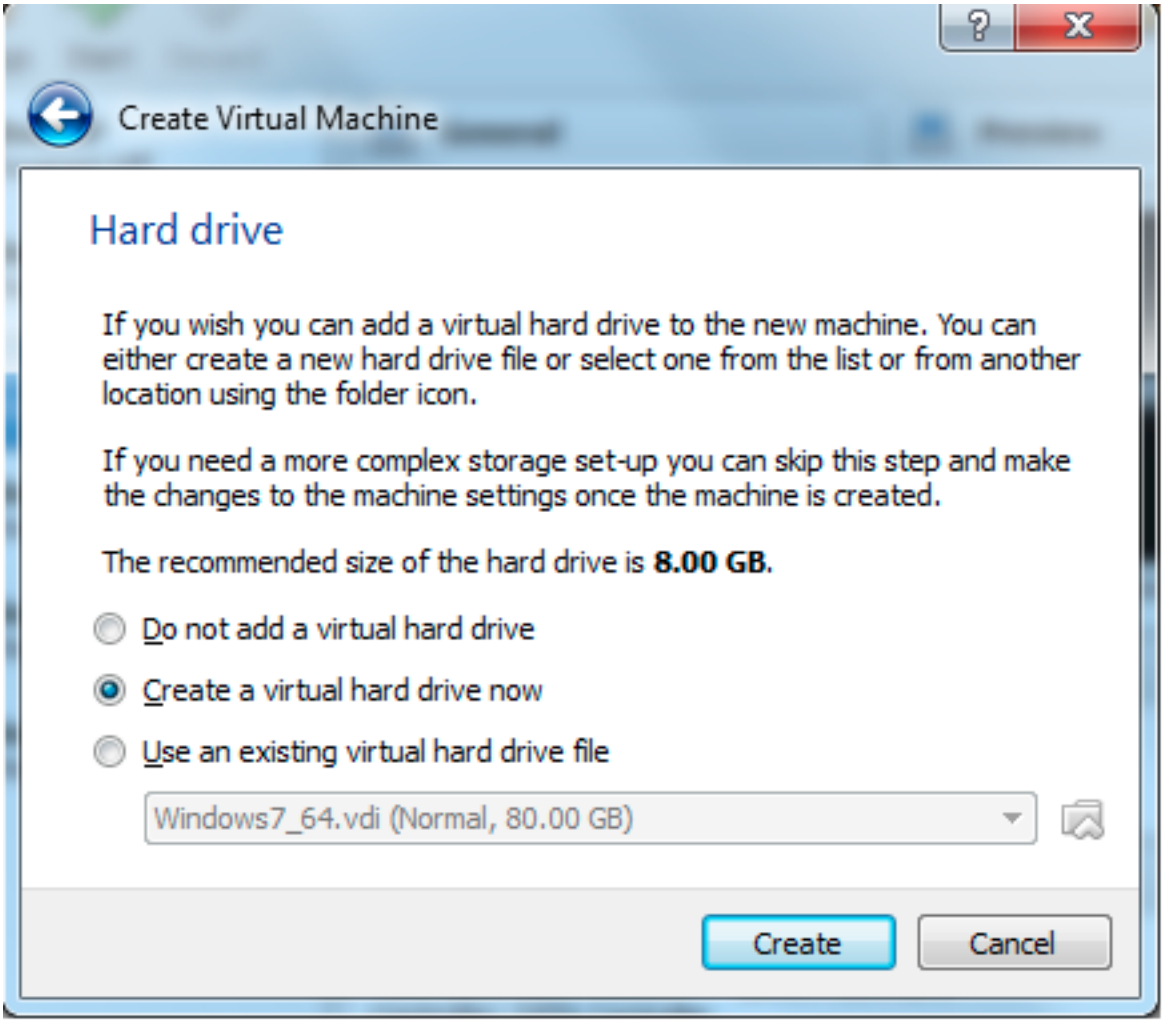
Versão:Ubuntu (64-bit)



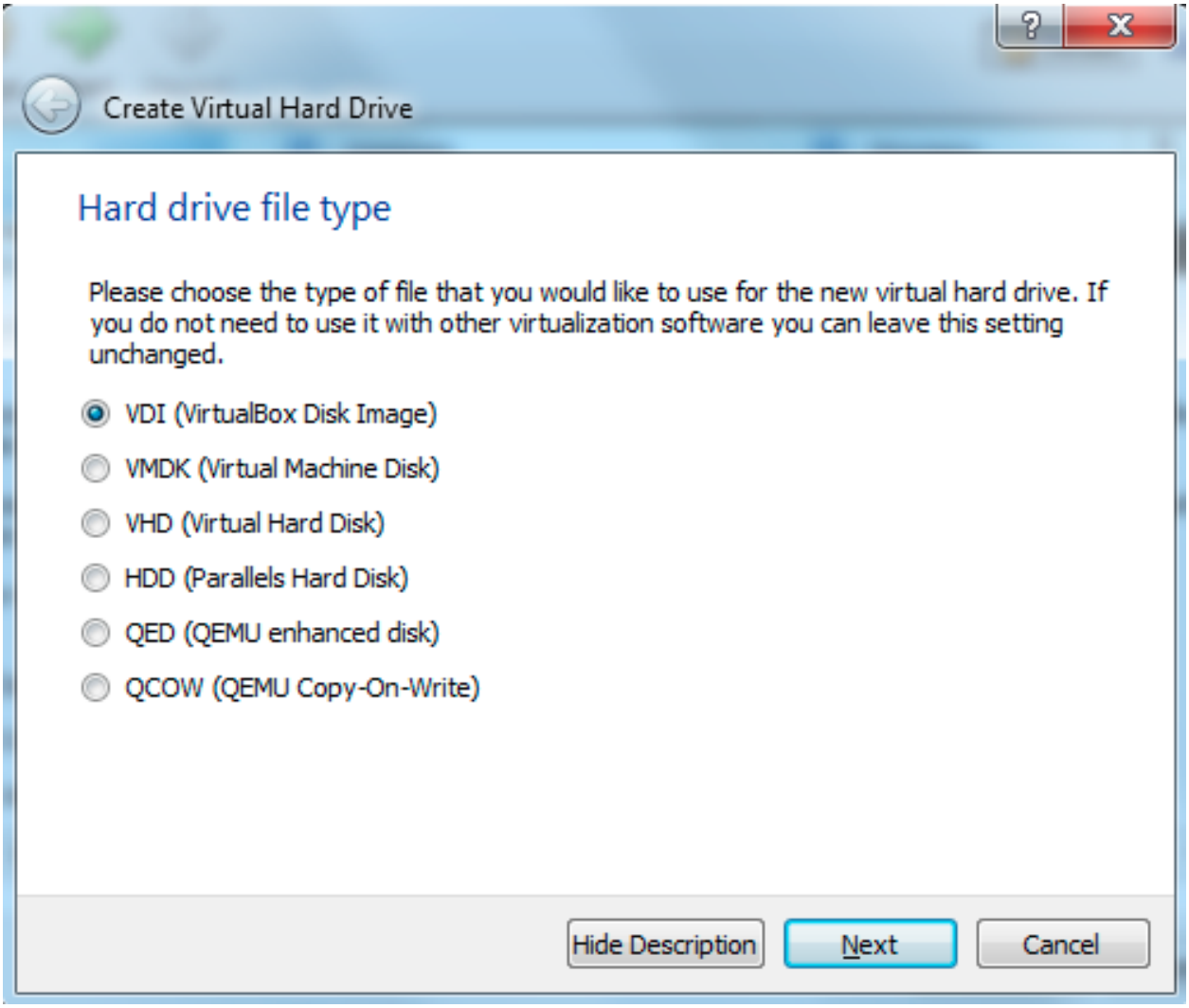
Selecioinar tamanho da memória: 2048.



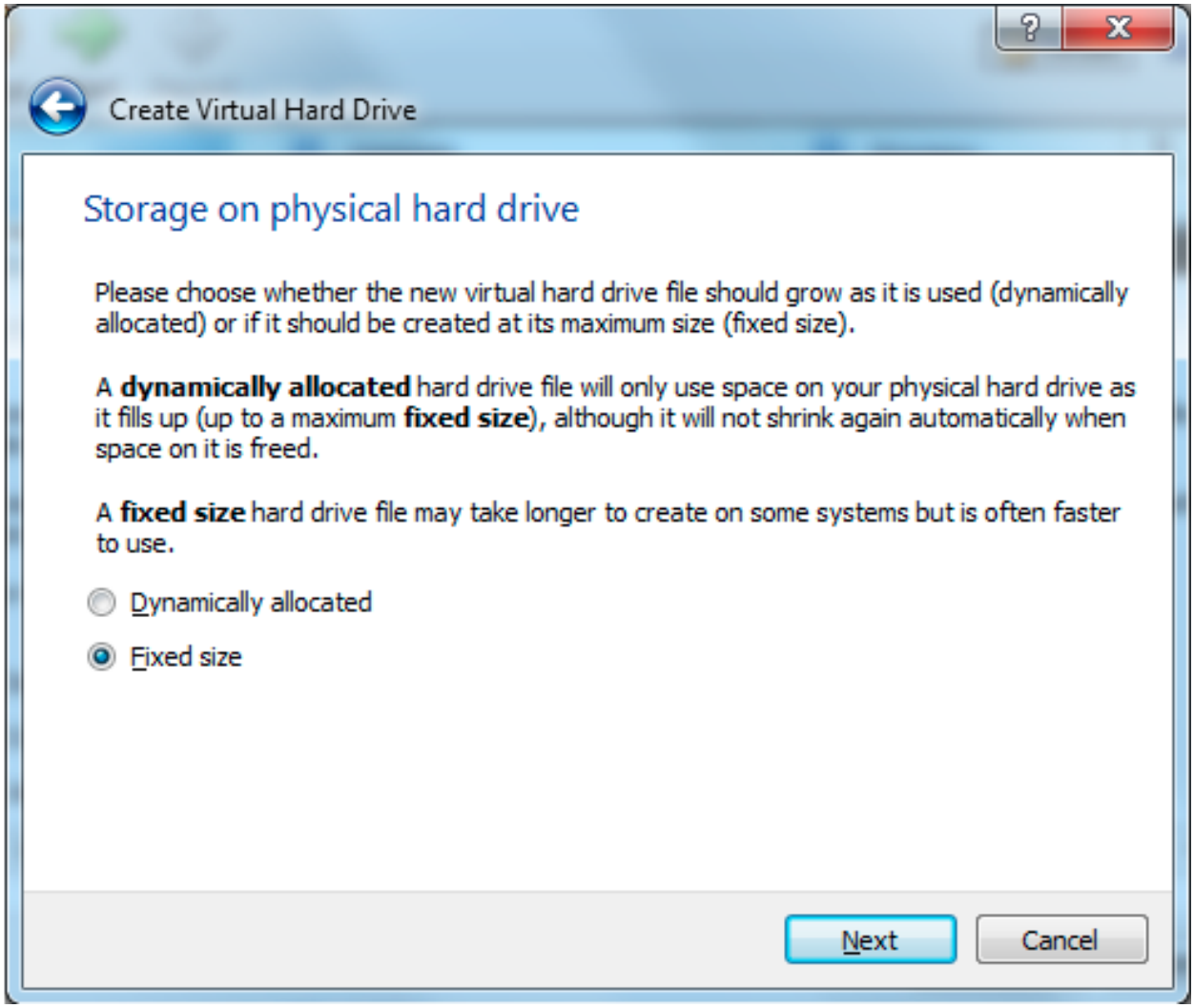
Criar um novo.



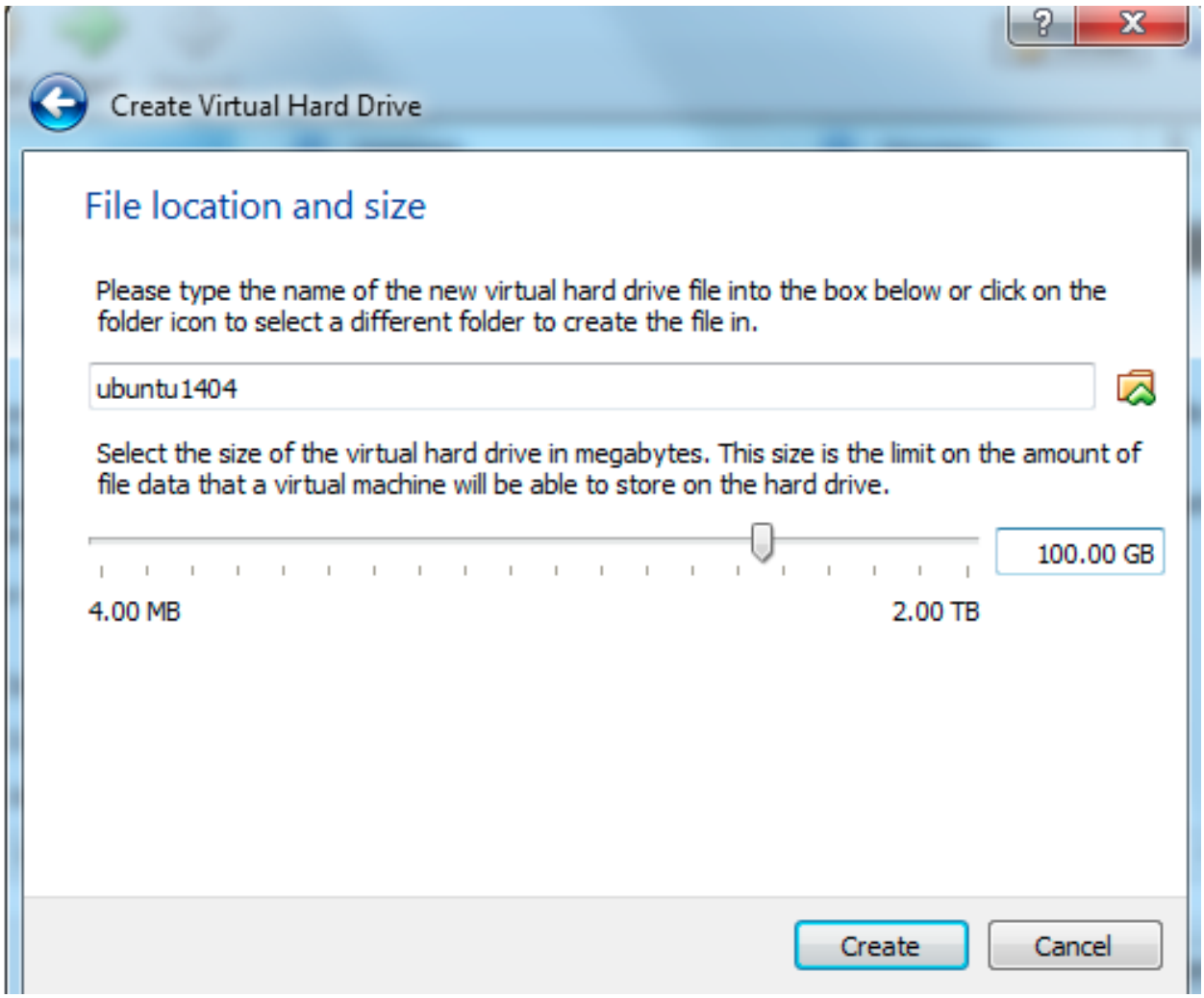
Selecionar: VDI.



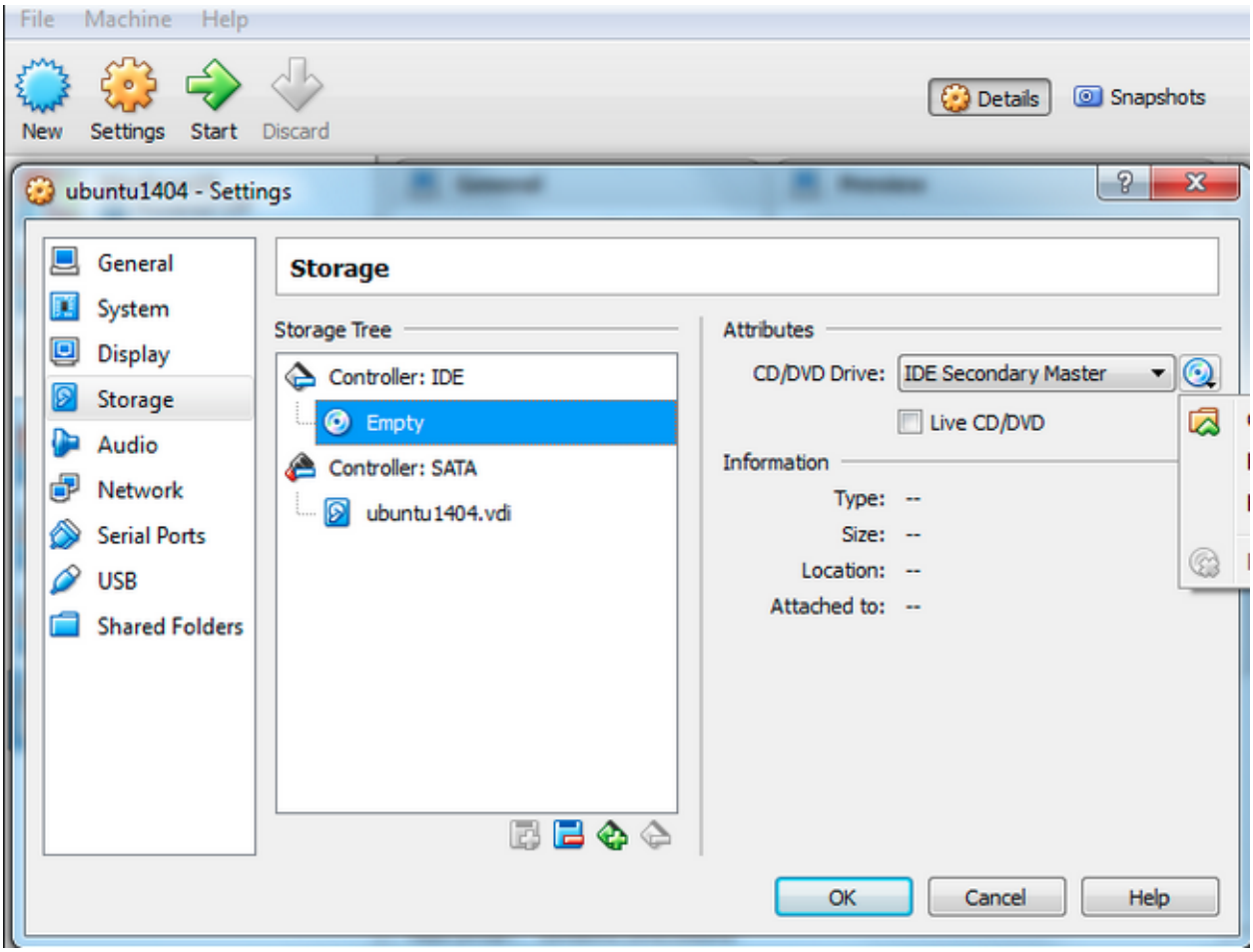
Selecionar: Dinamicamente Alocado.



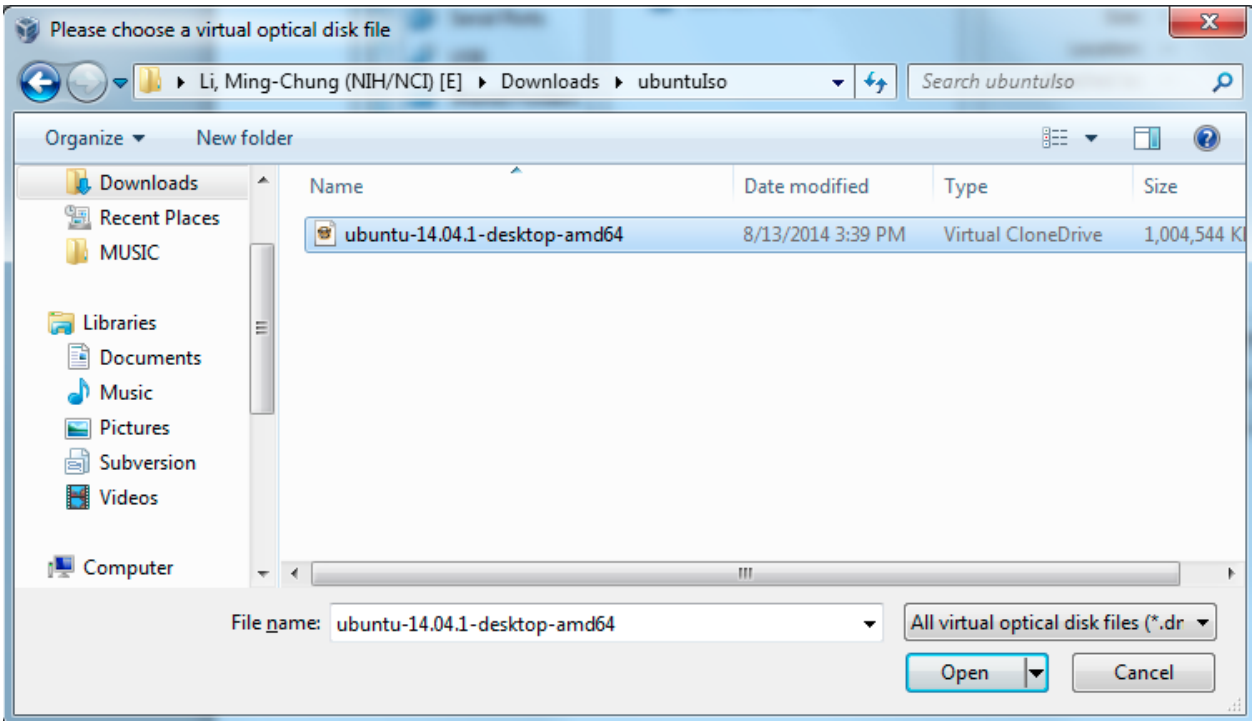
Localização e tamanho do arquivo 15,12



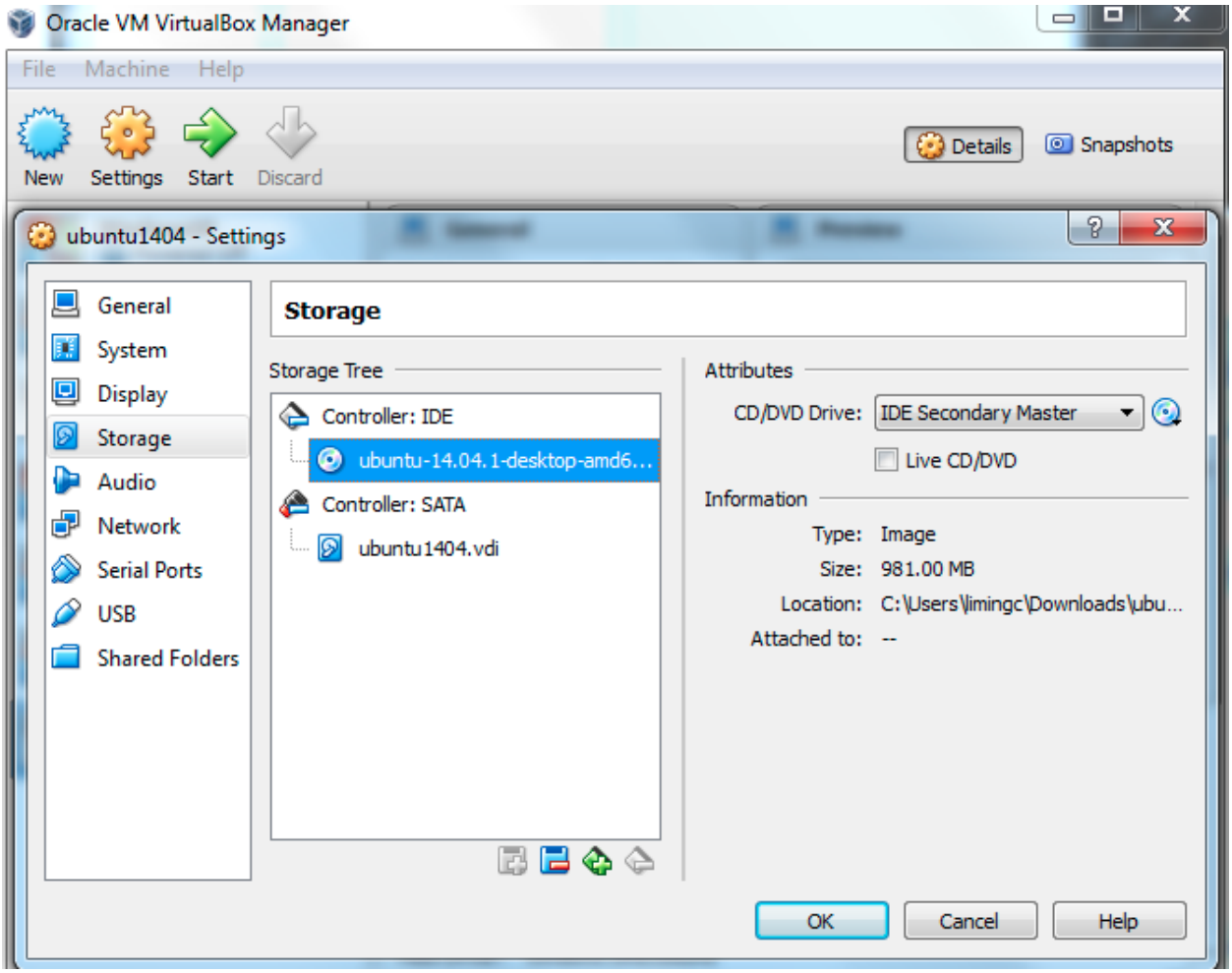
O Armazenamento estará vazio clique no Ide secundário Master.



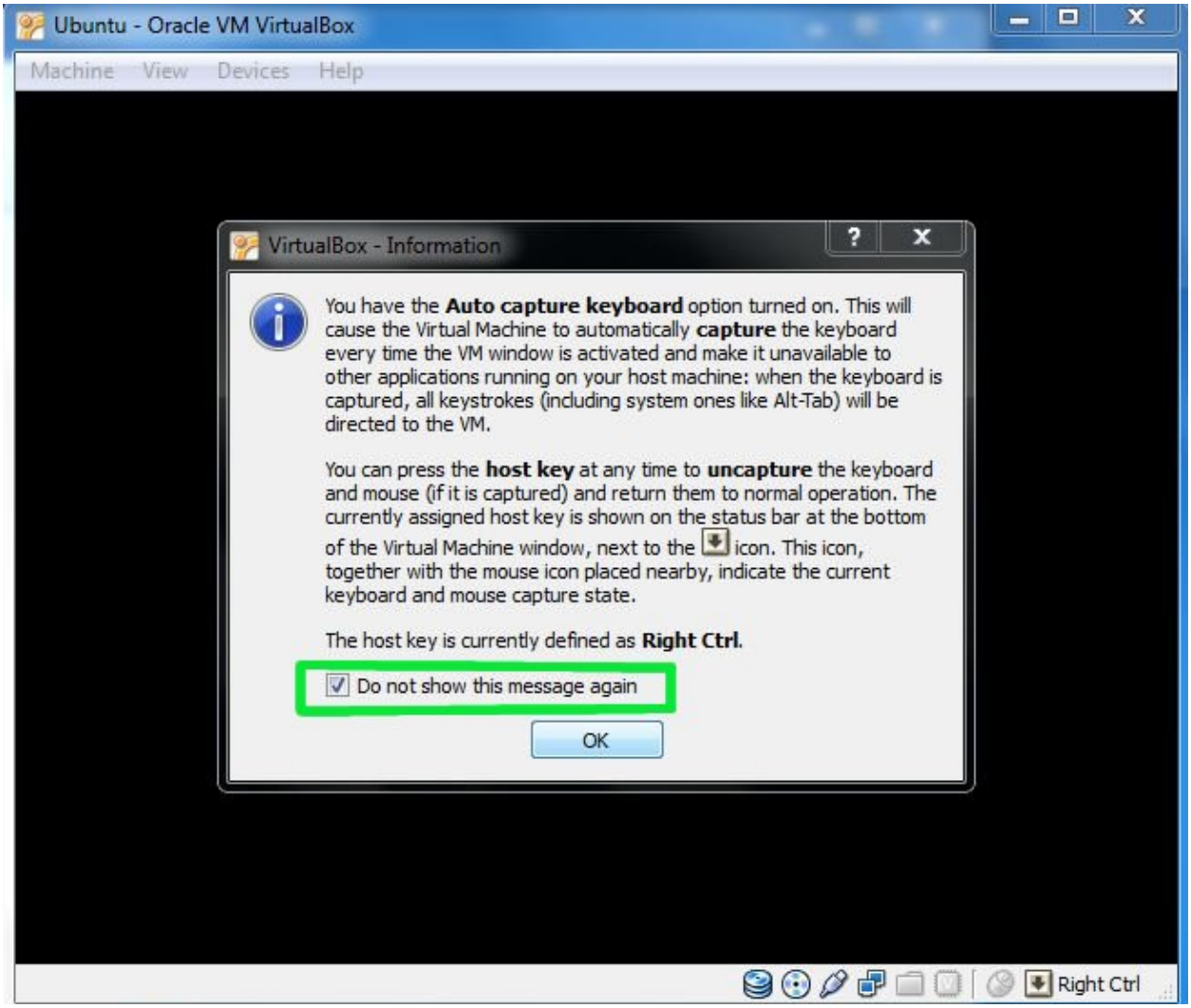
Escolha o arquivo do Ubuntu 17.10 que você baixou do site para seu computador.



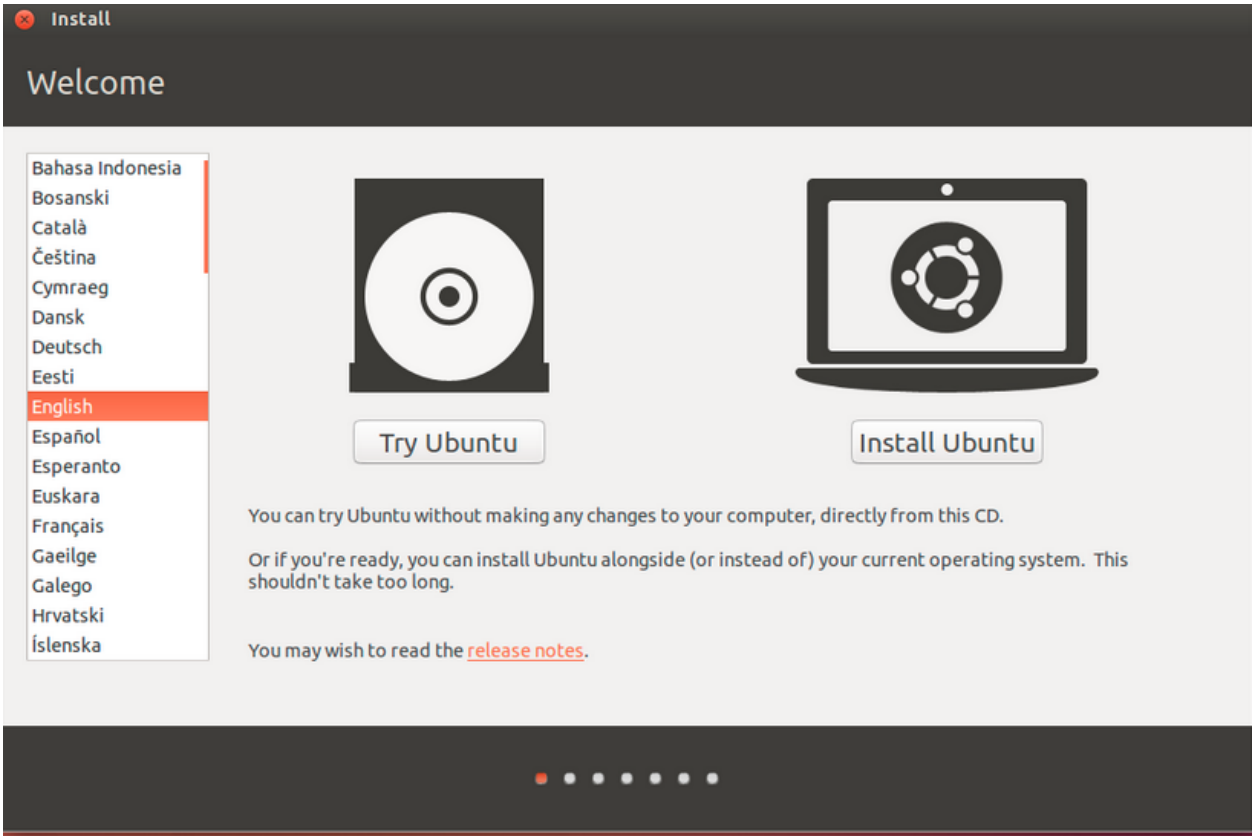
Observe se na imagem de CD aparece o nome do Ubuntu. Se sim, clique seta verde.



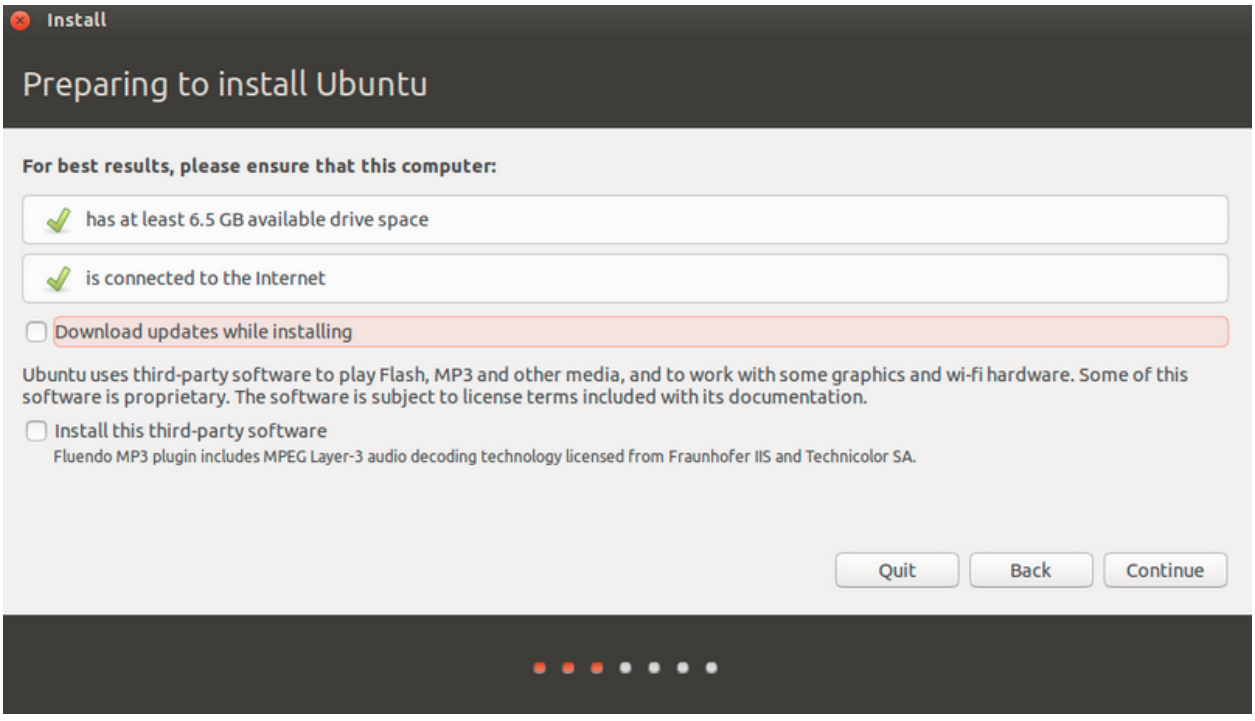
Remova a janela pop up, clicando no OK.



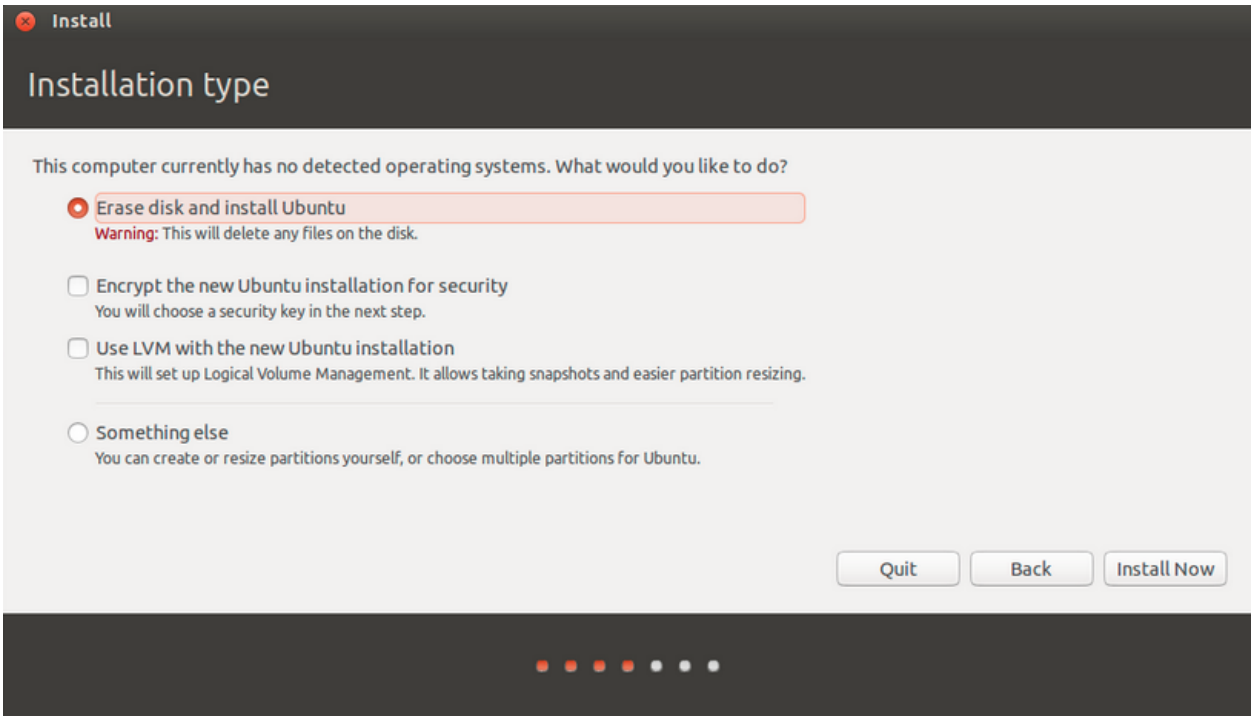
Espere alguns segundos até abrir a janela de instalção do Ububtu. Clique em instalar ubuntu.



Selecione Instal Updates e clique em CONTINUE

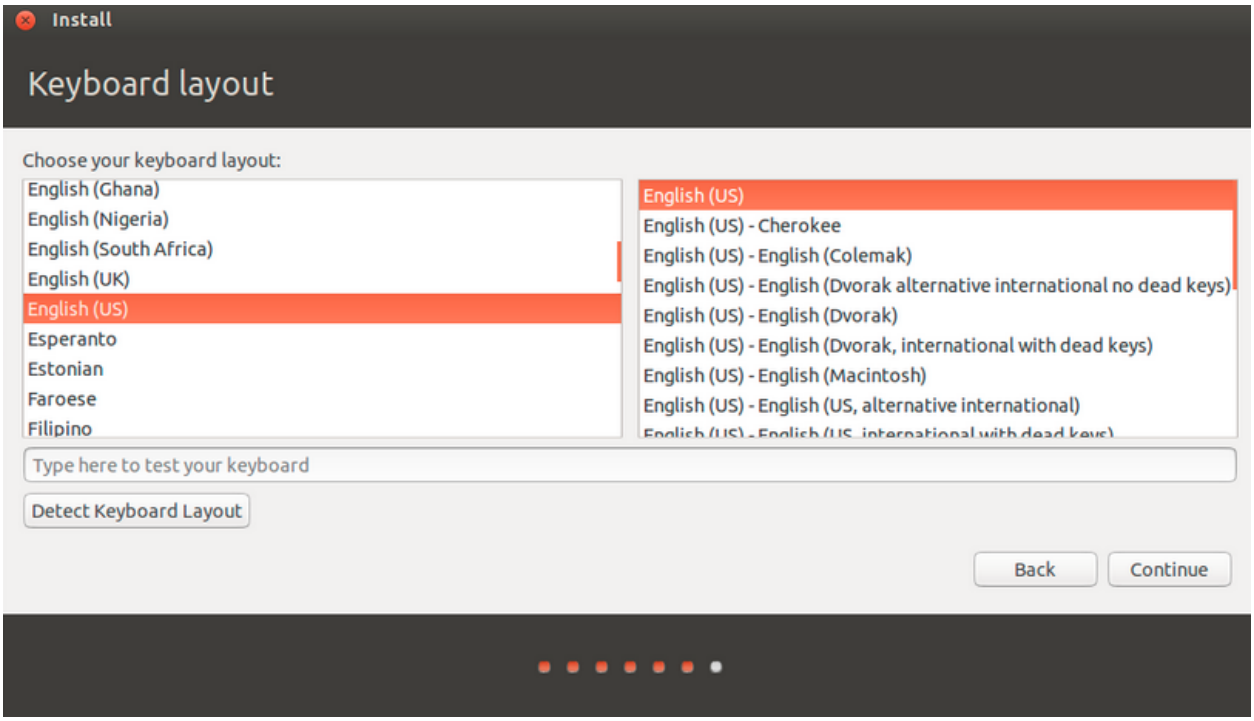


Selecione a primeira opção que é apagar o disco e instalar o Ubuntu. Clique em CONTINUE.

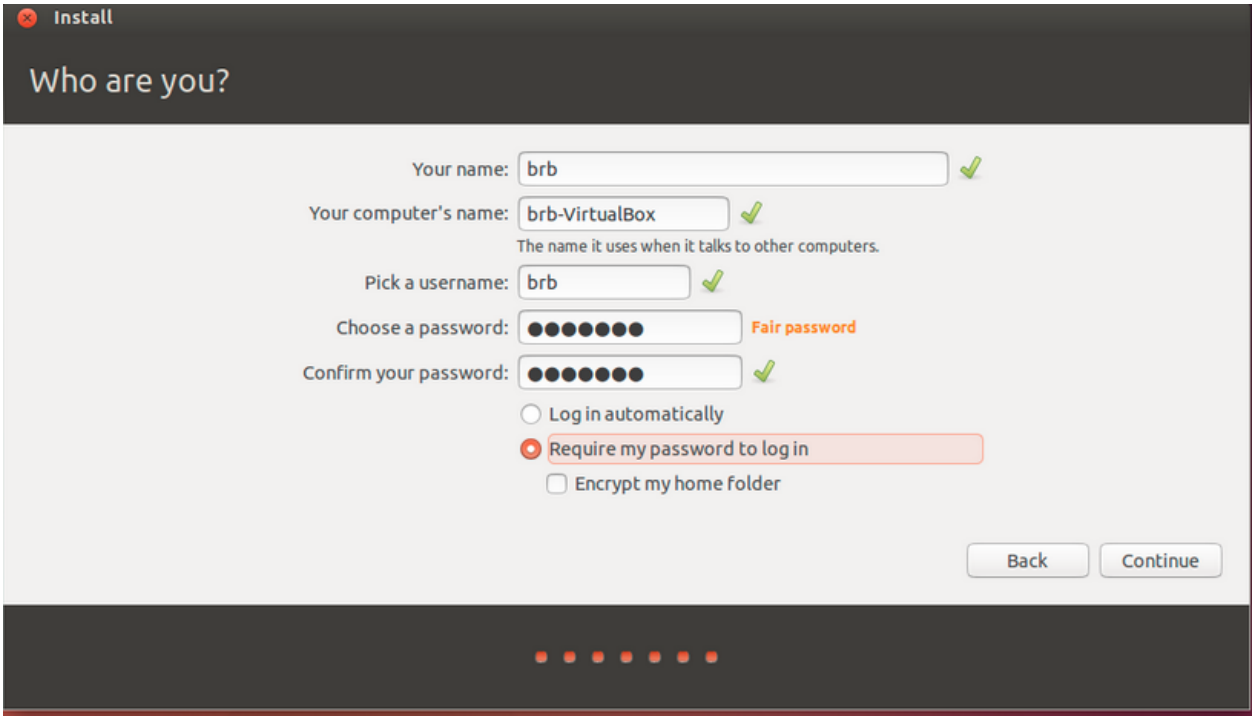


Escolha sua Cidade.

Escolha sua língua falada e teclado.

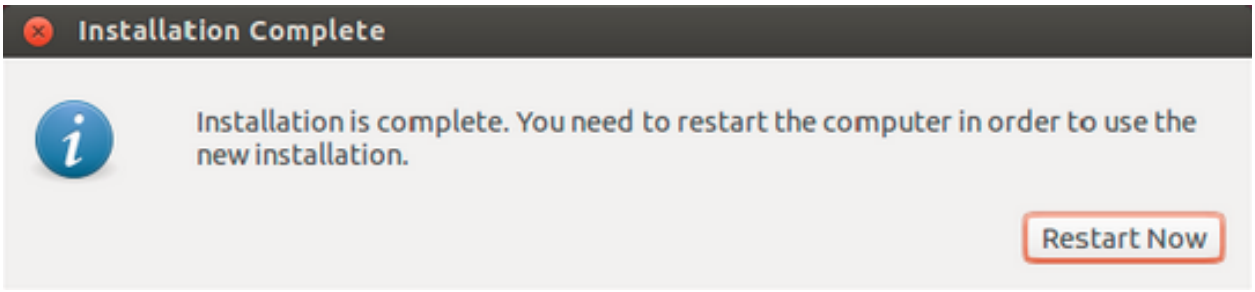


Na aba “Who are you” crie respectivamente: seu nome , o nome do seu computador, seu login, sua senha, confirme sua senah e selecione requerer senha.



Espere seu sistema instalar (pode ser que demore um longo tempo).

Dê OK na aba e seu Ubuntu estará instalado no seu VirtualBox.

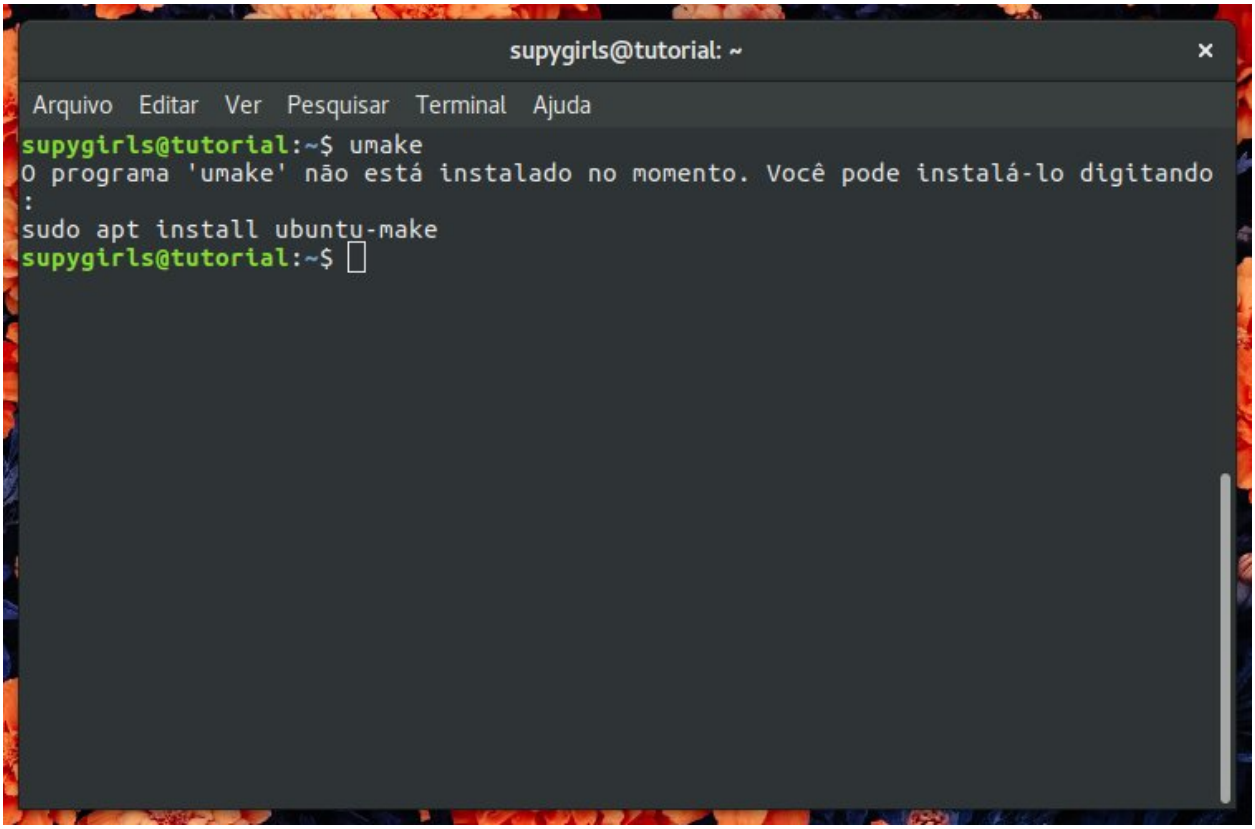


INSTALAR UMAKER

Instalador profissional de ferramentas de desenvolvedor - idepycharm

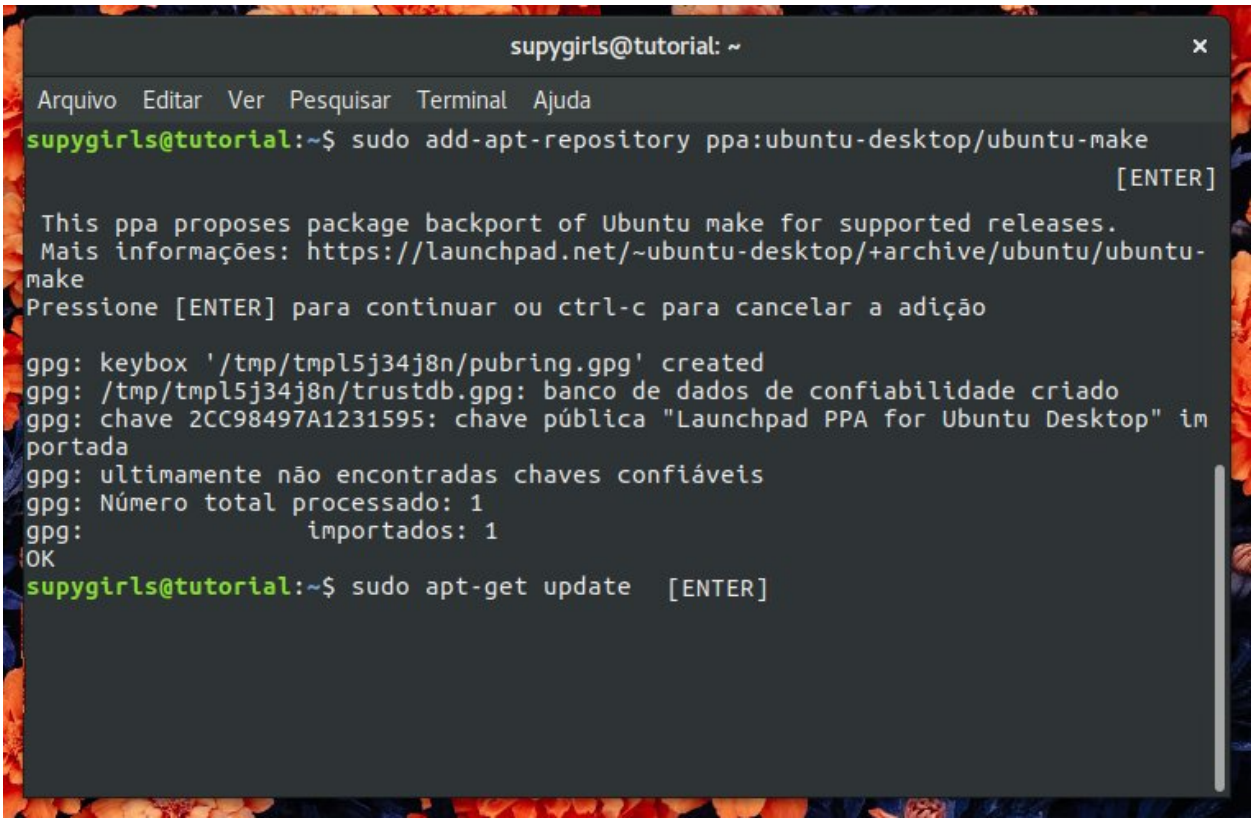
Terminal abre: Ctrl+Shift+t

Escreva o nome do programa: umake A informação gerada especifica se há o “ubuntu-maker” instalado no seu computador.

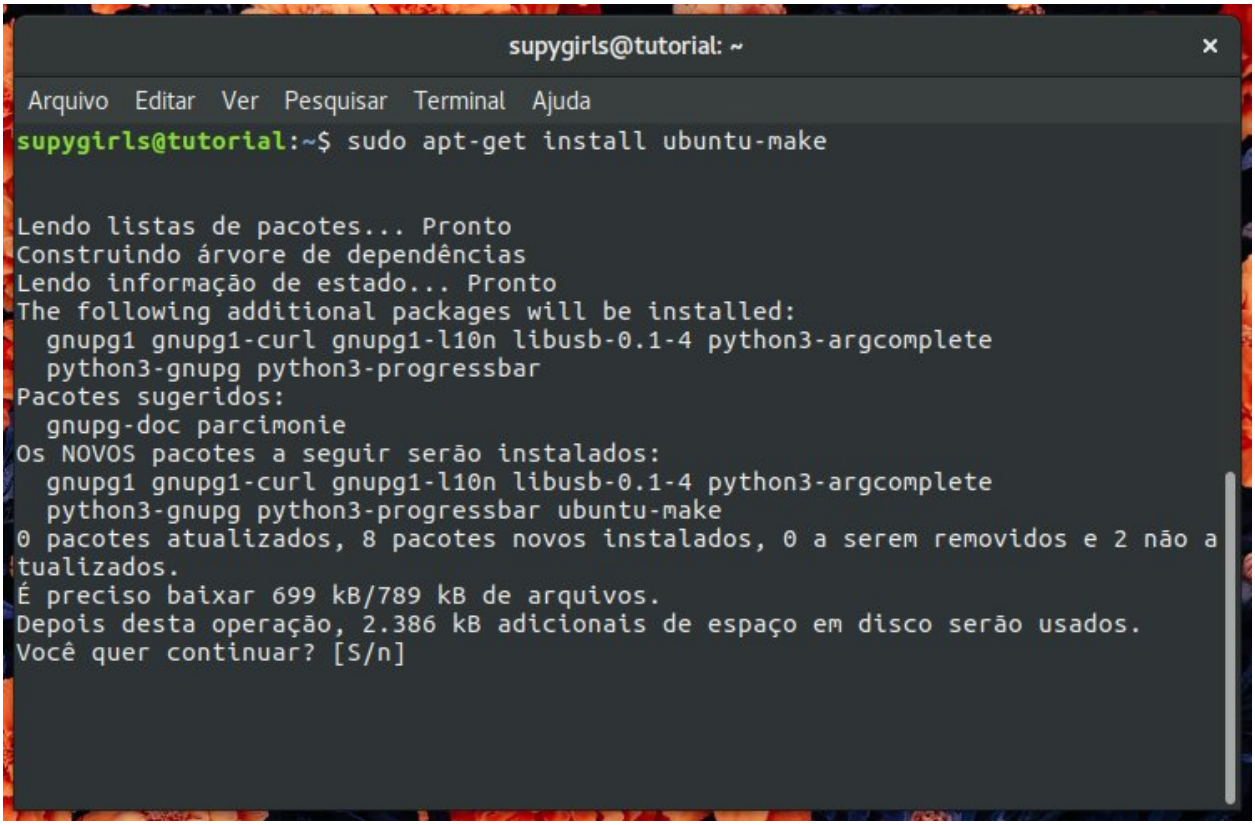


Atualize a biblioteca de repositórios

Digite: Sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make Sudo apt-get update

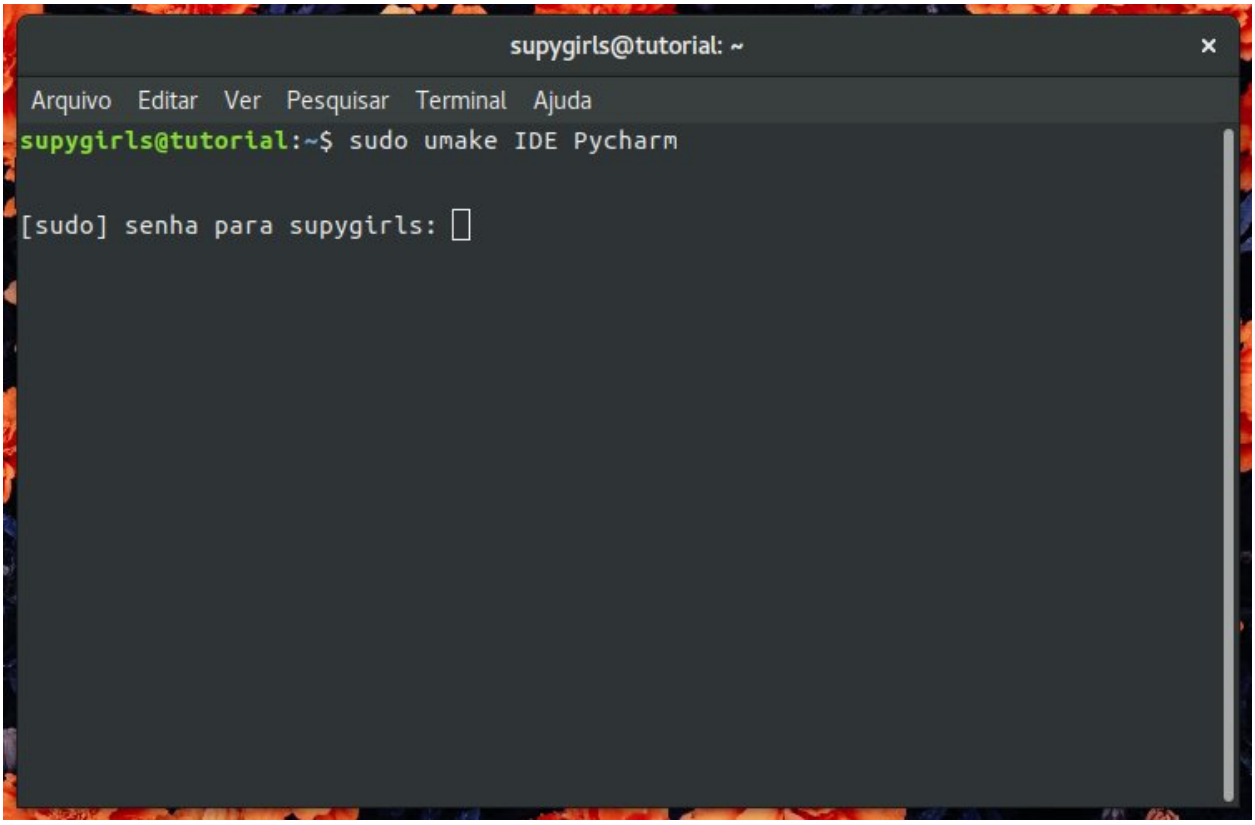


Digite: sudo apt-get install ubuntu-maker sudo(significa que vc é administrador e tem permissões diferenciadas) apt (pacotes do ubuntu) install chama o processo de instalação



INSTALAR PYCHARM

Digite: umake ide pycharm Insira a senha e pressione enter



Concluiu a instalação desligar janela de execução e iniciar seta verde.

1.1.2 CRIAR CONTAS

CONTA PROJETO PYCHARM

Início de projeto: file: settings project interpreter configuração create virtual environment Python3.5 Marcar: Inherit global site-packages Name: Marcar (No) Name:SuPyJogo OK

CONTA NO GITHUB

Username:

email:

password:

continue continue

participar do SuPyPerson Inca

CONTA SLACK

O que estamos fazendo: Formando um time profissional de desenvolvimento. Não usa windows pq tem muito virus. Ubunto profissional

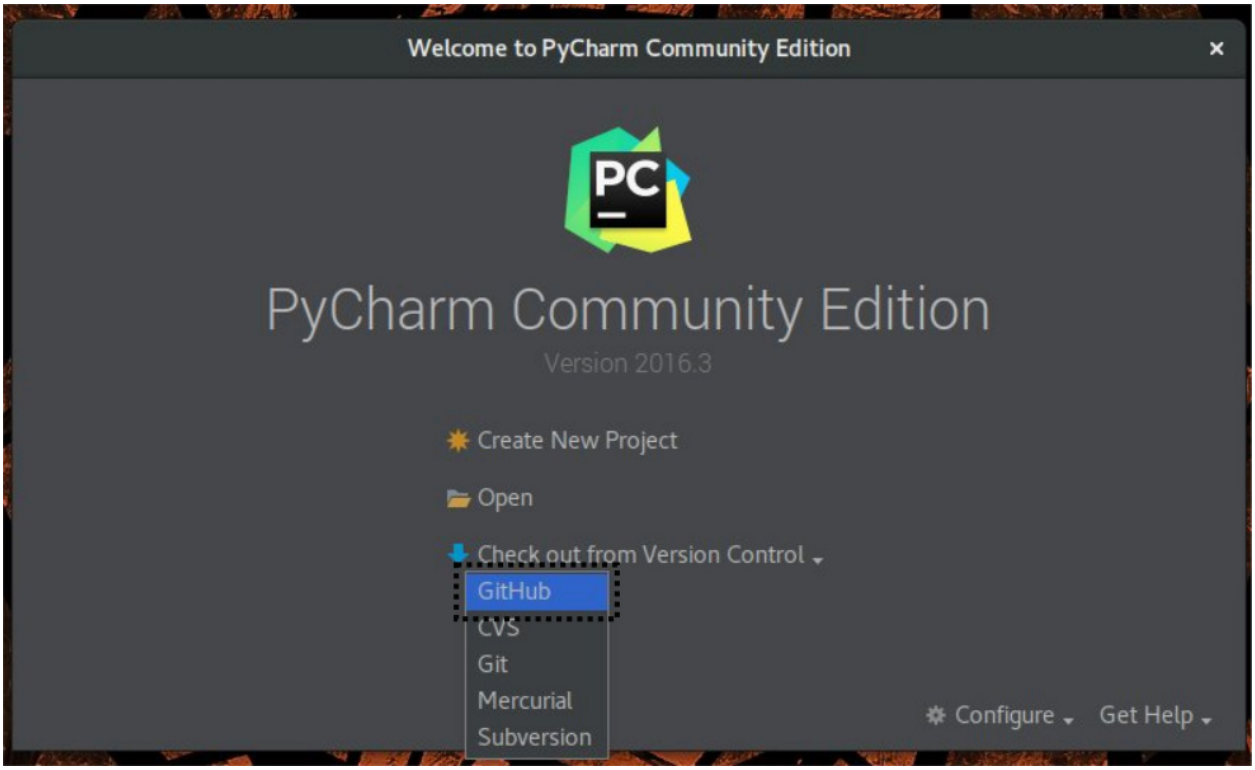
CONTA WAFFLE.IO

1.1.3 AÇÕES ENTRE PYCHARM E GITHUB

SINCRONIZANDO GITHUB E PYCHARM

Abra o IDE Pycharm e prima > Check out from Version Control <

Ao abrir as opções, prima > GitHub < para resgatar e logar na plataforma.

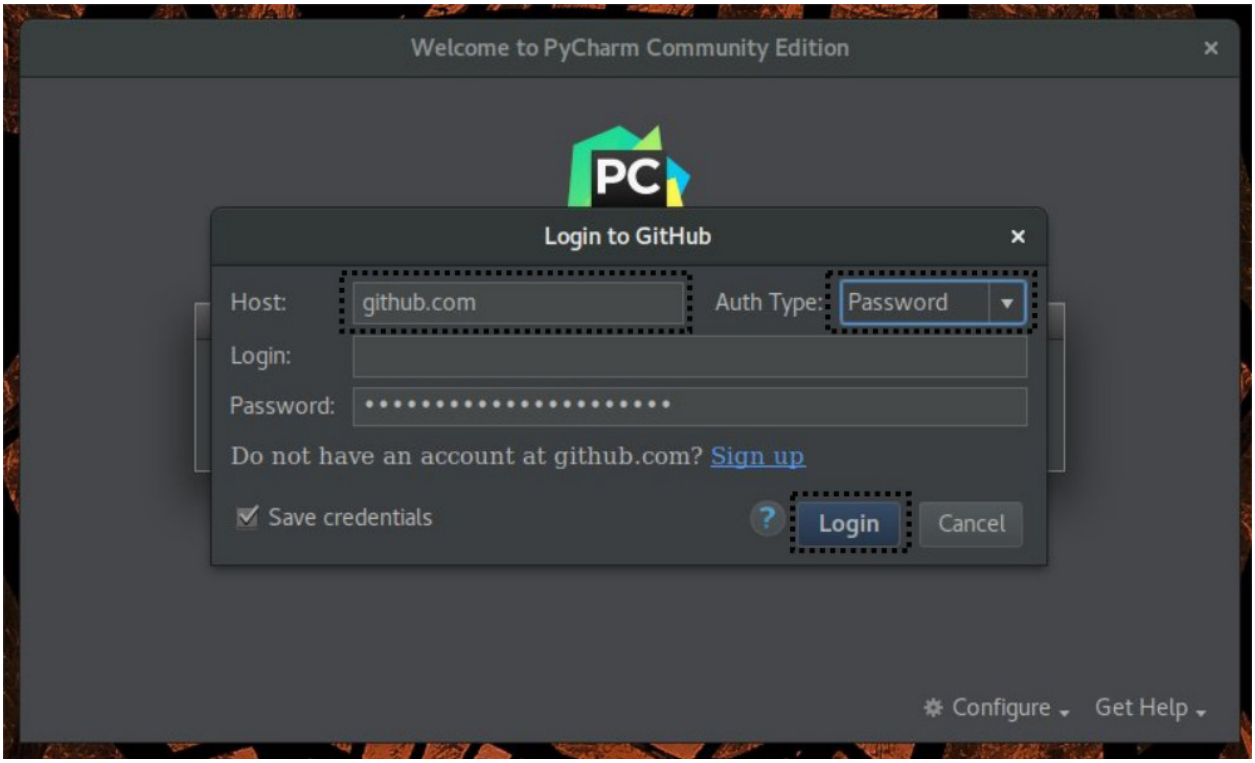


Insira suas informações de usuário e prima >Login<

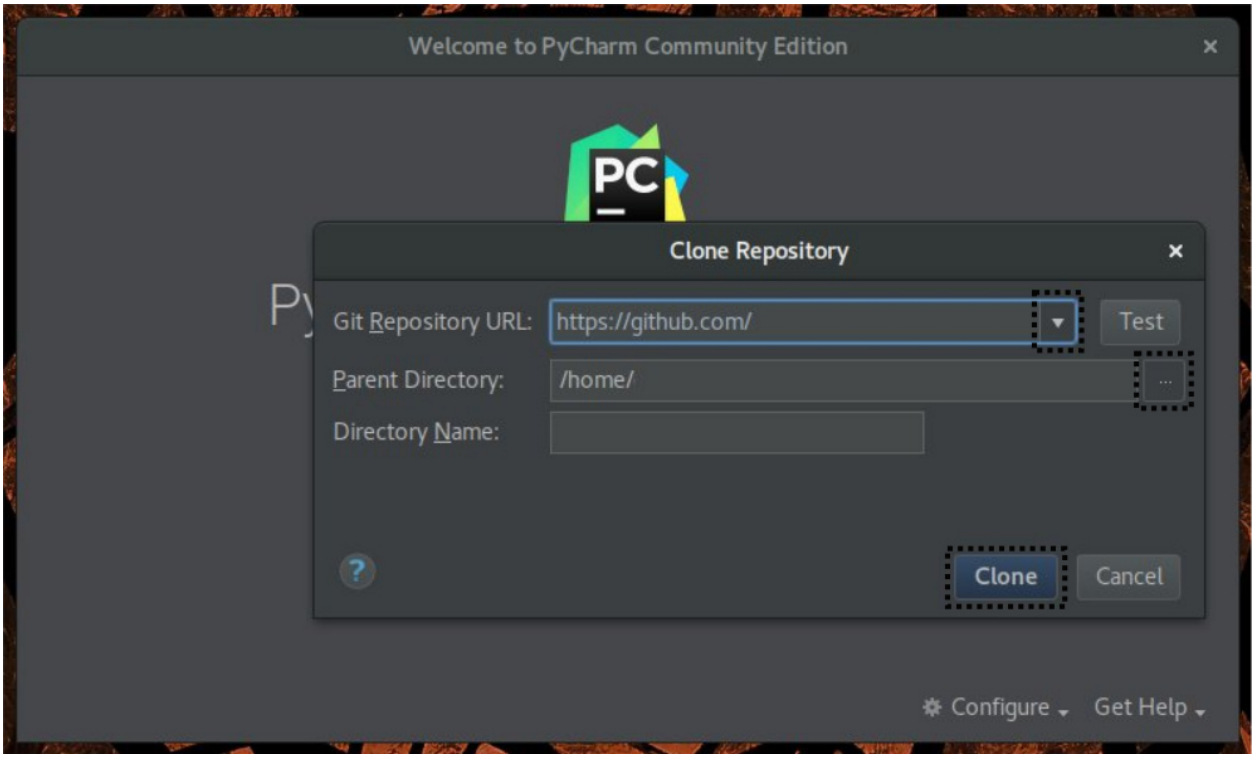
Atenção:

Host: github.com

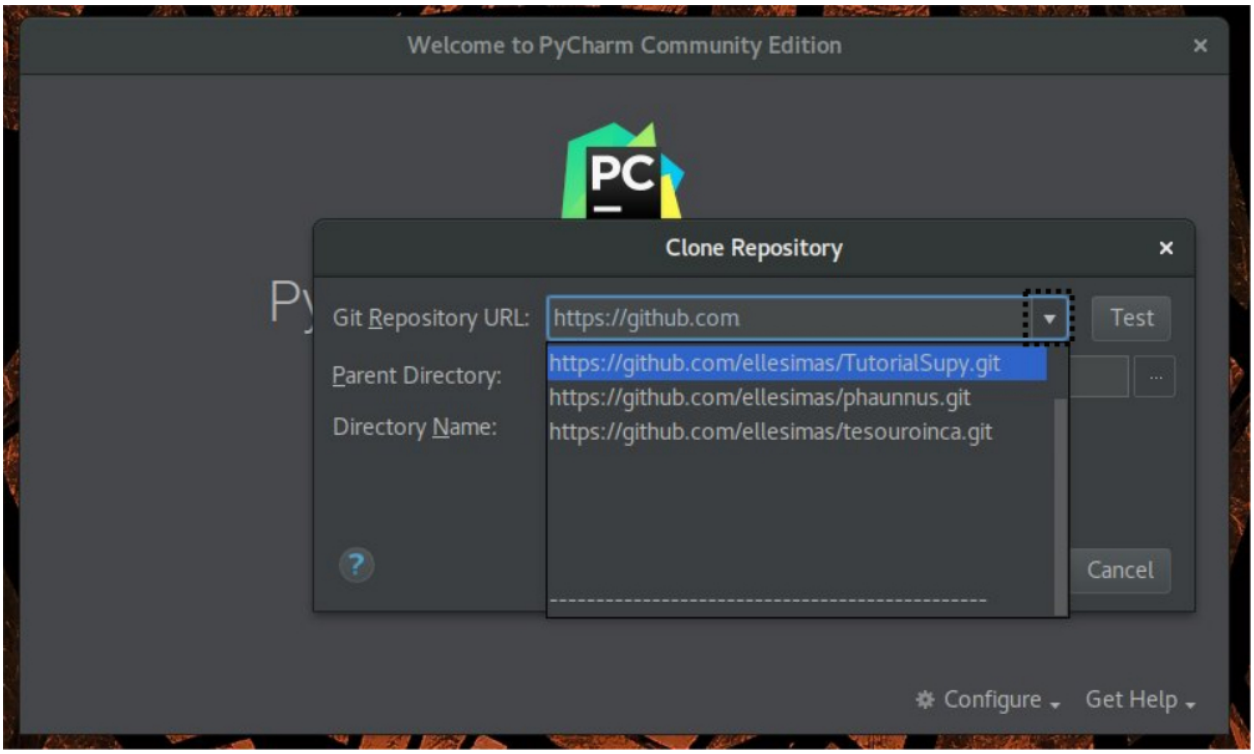
Auth Type: Password



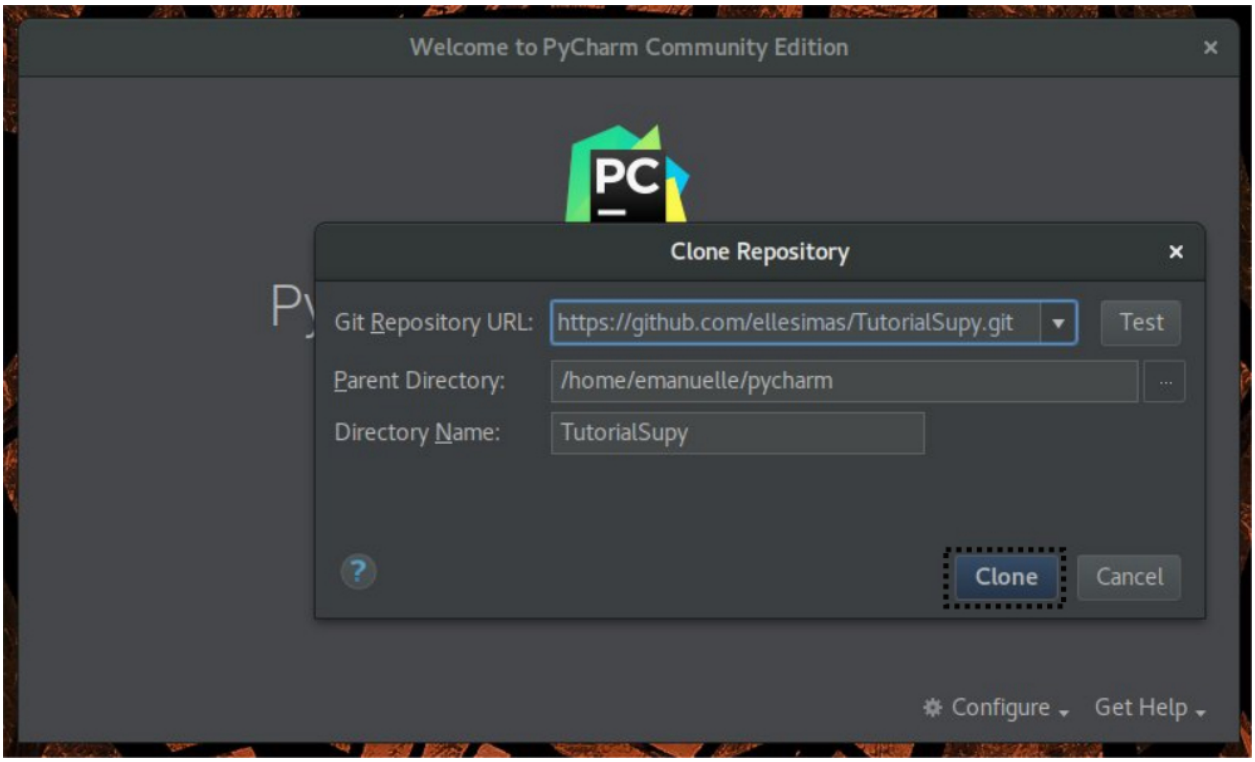
Na página seguinte, selecione o repositório do projeto >Git Repository URL< e a "pasta pai" >Parent Directory< -Veja Imagens- Feito isso, prima >Clone<



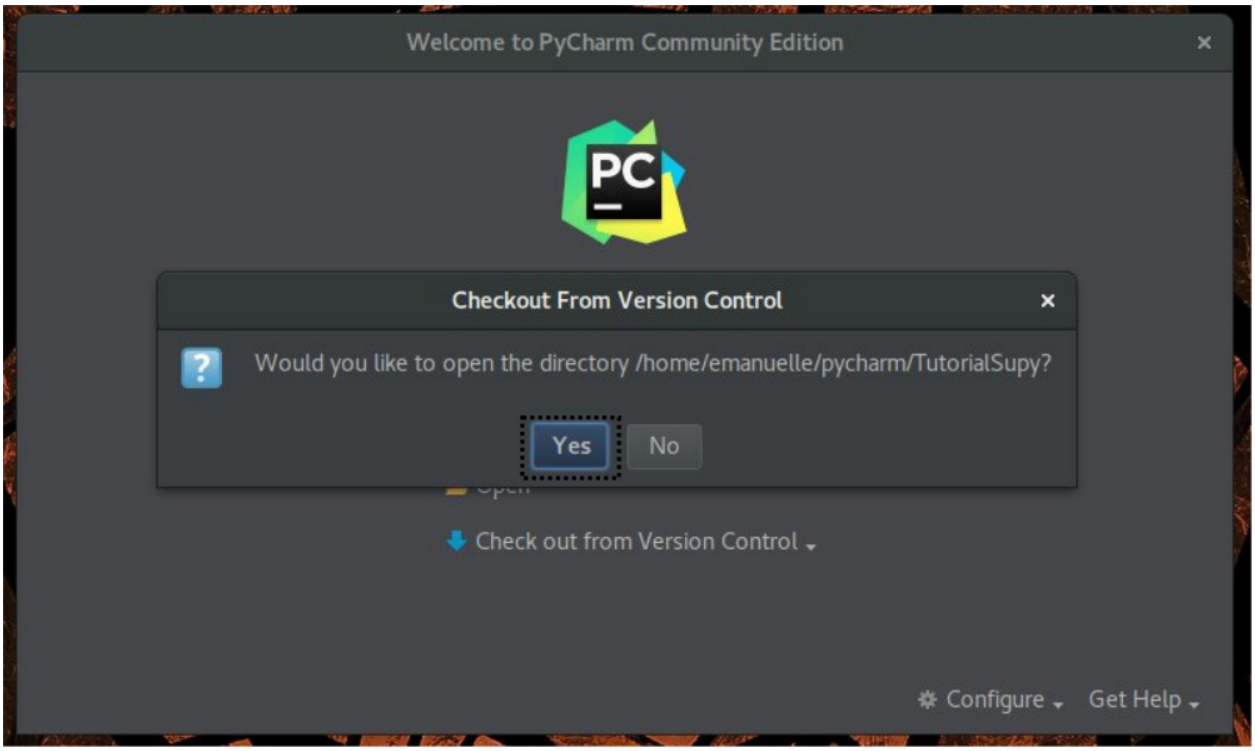
Selecione o repositório a sua escolha



Selecione a pasta "pai" de seu interesse.

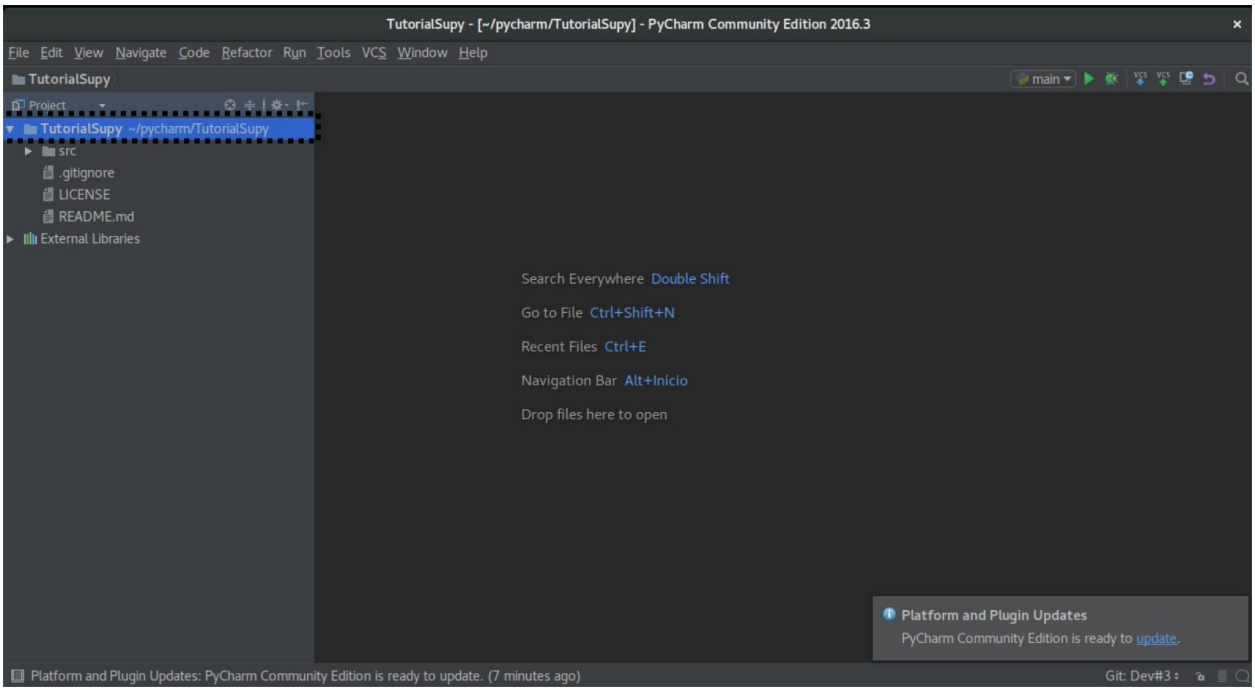


Abrirá um pop-up perguntando se você gostaria de abrio o diretório. Clique no sim.

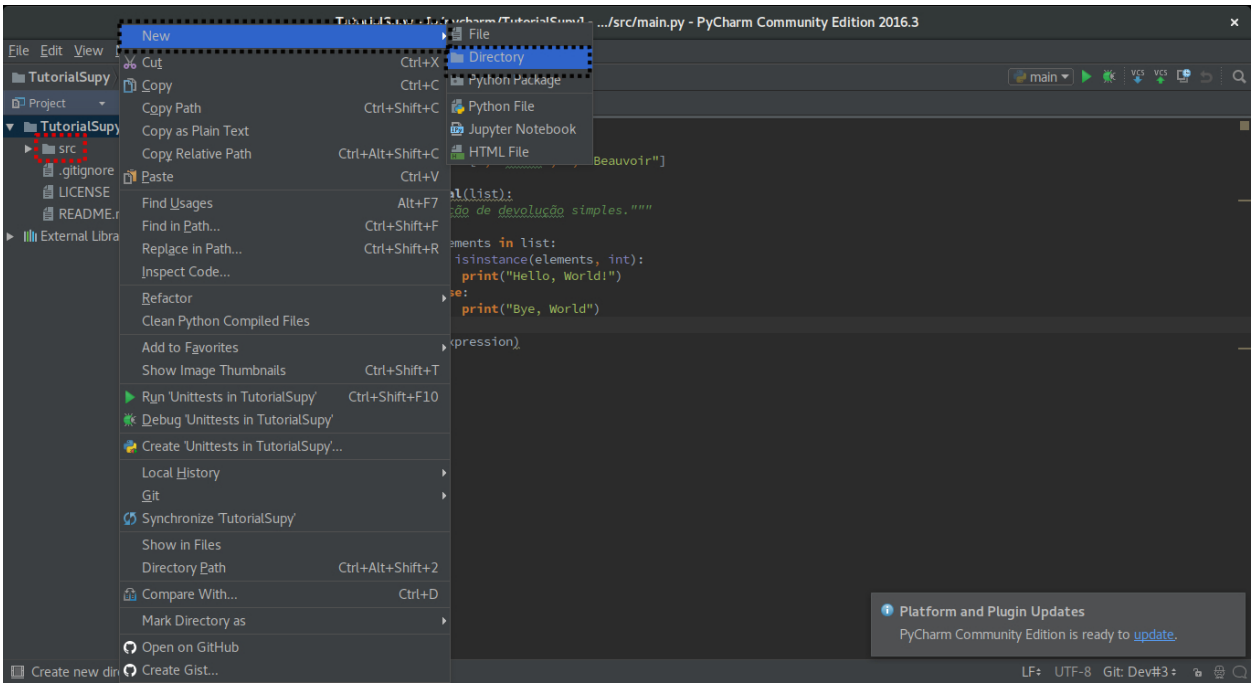


COMO ABRIR O PROJETO NO PYCHARM

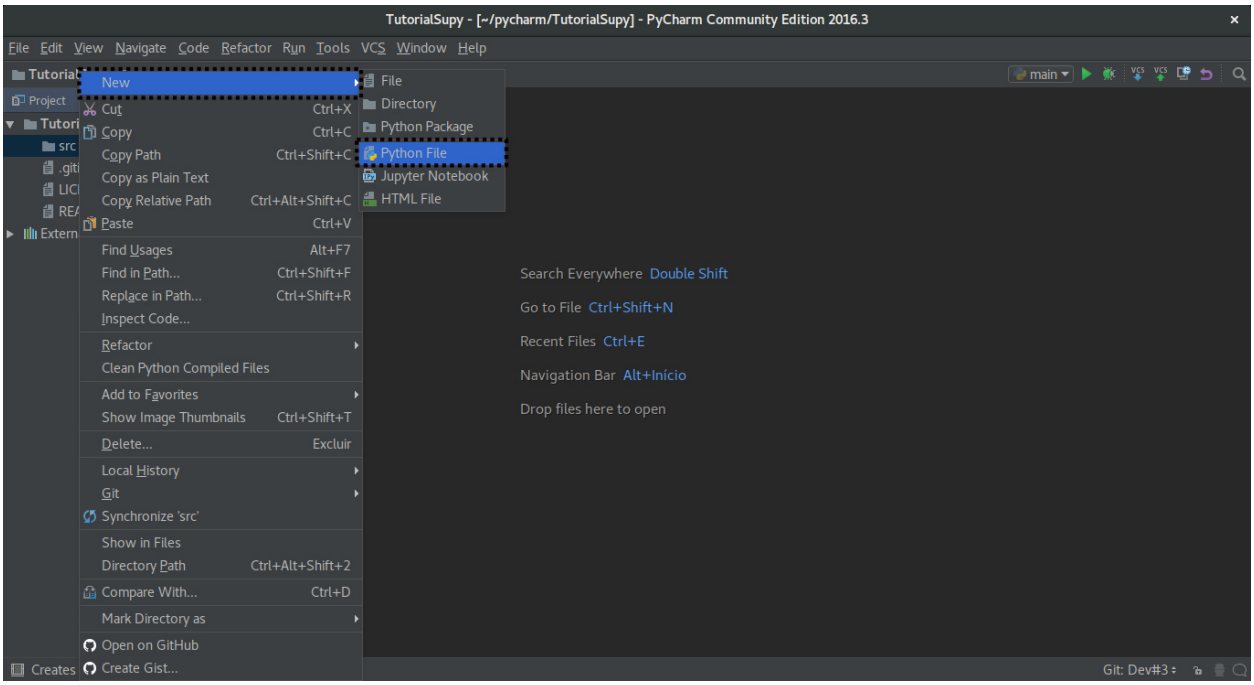
Observe o lado esquerdo do seu cursor e clique na primeira pasta.



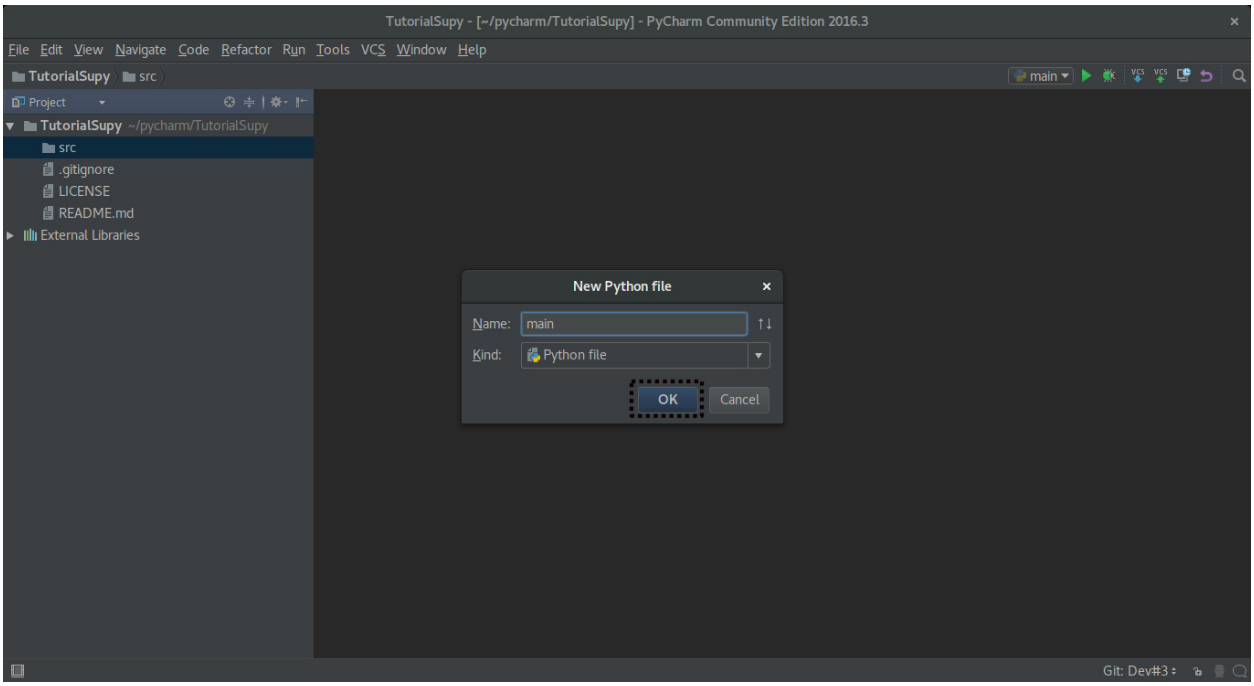
Clique com o botão direito sobre o projeto > New > Directory> src



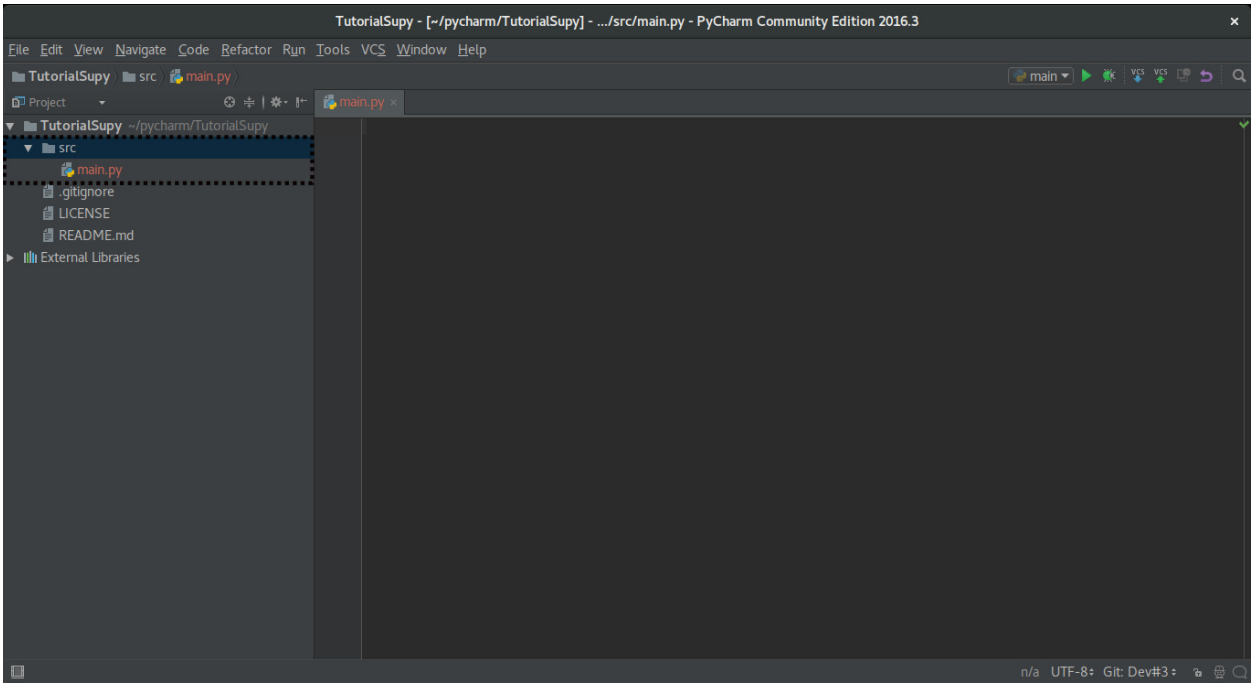
Clique com o botão direito sobre src> New > Python File > “main”



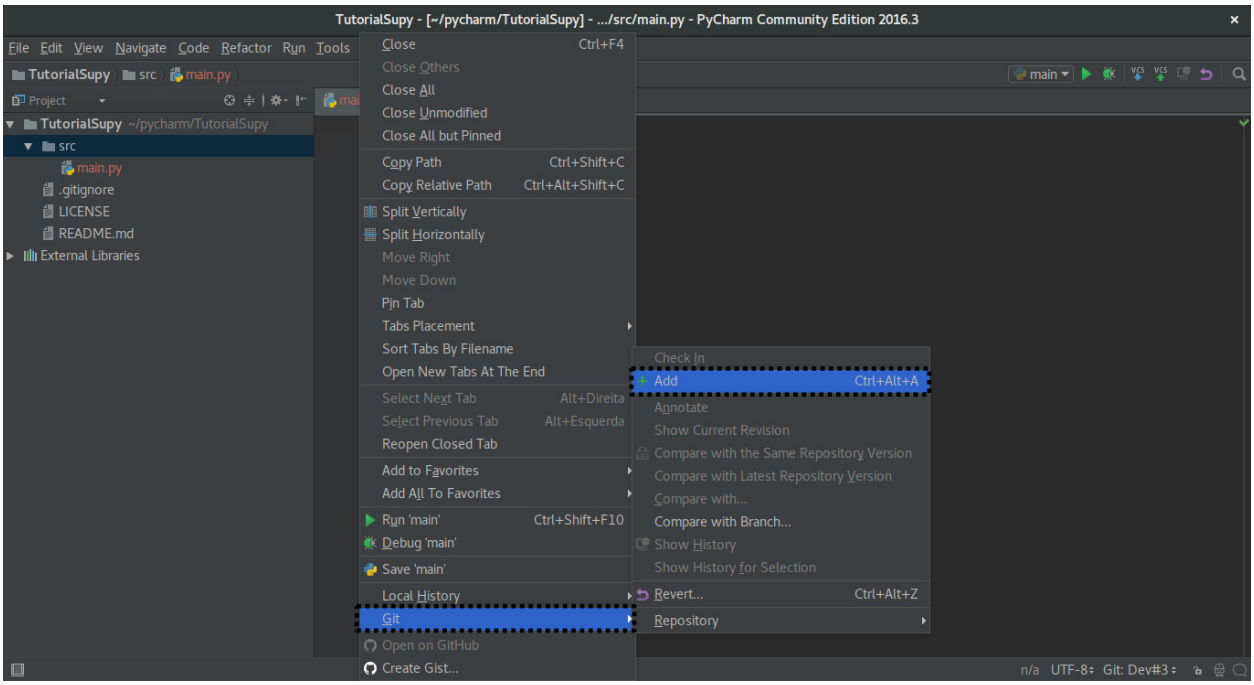
Vai aprir um pop-uppedindo para você confirmar o novo arquivo. Diga ok.



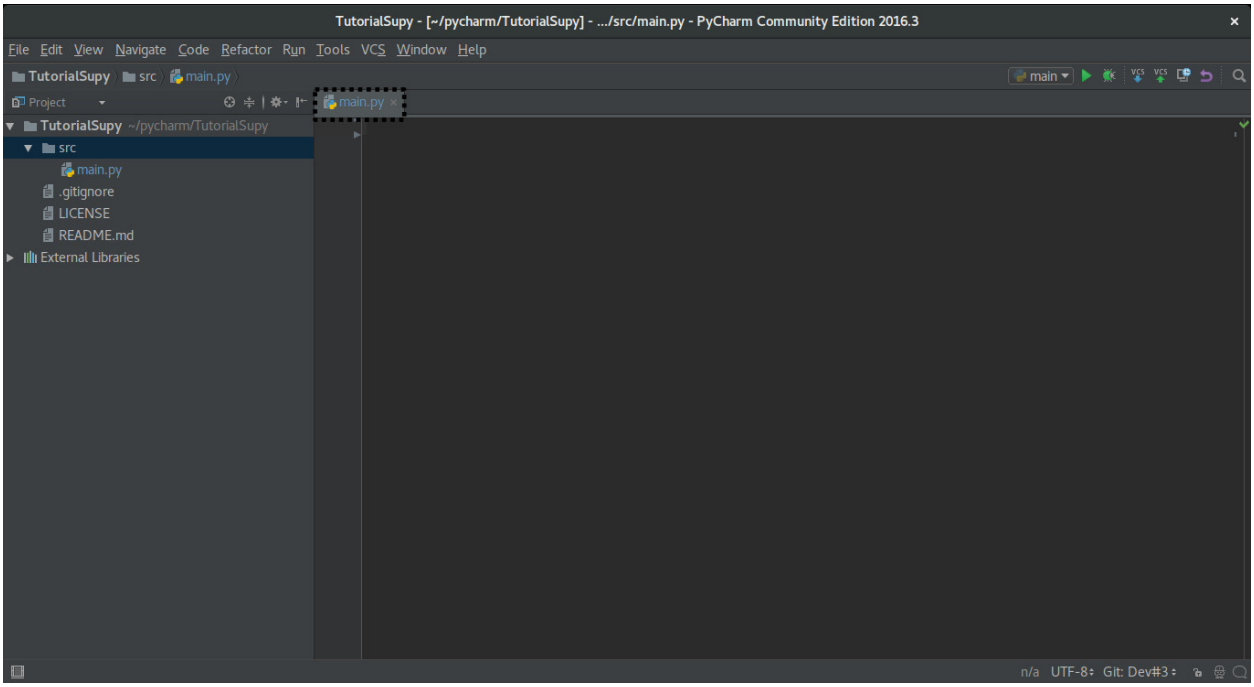
Surge a aba do novo arquivo em Python. Porem ela vai esta na cor vermelha.



Para que seja realmente adicionado o arquivo, clique com o botão direito em src > git > +add

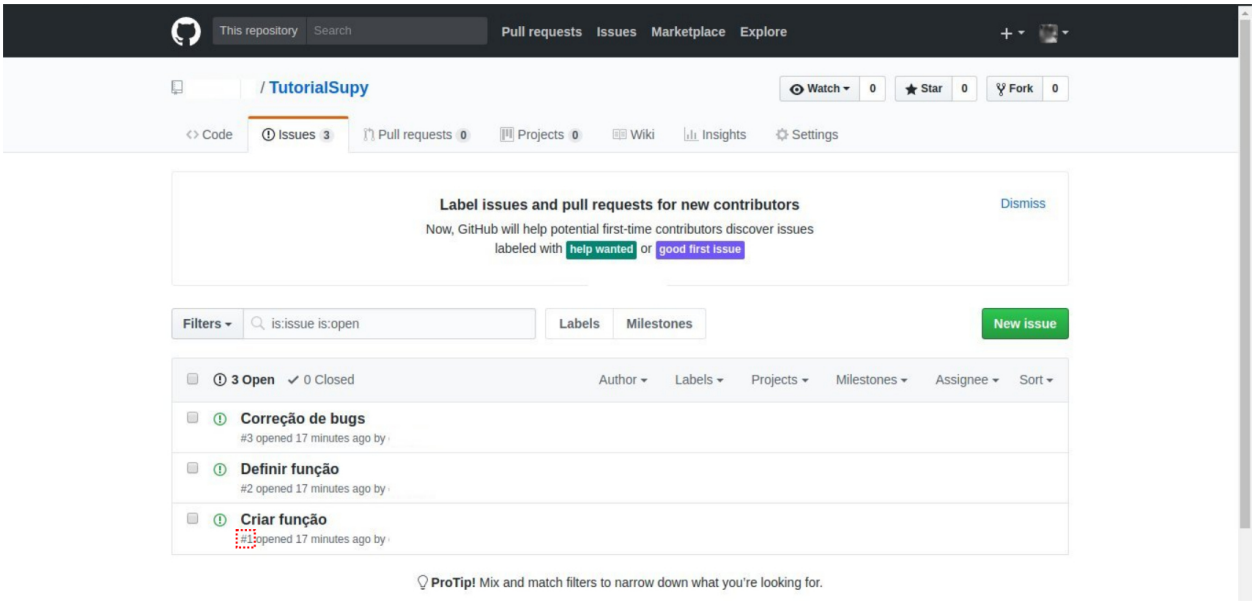


Se o arquivo mudar de cor, ele está corretamente adicionado.

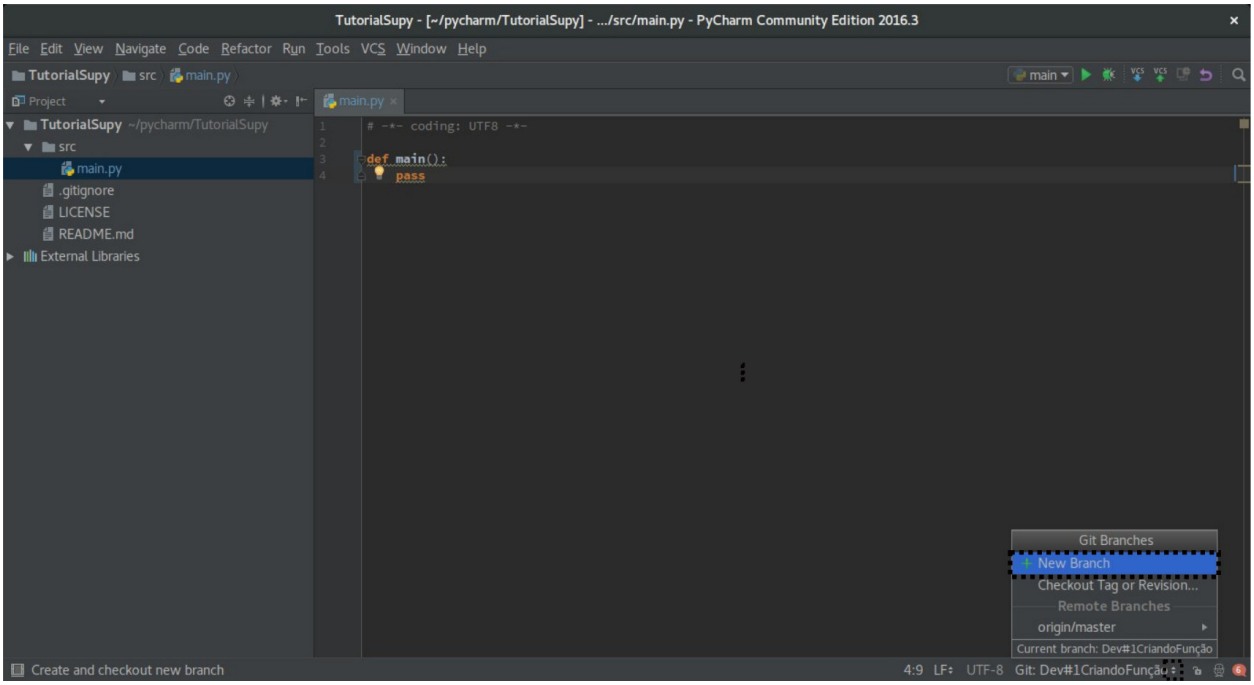


CRIAR UM <BRANCH>

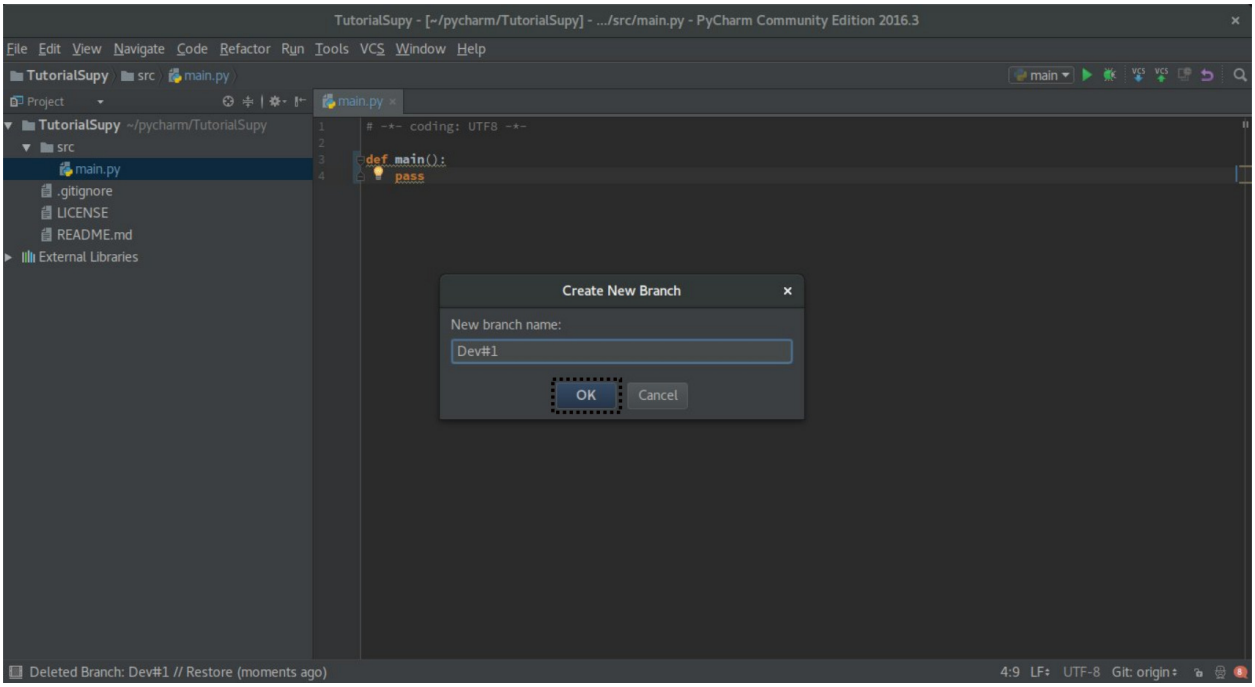
Localize o nome de seu issue.



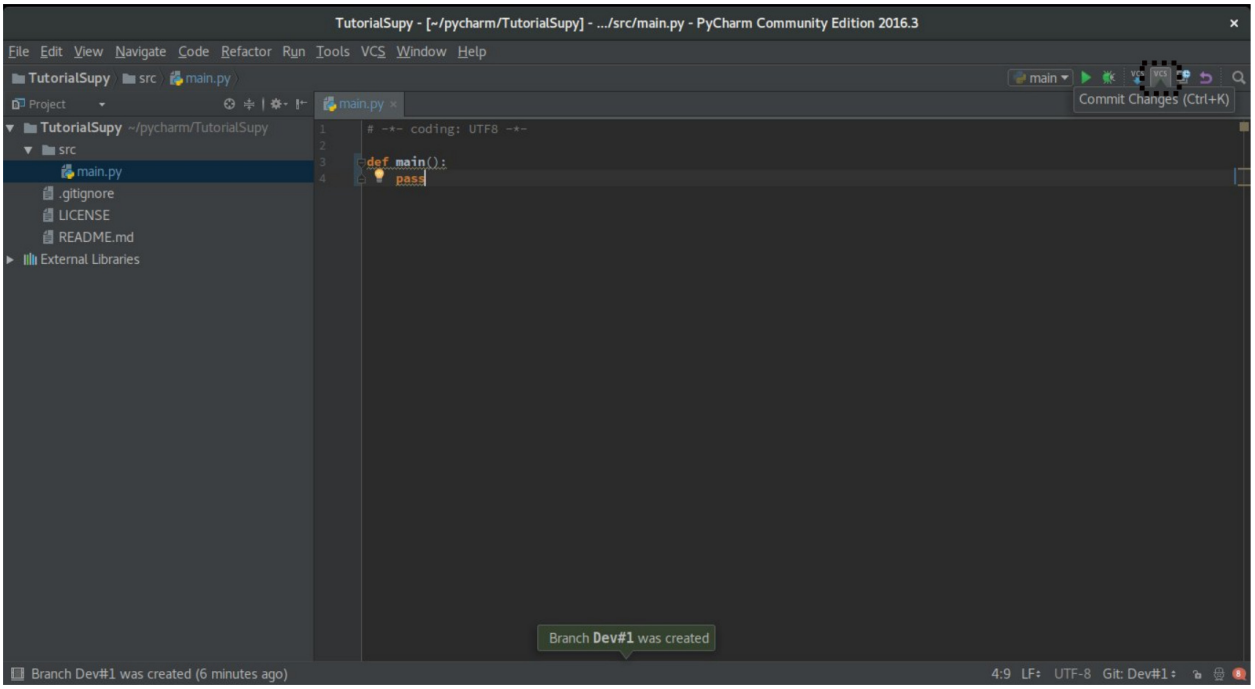
No pycharm, selecione o ícone setas > New Branch.



Nomeie seu Branch de forma a especificar o Issue que está trabalhando.



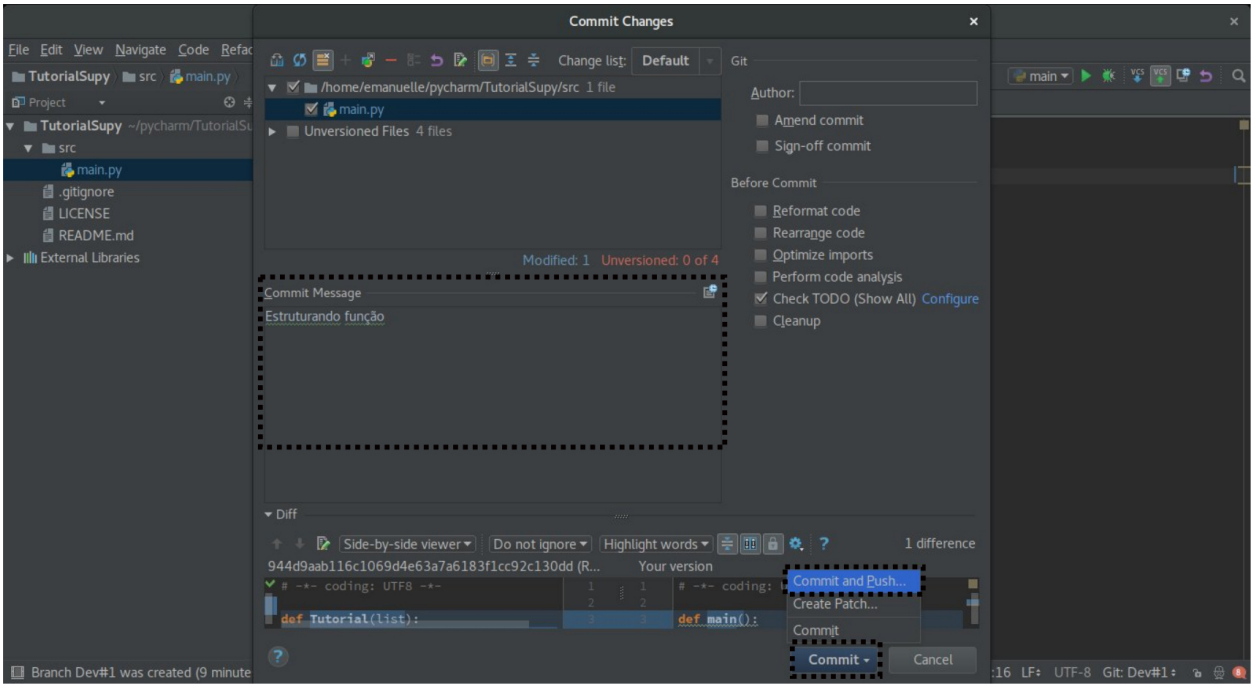
Na parte inferior da tela, aparecerá um balão dizendo que o branch foi criado.



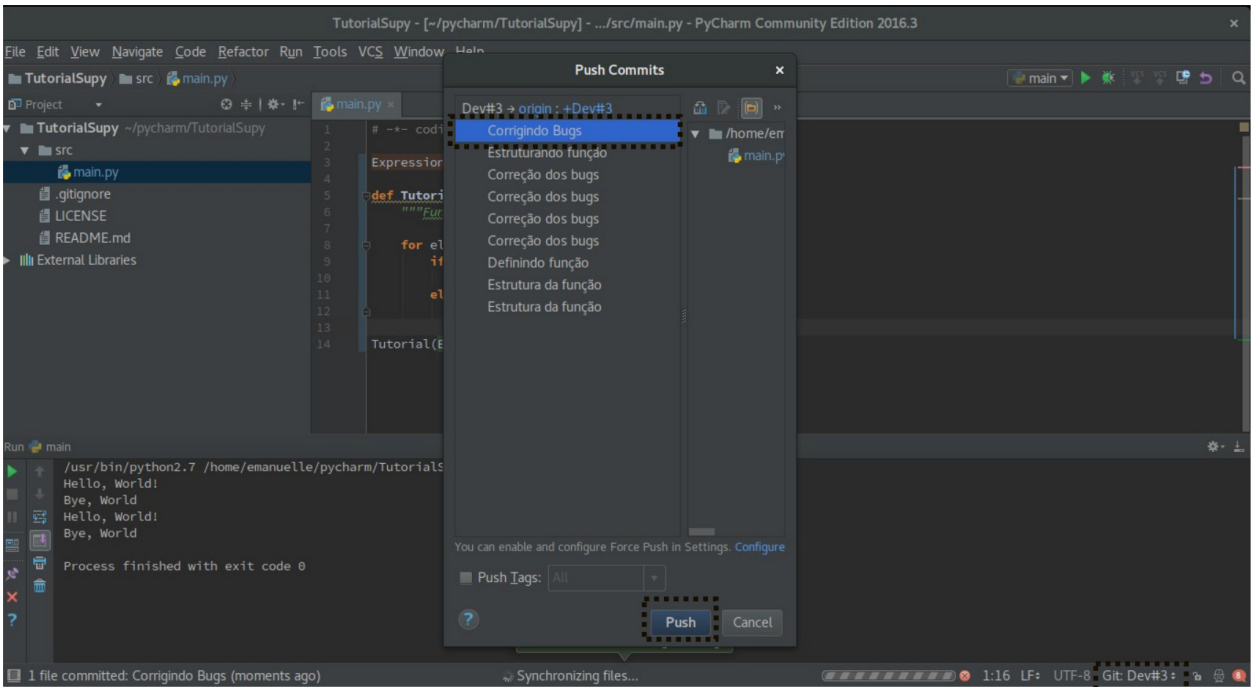
COMMIT

Como enviar mudanças.

Após programar algo que é novo, descreva o seu código > selecione 'commit' > prima commit and push.



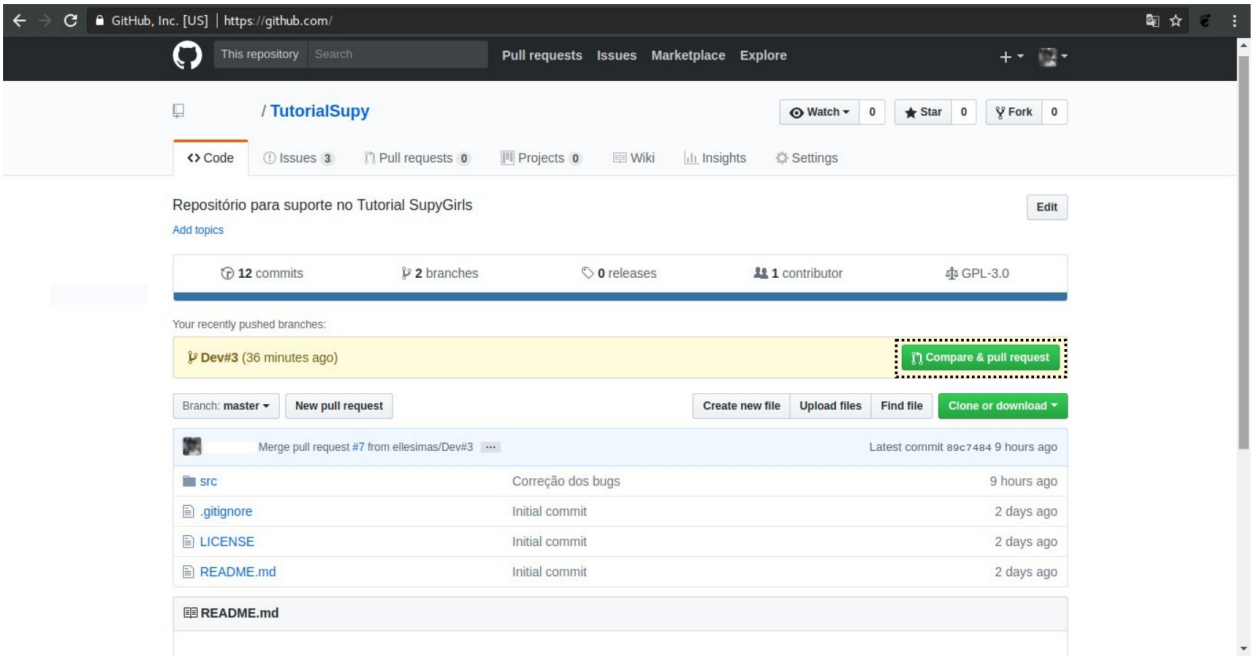
Aparecerá outra tela. Clique em push.



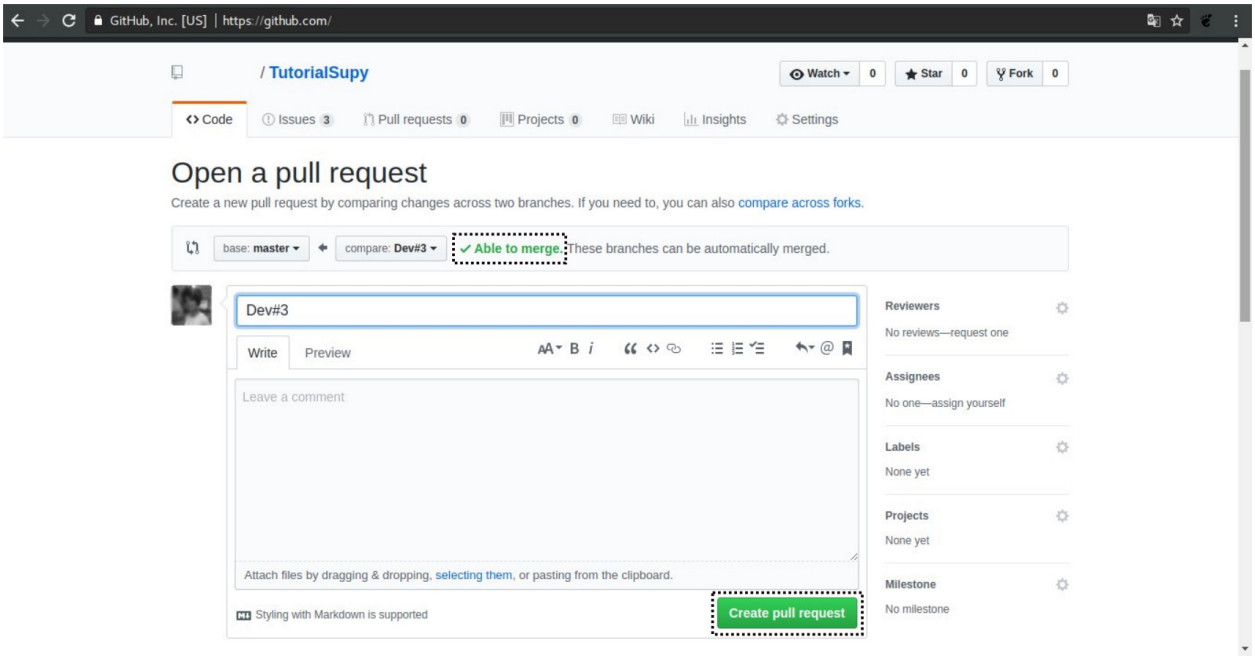
Pronto! Seu código foi enviado .

SALVAR MODIFICAÇÕES NA ORIGEM (MASTER)

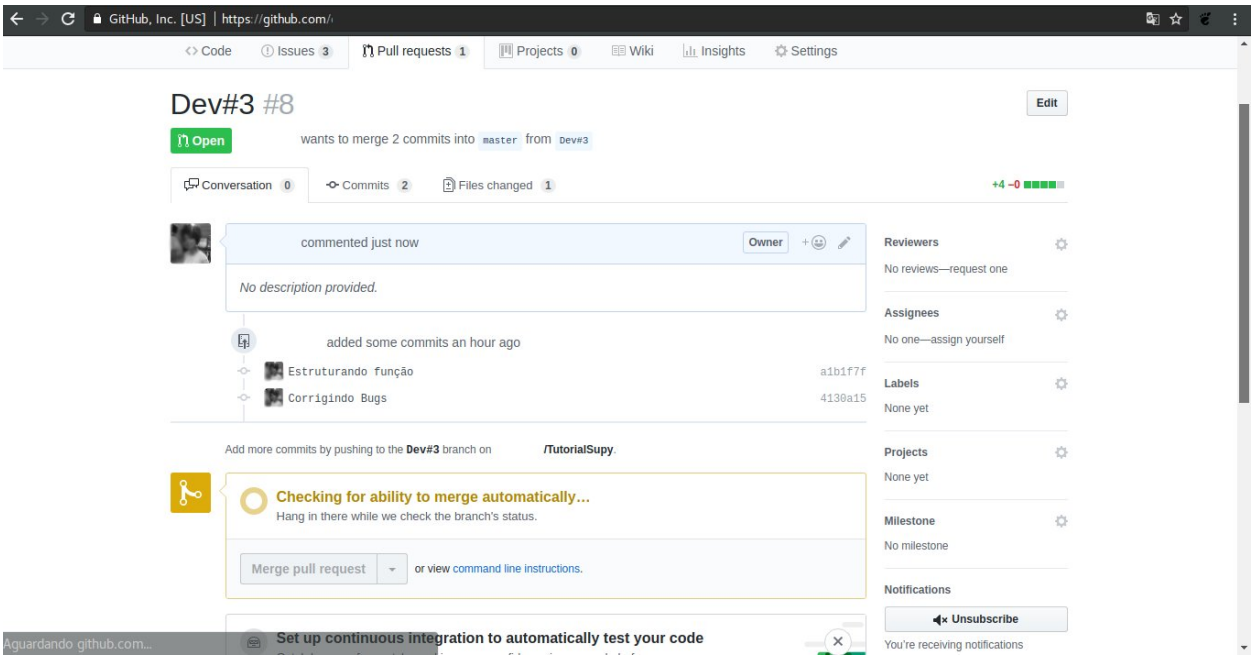
Depois de enviar as últimas modificações. Abra o Github na aba code e dê um >Compare & Pull Request<



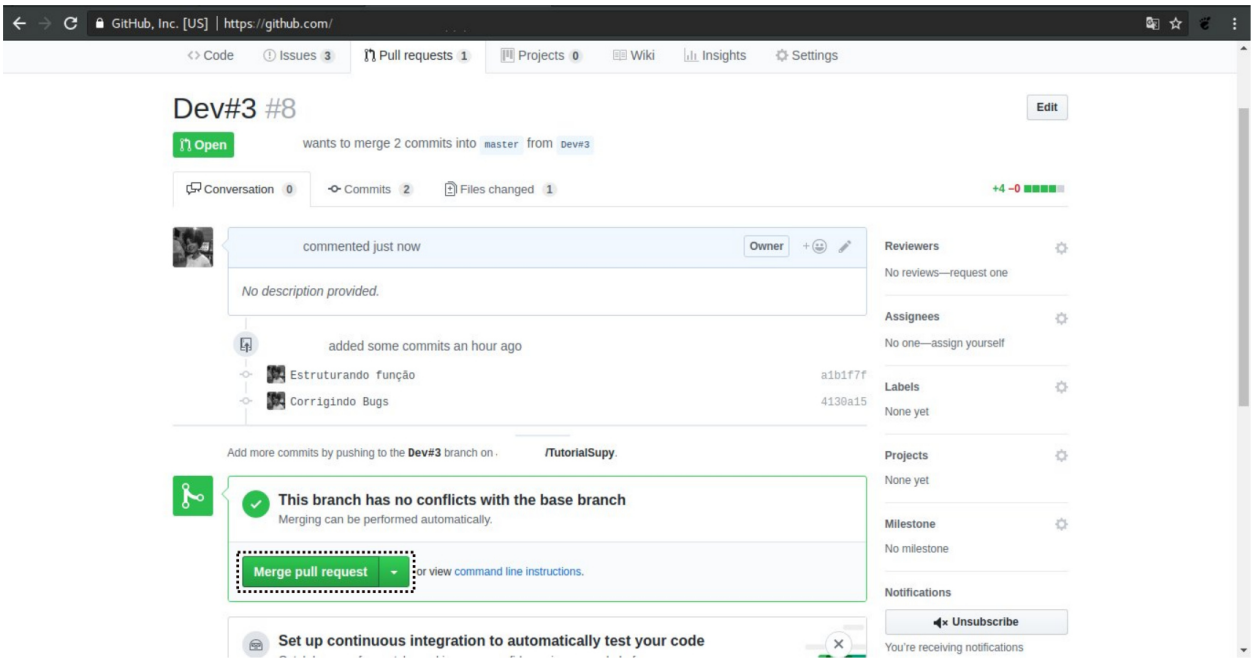
Clique em create and pull request.



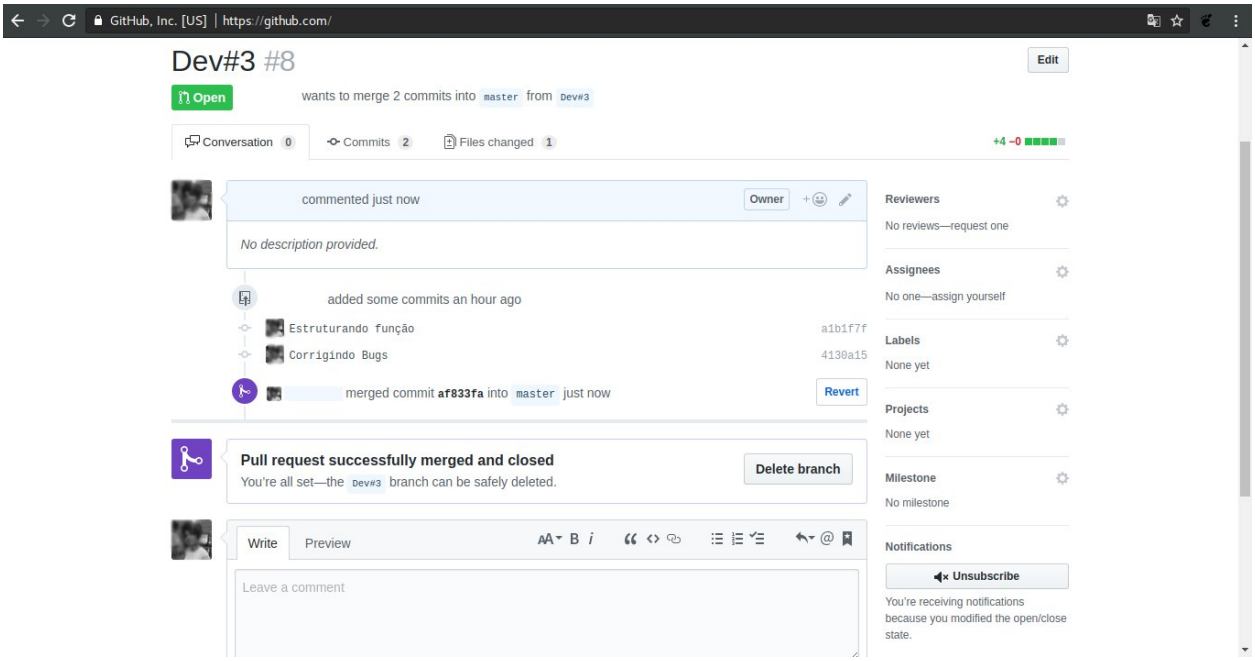
Aguarde o github acabar de checar



Clique em > Merge pull request<



O seu pull request foi bem sucedido!



1.1.4 PRÉ-REQUISITOS PARA INTRODUÇÃO À COMPUTAÇÃO

- **Abrir conta**
  - Github
  - IDE Pycharm Version: 2017.3.3 ou superior
  - Slack
  - Waffle.io

1.1.5 Bem Vindos ao Tutorial de documentação Sphinx!

COMO CRIAR UM PROJETO PARA GERAR UMA PAGINA WEB UTILIZANDO O SPHINX

O Sphinx é uma ferramenta que gera documentação simples mas sofisticada, gera de maneira facil arquivos em diversos formatos. Requisitos:

Python precisa estar instalado em sua máquina. Verifique a versão no seu terminal:

1 | $python –version

2 | $sudo apt install python3-pip

Java: veja se seu PC tem o java instalados.

1 | $java –version

2 | $sudo apt-get install default-jre

3 | $sudo apt-get install default-jdk

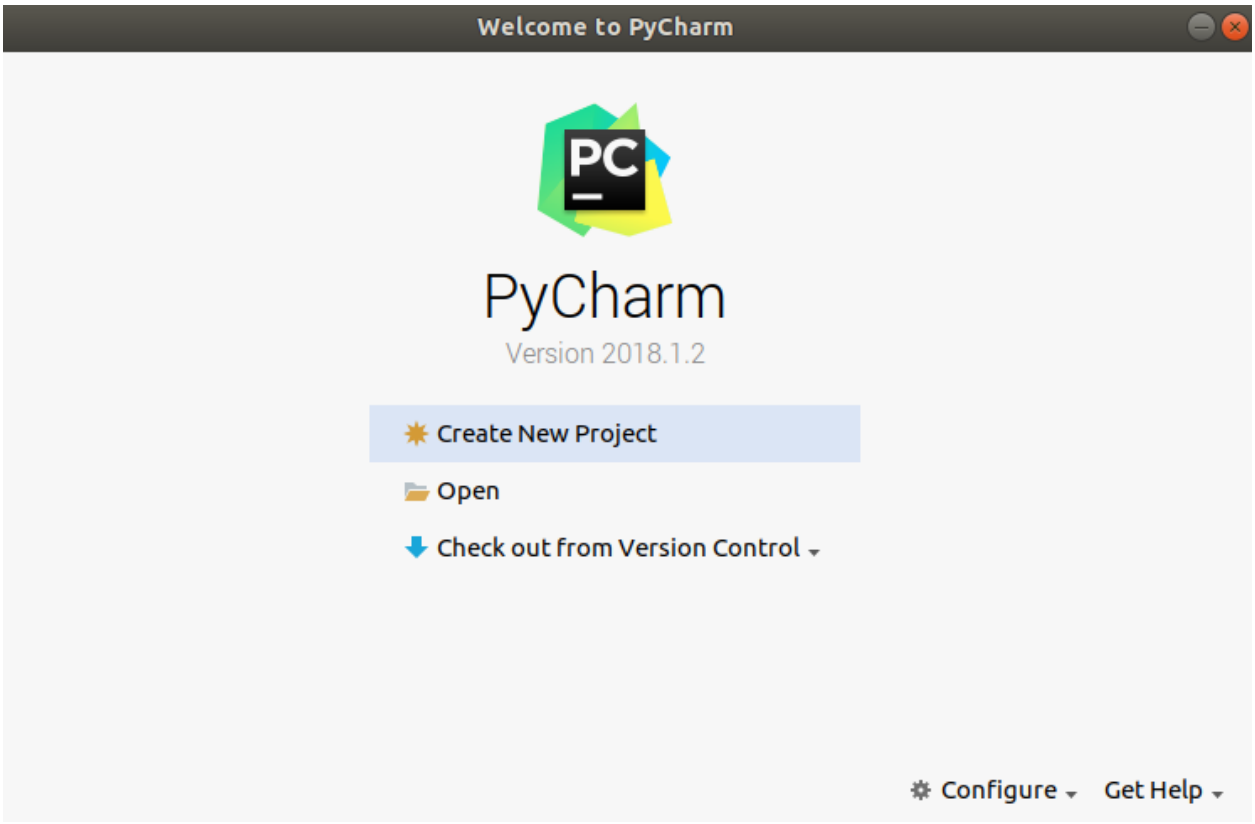
Pycharm também precisa estar instalado, no meu computador eu tenho a versão PyCharm Community Edition. Vamos utilizar o snap para instalar. No terminal:

1 | $sudo apt-get install snapd snapd-xdg-open

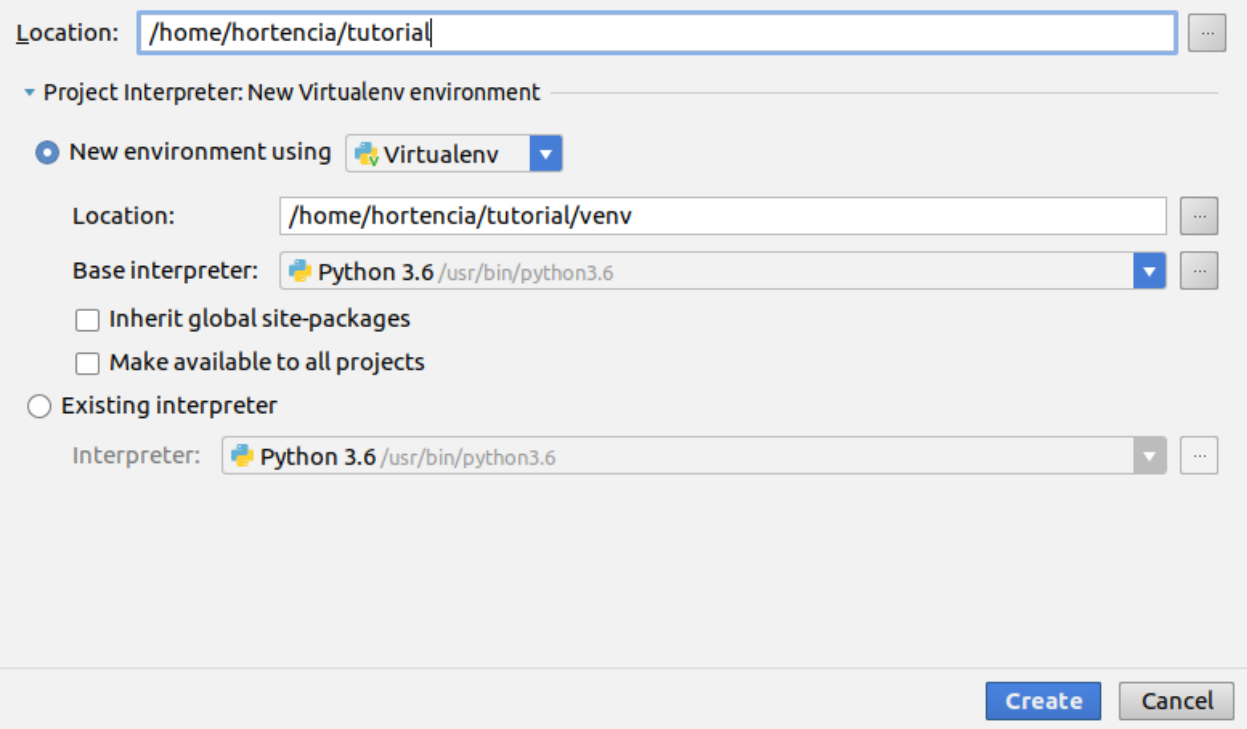
2 | $sudo snap install pycharm-community –classic

Agora só falta instalar o Sphinx!!! 1 | $sudo pip3 install sphinx

CRIANDO UM NOVO PROJETO NO SPHINX Abra a IDE PyCharm → Create New Project



Localização e nome do projeto → “dar o nome” → ir em create

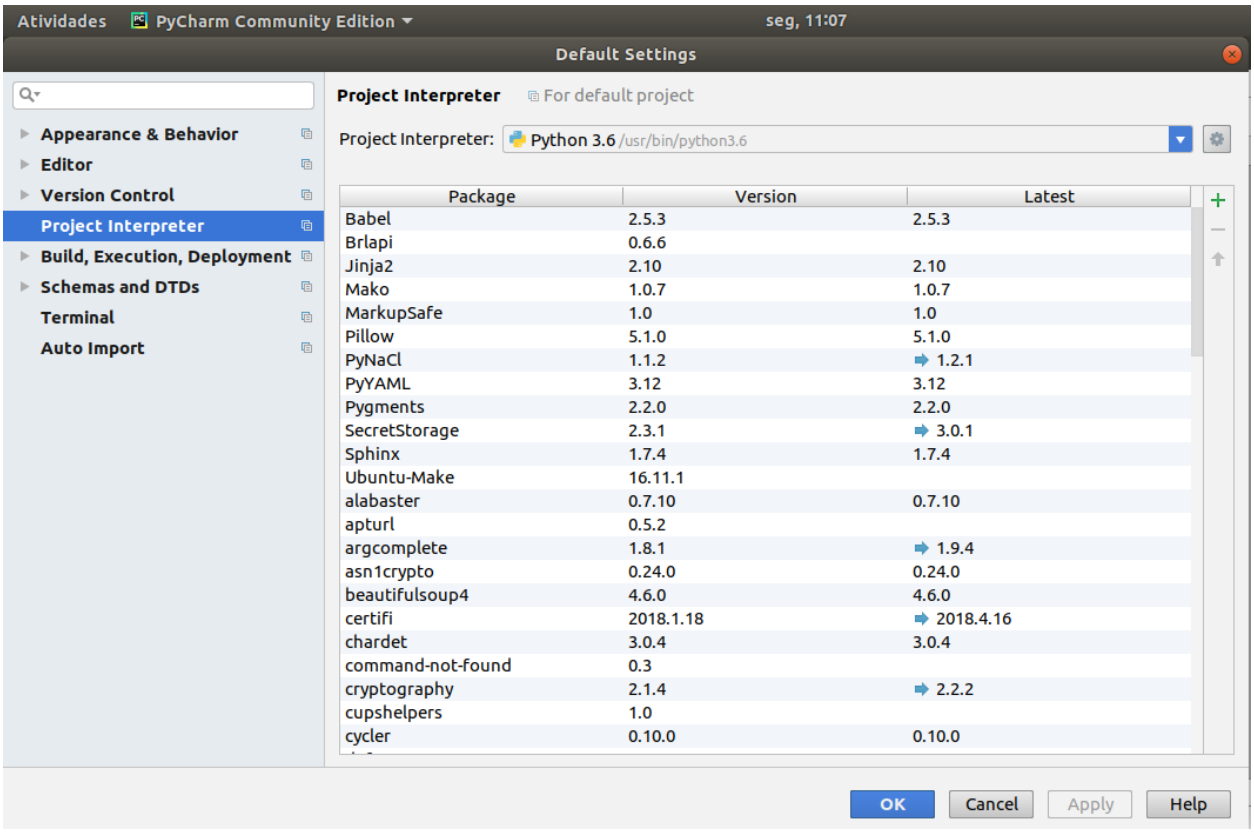


É importante que todas as ferramentas abaixo estejam visíveis. Para isso vá no menu view e marque:

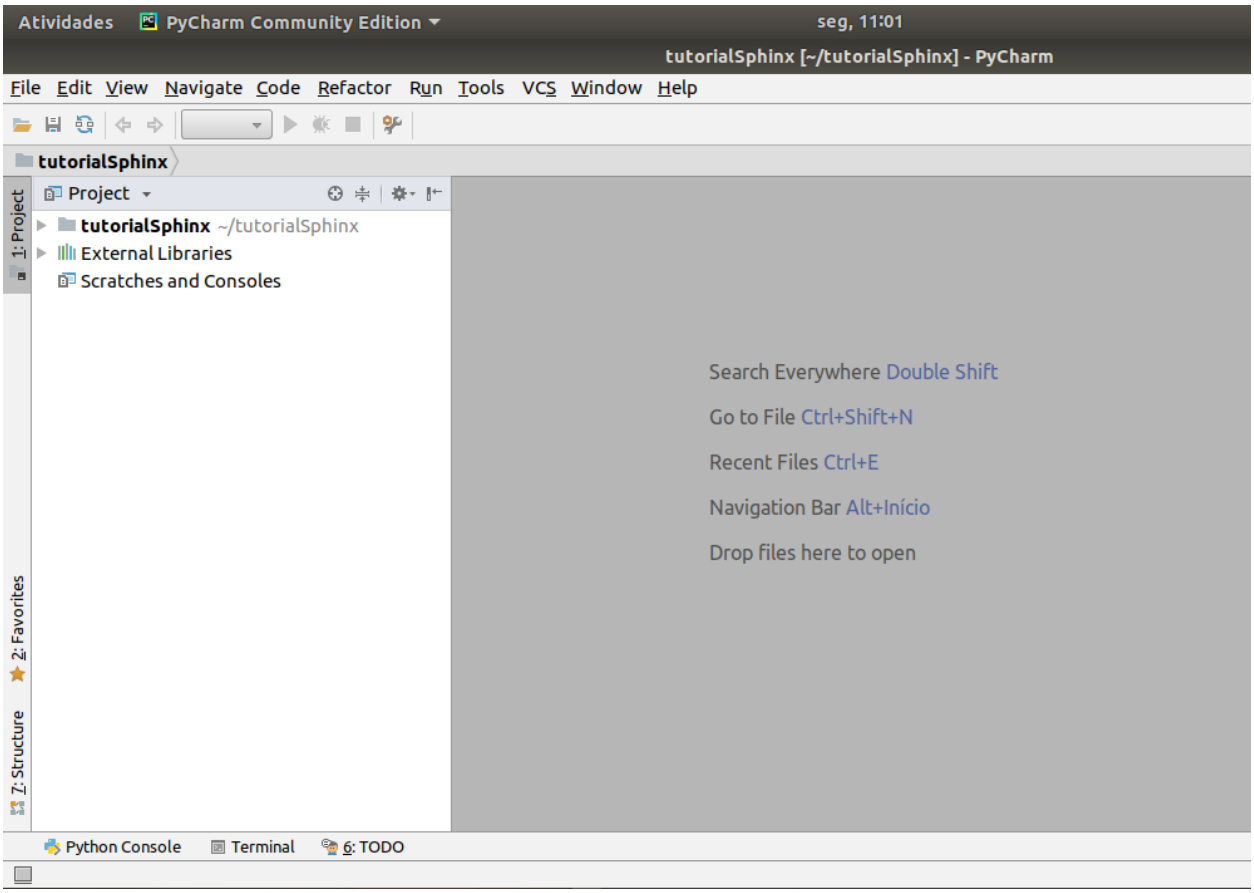
Tool bar Tool buttons Status Bar Navigation Bar

Verifique se o interpretador python 3.6 está instalado

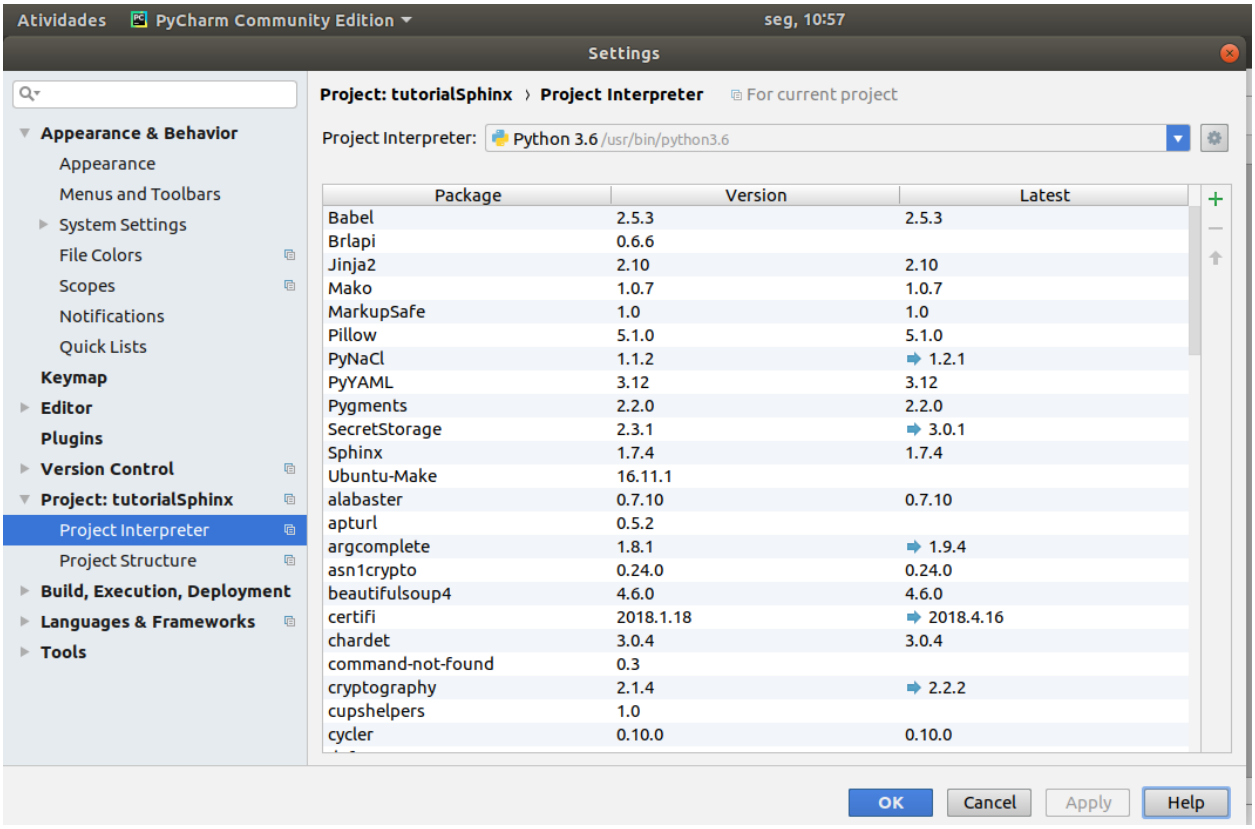
file → default settings – Project Interpreter – verifique se o Python instalado, se não clique na seta e adicione. Depois vá em apply e Ok



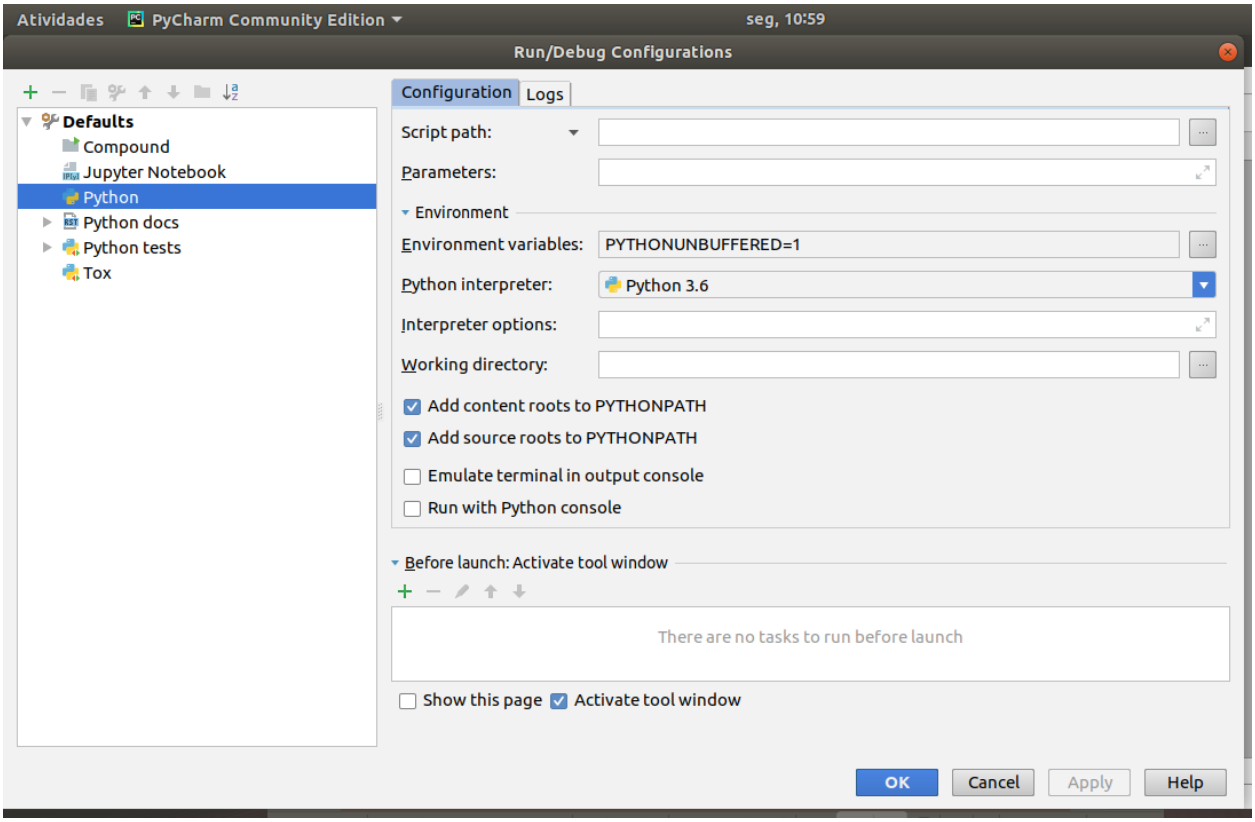
Verifique se o interpretador python 3.6 está instalado como interpretador no Sphinx Clique na chave de boca que aparece no menu



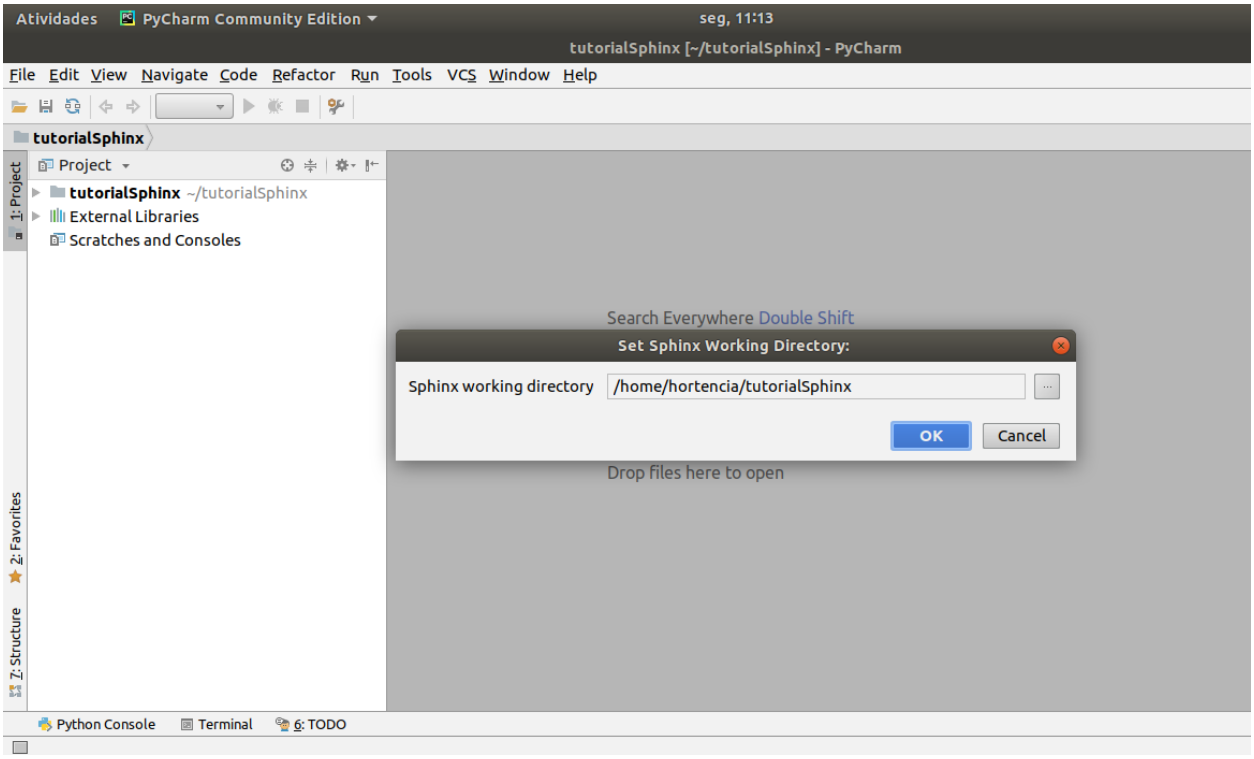
Project tutorialSphinx → Project Interpreter – verifique se o Python instalado, se não clique na seta e adicione. Depois vá em apply e Ok



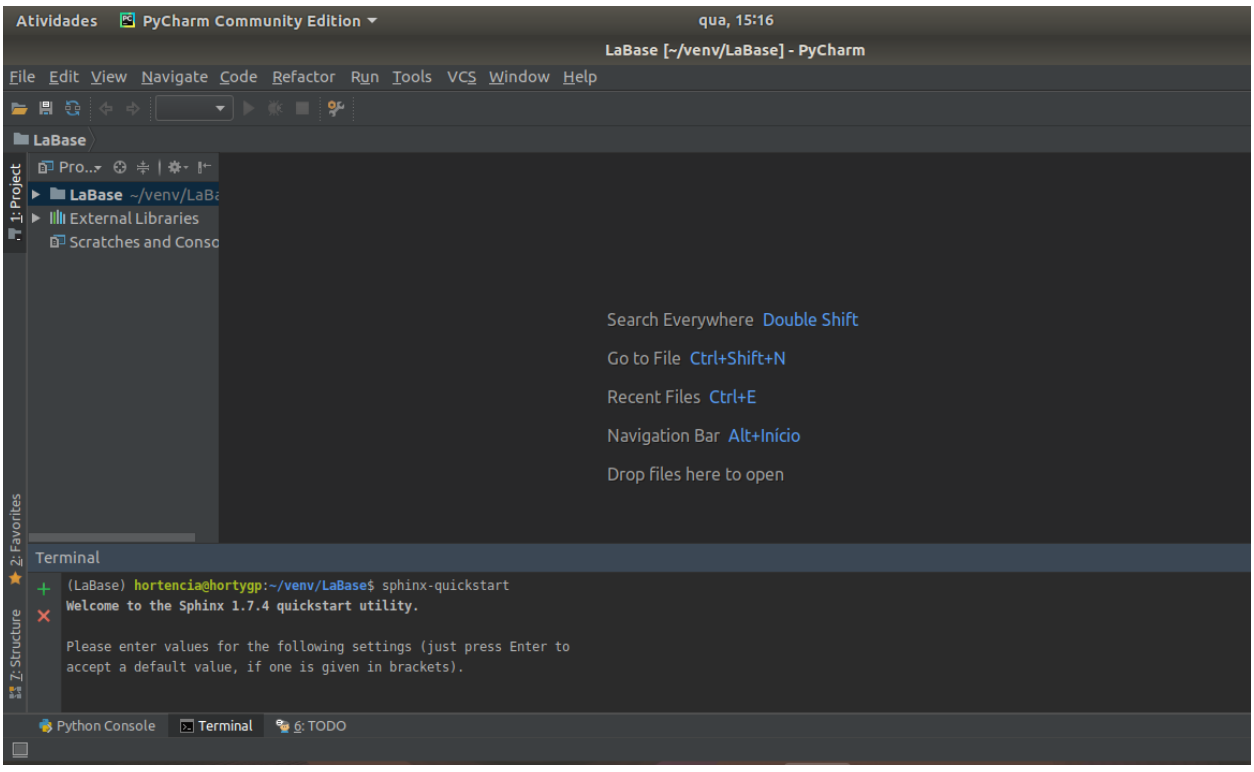
Verifique se o sue interpretador Python está rodando normalmente no Pycharm Run → edit configuration → default



Agora está faltando a ultima verificação!!!! Temos que ver se o Sphinx está aparecendo no menu: Vá no menu Tools e veriifque se o sphinx QuickStar está sendo usado → clica nele → vai aparecer a tela abaixo e clica ok



Ops!!!! ele não apareceu. . . . Se o Sphinx Quickstart não estiver lá no menu: vá no Terminal e digite sphinx-quickstart



Agora aparecem algumas perguntas para você trabalho, se quiser faça uma pesquisa sobre essas perguntas em http:

//docs.readthedocs.io/en/latest/getting_started.html e se informe.

PERGUNTAS E RESPOSTAS:

/usr/bin/sphinx-quickstart

Welcome to the Sphinx 1.7.4 quickstart utility.

Please enter values for the following settings (just press Enter to accept a default value, if one is given in brackets).

Selected root path: .

> You have two options for placing the build directory for Sphinx output. Either, you use a directory "_build" within the root path, or you separate "source" and "build" directories within the root path. Separate source and build directories (y/n) [n]: y
>
> Inside the root directory, two more directories will be created; "_templates" for custom HTML templates and "_static" for custom stylesheets and other static files. You can enter another prefix (such as ".") to replace the underscore.
>
> Name prefix for templates and static dir [_]: Enter
>
> The project name will occur in several places in the built documentation.
>
> Project name: TutorialSphinx
>
> Author name(s): Hortencia
>
> Sphinx has the notion of a "version" and a "release" for the software. Each version can have multiple releases. For example, for Python the version is something like 2.5 or 3.0, while the release is something like 2.5.1 or 3.0a1. If you don't need this dual structure, just set both to the same value.
>
> Project version []: 1.0
>
> Project release [1.0]: 1.0
>
> If the documents are to be written in a language other than English, you can select a language here by its language code. Sphinx will then translate text that it generates into that language. For a list of supported codes, see http://sphinx-doc.org/config.html#confval-language
>
> Project language [en]: pt-br
>
> The file name suffix for source files. Commonly, this is either ".txt" or ".rst". Only files with this suffix are considered documents.
>
> Source file suffix [.rst]: Enter
>
> One document is special in that it is considered the top node of the "contents tree", that is, it is the root of the hierarchical structure of the documents. Normally, this is "index", but if your "index" document is a custom template, you can also set this to another filename.
>
> Name of your master document (without suffix) [index]: Enter
>
> Sphinx can also add configuration for epub output: Do you want to use the epub builder (y/n) [n]: n
>
> Indicate which of the following Sphinx extensions should be enabled:
>
> autodoc: automatically insert docstrings from modules (y/n) [n]: y
>
> doctest: automatically test code snippets in doctest blocks (y/n) [n]: n
>
> intersphinx: link between Sphinx documentation of different projects (y/n) [n]: n
>
> todo: write "todo" entries that can be shown or hidden on build (y/n) [n]: n
>
> coverage: checks for documentation coverage (y/n) [n]: n
>
> imgmath: include math, rendered as PNG or SVG images (y/n) [n]: n

mathjax: include math, rendered in the browser by MathJax (y/n) [n]: n

ifconfig: conditional inclusion of content based on config values (y/n) [n]: n

viewcode: include links to the source code of documented Python objects (y/n) [n]: y

githubpages: create .nojekyll file to publish the document on GitHub pages (y/n) [n]: y

A Makefile and a Windows command file can be generated for you so that you only have to run e.g. 'make html' instead of invoking sphinx-build directly.

Create Makefile? (y/n) [y]: y

Create Windows command file? (y/n) [y]: n

Creating file ./source/conf.py.

Creating file ./source/index.rst.

Creating file ./Makefile.

Finished: An initial directory structure has been created.

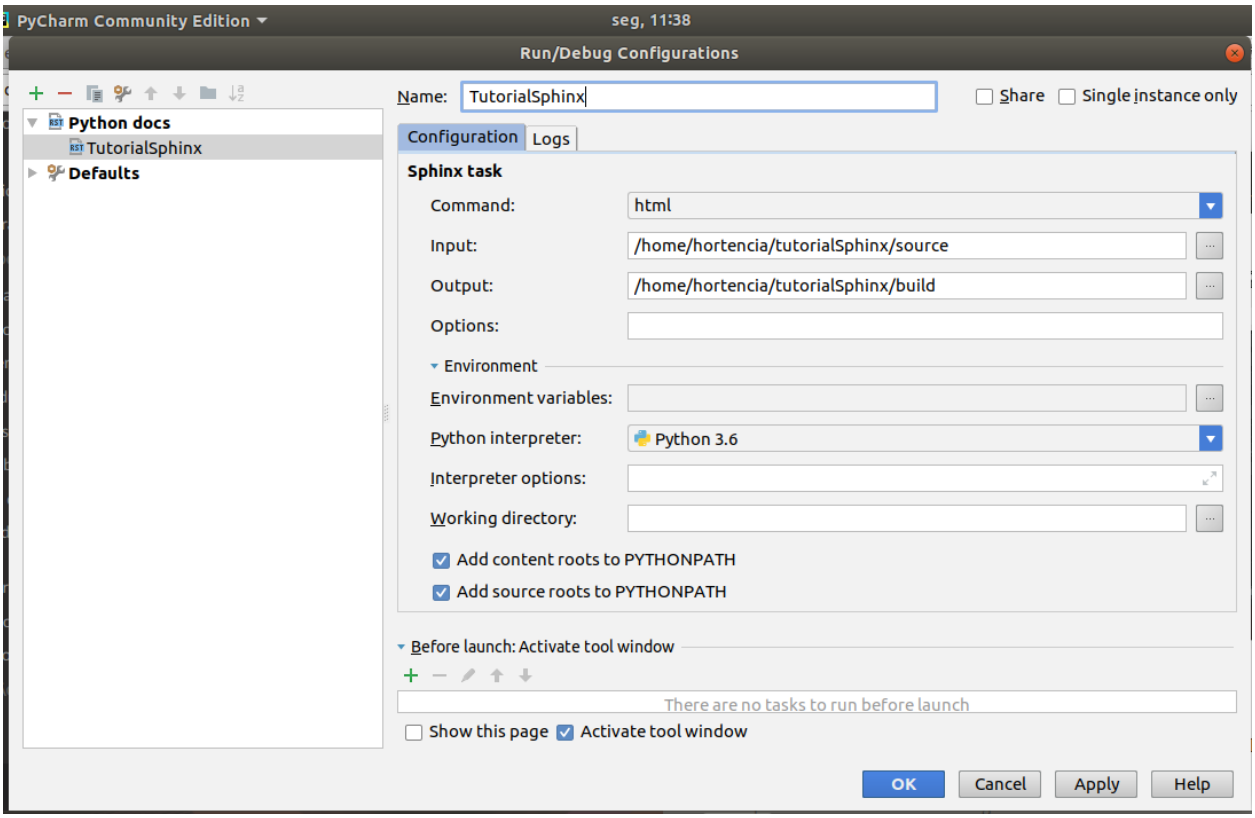
You should now populate your master file ./source/index.rst and create other documentation source files. Use the Makefile to build the docs, like so: make builder

where "builder" is one of the supported builders, e.g. html, latex or linkcheck.

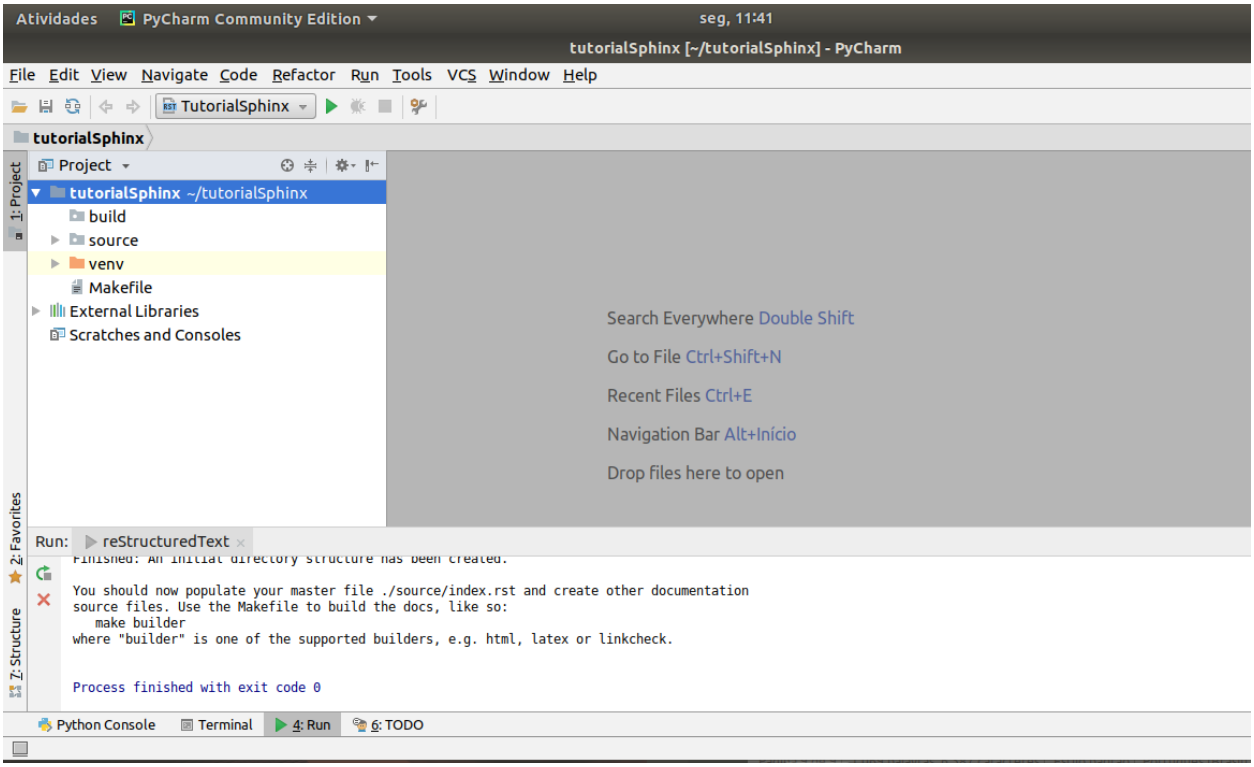
Process finished with exit code 0

Depois de responder as perguntas acima você deve criar um python docs em edit configurations Run → edit configuration → ir no sinal de adição

python docs sphinx task Importante!!!! Não pode esquecer de colocar o Nome input : source output: build Apply → ok



No menu –> Edit Configuration vai aparecer o nome que você deu para teu sphinx clicar em play a seta verde



No canto direito os navegadores instalados (google, firefox e opera) aparecem para rodar na web

1.1.6 Projetos com Python Orientado a objetos

POR QUE PROGRAMAR COM PYTHON ORIENTADO A OBJETOS?

O objetivo do Python Orientado a Objetos é a aproximação do mundo real.

Tudo iniciou com Alan Kay que inventou a Linguagem de programação SmallTalk e e um dos pais do conceito de programação orientada a objetos.

Para a linguagem de programação SmallTalk tudo é objeto e não há tipos primitivos.

1.1.7 O PARADIGMA

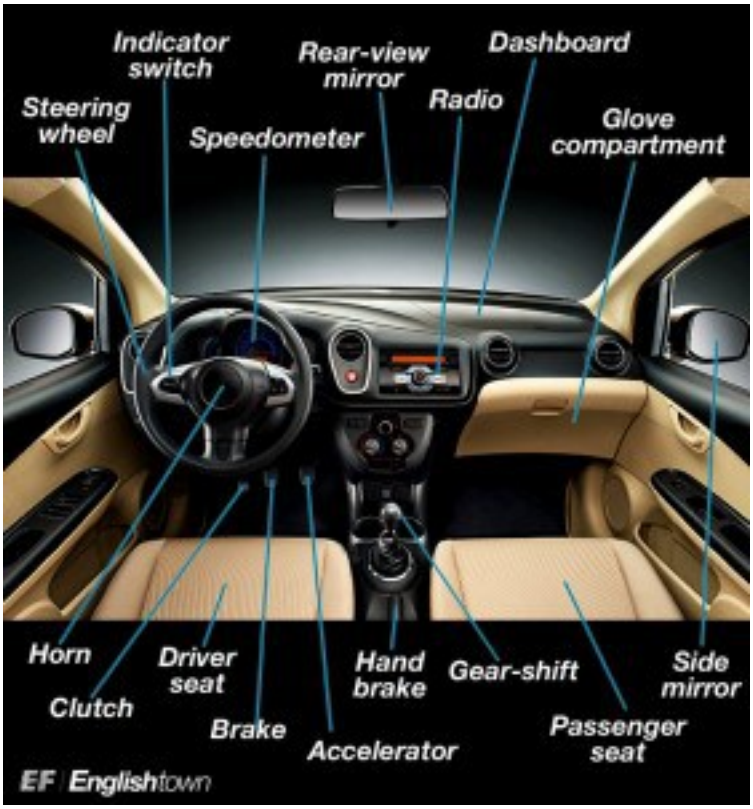
Paradigma é a forma que vamos programar e executar o nosso software.

1 | Paradigma procedural: é baseado no conceito de chamados e procedimentos conhecidos como rotinas, funções, métodos.

2 | Paradigma OO: composto por objetos que possuem propriedades que são

Um exemplo para entender melhor:

Quero construir um carro com foco nas funcionalidades: frear, acelerar, mudar marcha, isso faz com que o Orientado a Objetos se aproxime do mundo real!



VANTAGENS OO:

1 | CONFIABILIDADE É o isolamento entre as partes isso quer dizer que quando alteramos uma parte a outra não necessariamente é afetada.

2 | OPORTUNO Os componentes do projeto podem ser desenvolvidos paralelamente.

3 | FACIL DE MANTER Permite uma maior facilidade para atualizar o software.

4 | EXTENSÍVEL Você pode ter um carro com 10 funcionalidades, mas depois de um tempo criar uma nova versão com 50 funcionalidades, você não precisa recriar o objeto, você já tem várias partes prontas é só aproveitar!

5 | REUTILIZAVEL Criamos uma classe Carro para representar o carro. Podemos utilizar essa classe para um sistema de uma loja de carros. Podemos utilizar novamente para um sistema de oficinas de carros Aproveitamos essa classe para um ainfinidade de outros projetos.

6 | NATURAL Se aproxima do mundo real. Está focado mais nas funcionalidades do quenos detalhes da implementação. É mais fácil de compreender!

1.1.8 Classe e Objeto - Isso você tem que entender!!!

O que é um objeto? É algo material ou abstrato que pode ser percebido pelos sentidos e descrito por meio das suas caracteristicas, comportamentos e estado atual (status) Por exemplo “o carro”:

Como podemos descrever esse carro? Características: cor, marca, fabricante e outros As características do carro são os seus atributos(campos)! cor = preto, verde, azul, amarelo

Comportamentos: Acelerar, frear, ligar o som entre outros

Estado atual: É representado pelos valores dos atributos naquele momento que é analisado.

Importante: Podemos ter vários tipos de carros Ao criar um carro teremos um modelo, um molde, um formato.

Agora temos o carro e o molde para fazer o carro!

CARRO = OBJETO MOLDE = CLASSE

O que faz esse carro são os métodos Os métodos pertencem a uma classe métodos = acelerar, passar a marcha, diminuir a velocidade etc Através dos métodos conseguimos mexer com os atributos do objeto.

Todo objeto vem a partir de uma classe, a partir de um molde/modelo

A classe define quem são os atributos e métodos comuns que serão compartilhados por um objeto.

1.1.9 Instanciar o que é isso?

Quando temos uma classe e queremos gerar um objeto a partir dela então você faz o instanciamento.

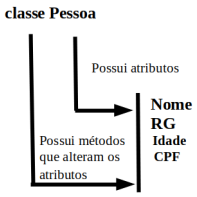
Instanciar é gerar um objeto a partir de uma classe!

c = Carro() –> c é instancia da classe Carro

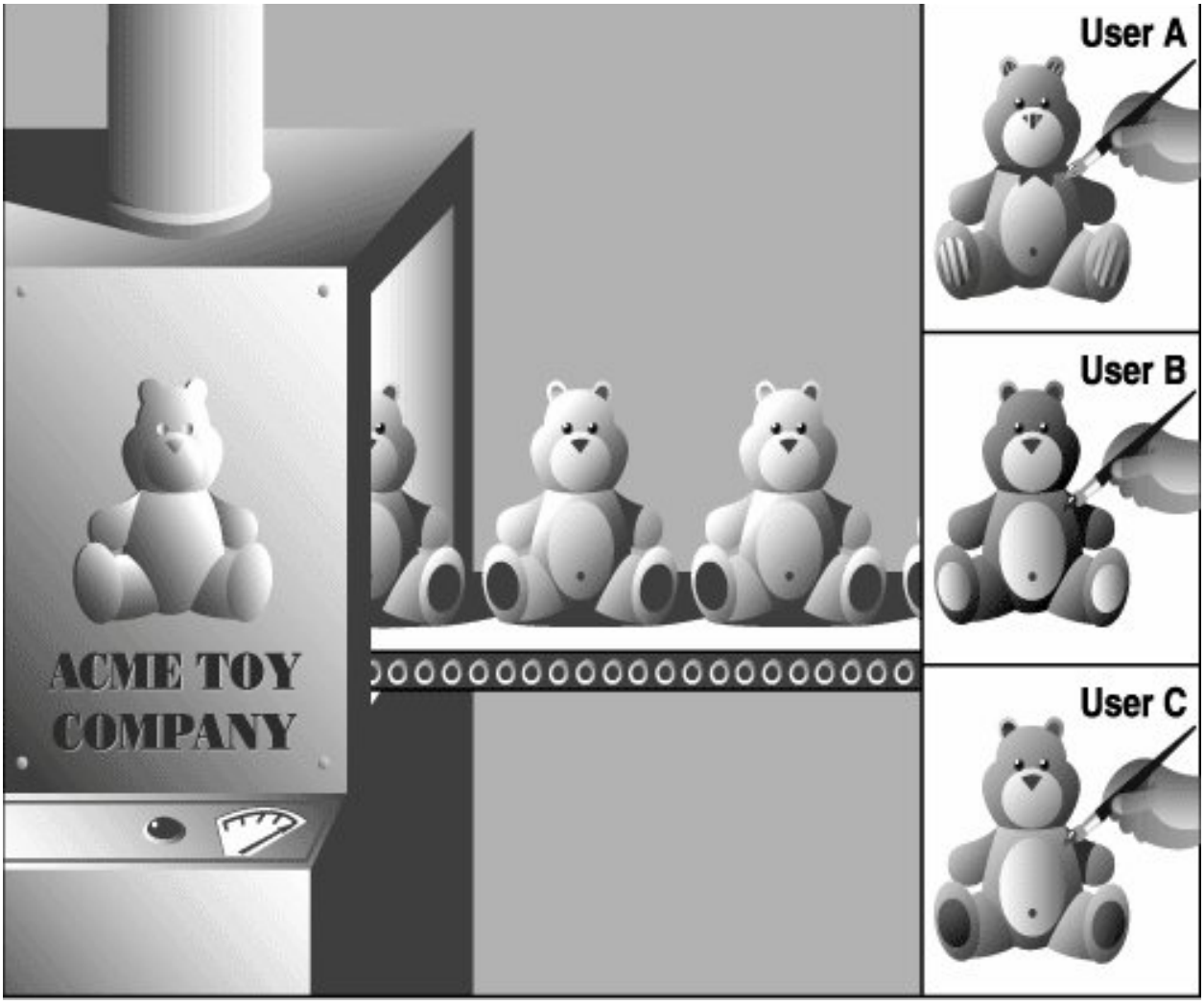
1.1.10 O que é uma classe?

Classe é um termo usado para um tipo de objeto As classes descrevem os objetos A classe serve como modelo, como molde para armazenar informações e realizar tarefas.

Uma classe pode conter vários elementos



Podemos ter várias classes, os objetos dessas classes são instanciados de forma que a execução do programa é vista como um conjunto de objetos relacionados que se comunicam envianod mensagens uns para outros.


User A
User B
User C
ACME TOY
COMPANY

CHAPTER 2

Bem Vindos ao Circo Voador da Programação Python

Aqui vamos ter uma introdução rápida de como programar jogos para Web usando Python. Na verdade vamos usar o Brython que é o Python que funciona dentro de um navegador web como o Firefox.

2.1 Como brincar com os desafios

Entre na plataforma <nome do seu projeto>.is-by.us. Clique no projeto determinado pelo instrutor. Vamos começar importando o circus para criar um jogo. Você faz a sua implementação e depois invoca o circus para conferir o resultado.

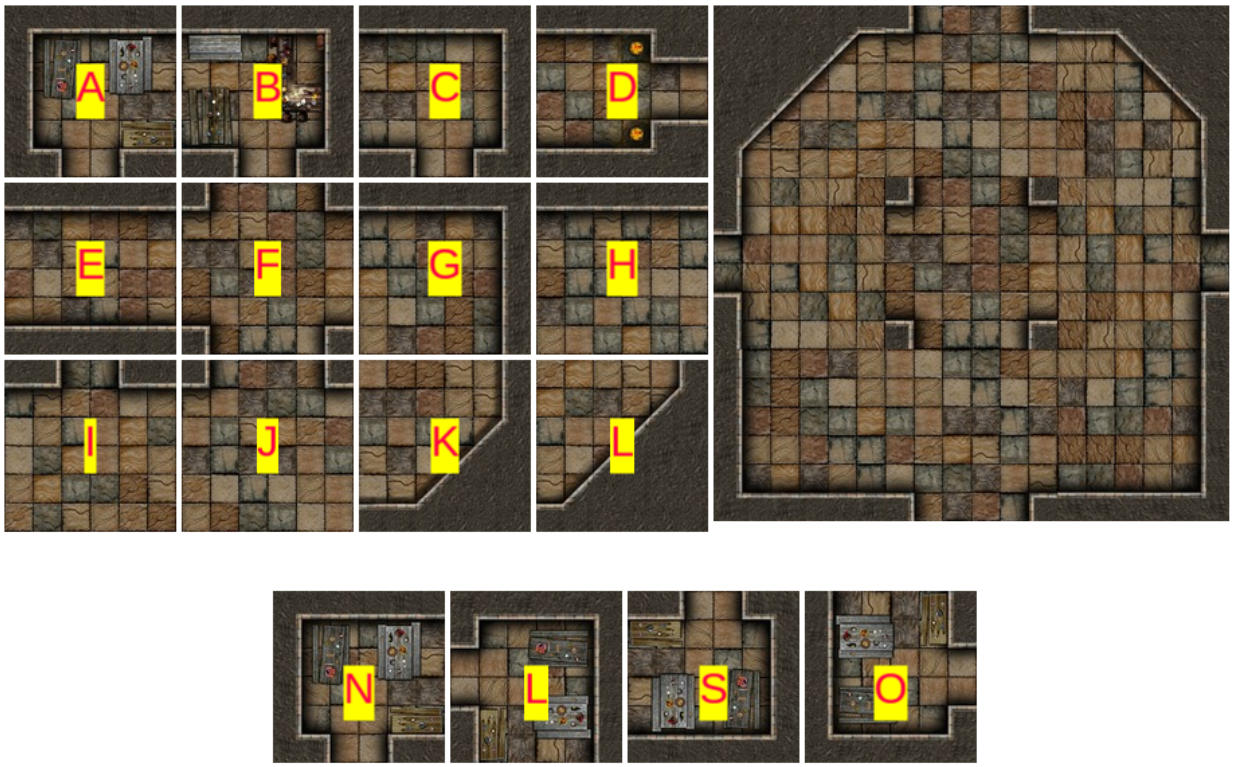
```
from _spy.circus.circus import circus
# faça aqui a sua implementação do desafio
if __name__ == "__main__":
    #circus(<ponha aqui o número do desafio e descomente a linha>, <parâmetro␣
↪indicado>)
```

Note: Procure ser cooperativo com a sua equipe.

2.2 Criando uma Câmara com Constantes

Uma constante é um valor que não se modifica ao longo de um programa. Em Python a constante é escrita com todas as letras maiúsculas como no nome TOPO_ESQUERDA abaixo.

Use os ladrilhos nomeados de A a L para montar a câmara mostrada à direita.




```
from _spy.circus.circus import circus

TOPO_ESQUERDA = "AN"
TOPO_DIREITA = "AN"
TOPO_CENTRO = "AN"
MEIO_ESQUERDA, CENTRO, MEIO_DIREITA = "AN", "AN", "AN"
FUNDO_ESQUERDA, FUNDO_CENTRO, FUNDO_DIREITA =  "AN", "AN", "AN"

# O comando abaixo voce vai entender no próximo desafio
circus(1, [[TOPO_ESQUERDA, TOPO_CENTRO, TOPO_DIREITA], [MEIO_ESQUERDA, CENTRO,
        MEIO_DIREITA], [FUNDO_ESQUERDA, FUNDO_CENTRO, FUNDO_DIREITA]])
```

Note: No texto "AN" a primeira letra determina o ladriho e a segunda se está girada para Norte, Leste, Sul ou Oeste.

2.3 Criando uma Câmara com Listas

Uma lista é um conjunto de coisas, pode ser um conjunto de números, letras, palavras ou qualquer outro objeto. Em Python a lista é escrita assim: *[<uma coisa>, <outra coisa>]*.

Use os ladrilhos nomeados de A a L para montar a câmara mostrada abaixo, consulte o exercício anterior.

```
from _spy.circus.circus import circus

MASMORRA = [[ "AN", "AN", "AN", "AN", "AN", "AN"],
```

(continues on next page)

(continued from previous page)

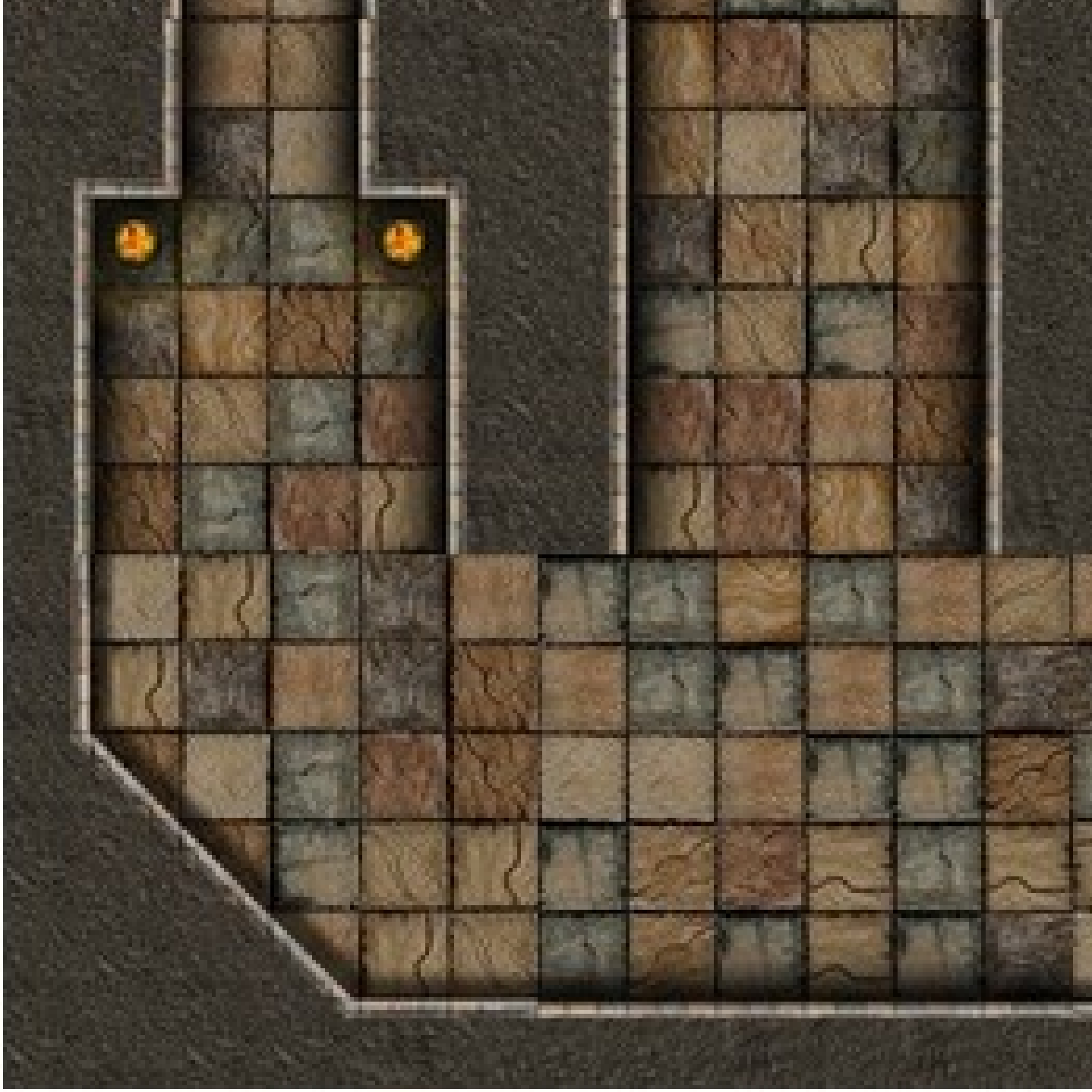
```
            [ "AN", "AN", "AN", "AN", "AN", "AN"],
            [ "AN", "AN", "AN", "AN", "AN", "AN"],
            [ "AN", "AN", "AN", "AN", "AN", "AN"],
            [ "AN", "AN", "AN", "AN", "AN", "AN"]
            ]

circus(2, MASMORRA)
```

Note: No texto "AN" a primeira letra determina o ladriho e a segunda se está girada para Norte, Leste, Sul ou Oeste.

2.4 Criando uma Câmara com Dicionário

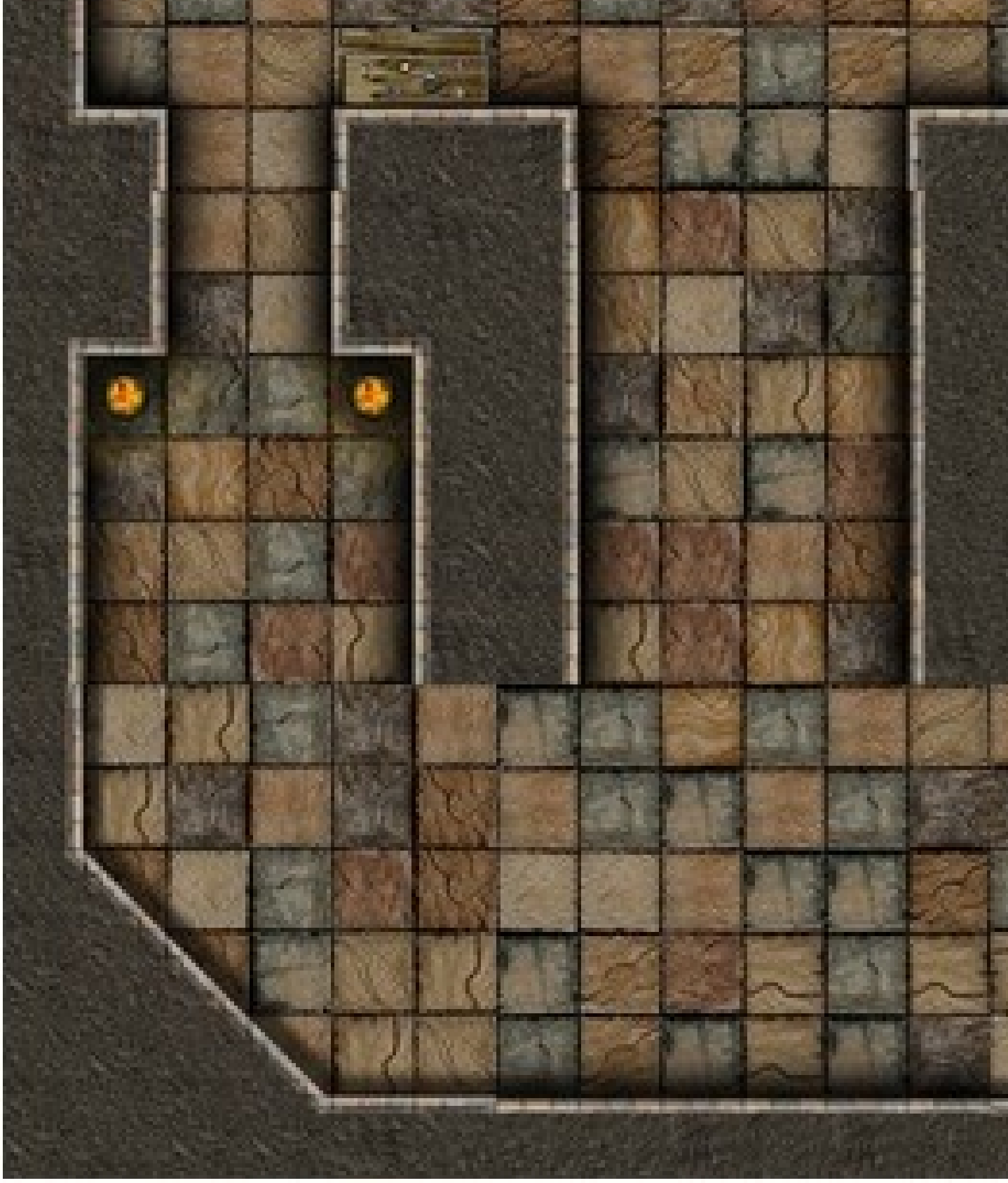
Uma lista é um conjunto de coisas, pode ser um conjunto de números, letras, palavras ou qualquer outro objeto. Em Python a lista é escrita assim: *{<umnome: umvalo>, <outronome: outrovalor>}*.

Use os ladrilhos nomeados de A a L para montar a câmara mostrada abaixo, consulte o exercício A. Descubra quais posições os nomes misteriosos indicam.



```
from _spy.circus.circus import circus

MASMORRA = {'Cahuitz': 'AN', 'Cauha': 'AN', 'Coycol': 'AN',
 'Huatlya': 'AN', 'Micpe': 'AN', 'Nenea': 'AN',
 'Pallotl': 'AN', 'Tetlah': 'AN', 'Zitllo': 'AN'}


circus(3, MASMORRA)
```

Note: No texto “AN” a primeira letra determina o ladrilho e a segunda se está girada para Norte, Leste, Sul ou Oeste.



```
from _spy.circus.circus import circus

MASMORRA = {'Cahuitz': 'AN', 'Cauha': 'AN', 'Coycol': 'AN',
 'Huatlya': 'AN', 'Micpe': 'AN', 'Nenea': 'AN',
 'Pallotl': 'AN', 'Tetlah': 'AN', 'Zitllo': 'AN'}
```

(continues on next page)

(continued from previous page)

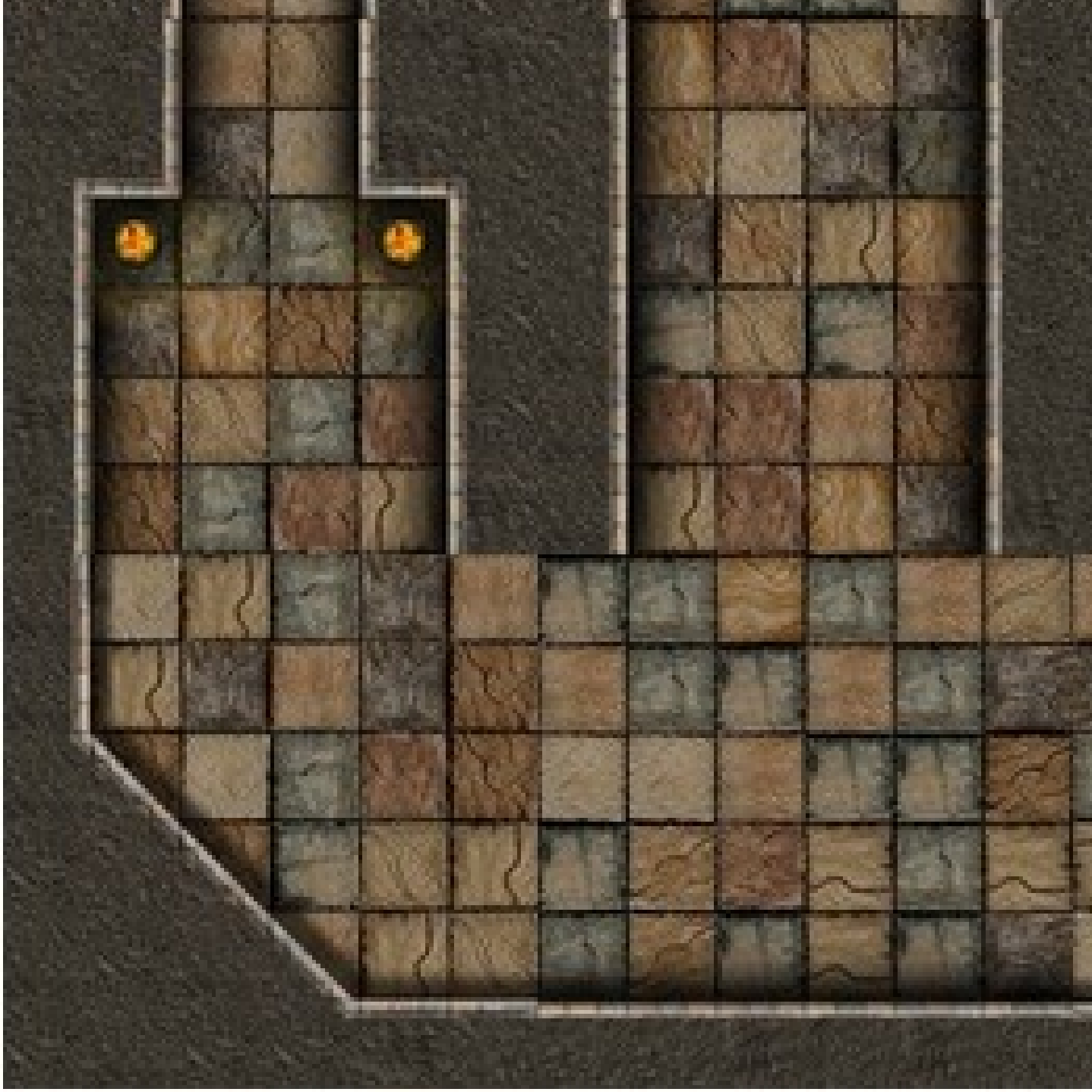
```
circus(4, MASMORRA)
```

Note: No texto "AN" a primeira letra determina o ladrilho e a segunda se está girada para Norte, Leste, Sul ou Oeste.

2.6 Uma Câmara Muito Instável

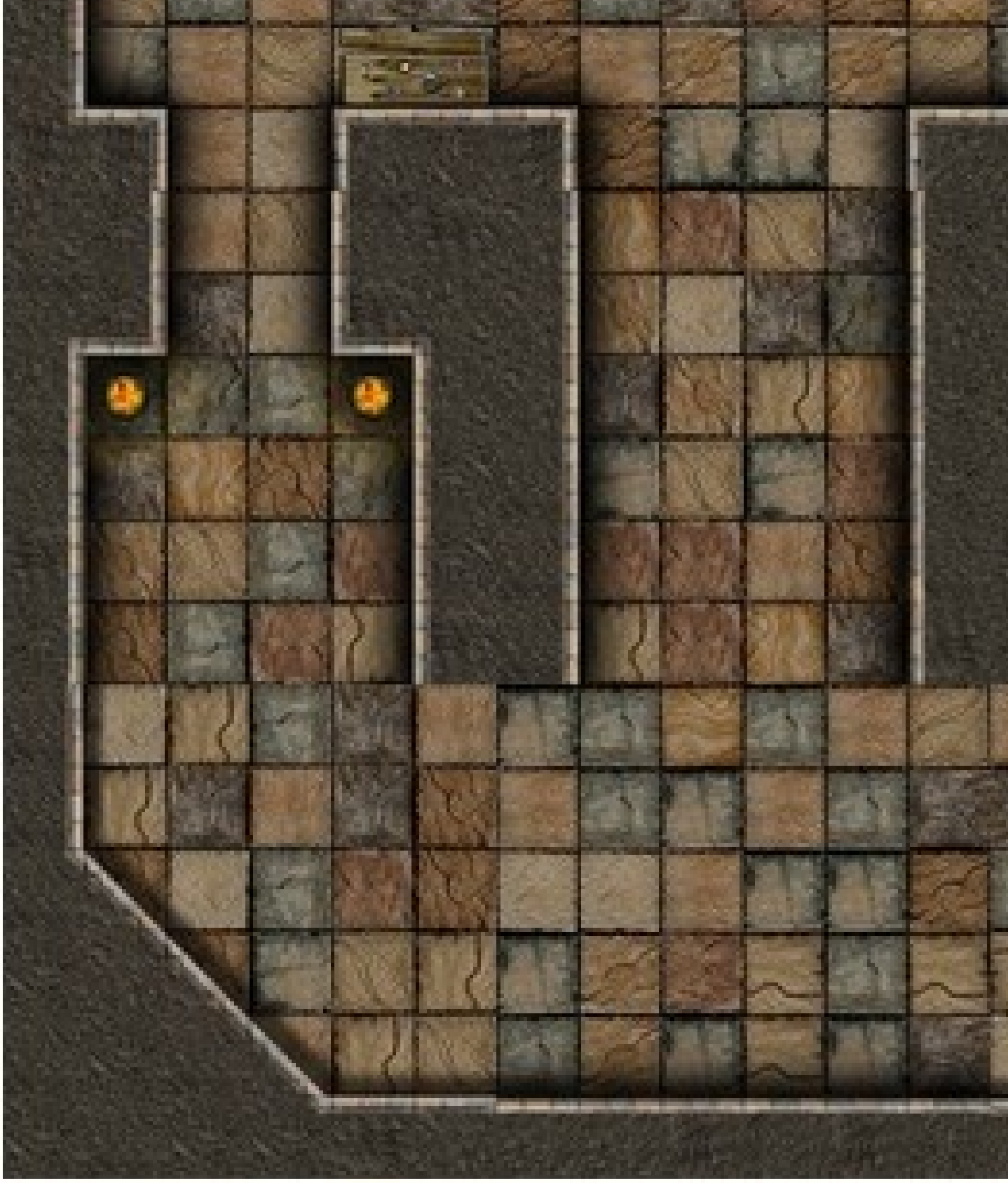
Esta câmara embaralha as posições e as direçoes se os ladrilhos não forem colocados corretamente. Você terá que usar uma estratégia para montar os ladrilhos antes que eles embaralhem.

Use os ladrilhos nomeados de A a L para montar a câmara mostrada abaixo, consulte o exercício A. Descubra quais posições os nomes misteriosos indicam.



```
from _spy.circus.circus import circus

MASMORRA = {'Cahuitz': 'AN', 'Cauha': 'AN', 'Coycol': 'AN',
 'Huatlya': 'AN', 'Micpe': 'AN', 'Nenea': 'AN',
 'Pallotl': 'AN', 'Tetlah': 'AN', 'Zitllo': 'AN'}


circus(5, MASMORRA)
```

Note: No texto “AN” a primeira letra determina o ladrilho e a segunda se está girada para Norte, Leste, Sul ou Oeste.



```
from _spy.circus.circus import circus

MASMORRA = {'Cahuitz': 'AN', 'Cauha': 'AN', 'Coycol': 'AN',
 'Huatlya': 'AN', 'Micpe': 'AN', 'Nenea': 'AN',
 'Pallotl': 'AN', 'Tetlah': 'AN', 'Zitllo': 'AN'}
```

(continues on next page)

(continued from previous page)

```
circus(6, MASMORRA)
```

Note: No texto "AN" a primeira letra determina o ladrilho e a segunda se está girada para Norte, Leste, Sul ou Oeste.

2.8 Primeiro Cenário do Jogo

Vamos começar importando o módulo Circus para criar um jogo baseado na biblioteca Phaser. Vamos criar uma organização em Python chamada *class* que vai ter duas operações *preload* e *create*. O objetivo é mostrar a imagem abaixo:

```
from _spy.circus.game import Circus


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus de poder criar um␣
↪jogo"""

    def preload(self):
        """Aqui no preload carregamos os recursos usados no jogo, neste caso a imagem␣
↪masmorra"""
        self.image("fundo", "http://<descubra um jeito de achar a url que vai ser␣
↪posta aqui>")

    def create(self):
        """Aqui colocamos a imagem masmorra na tela do jogo"""
        self.sprite("fundo")

if __name__ == "__main__":
    Jogo()
```

Note: Ainda é um programa bem simples.

2.9 Montando a Cena com Ladrilhos

Na maior parte dos jogos o cenário tem que ser montado a partir de uma folha de ladrilhos. As folhas de ladrilho são numeradas de 0 a n da esquerda para direita de cima para baixo. Esta é a folha de ladrilhos:

```
from _spy.circus.game import Circus


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus de poder criar um
↪jogo"""

    def preload(self):
        """Aqui no preload carregamos os recursos usados no jogo, neste caso a folha
↪de ladrilhos"""
        self.spritesheet("ladrilho", "http://<advinha!>", 128, 128, 12)
```

(continues on next page)

(continued from previous page)

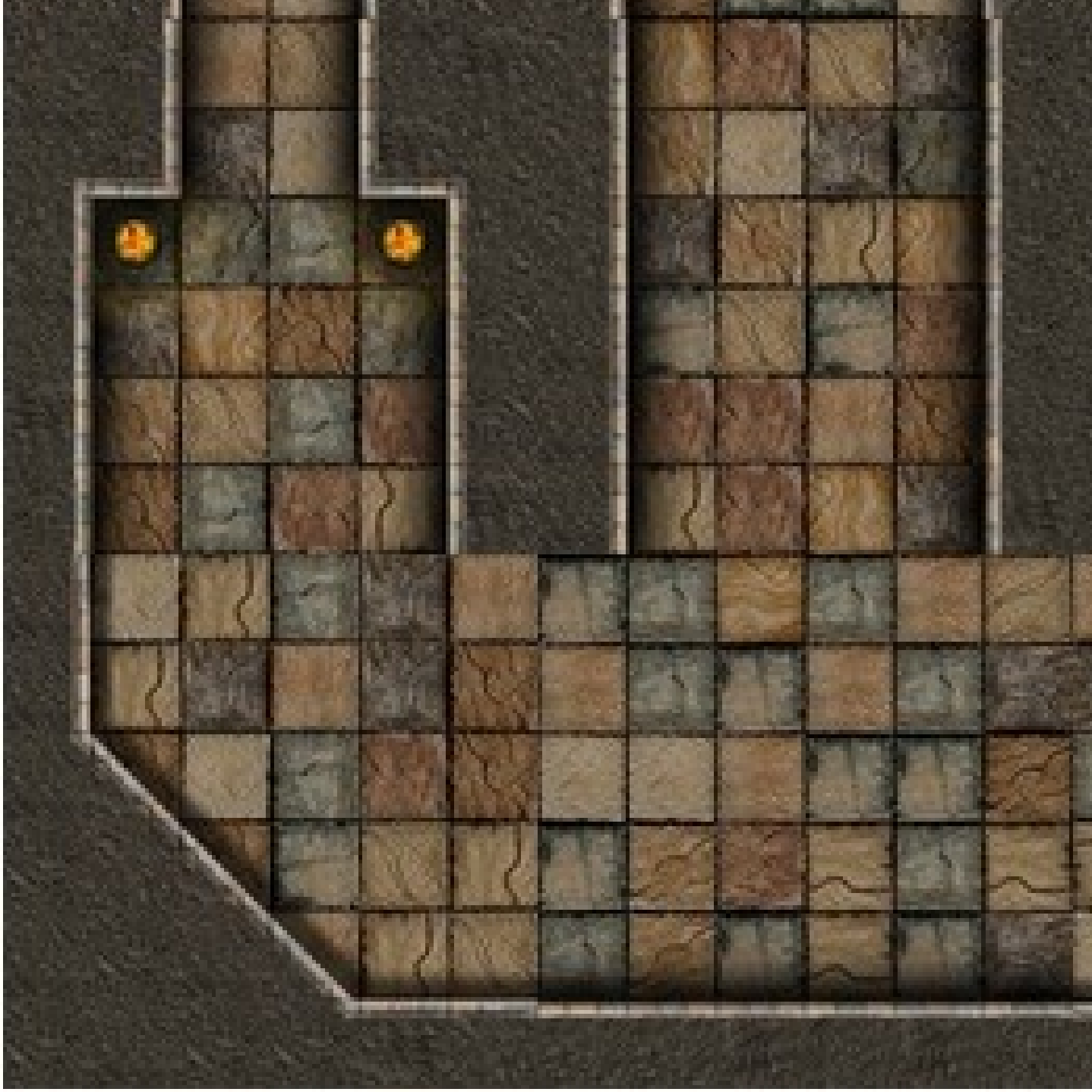
```
    def create(self):
        """Aqui colocamos cada ladrilho indicando a posição na tela e depois␣
↪selecionando o ladrilho"""
        um_ladrilho = self.sprite("ladrilho", 0, 0)
        um_ladrilho.frame = 5  # este número seleciona o ladrilho que vai ser colocado
        um_ladrilho = self.sprite("ladrilho", 0, 0)  # mude a posição do ladrilho
        um_ladrilho.frame = 5  # troque o ladrilho!
        # Coloque mais dois outros ladrilhos

if __name__ == "__main__":
    Jogo()
```

Note: Ainda é um programa bem simples.

2.10 Cena com Ladrilhos Girados

Vamos montar uma outra masmorrra, mas que agora requer que rotacionemos os ladrilhos. Veja os comando *angle* e *anchor* no código. A folha de ladrilhos continua esta:

Procure reproduzir este labirinto:



```
from _spy.circus.game import Circus


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus necessários para␣
↪criar um jogo"""

    def preload(self):
        """Aqui no preload carregamos os recursos usados no jogo, neste caso a folha␣
↪de ladrilhos"""
        self.spritesheet("ladrilho", "http://<advinha!>", 128, 128, 12)

    def create(self):
        """Aqui colocamos cada ladrilho indicando a posição na tela e depois␣
↪selecionando o ladrilho"""
```

(continues on next page)

(continued from previous page)

```
        um_ladrilho = self.sprite("ladrilho", 0, 0)
        um_ladrilho.frame = 5  # este número seleciona o ladrilho que vai ser colocado
        um_ladrilho.anchor.setTo(0.5, 0.5)  # este comando faz com que a rotação seja␣
↪no centro do ladrilho
        um_ladrilho.angle = 90  # está é a rotação em graus do ladrilho
        um_ladrilho = self.sprite("ladrilho", 0, 0)  # mude a posição do ladrilho
        um_ladrilho.frame = 5  # troque o ladrilho!
        um_ladrilho.anchor.setTo(0.5, 0.5)
        um_ladrilho.angle = 90  # troque o ângulo!
        # Coloque aqui o resto dos ladrilhos

if __name__ == "__main__":
    Jogo()
```

Note: Observe que ao colocar a âncora no centro do ladrilho, este centro também servirá para posicionar o ladrilho. Ajuste as cordenadas para que tudo fique correto.

2.11 Cena com Mais Ladrilhos Girados

Vamos montar uma outra masmorrra, um pouco maior. Em vez de copiar várias vezes o código, vamos usar uma *lista* de *tuplas* e o comando *for* A folha de ladrilhos continua esta:

Procure reproduzir este labirinto:

```
from _spy.circus.game import Circus


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus necessários para␣
↪criar um jogo"""
    MOSAICO = [
    [(5, 90), (5, 90), (5, 90), (5, 90)],
    [(5, 90), (5, 90), (5, 90), (5, 90)],
    [(5, 90), (5, 90), (5, 90), (5, 90)],
    [(5, 90), (5, 90), (5, 90), (5, 90)]
```

(continues on next page)

(continued from previous page)

```
    ]  # esta é uma lista de listas. Cada lista é uma linha e cada tupla dá o␣
↪ladrilho (5) e o ângulo (90)

    def preload(self):
        """Aqui no preload carregamos os recursos usados no jogo, neste caso a folha␣
↪de ladrilhos"""
        self.spritesheet("ladrilho", "http://<advinha!>", 128, 128, 12)

    def create(self):
        """Aqui colocamos um for que varre cada linha e cada coluna da matriz"""
        azulejo = self.MOSAICO
        for y in range(4):
            for x in range(4):
                frame, angle = self.MOSAICO[y][x]  # este comando desmembra a tupla em␣
↪duas variáveis, frame e angle
                um_ladrilho = self.sprite("ladrilho", 64+x*128, 64+y*128)  # cria e␣
↪muda a posição do ladrilho
                um_ladrilho.frame = frame
                um_ladrilho.anchor.setTo(0.5, 0.5)
                um_ladrilho.angle = angle

if __name__ == "__main__":
    Jogo()
```

Note: Observe que ao colocar a âncora no centro do ladrilho, este centro também servirá para posicionar o ladrilho. Ajuste as cordenadas para que tudo fique correto.

2.12 Cena com Muitos Ladrilhos Girados

Vamos montar a masmorra original, bem maior. Em vez de copiar várias vezes o código, vamos usar um *dicionário* de *tuplas* e o comando *for* A folha de ladrilhos continua esta:

Procure reproduzir este labirinto:

```
from _spy.circus.game import Circus

A, B, C, D, E, F, G = 0, 128, 256, 256+128, 512, 512+128, 512+256


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus necessários para␣
↪criar um jogo"""
    MOSAICO = {
    (A, A): (5, 90), (B, A): (5, 90), (C, A):(5, 90), (D, A):(5, 90), (E, A):(5, 90),␣
↪(F, A):(5, 90), (G, A):(5, 90),
```

(continues on next page)

(continued from previous page)

```
    (A, B): (5, 90), (B, B): (5, 90), (C, B):(5, 90), (D, B):(5, 90), (E, B):(5, 90),␣
↪(F, B):(5, 90), (G, B):(5, 90),
    (C, C): (5, 90), (B, C): (5, 90), (C, C):(5, 90), (D, C):(5, 90), (E, C):(5, 90),␣
↪(F, C):(5, 90), (G, C):(5, 90),
    (D, D): (5, 90), (B, D): (5, 90), (C, D):(5, 90), (D, D):(5, 90), (E, D):(5, 90),␣
↪(F, D):(5, 90), (G, D):(5, 90),
    (E, E): (5, 90), (B, E): (5, 90), (C, E):(5, 90), (D, E):(5, 90), (E, E):(5, 90),␣
↪(F, E):(5, 90), (G, E):(5, 90),
   }  # este é um dicionário de tuplas. Cada chave é a posição do ladrilho e cada␣
↪tupla dá o ladrilho (5) e o ângulo (90)

    def preload(self):
        """Aqui no preload carregamos os recursos usados no jogo, neste caso a folha␣
↪de ladrilhos"""
        self.spritesheet("ladrilho", "http://<advinha!>", 128, 128, 12)

    def create(self):
        """Aqui colocamos cada ladrilho indicando a posição na tela e depois␣
↪selecionando o ladrilho"""
        for (x, y), (frame, angle) in self.MOSAICO.items():
            # coloque o resto aqui

if __name__ == "__main__":
    Jogo()
```

Note: Observe que ao colocar a âncora no centro do ladrilho, este centro também servirá para posicionar o ladrilho. Ajuste as cordenadas para que tudo fique correto.

2.13 Colocando Personagens

Vamos criar uma outra classe para um personagem monstro. O monstro vai ser criado na invocação *__init__* do Jogo. Use a folha de ladrilhos para programar os monstros:

```
from _spy.circus.game import Circus, Actor


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus de poder criar um jogo"""
    def __init__(self):
        super().__init__()  # super é invocado aqui para preservar os poderes recebidos do Circus
        self.ladrilho_monstro = "monstro"
        self.monstro = Monstro(self.ladrilho_monstro, 0, 0, 0)

    def preload(self):
        """Aqui no preload carregamos a imagem masmorra e a folha de ladrilhos dos monstros"""
        self.image("fundo", "http://<descubra um jeito de achar a url que vai ser posta aqui>")
        self.spritesheet(self.ladrilho_monstro, "http://<advinha!>", 64, 63, 16*12)

    def create(self):
        """Aqui colocamos a imagem masmorra na tela do jogo"""
        self.sprite("fundo")


class Monstro(Actor):
```

(continues on next page)

(continued from previous page)

```
    """Essa  é a classe Monstro que controla os personagens do jogo"""
    def __init__(self, nome, frame, x, y):
        super().__init__()
        self.nome, self.frame, self.x, self.y = nome, frame, x, y

    def create(self):
        """Aqui colocamos o sprite do monstro e selecionamos o frame que o representa"
↪""
        self.monstro = "<troque isto pela criação do sprite, use self.nome, self.x e␣
↪self.y>"
        "escolha aqui o frame e use self.frame"
        self.monstro.anchor.setTo(0.5, 0.5)

if __name__ == "__main__":
    Jogo()
```

2.14 Colocando Mais Personagens

Vamos criar outros monstros. A operação *[f(x) for x in <Coleção>]* gera uma lista varrendo a Coleção dada e executando a função f(x) para cada x na Coleção.

```
from _spy.circus.game import Circus, Actor

MONSTROS = [(0, 0, 0), (0, 0, 0), (0, 0, 0)]


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus de poder criar um␣
↪jogo"""
    def __init__(self):
        super().__init__()  # super é invocado aqui para preservar os poderes recebidos␣
↪do Circus
        self.ladrilho_monstro = "monstro"
        self.monstro = [Monstro(self.ladrilho_monstro, frame, x, y) for frame, x, y in␣
↪MONSTROS]

    def preload(self):
        """Aqui no preload carregamos a imagem masmorra e a folha de ladrilhos dos␣
↪monstros"""
        self.image("fundo", "http://<descubra um jeito de achar a url que vai ser posta␣
↪aqui>")
        self.spritesheet(self.ladrilho_monstro, "http://<advinha!>", 64, 63, 16*12)

    def create(self):
        """Aqui colocamos a imagem masmorra na tela do jogo"""
        self.sprite("fundo")


class Monstro(Actor):
    """Essa  é a classe Monstro que controla os personagens do jogo"""
    def __init__(self, nome, frame, x, y):
        super().__init__()
        self.nome, self.frame, self.x, self.y = nome, frame, x, y

    def create(self):
        """Aqui colocamos o sprite do monstro e selecionamos o frame que o representa"""
        self.monstro = "<troque isto pela criação do sprite, use self.nome, self.x e␣
↪self.y>"
        "escolha aqui o frame e use self.frame"
        self.monstro.anchor.setTo(0.5, 0.5)

if __name__ == "__main__":
  Jogo()
```

2.15 Animando Personagens

Vamos criar uma animação para o monstro. Use os ladrilhos onde o monstro apresenta as diversas posições:

```
from _spy.circus.game import Circus, Actor


class Jogo(Circus):
    """Essa  é a classe Jogo que recebe os poderes da classe Circus de poder criar um␣
↪jogo"""
    def __init__(self):
        super().__init__()  # super é invocado aqui para preservar os poderes␣
↪recebidos do Circus
        self.ladrilho_monstro = "monstro"
        self.monstro = Monstro(self.ladrilho_monstro, 0, 0, 0)

    def preload(self):
        """Aqui no preload carregamos a imagem masmorra e a folha de ladrilhos dos␣
↪monstros"""
        self.image("fundo", "http://<descubra um jeito de achar a url que vai ser␣
↪posta aqui>")
        self.spritesheet(self.ladrilho_monstro, "http://<advinha!>", 64, 63, 16*12)

    def create(self):
        """Aqui colocamos a imagem masmorra na tela do jogo"""
        self.sprite("fundo")


class Monstro(Actor):
```

(continues on next page)

(continued from previous page)

```
    """Essa  é a classe Monstro que controla os personagens do jogo"""
    def __init__(self, nome, frame, x, y):
        super().__init__()
        self.nome, self.frame, self.x, self.y = nome, frame, x, y

    def create(self):
        """Aqui colocamos o sprite e o frame do monstro e acrescentamos a animação"""
        self.monstro = "<troque isto pela criação do sprite, use self.nome, self.x e␣
↪self.y>"
        "escolha aqui o frame e use self.frame"
        self.monstro.anchor.setTo(0.5, 0.5)
        self.monstro.animations.add('mon', [self.frame, self.frame+1, self.frame+2,],␣
↪4, True)
        self.monstro.play('mon')

if __name__ == "__main__":
    Jogo()
```

2.16 Movimentando Personagens

Vamos movimentar monstros. Para isso é preciso acrescentar a função update na classe Monstro.

```
from _spy.circus.game import Circus, Actor
from random import random

MONSTROS = [(100, 100, 100), (0, 200, 100), (50, 300, 100)]
DIR = [(1,0), (1,1), (0,1), (-1,1), (-1,0), (-1,-1), (0,-1), (1,-1)]


class Jogo(Circus):
   """Essa  é a classe Jogo que recebe os poderes da classe Circus de poder criar um␣
↪jogo"""
   def __init__(self):
      super().__init__()  # super é invocado aqui para preservar os poderes recebidos␣
↪do Circus
      self.ladrilho_monstro = "monstro"
      self.monstro = [Monstro(self.ladrilho_monstro, frame, x, y) for frame, x, y in␣
↪MONSTROS]

   def preload(self):
      """Aqui no preload carregamos a imagem masmorra e a folha de ladrilhos dos␣
↪monstros"""
      self.image("fundo", "http://<descubra um jeito de achar a url que vai ser posta␣
↪aqui>")
      self.spritesheet(self.ladrilho_monstro, "http://<advinha!>", 64, 63, 16*12)

   def create(self):
      """Aqui colocamos a imagem masmorra na tela do jogo"""
      self.sprite("fundo")


class Monstro(Actor):
    """Essa  é a classe Monstro que controla os personagens do jogo"""
    def __init__(self, nome, frame, x, y):
        super().__init__()
        self.nome, self.frame, self.x, self.y = nome, frame, x, y
        self.first = True
        self.direction = 0

    def create(self):
        """Aqui colocamos o sprite do monstro e selecionamos o frame que o representa
↪"""
        self.monstro = "<troque isto pela criação do sprite, use self.nome, self.x e␣
↪self.y>"
        "bote aqui self.??o que??".frame = self.frame
        self.monstro.anchor.setTo(0.5, 0.5)
        "ponha aqui a animação"
        self.enable(self.monstro)

    def update(self):
        player = self.monstro
        def redirect():
            self.first = False
            self.direction = d = int(random() * 8.0)
            x, y = DIR[d]
            return x * 150, y * 150

        player.angle = (self.direction*45+270) % 360
        if int(random() + 0.02) or self.first:
```

(continues on next page)

(continued from previous page)

```
            player.body.velocity.x, player.body.velocity.y = redirect()

if __name__ == "__main__":
  Jogo()
```

CHAPTER 3

Bem vindo à Documentação SuperPython

3.1 SuperPython - Introdução

3.2 SuperPython - Manual

Este ambiente permite rodar um game stand alone e criar módulos que não estão no menu.

Para rodar um game basta chamar <projeto>.is-by.us/code/_<modulo>. O submódulo *main.py* será importado.

Para criar ou acessar um módulo fora do menu basta clicar a letra **O** no canto inferior direito do menu principal.

A biblioteca Phaser está disponível baixada do CDN. Use: *from browser import window; window.Phaser*

O projeto que se está usando pode ser caracterizado como a primeira palavra da url.

Pode se importar novos módulos adicionando o prefixo (módulo) *_spy* na frente do nome do módulo que se quer importar.

3.3 SuperPython - Modulos

SuperPython é programado em Brython

Funcionalidades Documentadas:

- Modelo do SuperPython : Entidades Basicas *SuperPython - Módulos Principais*
- Testes Unitários do SuperPython : *SuperPython - Módulos de Teste*

3.4 Notas de Lançamento V. 2.1.0

SuperPython

3.4.1 Milestone

musicalmice - Interface remota do instrutor

3.4.2 Aspectos do Lançamento

Este lançamento adiciona e divide as abas de tutoriais, comporta as funcionalidades 3D , inclui tutorial Sphynx e corrige "Burger Menu"e concertar a recuperaço de códigos do banco de dados.

Destaques dos Aspectos

Adaptação para melhor visualizar a plataforma a partir do celular.

Aspecto #1

Adição e divisão de abas nos tutoriais.

Aspecto #2

Adição de elementos em 3D para a construção de games.

Aspecto #3

Inclusão do tutorial do Jogo do Tesouro.

3.4.3 Melhoramentos nos Tutoriais

Melhoramentos #1

Melhor separação da área de tutoriais.

Melhoramento #2

Adição do tutorial : Vitollino. Tutorial que ensina a fazer um jogo enquanto se jogo o próprio jogo que está sendo feito.

Melhoramento #3

Adição do tutorial: SuPyGirls Connection. Tutorial Sphinx que ensina a criar elementos básicos de um jogo.

Melhoramento #4

Quebra de turorial em vários rsts.

Melhoramento #5

Adição do tutorial: Super Python.

Melhoramento #6

Dividir os tutoriais.

Melhoramento #7

Adição de abas nos tutoriais.

3.4.4 Melhoramentos no Jogo em 3D

Melhoramentos #1

Adiciona biblioteca glowscript. Biblioteca em javascript que suporta WEBGL

Melhoramento #2

Criação de módulo VPython no Vitollino.

Melhoramento #3

Inclusão das funções extrusion e ellipsoid.

3.4.5 Melhoramentos no Jogo do Tesouro

Melhoramentos #1

Desenvolvimento do código que virtualiza o jogo do Tesouro .

Melhoramento #2

Inclusão do jogo como recurso da plataforma SuPyGirls.

Melhoramento #3

Desafio de aprendizado em Inteligência Artificial na construção dos jogadores robóticos do jogo do Tesouro.

3.4.6 Consertos

Conserto #01

Correção Burger menu.

Conserto #02

Correção do code controller.

3.4.7 Questões e Problemas Conhecidos

Funcionalidade remota não foi implementada devido a avanços em outra áreas do projeto.

3.4.8 Lançamentos Anteriores e Posteriores

Lançamento Anterior: Junho 2018 *Lançamento 2.0.0*

3.5 Lançamentos Anteriores

3.5.1 Notas de Lançamento V. 2.0.0

SuperPython

Milestone

pearlharbour - Nova interface remodelada

Aspectos do Lançamento

Este lançamento inaugura uma interface renovada, integrando o acesso à várias plataformas que estavam sendo desenvolvidas em separado. Devido à esta mudança estrutural a vesão foi atualizada de 1.x.x para 2.0.0

Destaques dos Aspectos

A integração das plataformas anteriores, adição de tutoriais e documentação e o novo design responsivo são os maiores destaques deste lançamento.

Aspecto #1

Acesso integrado às plataformas SuperPython, Kwarwp e SuPyGirls.

Melhoramentos

Importar o caminho para outros módulos de programação.

Melhoramento #1

Experimentar a importação relativa da fonte Brython.

Melhoramento #2

Importar o caminho Brython.

Melhoramento #3

Obter Vitollino a partir do Github.

Melhoramento #4

Superpython - WebIDE para ensino de Python com Games importado.

Melhoramento #5

Possibilidade de salvar o novo código do usuário.

Melhoramento #5

Adicionar Kwarwp.

Melhoramento #6

Criar novo módulo no Github.

Aspecto #2

Adição de tutoriais e documentação.

Melhoramentos

Inclusão de tutoriais sobre cada plataforma.

Melhoramento #1

Acréscimo do Tutorial SuPyGirls Connection.

Melhoramento #2

Adicionar documentação SuperPyhton.

Melhoramento #3

Adição do Tutorial Vitollino.

Melhoramento #4

Adicionar Tutorial Introdução à Computação.

Melhoramento #5

Dividir os tutoriais em diversos .rst.

Aspecto #3

Interface com novo design mais responsivo.

Melhoramentos

Detalhamento visual.

Melhoramento #1

Interface para informar o repositório e apresentar a imagem e os nomes.

Melhoramento #2

Refinamento gráfico usando o projeto CSS Bulma.

Melhoramento #3

Destacar tema Cloud para o pythonanywhere.

Melhoramento #4

Tela de escolher projetos.

Melhoramento #5

Melhorar responsividade.

Melhoramento #6

Acréscimo de tela de escolher projetos.

Melhoramento #7

Adicionar imagens dos projetos com links.

Consertos

O código foi consertado.

Conserto #01

Conserto de controllador do código.

Questões e Problemas Conhecidos

1. Ainda falta melhoria quanto à escrita dos textos dos tutorias.
2. Há necessidade de uma melhor disposção visual tanto dos textos quanto das figuras/gifs nas páginas.

Lançamentos Anteriores e Posteriores

Próximo Lançamento: Maio 2018 Lançamento 2.1.0

Lançamento Anterior: Fevereiro 2018 *Lançamento 1.1.3*

3.5.2 Lançamentos Anteriores

Notas de Lançamento V. 1.1.3

SuperPython

Milestone

hidenseek - Suporte a importação cruzada

Aspectos do Lançamento

Destaques dos Aspectos

Este ambiente permite criar novos módulos ou arquivos.

Aspecto #1

Para criar ou acessar um módulo fora do menu basta clicar a letra **O** no canto inferior direito do menu principal.

Melhoramentos

Novo menu visual e gif de carregamento.

Melhoramento #1

O novo menu visual é simétrico e suporta até cinquenta usuários.

Melhoramento #2

Um gif animado com engrenagens rodando indica a carga do editor.

Consertos

O login, a url de projeto, novos módulos e importação de módulos foram consertados

Conserto #01

Quando se loga em um módulo ele fica tarjado no menu principal.

Conserto #01

Quando se loga em um módulo ele fica tarjado no menu principal.

Conserto #02

O projeto que se está usando pode ser caracterizado como a primeira palavra da url.

Conserto #03

Pode se criar e editar novos módulos e ou arquivos pelo menu extra principal.

Conserto #04

Pode se importar novos módulos adicionando o prefixo (módulo) *_spy* na frente.

Questões e Problemas Conhecidos

1. Ainda está congelado na versão antiga do Brython 3.0.2
2. Aparece uma imagem espúria *em construção* quando roda o jogo *stand alone*.

Lançamentos Anteriores e Posteriores

Próximo Lançamento: Abril 2018 *Lançamento 2.0.0*

Lançamento Anterior: Novembro 2015 *Lançamento 1.1.2*

Lançamentos Anteriores

Notas de Lançamento V. 1.1.2

SuperPython

Milestone

hidenseek - Suporte a importação cruzada

Aspectos do Lançamento

Destaques dos Aspectos

Este ambiente permite rodar um game stand alone e criar módulos que não estão no menu. Agregando a lib do Phaser.

Aspecto #1

Para rodar um game basta chamar <projeto>.is-by.us/code/_<modulo>. O submódulo *main.py* será importado.

Aspecto #2

Para criar ou acessar um módulo fora do menu basta clicar a letra **O** no canto inferior direito do menu principal.

Aspecto #3

A biblioteca Phaser está disponível baixada do CDN.

Melhoramentos

Novo menu visual e gif de carregamento.

Melhoramento #1

O novo menu visual é simétrico e suporta até cinquenta usuários.

Melhoramento #2

Um gif animado com engrenagens rodando indica a carga do editor.

Consertos

Ajuste da estrutura de configuração para facilitar os testes

Conserto #01

Adiciona o arquivo *vendor.py* para eliminar lib no import do bottle.

Questões e Problemas Conhecidos

1. Ainda está congelado na versão antiga do Brython 3.0.2
2. Aparece uma imagem espúria *em construção* quando roda o jogo *stand alone*.

Lançamentos Anteriores e Posteriores

Próximo Lançamento: A ser definido *Lançamento 1.1.3*

Lançamento Anterior: Novembro 2015 *Lançamento 1.1.1*

Notas de Lançamento V. 1.1.1

SuperPython

Milestone

hidenseek - Suporte a importação cruzada

Aspectos do Lançamento

Destaques dos Aspectos

Este ambiente permite a edição de códigos e importação do módulo de um outro projeto.

Aspecto #1

o nome __name__ é atribuído "__main__" no programa principal.

Melhoramentos

Versão para uso do Google Application Engine e PyBuilder.

Melhoramento #1

Foi melhorada a configuração para uso do construtor de aplicativos PyBuilder. A configuração *build.py* define os aspectos necessários para verificar testes, cobertura, cabeçalhos e detalhes para deployment.

Consertos

Conserto #01

o nome __name__ é atribuído "__main__" no programa principal.

Conserto #02

A carga do módulo principal é feita por AJAX, evitando o congelamento do HTML.

Questões e Problemas Conhecidos

A funcionalidade ainda é muito simples, requer melhorias.

Uma nova versão deve suportar o monitoramento da atividade dos alunos.

Lançamentos Anteriores e Posteriores

Próximo Lançamento: Maio 2016 *Lançamento 1.1.2*

Lançamento Anterior: Julho 2015 *Lançamento 0.1.0*

Notas de Lançamento V. 0.1.0

SuperPython

Milestone

Bruce - Protótipo da inteface Gráfica

Aspectos do Lançamento

Destaques dos Aspectos

Este ambiente inicial permite apenas o teste da interface com o usuário

Aspecto #1

O ambiente apenas apresenta o visual da interface mas não permite nenhuma interação.

Melhoramentos

Versão para uso do Google Application Engine e PyBuilder.

Melhoramento #1

Foi adicionada uma configuração para uso do Google_Cloud. A configuração *app.yaml* define os aspectos necessários para rodar no servidor do Goolge Application Engine.

Melhoramento #2

Foi adicionada uma configuração para uso do construtor de aplicativos PyBuilder. A configuração *build.py* define os aspectos necessários para verificar testes, cobertura, cabeçalhos e detalhes para deployment.

Consertos

Nenhum conserto notável.

Questões e Problemas Conhecidos

A funcionalidade ainda é muito simples, requer melhorias.

Uma nova versão deve suportar o monitoramento da atividade dos alunos.

Lançamentos Anteriores e Posteriores

Próximo Lançamento: A ser definido *Lançamento 0.2.0*

3.6 SuperPython - Módulos Principais

3.6.1 Módulo Cliente

Brython front end client.

class view._core.main.**Main**(*br*)

Bases: object

error(*error*)

play()

post_id(*ev*, *form_id='ident-form'*, *address='_claim/'*, **_*)

scorer(*score*, *log='__score__.py'*)

start(*navigate=[['Edit', '_edit']]*)

view._core.main.**main**(***kwargs*)

Brython front end client.

class view._core.supygirls_factory.**Dialog**(*gui*, *text='xxxx'*, *act=<function Dialog.<lambda>>*)

Bases: object

action(*extra*)

del_err()

get_text()

hide()

remove()

set_csl(*text*)

set_err(*text*)

set_text(*text*)

show()

textarea(*text*, *style={'background': 'rgba(10, 10, 10, 0.5)', 'border': 1, 'borderColor': 'darkslategrey', 'color': 'navajowhite', 'flex': '3 1 auto', 'margin': '0', 'padding': 10, 'position': 'relative', 'resize': 'none', 'width': '99%'}*)

class view._core.supygirls_factory.**EmpacotadorDeImagem**(*canvas*, *glyph*, *x*, *y*, *dx*, *dy*)

Bases: object

do_move(*x*, *y*, *image=None*)

do_remove()

do_translate(*x*, *y*)

mover(*x*, *y*, *image=None*, *_, **__)

remove()

translate(*x*, *y*)

class view._core.supygirls_factory.**GUI**(*width=800*, *height=600*, *code=''*, *codename=''*, ***kwargs*)

Bases: view._core.supygirls_factory._GUI

O terreno onde o Festival Kuarup é apresentado

executa_acao(*dialog*, *action=None*)

get_code()

view._core.supygirls_factory.**random**() → x in the interval [0, 1).

See also:

Module client.superpython.core

Note: Unidade de Modelo Cliente.

3.6.2 Módulo Servidor

Controlador Principal

See also:

Module server.controllers.main_controller

Note: Controlador principal da funcionalide Web.

Controlador de Código

See also:

Module server.controllers.main_controller

Note: Controla também a invocação de game stand alone.

Controlador de Edição

See also:

Module `server.controllers.main_controller`

Note: Controla também a invocação de múdulos __init__.

3.7 SuperPython - Módulos de Teste

See also:

Modules *SuperPython - Módulos Principais*

Note: Módulo principal de testes unitários.

CHAPTER 4

Bem Vindos ao Tutorial SuPyGirls Connection

Bora programar!

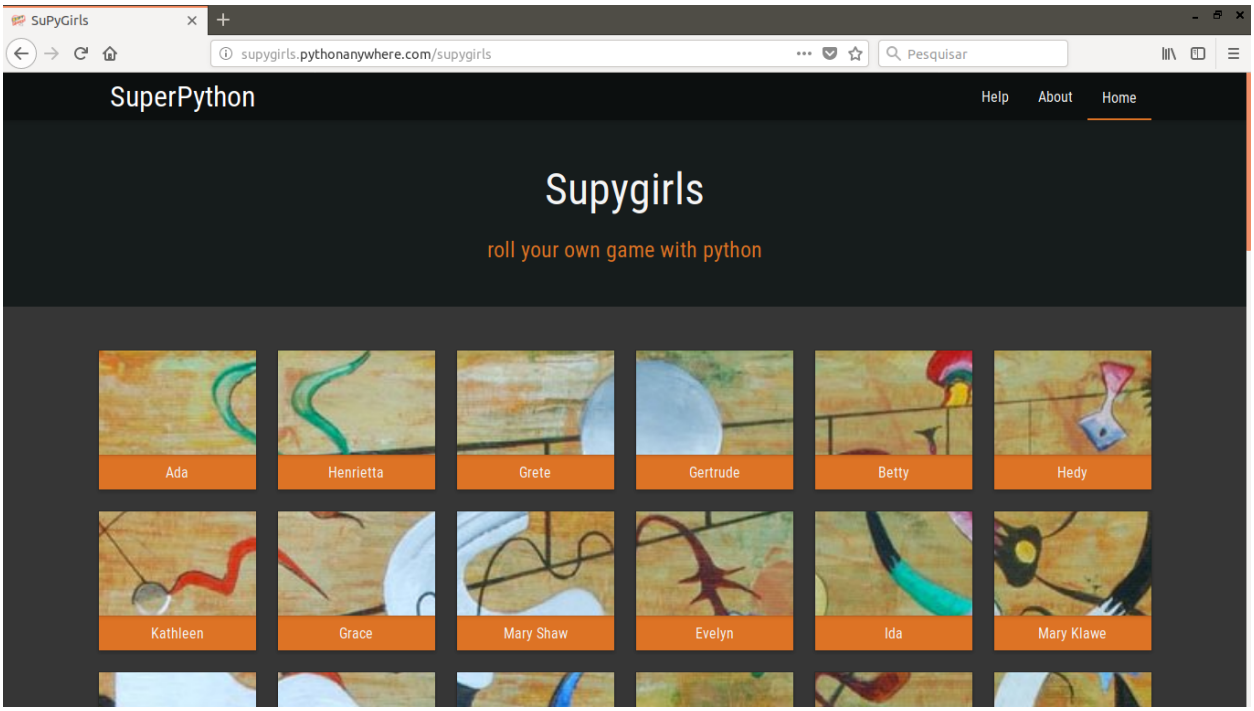
Bem vindos ao SuPyGirls, hoje nós vamos começar a nos aventurar pelo incrível mundo do desenvolvimento de Jogos.

A plataforma SuperPython é um ambiente de desenvolvimento online de Jogos em python que você pode acessar de qualquer lugar através da Internet. Nela você vai encontrar algumas ferramentas pensadas para te ajudar a aprender a programar enquanto constrói jogos. Esse tutorial vai focar no uso da ferramenta Vittolino para a construção de histórias em quadrinhos.

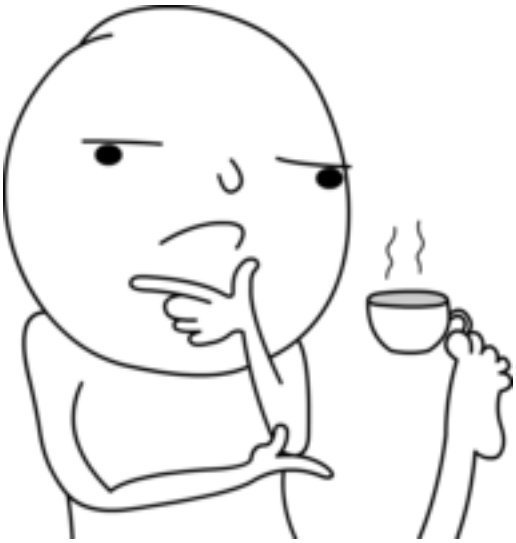
Primeiro, devemos acessar o endereço da plataforma SupyGirls do SuperPython, que é esse aqui ó:

http://supygirls.pythonanywhere.com/supygirls

O que deve te levar direto pra cá:

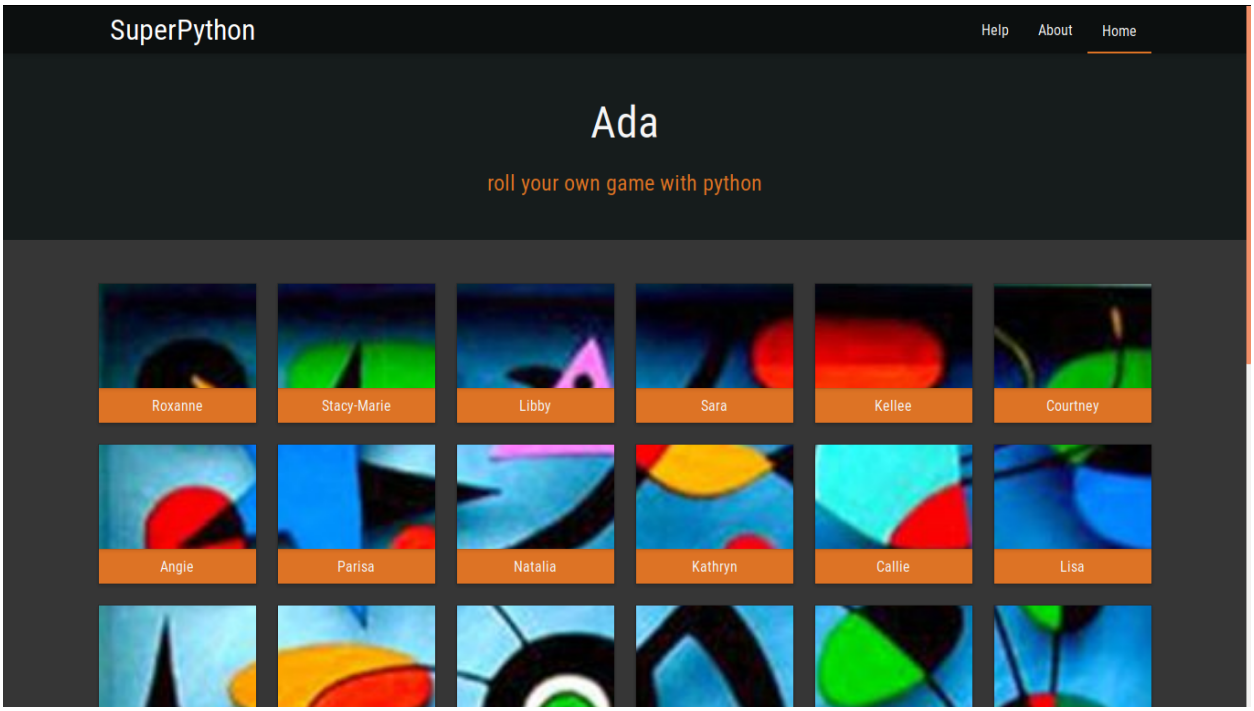


Cada quadrado nomeado nesse portal é uma sala para uma turma de desenvolvimento de jogos. Você deve entrar na sala da sua turma que seu professor ou professora indicar (se ele tiver enrolando, pede pra ele dizer logo qual é, nós queremos programar!). Mas o que será que todos esses nomes tem em comum?



(pausa reflexiva)

Na sala da sua turma, existem mais quadrados nomeados. Cada quadrado nomeado é um editor de jogos, ou seja, se você clicar em qualquer um deles você vai ser levado para um ambiente onde já vai poder começar a contruir com todas as doideiras que se passam dentro da sua cabeça perturbada!



Mais nomes... suspeito...

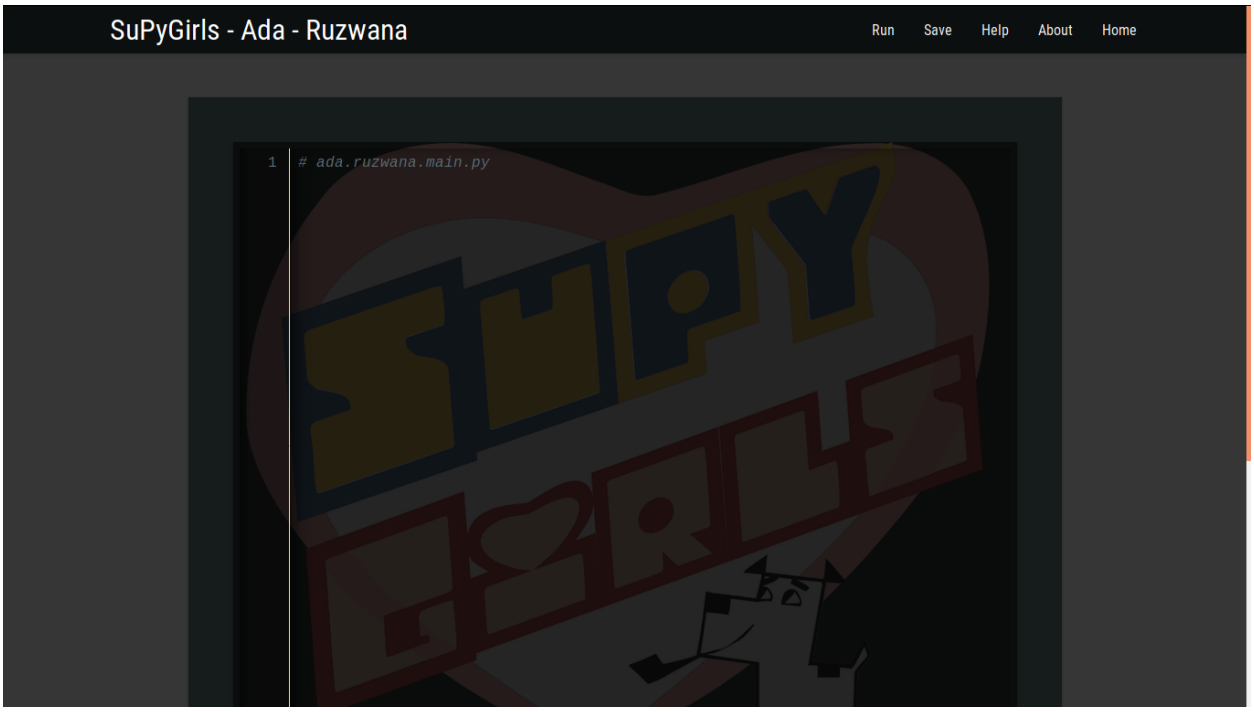
Enfim, de volta ao que importa, JOGOS!

4.1 Vittolino - Conte Estórias

Pronto, finalmente chegamos ao editor. Essa é a tela de início. Nela você pode observar no canto superior esquerdo o nome do projeto, o nome da Sala da sua turma e o nome do editor que você escolheu. Para continuarmos, precisaremos escolher a opção Edit.

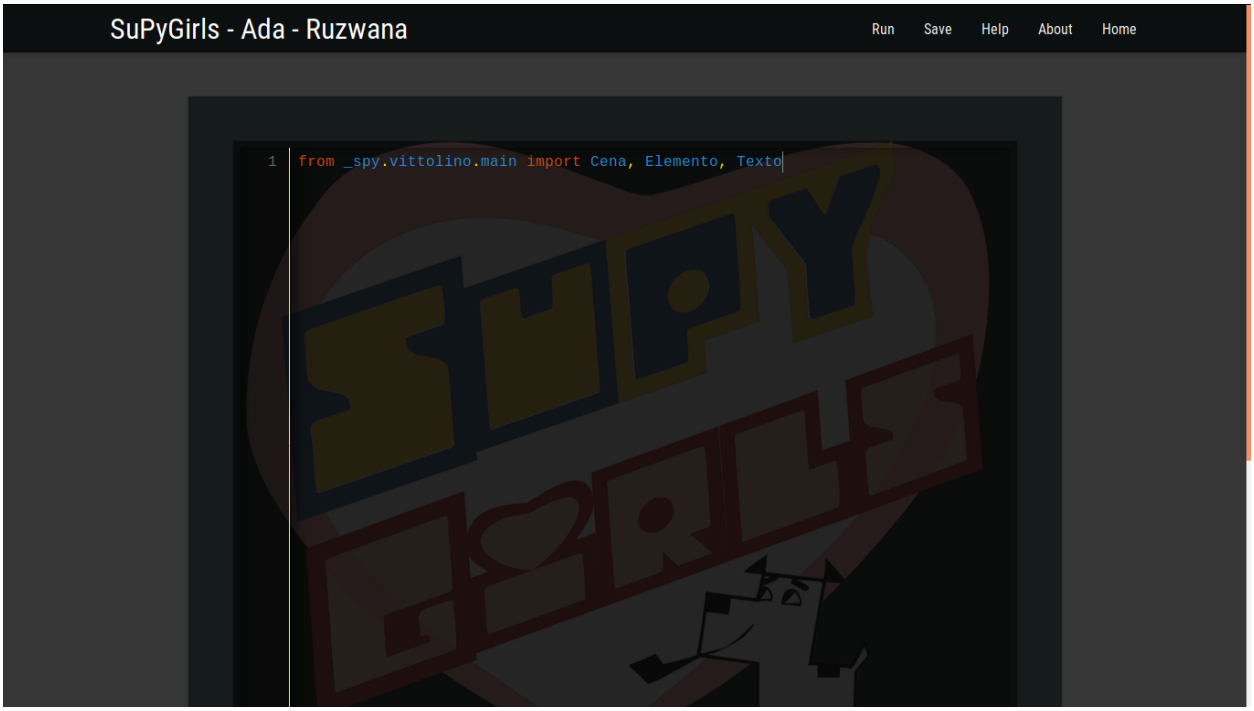
Agora chegamos ao editor do código Python. Esse código é como você vai descrever tudo o que o computador precisa fazer para ele construir o jogo que você imaginar. No momento, ele só tem uma linha. Essa linha começa com o caracter '#', o que significa que ela não será lida pelo computador, ela é um comentário feito para os seres humanos lerem. Eles servem para facilitar a leitura do código pelo programador.

Especificamente, essa linha é apenas uma marcação do título do documento de texto que nós vamos editar. Mas se você ficou confuso, não se preocupe, porque nós vamos apagá-la!



Agora, na primeira linha, nós vamos escrever o seguinte código:

from _spy.vittolino.main import Cena, Elemento, Texto



Que doido, o editor coloriu automaticamente algumas palavras para nós, o que isso deve significar: from _spy.vittolino.main import Cena, Elemento, Texto

4.2 A Arquitetura do Vittolino

```
from _spy.vittolino.main import Cena, Elemento, Texto
```

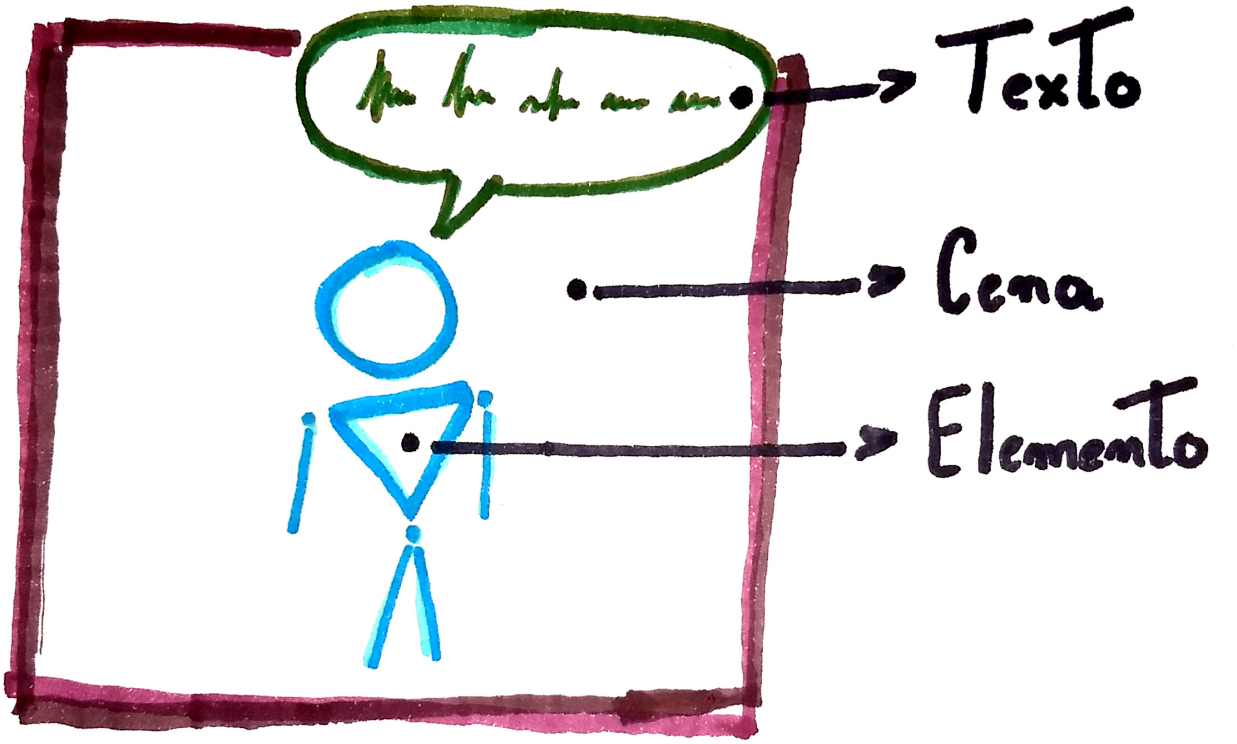
Agora é uma boa hora pra explicar como o vittolino funciona. Vamos analisar essa frase que nós acabamos de inserir no nosso código. Não se preocupe se não entender essas coisas agora, programar é do tipo de coisa que se aprende fazendo!

from Palavra em inglês que significa "de". ex.: Ele veio de dentro do oceano congelado da lua de Júpiter!

import Palavra em inglês que significa "importar". ex.: Mês passado eu importei 3 cangurus Uzbequistaneses!

Ou seja, de _spy.vittolino.main importar Cena, Elemento, Texto

A história em quadrinhos que vamos construir com o Vittolino é estrturada da seguinte maneira:



Cada quadro é uma Cena, que possui Elementos que, por sua vez, possuem um Texto. Sabendo disso, o que a nossa primeira linha de código deve pedir para o computador fazer?

Dentro do Vittolino, para construirmos algo que tem uma imagem (uma Cena ou um Elemento), precisamos que esta imagem esteja na Internet. Isso acontece porque quando formos pedir pro computador construir uma dessas coisas, a gente deverá dar para ele um link de uma imagem.

Agora é o momento de criar!

Antes de começarmos a implementar o jogo, precisamos pensar em qual história iremos contar! Busque na internet uma imagem de um cenário e uma imagem de um personagem para entrar no primeiro quadro de sua história. Se quiser já planejar toda a história e ir buscando as imagens, façam! Só não se esqueça de salvar e identificar os links que for encontrando.

Para o nosso exemplo, nós escolhemos os seguintes links:

Cena (Imagem de um Castelo)

https://img.elo7.com.br/product/original/1AD3471/painel-1x0-70-salao-de-festa-salao-de-festa.jpg

Elemento (Imagem de uma Barbie)

http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-barbie-20.png

Para copiar o link de uma imagem, clique com o botão direito nela e escolha a opção “Copiar Endereço da Imagem”, mas atenção! Existe a possibilidade de uma imagem que você queira pegar esteja com seu endereço protegido, por isso é sempre bom testar os links que vocês selecionarem. Perceba que se você clicar em alguns dos links acima, você é redirecionado para uma página que possui apenas a imagem e nada mais.

Nós podemos ir salvando os links no próprio código ou no bloco de notas. Se você escolher salvar no próprio código, você tem duas opções:

Salvar em um comentário. Utilize o '#' no começo de uma linha e escreva sua anotação depois. Salvar numa variável. Escolha um nome sem espaços e único e escreva nome = "link". O link deve estar entre aspas.

```
1  from _spy.vittolino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
```

Se você tem uma alma mais desenhista e quer criar os personagens e os cenários, também pode! Desenhe no computador ou tire fotos de algum desenho à mão e faça o upload do arquivo para um site de hosting de imagens, existem vários na Internet. Depois é só salvar os links onde suas imagens foram guardadas.

Também é o momento de se pensar nas falas de cada personagem em cada cena!

Para nosso exemplo, nós escolhemos o seginte Texto para a Barbie no Castelo:

"Vim para o castelo para me abrigar de um ataque de dragões."

Não esqueça de anotar em algum lugar, no bloco de notas, no código (da mesma maneira dos links) ou no próprio papel.

Pronto! Agora que temos o primeiro quadro planejado e preparado, podemos começar a implementar em código.

4.3 Implementação do Primeiro Quadro

Agora precisamos pedir ao computador duas coisas para o nosso Jogo começar a andar. Primeiro, precisamos pedir pra ele anotar como construir a nossa história, e depois que contarmos pra ele, devemos pedir pra que ele realize a construção da nossa história.

```
1  from _spy.vittolino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
5
6  def Historia():
7    #Aqui é onde dizemos pro computador como a história é construída.
8    #Perceba que todas as linhas começam com um "espaço".
9    #Esse espaço é na realidade uma "indentação", e pode ser feito
10   #com a tecla "tab" a segunda tecla abaixo da tecla ESC;
11
12 Historia() #Aqui é onde nós pedimos ao computador que execute aquilo que
13 ➤   ➤     #ensinamos a ele de como construir nossa história.
14   ➤ ➤     #Perceba que na linha 12 não há mais a identação,mostrando pro
15     ➤     #computador que acabaram as instruções de como se construir a
16       ➤   #Historia.
17
```

CALMA

Para contar para o computador como fazer algo, nós usamos a palavra-chave def. Ela significa definir, ou seja, você está definindo algo novo para o computador, algo que ele não sabia. Mas para usar essa palavra da seguinte maneira:

def NomeDaquiloQueVoceVaiEnsiarSemEspacosNemAcento():

Logo após essa instrução, nós escrevemos todas as instruções daquilo que estamos ensinando ao computador de maneira indentada. A indentação é um tipo de espaço especial que diz para o computador que as instruções indentadas em sequência fazem parte daquilo que você quer ensinar a ele.

```
1  from _spy.vittolino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
5
6  def Historia():
7    #Aqui é onde dizemos pro computador como a história é construída.
8    #Perceba que todas as linhas começam com um "espaço".
9    #Esse espaço é na realidade uma "indentação", e pode ser feito
10   #com a tecla "tab" a segunda tecla abaixo da tecla ESC;
11
12 Historia() #Aqui é onde nós pedimos ao computador que execute aquilo que
13 ➤    ➤     #ensinamos a ele de como construir nossa história.
14   ➤  ➤     #Perceba que na linha 12 não há mais a identação,mostrando pro
15     ➤      #computador que acabaram as instruções de como se construir a
16       ➤    #Historia.
17
```

mais uma vez, agora com mais calma.

Agora precisamos dizer para o computador, como construir a Historia. Primeiro vamos precisar de uma Cena.

Nossa primeira Cena é o Castelo, mas não é porque nós salvamos o link no código que o computador sabe que essa é a nossa intenção. Nós precisamos dizer de maneira clara pra ele. Então a primeira instrução de como construir nossa história vai ser:

nomeCena = Cena(img = “link”)

No caso do nosso exemplo:

```
1  from _spy.vittolino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
5
6  def Historia():
7  ➤    cenaCastelo = Cena(img = "https://img.elo7.com.br/product/original/1AD347
8  Historia()
```

Não se esqueçam de fechar as aspas e os parenteses! Agora falta muito pouco pra podermos fazer o nosso primeiro teste. Uma vez que já contamos que existe uma cena na nossa história, temos que falar pra cena “rodar”, fazemos isso através da função .vai() da própria cena. Nós escrevemos:

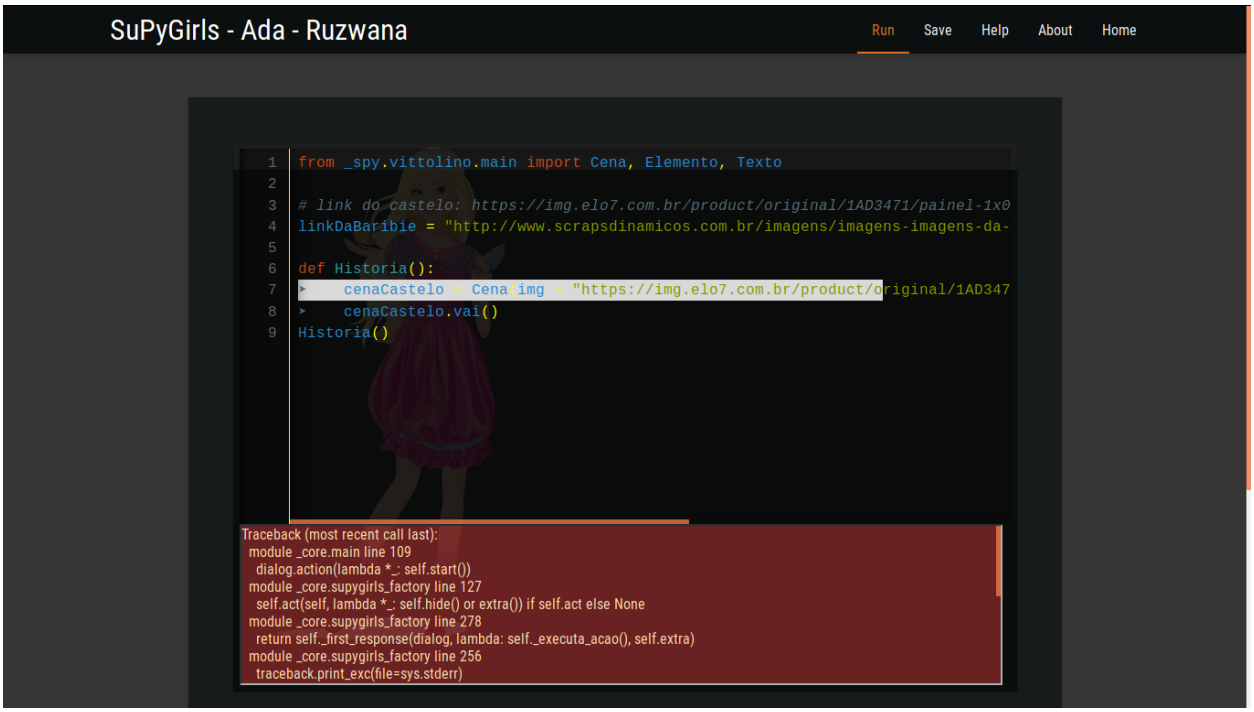
nomeCena.vai()

No caso do nosso exemplo:

```
from _spy.vittolino.main import Cena, Elemento, Texto

# link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-

def Historia():
    cenaCastelo = Cena(img = "https://img.elo7.com.br/product/original/1AD347
    cenaCastelo.vai()
Historia()
```

Agora vamos para o nosso primeiro teste! Procure o botão no canto direito superior escrito “Run” e reze para tudo dar certo!

RUFEM OS TAMBORES . . . .



ERROR

Tivemos um erro, nosso teste falhou. Alguma coisa em algum lugar deu errado. Algo de errado não está certo. Programar é uma constante batalha com os erros que assombram o mais astuto de todos os programadores. O computador não

pensa por nós, então qualquer erro, por menor que seja, tem o potencial de fazer nosso programa parar de funcionar.

"Mas o que eu faço?!", vocês podem perguntar.

Chequem tudo. Cheque se você usou as letras maiúsculas certas, se fechou todos os parênteses, se escreveu tudo certinho, se esqueceu ou não usou aspas, qualquer coisa pode ter acontecido. A mensagem em vermelho te dá uma dica do que pode ter acontecido no seu código. Muitas vezes ela diz precisamente onde o seu erro aconteceu, mas muitas vezes ela mais atrapalha do que ajuda. Nessas horas é bom pedir ajuda pro colega ao lado ou na Internet para tentar descobrir qual o seu erro. Sabe quando a gente tá procurando alguma coisa em casa e vem alguém e simplesmente acha? Na programação, também acontece. Às vezes.

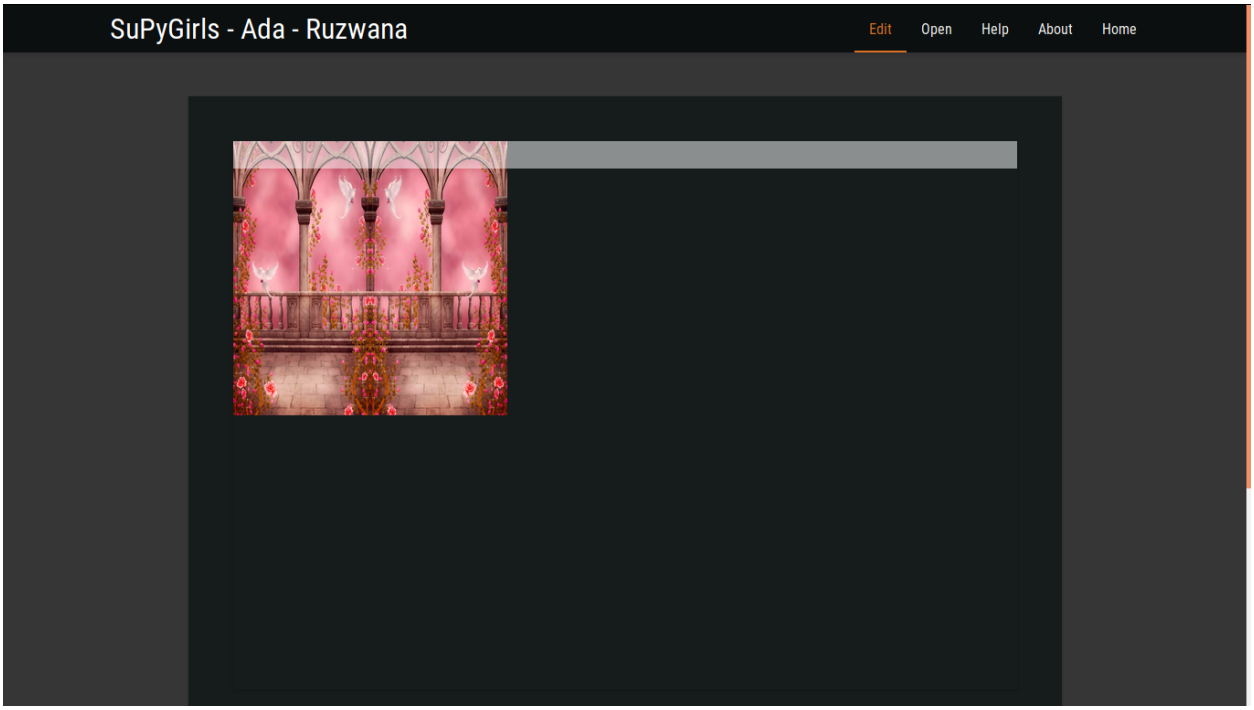
4.4 É Vitollino, não Vittolino!

Achei o meu erro!!

Eu vinha escrevendo Vitollino errado desde o começo, inclusive aqui nesse tutorial. Eu estava escrevendo com dois t's ao invés de dois l's. Erros pequenos como esses podem custar horas, se não dias de dor de cabeça, por isso, preste mais atenção que eu ao programar! Vamos consertar o código e ver se agora funciona.

```
1  from _spy.vitollino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
5
6  def Historia():
7      cenaCastelo = Cena(img = "https://img.elo7.com.br/product/original/1AD347
8      cenaCastelo.vai()
9  Historia()
```

AGORA VAI!



FOOOOOOOOOOOO951OOOOOOOIIIII!!!!!1!!!!!!!111 Existe uma sensação de alívio única de achar um erro que estava travando o seu código. Nossa Cena ainda não faz nada além de simplesmente aparecer, mas pra quem não tinha nem isso, acho digno de uma comemoração.

A próxima coisa que precisamos fazer é introduzir o personagem, faremos isso de maneira semelhante do que foi feito com a cena.

nomePersonagem = Elemento(img = "link", tit = "Titulo", style = dict (top = v1, left = v2, height = v3, width = v4))

Para introduzir a Cena, nós apenas precisávamos passar a imagem da Cena. O Elemento precisa de um pouco mais que isso, precisamos dizer qual é o título daquele elemento (opcional) e um tal de style. Style é a maneira do Vitollino descrever a altura, a largura, e a posição do Elemento em Cena. Substituímos cada um daqueles valores por algum que se adeque a nossa história.

No nosso exemplo, como tinhamos o link da Barbir numa variável, podemos dar o nome da variável ao invés do link entre aspas. O nosso exemplo fica assim:

```
1  from _spy.vitollino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
5
6  def Historia():
7      cenaCastelo = Cena(img = "https://img.elo7.com.br/product/original/1AD347
8      barbie = Elemento(img=linkDaBarbie,
9                        tit="Barbie",
10                       style=dict(left=150, top=60, width=60, height=200))
11     cenaCastelo.vai()
12 Historia()
```

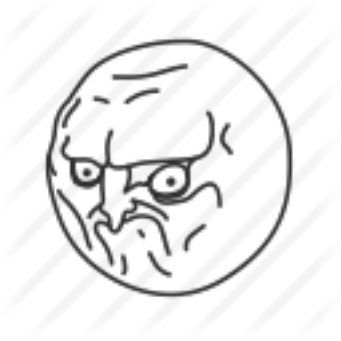
Não se preocupem! Nós podemos pular linha no meio da declaração de uma função() se isso ajudar a leitura.

Vamos testar mais uma vez para ver se está tudo em ordem.

```
1  from _spy.vitollino.main import Cena, Elemento, Texto
2
3  # link do castelo: https://img.elo7.com.br/product/original/1AD3471/painel-1x0
4  linkDaBaribie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-
5
6  def Historia():
7      cenaCastelo = Cena(img = "https://img.elo7.com.br/product/original/1AD347
8      barbie = Elemento(img=linkDaBarbie,
9                        tit="Barbie",
10                       style=dict(left=150, top=60, width=60, height=200))
11     cenaCastelo.vai()
12 Historia()
```

```
Traceback (most recent call last):
  module _core.main line 109
    dialog.action(lambda *_: self.start())
  module _core.supygirls_factory line 127
    self.act(self, lambda *_: self.hide() or extra()) if self.act else None
  module _core.supygirls_factory line 278
    return self._first_response(dialog, lambda: self._executa_acao(), self.extra)
  module _core.supygirls_factory line 256
    traceback.print_exc(file=sys.stderr)
```

ERRO DE NOVO!

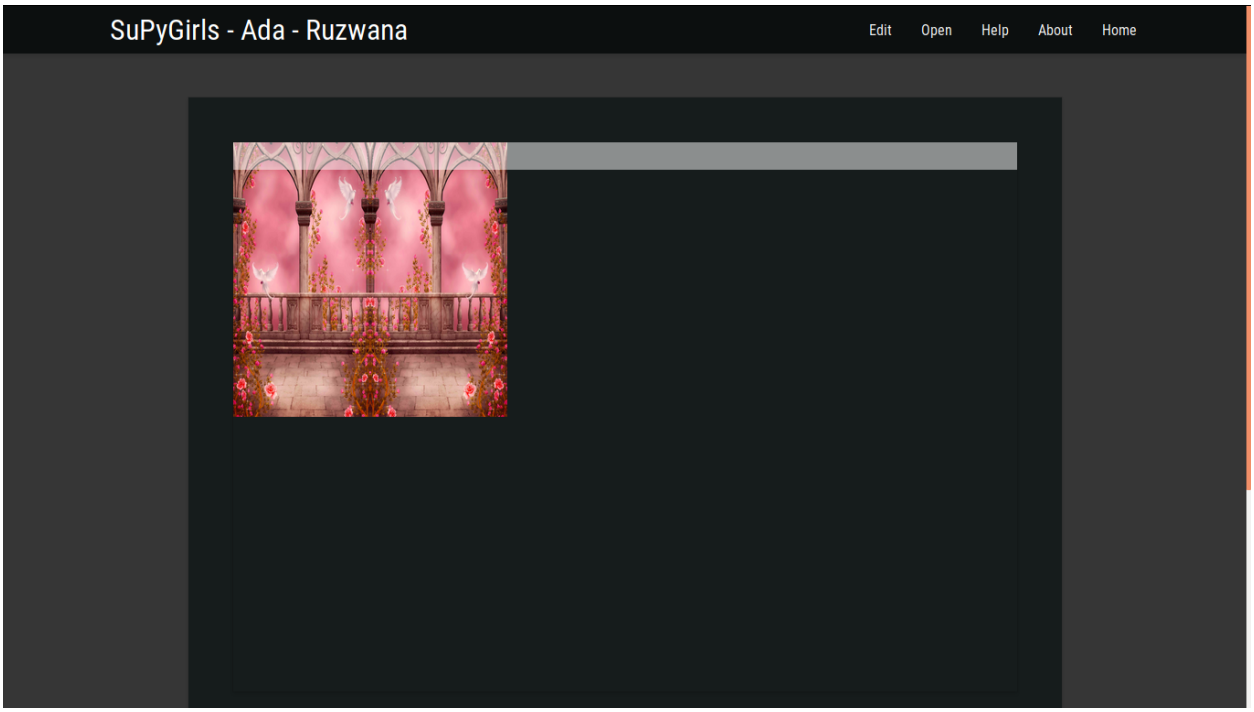


Erro de Novo!! O que será que deu errado dessa vez?!

Ah, já achei... Será que você consegue achar o meu erro sem ler qual é? (Duvido, rs)

Praqueles que não acaharam, eu escrevi linkDaBaribie quando eu guardei o meu link, e na hora de escrever na construção do Elemento, eu escrevi linkDaBarbie certo. Uma letra fora aqui e ali e nada funciona mais. Vamos testar mais uma vez. Também vou aproveitar para trocar o link do Castelo por uma varável, para arrumar melhor o meu código.

```
from _spy.vitollino.main import Cena, Elemento, Texto

linkDoCastelo =" https://img.elo7.com.br/product/original/1AD3471/painel-1x0-7
linkDaBarbie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-b
def Historia():
    cenaCastelo = Cena (img = linkDoCastelo)
    barbie = Elemento(img=linkDaBarbie,
                      tit="Barbie",
                      style=dict(left=150, top=60, width=60, height=200))
    cenaCastelo.vai()

Historia()
```



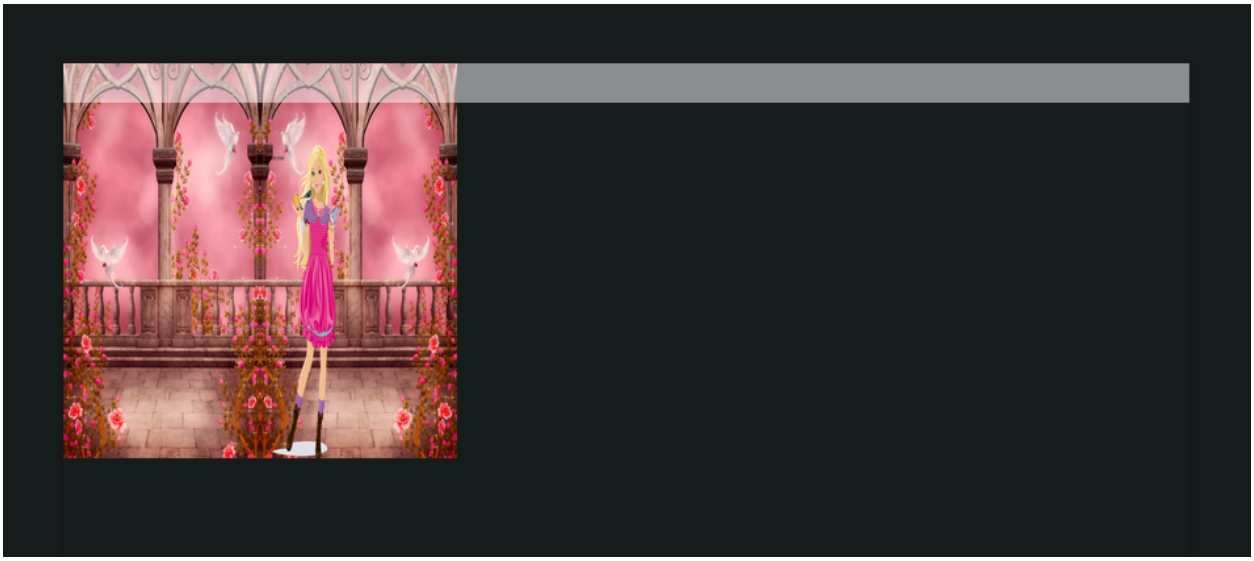
Sem erros!

Não obtivemos nenhuma mensagem de erro dessa vez, mas a Barbie ainda não apareceu. Isso é porque assim como tinhamos que mandar a cena "rodar", temos que mandar o nosso personagem "entrar em cena". Nós fazemos isso através da função .entrar() do Elemento. Diferente da função .vai(), nós precisamos passar a Cena na qual o Elemento vai entrar na função .entrar().

nomePersonagem.entrar(nomeCena)

O nosso exemplo fica assim:

```
1  from _spy.vitollino.main import Cena, Elemento, Texto
2
3  linkDoCastelo =" https://img.elo7.com.br/product/original/1AD3471/painel-1x0-7
4  linkDaBarbie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-b
5  def Historia():
6      cenaCastelo = Cena (img = linkDoCastelo)
7      barbie = Elemento(img=linkDaBarbie,
8                        tit="Barbie",
9                        style=dict(left=150, top=60, width=60, height=200))
10     barbie.entra(cenaCastelo)
11     cenaCastelo.vai()
12
13 Historia()
```



Sucesso!

Agora só falta inserirmos o Texto para nossa primeira página ficar completa! E assim como antes, nós iremos introduzi-lo de maneira semelhante ao Elemento e ao Personagem:

nomeTexto = Texto(nomeCena, “texto”)

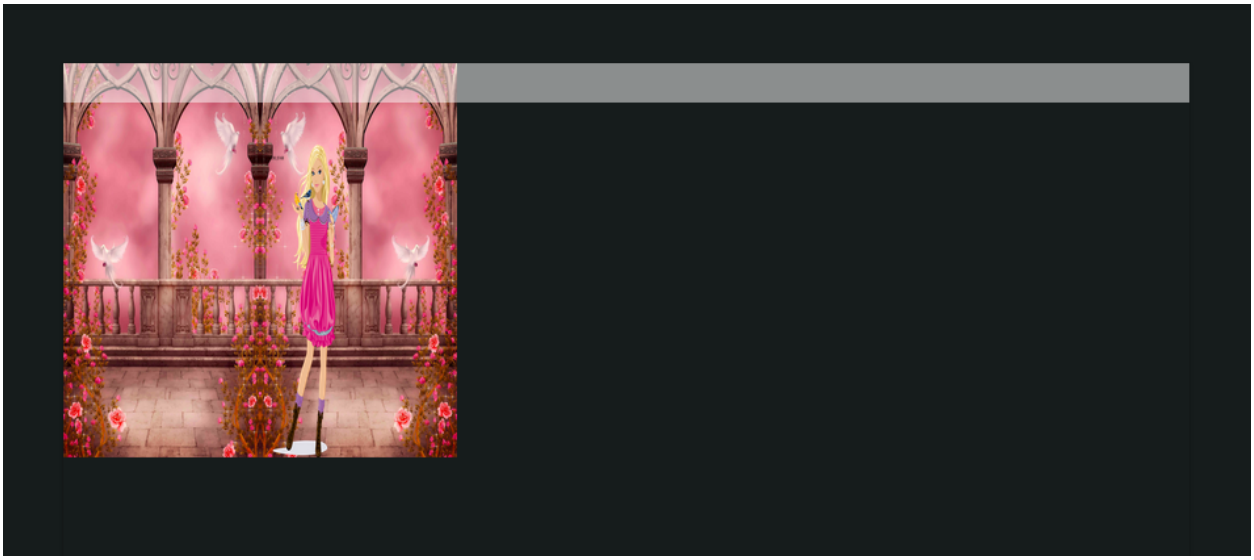
Assim como o Elemento e Cena, não basta apenas contar para o computador que o Texto existe, temos que associa-lo a algum Elemento dentro da Cena que ele se encontra. Fazemos isso associando a função .vai() do Texto com a função .vai() do elemento que se deseja associar. Para se associar função, nós não escrevemos os parênteses:

nomeElemento.vai = nomeTexto.vai

No nosso exemplo, fica assim:

```
1  from _spy.vitollino.main import Cena, Elemento, Texto
2
3  linkDoCastelo =" https://img.elo7.com.br/product/original/1AD3471/painel-1x0-7
4  linkDaBarbie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-b
5  def Historia():
6      cenaCastelo = Cena (img = linkDoCastelo)
7      barbie = Elemento(img=linkDaBarbie,
8                        tit="Barbie",
9                        style=dict(left=150, top=60, width=60, height=200))
10     barbie.entra(cenaCastelo)
11     txtBarbie = Texto(cenaCastelo,
12                       "Vim para o castelo para me abrigar de um ataque de drag
13     txtBarbie.vai = barbie.vai
14     cenaCastelo.vai()
15
16 Historia()
```

E agora o último teste da nossa primeira página!

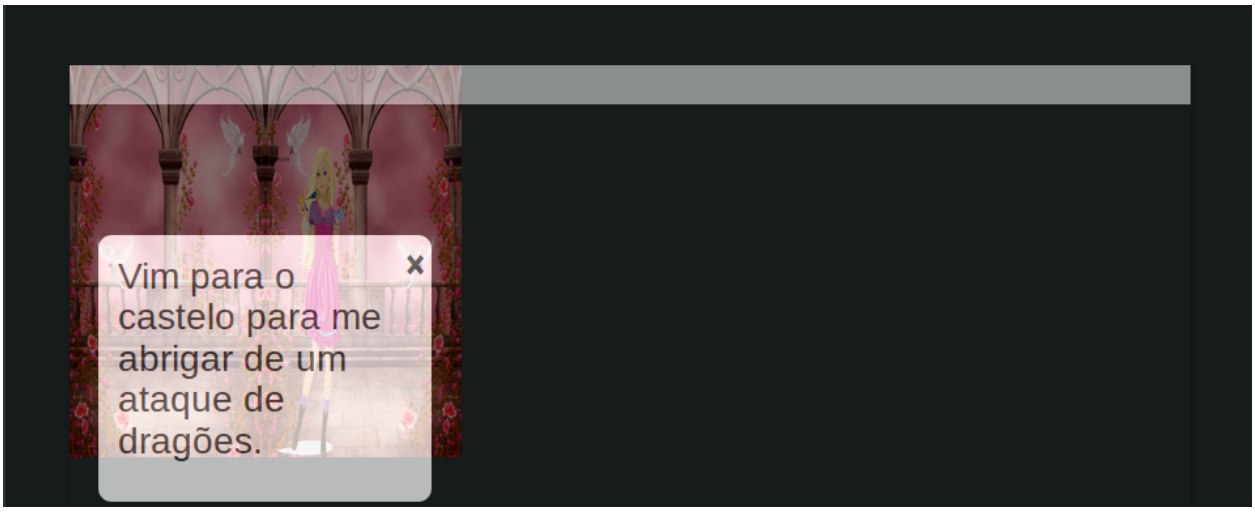


Sem texto, mais um erro!

O erro que cometi agora foi a ordem da operação txtBarbie.vai = barbie.vai. Escrito dessa maneira, eu estou dizendo ao computador para fazer ação do texto da barbie ser igual a ação da barbie, que é nula. O que eu quero na verdade é justamente o oposto, que a barbie copie a ação do texto, logo deve ser, barbie.vai = txtBarbie.vai.

```
1  from _spy.vitollino.main import Cena, Elemento, Texto
2
3  linkDoCastelo =" https://img.elo7.com.br/product/original/1AD3471/painel-1x0-7
4  linkDaBarbie = "http://www.scrapsdinamicos.com.br/imagens/imagens-imagens-da-b
5  def Historia():
6      cenaCastelo = Cena (img = linkDoCastelo)
7      barbie = Elemento(img=linkDaBarbie,
8                        tit="Barbie",
9                        style=dict(left=150, top=60, width=60, height=200))
10     barbie.entra(cenaCastelo)
11     txtBarbie = Texto(cenaCastelo,
12                       "Vim para o castelo para me abrigar de um ataque de drag
13     barbie.vai = txtBarbie.vai
14     cenaCastelo.vai()
15
16 Historia()
```

E agora sim, o último teste da nossa primeira página!



AEAEAEAEAEAEAEAEAEAEAE

CHAPTER 5

Bem Vindos ao Tutorial Kwarwp

Bora programar!

Bem vindos ao SuPyGirls, hoje nós vamos começar a nos aventurar pelo incrível mundo do desenvolvimento de Jogos.

A plataforma SuperPython é um ambiente de desenvolvimento online de Jogos em python que você pode acessar de qualquer lugar através da Internet. Nela você vai encontrar algumas ferramentas pensadas para te ajudar a aprender a programar enquanto constrói jogos. Esse tutorial vai focar no uso da ferramenta Vittolino para a contrução de histórias em quadrinhos.

Primeiro devemos acessar o endereço da plataforma SupyGirls do SuperPython, que é esse aqui ó:

http://supygirls.pythonanywhere.com/supygirls

O que deve te levar direto pra cá:

Cada quadrado nomeado nesse portal é uma sala para uma turma de desenvolvimento de jogos. Você deve entrar na sala da sua turma que seu professor ou professora indicar (se ele tiver enrolando, pede pra ele dizer logo qual é, nós queremos programar!). Mas o que será que todos esses nomes tem em comum?

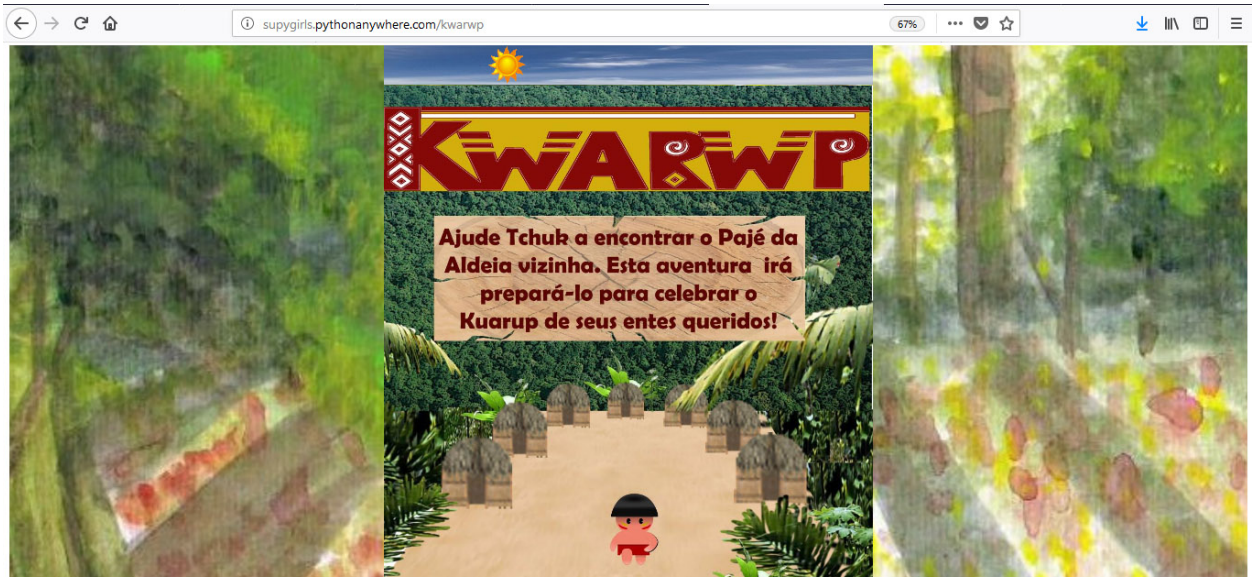
Na sala da sua turma, existem mais quadrados nomeados. Cada quadrado nomeado é um editor de jogos, ou seja, se você clicar em qualquer um deles você vai ser levado para um ambiente onde já vai poder começar a contruir todas as doideiras que sem passam dentro da sua cabeça perturbada!

Mais nomes... suspeito...

Enfim, de volta ao que importa, JOGOS!

5.1 Kwarwp - Apresentação

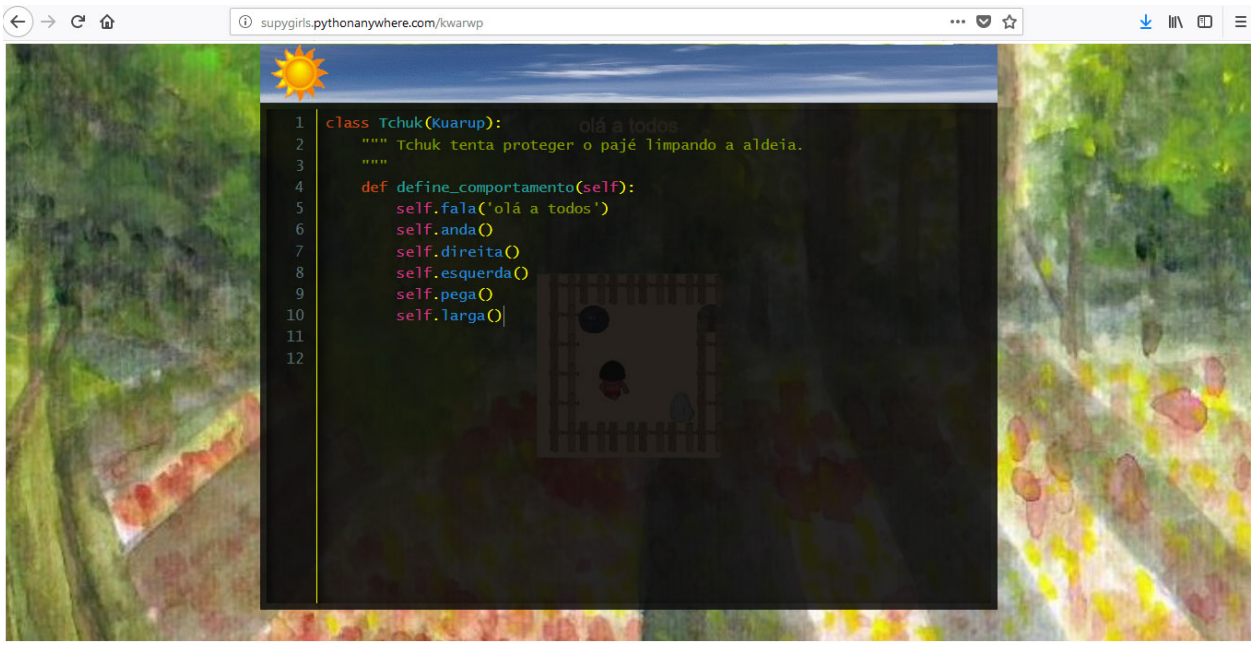
Kwarwp é um Game que nos ensina a linguagem de programação Python enquanto ajudamos o pequeno Índio Tchuk a vencer vários desafios que o levarão ao seu objetivo.



Tchuk é um sobrevivente de sua Aldeia que foi incendiada. Segundo a tradição, ele precisa celebrar o Kuarup de seus entes queridos, que sigifica a última homenagem aos que se foram.

Para alcançar seu objetivo, Tchuk sai em busca do Pajé da Aldeia vizinha, pois ele é o único que pode prepará-lo para esta missão.

De cenário em cenário, Tchuk procura pelo Pajé através da sua ajuda: Cada frase organizada na programação em Python pode guiá-lo na direção certa!



Mas cuidado! Se Tchuk, apenas, passar por cada cenário, sem se preocupar em eliminar os perigos existentes, o Pajé pode se ferir e tudo estará perdido!

5.2 Kwarwp - Criando Mapas

Caso você queira contribuir com desafios, você pode criar novos desafios

Kwarwp/_static/captela3.png

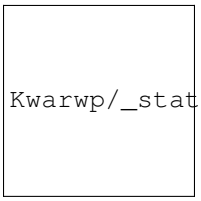
Agora chegamos ao editor do código Python. Esse código é como você vai descrever tudo o que o computador precisa fazer para ele construir o jogo que você imaginar. No momento ele só tem uma linha. Essa linha começa com o caracter '#', o que significa que ela não será lida pelo computador, ela é um cometário feito para seres humanos lerem. Eles servem para facilitar a leitura do código pelo programador.

Especificamente, essa linha é apenas uma marcação do título do documento de texto que nós vamos editar. Mas se você ficou confuso, não se preocupe, porque nós vamos apagá-la!

Kwarwp/_static/captela4.png

Agora na primeira linha nós vamos escrever o seguinte código:

from _spy.vittolino.main import Cena, Elemento, Texto



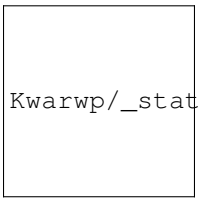
Que doido, o editor coloriu automaticamente algumas palavras para nós, o que isso deve significar: from _spy.vittolino.main import Cena, Elemento, Texto

5.3 Vittolino - Explorando desafios

Pronto, finalmente chegamos ao editor. Essa é a tela de início. Nela você pode observar no canto superior esquerdo o nome do projeto, o nome da Sala da sua turma e o nome do editor que você escolheu. Para continuarmos, precisaremos escolher a opção Edit.

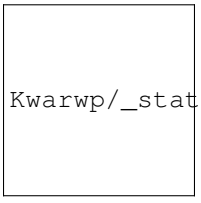
Agora chegamos ao editor do código Python. Esse código é como você vai descrever tudo o que o computador precisa fazer para ele construir o jogo que você imaginar. No momento ele só tem uma linha. Essa linha começa com o caracter '#', o que significa que ela não será lida pelo computador, ela é um cometário feito para seres humanos lerem. Eles servem para facilitar a leitura do código pelo programador.

Especificamente, essa linha é apenas uma marcação do título do documento de texto que nós vamos editar. Mas se você ficou confuso, não se preocupe, porque nós vamos apagá-la!



Agora na primeira linha nós vamos escrever o seguinte código:

from _spy.vittolino.main import Cena, Elemento, Texto



Que doido, o editor coloriu automaticamente algumas palavras para nós, o que isso deve significar: from _spy.vittolino.main import Cena, Elemento, Texto

5.4 Kwarwp - Tutorial Funcional

Chegamos ao tutorial funcional. estou fazendo algumas alterações, mas todos podem alterar. Nela você pode observar no canto superior esquerdo o nome do projeto, o nome da Sala da sua turma e o nome do editor que você escolheu. Para continuarmos, precisaremos escolher a opção Edit.

Agora chegamos ao editor do código Python. Esse código é como você vai descrever tudo o que o computador precisa fazer para ele construir o jogo que você imaginar. No momento ele só tem uma linha. Essa linha começa com o caracter '#', o que significa que ela não será lida pelo computador, ela é um cometário feito para seres humanos lerem. Eles servem para facilitar a leitura do código pelo programador.

Especificamente, essa linha é apenas uma marcação do título do documento de texto que nós vamos editar. Mas se você ficou confuso, não se preocupe, porque nós vamos apagá-la!

Agora na primeira linha nós vamos escrever o seguinte código:

from _spy.vittolino.main import Cena, Elemento, Texto

Que doido, o editor coloriu automaticamente algumas palavras para nós, o que isso deve significar: from _spy.vittolino.main import Cena, Elemento, Texto

CHAPTER 6

Tutorial da Plataforma SuperPython

Este é um breve tutorial de imersão na plataforma SuperPython.

Code Author Carlo E. T. Oliveira

Affiliation Universidade Federal do Rio de Janeiro

Homepage Programa Superpython

6.1 CONTEÚDO

6.1.1 Introdução ao Python

Note: Este arquivo compreende uma fugaz introdução ao python.

Este documento pretende tornar familiar, à primeira vista, contextualizações importantes e sintaxes frequentes no uso linguagem.

Espero que tão logo este documento seja parco frente a sua fome pythônica e você precise enfrentar o árduo plano das documentações :D

ÍNDICE

O que é Python?

Python é uma linguagem de programação! Tcharan!!!!!!

Uma linguagem de programação de alto nível arquitetada para ser, simultâneamente, compreensível e poderosa!

Criado por Guido van Rossum e lançado em 1991, a estura de controle (sintaxe) requerida pelo interpretador e a abordagem orientada a objeto auxiliam o programador a desenvolver códigos organizados e claros, sejam eles extensos ou breves.

Você pode ler mais sobre no Python.org

PARADIGMAS DE PROGRAMAÇÃO

O Paradigmas de programação são um conglomerado de classificações atribuidas às estruturas de código (sintaxe) que o programador utiliza. Para ser mais claro, existem diversas linguagens de programação e também diversas formas de externalizar suas soluções através delas; estas soluções resultam em uma estrutura que pode ser classificada como um determinado paradigma.

Ahhhhhhh! Mas pra quê isso?

Bem, *isso* é resultado de um longo período de aprimoramento das linguagens de programação! Inicialmente a programação era difícil, requeria um graaande conhecimento, sobre a linguagem e computadores, e atenção do programador pois a escrita era de **baixo nível**, ou seja, eram compilados de instruções diretas para o computador (Brinca um pouquinho aqui: Código Binário). Tudo muito complexo, específico, engessado.

Note: Exemplo de linguagem baixo nível: Código Binário, Assembly

---

Com a caminhar da tecnologia as demandas passaram a ser outras! Muito trabalho para pouco programador e muita criatividade para linguagens que não conseguiam acompanhar!!

Daí surgem as linguagens de **alto nível**! As de terceira geração seguiam o paradigma procedural, e descreviam especificamente quais procedimentos utilizar para resolver o problema em específico. E mais uma vez tudo dependia do conhecimento profundo do desenvolvedor e a programação ainda não era nada intuitiva.

Note: Exemplo de linguagem alto nível (Terceira Geração): COBOL,FORTRAN...

Observando, o nível da linguagem é dado de acordo com o grau de proximidade entre a estrutura de programação e a estrutura da nossa língua! Nesse grupo estão as linguagens C, C++, JAVA, [...] e nosso amadinho PYTHON!

Voltemos aos paradigmas...

Como dito, existem diversos paradigmas! Mas neste documento focaremos em três: Programação Procedural, Programação Estruturada e a Programação Orientada a Objeto.

Programação Procedural

Bem como o nome diz, se trata de uma programação centrada em procedimentos. Este paradigma de programação apresenta-se comumente em scripts corridos que determinavam, diretamente, as ações a serem tomadas pelo computador.

Exemplo de código seguindo o paradigma procedural na linguagem Assembly:

```
 lea si, string ; Atribui SI ao endereço de string.
 call printf    ; Coloca o endereço atual na pilha e chama o processo printf

 hlt            ; Encerra o computador.
 string db "Ola mundo!", 0

 printf PROC
     mov AL, [SI] ; Atribui à AL o valor no endereço SI.
     cmp AL, 0    ; Compara AL com nulo.
     je pfend     ; Pula se comparação der igual.

     mov AH, 0Eh
     int 10h      ; Executa uma função da BIOS que imprime o caractere em AL.
     inc SI       ; Incrementa em um o valor de SI.
     jmp printf   ; Pula para o início do processo.

     pfend:
     ret          ; Retorna para o endereço na posição atual da pilha.
printf ENDP
```

Em python poderíamos conseguir o mesmo resultado:

```
print("Olá, Mundo!") #Teste aí no seu console! :D
```

Programação Estruturada

Bem como o nome diz, se trata de uma programação centrada na estrutura. Este paradigma de programação apresenta-se comumente em blocos únicos, centrados na sequência, decisão e iteração (loops, condicionais...).

Flui bem em projetos breves. Já em projetos extensos a chance de uma única alteração descarrilhar toodo o programa é relevante!

Exemplo de código seguindo o paradigma estruturado:

```
def soma(*args):
    resultado = 0
    for numero  in args:
        resultado += numero
        print("Soma= ", resultado)

soma(1,2,3)
```

Programação Orientada a Objeto (OO)

See also:

Você pode ver outra explicação sobre OO aqui: *Projetos com Python Orientado a objetos*

Bem como o nome diz, se trata de uma programação centrada nos objetos. O objeto na OO é tudo aquilo que carrega, conjuntamente, propriedades e operações de uma classe.

Este paradigma de programação apresenta-se, comumente, em diversos blocos com comportamentos singulares, técnica denominada encapsulamento, e blocos de funcionamento conjunto.

Diferente da programação procedural, a estrutura de um código orientado a objeto permite a solução de problemas pontuais e a adição ou subtração de novos comportamentos a qualquer momento, sem que a porção funcional do código

sofra. Outro ganho no uso do paradigma OO é a reutilização do código (princípios de HERANÇA e POLIMORFISMO)

Exemplo do código anterior seguindo o paradigma OO:

```
class Boneca():
     def __init__(self, cabelo, cor, roupa, modelo=""):
          self.modelo = modelo
          self.cabelo = cabelo
          self.cor = cor
          self.roupa = roupa

     def fala(self):
          # Codigo para a boneca falar

     def anda(self):
          # Codigo para a boneca andar
```

Tip: Você pode acessar o conteúdo de Programação Orientada Objeto acessando o tópico *POO- Programação Orientado a Objeto*

Python: Sintaxe Básica

Note: Os Tópicos abaixo, e outros mais aprofundados, podem ser encontradas na Documentação Python

Variáveis

```
#Teste esse código no seu console

nome_da_variavel = "valor da variavel"
nome_da_outra_variavel = 5362543
nome_da_outra_outra_variavel = [a,b,c,d,e,f,g,h]

#decalração de múltiplas variáveis
nome_da_variavel, nome_da_variavel_dois = "variavel_um", "variavel_dois"

"""Imprime na tela o valor da variavel"""
print(nome_da_variavel)
```

Warning: É indicado não começar sua variável com:

- número

Warning: **O python tem alguns nomes reservados:**

'False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'

Espaço de nome

"Os namespaces são uma ótima ideia - vamos fazer mais disso!" - The zen of python

Se imagine em uma sala de aula com mais 10 pessoas. 50% delas tem nome com grafia e sobrenomes idênticos e a outra metade são apenas idênticos na aparência. Seu trabalho é diferenciá-los. Qual seria sua estratégia?

O mesmO pode acontecer quando programamos. Dentro do nosso módulo é fácil criarmos um script sem nomes repetidos, porém, bem mais trabalhoso quando estamos usando módulos externos.

Tudo no python (strings, listas, funções...) é um objeto, e todo objeto recebe um id equivalente tanto para o atributo quanto para a atribuição:

```
#teste o código abaixo no seu console
Maria_Maia = 4
print('id(Maria_Maia) =', id(Maria_Maia)) # id 140071085578048

Maria_Maia= Maria_Maia + 1
print('id(Maria_Maia_plus_um) =', id(Maria_Maia)) # id 140071085578080
print('id(5) =', id(5)) # id 140071085578080

Josefa = 4
print('id(Josefa) =', id(Josefa)) # id 140071085578048
print('id(4) =', id(4)) # id 140071085578048
```

Para evitar conflitos o Python tem um sistema, nomeado **namespace**, para **garantir que todos os nomes atribuidos aos objetos (variáveis, funções, classes...) do programa sejam exclusivos**, evitando qualquer conflito. Quando você nomeia algum objeto, este passa a ser mapeado com o nome determinado, podendo, também, nomes diferentes mapearem o mesmo objeto ou nomes iguais mapearem objetos diferentes:

```
#teste o código abaixo no seu console
x = "Qual foi?" # namespace global
def mostra_o_X_ai():
  x = "E aiiiiiiii!" #namespace local
  print(x)

print(x) # Qual foi?
mostra_o_X_ai() # E aiiiiiiii!
```

Olha que situação interessante! Para o Python o que determina qual 'X' deve ser apresentado é o **Escopo**;

Escopo

O escopo do nome é o **local** onde determinada variável é acessível; sendo determinado pelo *bloco de instrução* a qual ele pertence.

```
#teste o código abaixo no seu console
zero = 0 # Bloco de instrução 0; variável global
  um = 1 # Bloco de instrução 1; variável local
   dois = 2 # Bloco de instrução 2; variável local
     .
     .
     .
                    número_indefinido = inf # Bloco de instrução n; variável local
```

O escopo de nome tem a função de classificar quais nomes de variáveis, funções e classes estão acessíveis em cada bloco de instrução. Quanto mais próximo de n está o escopo da variável requerida, mais restrito é o acesso a este

objeto. É importante ressaltar que cada variável é global internamente ao bloco que pertence, e local externamente ao bloco que pertence. Esta definição é O escopo é importante para expressão de hierarquias.

```
#teste o código abaixo no seu console
VAR_GLOBAL="Bóson Treinamentos em Tecnologia"
def escreve_texto():
    VAR_LOCAL="Fábio dos Reis"
    print("Variável global: ", VAR_GLOBAL)
    print("Variável local: ", VAR_LOCAL)
print("Executando a função escreve_texto:")

escreve_texto()

print("Tentando acessar as variáveis diretamente:")
print("Variável global: ", VAR_GLOBAL)
print("Variável local: ", VAR_LOCAL) # Tentativa de chamar uma variável local como se
↪fosse global
```

Fonte exemplo: Bosontreinamentos

Dados: Type e Id

- Boolean

```
#teste o código abaixo no seu console
"""Booleano é um estado em python, composto de dois valores: Verdadeiro ou falso."""
print(10 > 9) # True
print(10 == 9) # False
print(10 < 9) # False
```

- Inteiro

```
#teste o código abaixo no seu console

""" Numeros sem parte decimal recebem o tipo 'inteiro'(int) """
inteiro_um = 12
inteiro_dois = -45
type(inteiro_um)
```

- Flutuante

```
""" Numeros com parte decimal recebem o tipo 'flutuante'(float) """
flutuante_um = 12.4
flutuante_dois = -45.6
type(flutuante_um)
```

- Complexo

```
""" Numeros com parte real e imaginnária recebem o tipo 'complex'"""
complexo_um = 12+3j
complexo_dois = 15-7j
type(complexo_um)
```

- String

```
""" Tudo, TUDO MESMO, que está entres aspas é string no python"""
string_um = "12+3j"
string_dois = "Oi! Espero que esteja tudo bem aí!"
type(string_um)


"""Tudo no python carrega uma identidade, um Id"""
id(insira_uma_variavel_aqui) # substitua por alguma variável qualquer
```

Tip: Quando estiver brincando com strings busque explorar os Metodos:

- join()
- translate()
- maketrans()
- upper()
- lower()
- strip()
- find()
- replace()

Estrutura de Dados

- Lista

```
#teste o código abaixo no seu console

""" Tudo que está entre colchetes [] é lista no python"""
lista_vazia = []
lista_um = [1,2,3,[1,2,3[1,2,3]]] #quantas listas tem aqui dentro?
lista_dois = ["oi",1,4.3,4+9j]
type(lista_um)
```

Tip: Quando estiver brincando com listas busque explorar os Metodos:

- len()
- index()
- append()
- extend()
- insert()
- remove()
- count()
- pop()
- reverse()
- sort()

- copy()
- clear()

- Dicionário

```
""" Tudo que tem uma chave e um valor é um dicionário no python"""
dicionario_um = {"um":"1","dois":2,"cachorro":"buldog"}
dict_vazia = {}
type(dicionario_um["um"])
type(dicionario_um["dois"])

dicionario_um.keys()
decionario_um.values()
```

- Tupla

```
""" Valores entre parêntesis () são uma tupla no python. Elas são imutáveis!"""
tupla_um = (1,2,3,4,5)
tupla_vazia = (,)
type(tupla_um)
```

- Set

```
""" Valores entre chaves {} são um conjunto (set) em python"""
set_um = {1,2,3,4,"5","e ae"}
type(set_um)
```

Manipulação das Estruturas de dados

- Lista
- Tuplas
- String
- Set

Compreensão e Expressão Geradora

Como dito anteriormente, o Python é uma linguagem poderosíssima! E alguns conceitos do python funcionam como atalhos na resolução de problemas computacionais.

Abaixo compilamos três funcionalidade muito poderosas da linguagem:

A compreensão é análoga a notação de conjuntos da matemática. Lembra?

1. {x ^ 2: x é um número natural menor que 10}
2. {x: x é um número inteiro menor que 20, x é ímpar}
3. {x: x é uma letra na palavra 'MATEMÁTICA', x é uma vogal}

Exemplo: Vooo-Insights

O tipo de compreensão dependerá do tipo de dado (Type) que você quererá como output.

- Compreensão de Listas

A compreensão de listas é utilizada onde, comumente, na busca por uma lista como output, usaríamos o loop.

Logo, onde antes nós faríamos:

```
lista = []
for i in range(13):
    lista.append(i**2)

print(lista)
```

Com a compreensão de lista conseguimos atribuir a construção da mesma lista da seguinte forma:

```
nueva_lista = [numero**2 for numero in range(13)]
print(nueva_lista)
```

A Sintaxe da compreensão de lista é:

```
[expressão(variável) for variável in conjunto_input [predicate][, ...]]
```

- Compreensão de dicionários
- Compreensão de Conjuntos

Built-in

Os built-ins são funções integradas ao python prontinhas para uso!

Veja mais em: BUILT-IN PYTHON.ORG

- Zip

A função *zip()* toma como argumento iteráveis (list, dict, string...) e as agrega a uma tupla.

Sintaxe da função:

```
zip(*iteravel)
```

Aplicação da função:

```
#gerando as variáveis
lista_quantidade = [1, 2, 3]
lista_alimentos = ['banana', 'laranja', 'maca']
lista_qualidade =  ['estragado','maduro','verde']

#nenhum iterável foi passado
empty_zip = zip()
print(empty_zip)
# Converting iterator to list
resultado_empty_list = list(empty_zip)
print(resultado_empty_list)

# Iteraveis passados
lista_um = zip(lista_alimentos, lista_quantidade)
lista_dois = zip(lista_alimentos, lista_quantidade,lista_qualidade)


# Convertendo em conjunto de tuplas
primeiro_zip= set(lista_um)
segundo_zip = set(lista_dois)
```

(continues on next page)

(continued from previous page)

```
print(primeiro_zip)
print(segundo_zip)
```

Tip: Os iteráveis passados podem não corresponder em quantidade! Teste no seu console:

LISTA_GRANDE=[‘UM’,’DOIS’,’TRES’,’QUATRO’]

LISTA_PEQUENA = [1,2,3]

- Slice

A função *slice()* pode ser usado para fatiar obejtos sequenciais (strings, bytes, listas, tuplas, conjunto)...

Sintaxe da função:

```
slice(inicio, parar, pulo)
```

Aplicação da função:

```
result1 = slice(1)
print(result1) # default

result2 = slice(1, 5, 2)
print(slice(1, 5, 2))

LISTA_GRANDE=['UM','DOIS','TRES','QUATRO','CINCO','SEIS']

fatia_lista = slice(1) # corte no index 1
#fatia_lista = slice(0,4) # corte do index 0 ao 4
#fatia_lista = slice(0,-1,2) # corte do index 0 ao ultimo index pulando 2
print(LISTA_GRANDE[fatia_lista])
```

Operadores

Os operadores python servem para designar **relações** entre as variáveis desejadas.

Veja alguns exemplos abaixo:

- Operadores aritméticos

OPERADORES ARITIMÉTICOS			
OPERADOR	*TIPO*	*VALOR | EXEMPLO*	
+	Adição	Realiza a soma entre dois valores.	10+7+4
-	Subtração	Realiza a subtração entre dois valores.	-10-7-4
*	Multiplicação	Realiza a multiplicação entre dois valores.	3*4
/	Divisão	Realiza a divisão entre dois valores.	10/5
//	Divisão	Retorna a parte inteira da divisão	10//5
%	resto	Retorna o resto da divisão entre dois valores.	4%2
**	Exponenciação	Multiplicação de um número por ele mesmo n vezes	4**2

```
# Teste esse código no seu console!
n = 2
```

(continues on next page)

(continued from previous page)

```
z = 4

a = n+z
b = n-z
c = n*z
d = n/z
e = n%z
f = n**z

print(a)
```

- Operadores de atribuição

Os Operadores de Atribuição Compostos realizam uma operação e em seguida, atribuem o resultado da operação para a variável que está a esquerda do operador de atribuição.

OPERADORES DE ATRIBUIÇÃO		
OPERADOR	*TIPO*	*VALOR*
=	igualdade	Atribui à variável da esquerda o valor à direita
+=	Adição	Realiza a soma entre dois valores.
-=	Subtração	Realiza a subtração entre dois valores.
*=	Multiplicação	Realiza a multiplicação entre dois valores.
/=	Divisão	Realiza a divisão entre dois valores.
%=	Módulo	Retorna o resto da divisão entre dois valores.
**	Exponenciação	Multiplicação de um número por ele mesmo n vezes
&=		Equivale a a = a & 8

```
# Teste esse código no seu console!
n = 2
z = 4

n += z # resultado igual a 6
n -= z # resultado igual a -2
n *= z # resultado igual a 8
n /= z
n %= z
n **= z

print(a)
```

- Operadores lógicos

Os operadores lógicos unem expressões lógicas retornando um valor lógico binário compreendido entre não atendimento da lógica (Falso) ou atendimento da lógica (Verdadeiro). Este tipo de dado (sim e não, zero e um, verdadeiro e falso) é chamado Booleano e, no python, as constantes True e False são reconhecidas como pertencentes ao tipo de dado bool:

```
#Teste no seu console
type(True) # <class 'bool'>
type(False) # <class 'bool'>
type(1 == 1) # <class 'bool'>
```

OPERADORES LÓGICOS		
OPERADOR	*VALOR*	*RESULTADO*
and	True se as duas expressões forem verdadeiras	Se a primeira expressão é verdadeira, o resultado será a segunda expressão.
or	False se, e somente se, duas expressões forem falsas	Se a primeira expressão é falsa, o resultado seré a segunda expressão.
not	Muda o valor do argumento: not True é False, not False é True	Booleano
in	True se receber um o item a ser verificado	Booleano

Combinações And:

AND	False	True
False	False	False
True	False	True

Combinações Or:

OR	False	True
False	False	True
True	True	True

```
#Teste no seu console
print("0 and 1:", bool(0 and 1))
print(0 and 1)
print("\n")

print("1 and 0:", bool(1 and 0))
print(1 and 0)
print("\n")

print("0 and 2:",bool(0 and 2))
print(0 and 2)
print("\n")

print("2 and 0:",bool(2 and 0))
print(2 and 0)
print("\n")

print("1 and 2:",bool(1 and 2))
print(1 and 2)
print("\n")

print("3 and 2:",bool(3 and 2))
print(2 and 3)
print("\n")

print("0 or 1:", bool(0 or 1))
print(0 or 1)
print("\n")

print("0 or 0:", bool(0 or 0))
```

(continues on next page)

(continued from previous page)

```
print(0 or 0)
print("\n")

print("\n")
print("not 0:", bool(not 0))
print(not 0)

print("\n")
print("not 1:", bool(not 1))
print(not 1)
print("\n")

print(2 in (2, 3)) # Saída True
print(2 is 3) # Saída False
```

Note: #SyntaxWarning: "is" with a literal add ao python 3.8 O compilador agora produz um SyntaxWarning quando as verificações de identidade (is e is not) são usadas com certos tipos de literais (por exemplo, strings, números). Muitas vezes, eles podem funcionar por acidente no CPython, mas não são garantidos pela especificação da linguagem. O aviso aconselha os usuários a usarem testes de igualdade (== e! =). (Contribuição de Serhiy Storchaka em bpo-34850.)

```
#Teste no seu console
print('1. Idoso')
print('2. Gestante')
print('3. Cadeirante')
print('4. Nenhum destes')
resposta=int( input('Você é: ') )

if (resposta==1) or (resposta==2) or (resposta==3) :
    print('Você tem direito a fila prioritária')
else:
    print('Você não tem direito a nada. Vá pra fila e fique quieto')
```

Exemplo resgatado em Python Progressivo

```
#Teste no seu console
mes= input('Qual o mês?')
dia_um= int(input('Que dia é hoje?'))
dia_dois= int(input('Que dia é amanhã?'))

if dia_um and dia_dois < 30 :
  print("Ainda estamos em", mes)
else:
  print("Estamos próximos do próximo mês!")
```

```
int_x = int(input("Manda um inteiro aí!"))

int_y = int(input("Manda outro aí!"))


if (int_x == 10) or (int_y < 20):

    print("Uma das duas expressões é verdadeira!")

else:
```

(continues on next page)

(continued from previous page)

```
    print("Ambas são falsas!")
```

- Operadores de identidade

OPERADORES DE IDENTIDADE	
OPERADOR	*VALOR*
is	Retorna verdadeiro quando as variáveis são idênticas (referem-se ao mesmo objeto)
is not	Retorna verdadeiro quando as variáveis nãp são idênticas (variáveis que não se referem ao mesmo objeto)

```
#Teste esse código no seu console
a = 3
b = 3
print(a is b) #True
print(a is not b) #False
```

- Operadores de comparação

OPERADORES COMPARATIVOS		
OPERADOR	*VALOR*	*RESULTADO*
>	True se o valor à esquerda é maior que o valor a direita	Se a primeira expressão é verdadeira, o resultado será a segunda expressão.
<	True se o valor à esquerda é menor que o valor a direita	Se a primeira expressão é falsa, o resultado seré a segunda expressão.
==	True se o valores à esquerda e a direita são equivalentes	Booleano
!=	True se o valor à esquerda é diferente ao da direita	Booleano
>=	True se o valor à esquerda é maior ou igual ao da direita	Booleano
<=	True se o valor à esquerda é menor ou igual ao da direita	Booleano

```
#Teste esse código no seu console
a = 3
b = 3
print(a>b) #True
print(a==b) #False
```

- Operadores de Membro

OPERADORES DE MEMBROS	
OPERADOR	*VALOR*
in	True se o valor está contido do conjunto investigado
not in	True se o valor não está contido no conjunto investigado

```
a = 10
b = 2
```

(continues on next page)

(continued from previous page)

```
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
  print ("a - esta na lista")
else:
  print ("a - não está na lista")

if ( b not in list ):
  print ("b - não está na lista")
else:
  print ("b - está na lista")
```

Iterações

Iterar é repetir algo.

- **CONDICIONAIS**

São estruturas que executam a **verificação** de estados com base nos argumentos passados.

As verificações são feitas pelos operadores condicionais que comparam os valores passados e retornam Verdadeiro ou Falso.

SE condição *ENTÃO* comando

Veja alguns abaixo:

OPERADORES CONDICIONAIS		
OPERADOR	*TIPO*	*VALOR*
==	Igualdade	Verifica a igualdade entre dois valores.
!=	Igualdade	Verifica a diferença entre dois valores.
>	Comparação	Verificar se o valor A é maior que o valor B.
<	Comparação	Verifica se o valor A é menor que o valor B.
>=	Comparação	Verifica se o valor A é maior ou igual ao valor B.
<=	Comparação	Verifica se o valor A é menor ou igual ao valor B.
In	Seqüência	Verifica se o valor A está contido em um conjunto.

A sintaxe de uma **condicional simples** é:

```
if operacao > valor_comparativo:
  print("operacao é maior que valor_comparativo") # Observe a identação!!
```

A sintaxe de uma **condicional composta** é:

```
if operacao > valor_comparativo:
  print("operacao é maior que valor_comparativo")
else:
  print("operacao não é maior que valor_comparativo")
```

A sintaxe de uma **condicional aninhada** é:

```
if operacao > valor_comparativo:
  print("operacao é maior que valor_comparativo")
elif operacao = valor_comparativo:
  print("operacao é igual que valor_comparativo")
else:
  print("operacao não é maior que valor_comparativo")
```

- **LOOP FOR**

Os Loops são laços de repetição (iterações) através de sequências (listas, tuplas, dicionários, conjuntos, strings...).

Com os loops é possível executar um conjuntos de instruções para cada item de um iterável.

Exemplos simples abaixo:

```
animais = ["leão", "macaco", "águia"]
for x in animais:
    print(x)
```

```
for x in "paralelepipedo":
   print(x)
```

Declaração de quebra:

```
# Pause o print de x quando x for macaco
caco = ["leão", "macaco", "águia"]
for x in caco:
    if x == "macaco":
       break
    print(x) #leão
```

Declaração de continuação:

```
caco = ["leão", "macaco", "águia"]
for x in caco:
  if x == "macaco":
    continue
  print(x)
```

Listas aninhadas:

```
 lista = [[1,2,3,4,5],[6,7,8,9],[10,11,12],[13,14,15]]

#print da lista
for x in lista:
    print(x)

 #print das listas aninhadas
 for x in lista:
    for y in x:
      print(y)
```

Uso de funções:

```
for x in range(9):
  print(x)
```

- **LOOP WHILE**

```
  i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Funções

Na programação a função é um bloco de código que realiza determinada tarefa que precisam ser executadas diversas vezes ou em momentos específicos.

A estrutura da função requer `nome da função`, `parâmetro` e um `corpo` que representa o comportamento da função.

- `Nome da função`: É um nome arbitrário e será usado para **chamar** a função.
- `parâmetro`: São os valores necessários para que o comportamento seja possível. O parâmetro pode ser uma lista, string, número... **dependerá** do comportamento esperado para a função.
- `corpo`: Corpo é a instrução da função. É as ações que ela deverá tomar sobre os parâmetros parâmetros passados.

```
#estrutura da função
def nome_da_função(parâmetro): # def é um termo reservado do python para dizer que é␣
↪uma função
    corpo
    corpo
    corpo
```

Warning: Observe o *Escopo* do `corpo` da função. A *identação* é interna ao `def`.

Observe o exemplo de função abaixo:

```
# A função 'diga_o_nome' imprime sempre o nome que for digitado
def diga_o_nome(nome): # 'diga_o_nome é o Nome da função; 'nome' é o parâmetro da␣
↪função
    print(nome)         # função python print() é o corpo da função

diga_o_nome("Gabriela") #observe como a função é chamada.
                        # "Gabriela" é o ARGUMENTO da função 'diga_o_nome'
```

Note: Os parâmetros chamam-se `parâmetro` no cabeçalho da função. Quando chamamos a função, como em `diga_o_nome_("Gabriela")`, o valor que fica dentro do parêntesis é chamado `argumento`.

Você pode criar funções que não requerem parâmetros. Estas funções **sempre retornarão o mesmo resultado**.

```
# A função 'diga_o_nome' imprime sempre o nome que for digitado
def diga_o_nome(): # 'diga_o_nome é o Nome da função
    nome = Gabriela     # observe que na ausência de parâmetros alguns valores␣
↪precisam ser declarados
```

(continues on next page)

(continued from previous page)

```
    print(nome)         # função python print() é o corpo da função

diga_o_nome() #observe como a função é chamada
```

Como dito acima, as funções também são usadas quando determinados comportamento só deve ser chamado em horas oportunas. Observe o código abaixo:

Tip: Teste o código abaixo no seu console!

```
# Operação fora da função

# o código:
n1 = int(input('Chuta um número:'))
n2 = int(input('Chuta mais um número'))
soma = n1 + n2
print("O resultado:", soma)

# Mesma operação dentro da função
def soma_FUN():
    n1 = int(input('Digite o Primeiro Número:'))
    n2 = int(input('Digite o Segundo Número:'))
    print("O resultado da função soma_FUN:", n1 + n2)

soma_FUN()
```

Funções Anônimas/Lambda

Uma forma mais *elegante* de programar é a construção de *funções lambda* ou *função anônima*.

A função lambda tem a seguinte sintaxe:

```
lambda argumentos da função: expressão/ação da função
```

Observe o exemplo abaixo:

```
dobro = lambda x: x*2
print(dobro(5))
```

Parâmetros Ordinais e Nomeados

Retomando, parâmetros são **valores** que serão utilizados pelo corpo da função para exercer alguns comportamentos. Quando a função não pede parâmetros, geralmente, as variáveis do corpo exercem tal função.

O parâmetros podem ser **ordinais** ou **nomeados**, ou seja, dependentes da posição ou do nome. Por exemplo: centagem

```
# Uma função que calcula a porcentagem de um valor.
def porcento(valor,porc=100):
    print(valor*(porc/100))


porcento(100) # 100
porcento(100,50) # 50
```

a função `porcento` pede: **parâmetro ordinal** `valor` e o **parâmetro nomeado** `porc` que, por ser nomeado, é o valor padrão/default da função, ou seja, sempre que chamarmos a função o argumento `porc` = `100`

Vejamos um outro exemplo:

```
# Uma função que calcula descontos e porcentagens acumulativas.
def porcento_desconto(valor,descnt,porc=100):
    prctgm = valor*(porc/100)
    print(int(prctgm-(prctgm*(descnt/100)))) # o int() é uma função python que␣
↪retorna apenas os valores sem a casa decimal (inteiros).


porcento_desconto(100) # TypeError: porcento() missing 1 required positional␣
↪argument: 'descnt'
porcento_desconto(100,0) # 100
porcento_desconto(0,100) # 0
porcento_desconto(100,50) # 50
porcento_desconto(100,50,50) # 25
```

Tip: Observe que no caso de **parâmetros ordinais** a ordem do chamado importa no resultado!!!!

Na função `porcento` pede: o **parâmetro ordinal** `valor`, o **parâmetro ordinal** `descnt`, e o **parâmetro nomeado** `porc` que torna `100` o valor padrão/default da função.

Warning: Todo **parâmetro ordinal** precisa ser passado no chamamento da função.

Veja alguns exemplos de funções python: Funções Python

POO- Programação Orientado a Objeto

TUDO NO PYTHON É OBJETO!

Grave esta frase. Retomaremos ela mais tarde.

Herança

Polimorfismo

Tópicos Avançados

Bem-vindo a seção de tópicos avançados!!

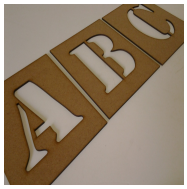
Não se assuste. **Não é um espaço para tópicos difíceis**, são apenas tópicos que requerem um conhecimento sólido sobre os tipos de dados e estruturas sintáticas que ele utiliza.

- **EXPRESSÃO REGULAR (RE)**

Expressões regulares (chamadas REs, ou regexes ou padrões de regex) são essencialmente uma pequena linguagem de programação altamente especializada embutida em Python e feita disponível através do `re` módulo. Através das expressões regulares é possível encontrar facilmente sequências, padrões, dentro de uma string.

Fonte: Python org

Imagine um stencil com padrão *ABC*:



Agora imagine uma string como esta abaixo:

```
texto = """ fbusfGFHHFdhbsfhbsjfjjfgjjbFHFHFHFGHsahbdshdFHFGJFGJFGHFGHFHbsfgjgjsfjjsfD
            GhjshdvuvJfghfgjsgfgjfgjjjADABCHJFJFGFHFHHFHHgshfgjJdfhhHFHFHFGHFGsfghggDG

↪FDGHJHGjhfhgHGFHJGFhgfhgfhgfHFHGHGChgchgchgcCHGHjhvhvhgCHGCHJGChgcjhgcJHGCH
        """
```

Como você faria para descobrir se o padrão do stencil *ABC* está na string *texto*?

Existem diversas formas: iterações, fatiamento de string e lista, built-in zip, etc. E há a `expressão regular`!

Ao utilizar o `re` a pergunta que queremos responder é: "Essa string corresponde a este padrão?" ou "Existe algum lugar nesta string que corresponda a este padrão?". O `re` também pode ser utilizados para manipular strings!

Antes de prosseguirmos deixe-me familiar você a alguns detalhes:

- **METACARACTERES**

Em sua grande maioria as letras e os caracteres correspondem a si mesmos. Por exemplo, a letra *a* corresponderá a qualquer letra *a* presente na string *"Eu fui à feira e não sabia o que comprar, sabia que havia esquecido a lista"* independente de ser o *a* que você procura!

```
string = "Eu fui à feira e não sabia o que comprAr, sabia que havia esquecido a lista"
stencil = "a"

print(stencil in string, string.count(stencil)) #True, 9
# Observe que a letra maiúscula foi ignorada pelo método count
```

Mas há desvios nesta esta regra, os metacaracteres não correpondem a si mesmo mas sim a *marcadores* de exceções.

Os metacaracteres são os marcadores do seu stencil:

METACARACTERES		
METACARACTER	*VALOR*	*EXPRESSÃO*
[]	Corresponde a uma lista de ocorrências dos caracteres desejados	*[ABC]*; *[A-E]'equivale a [ABCDE]; '[^ABC]* complementa excluindo o set
.	Corresponde a uma string de acordo com a quantidade de pontos	. ; ..
^	Corresponde a string que **inicia** com determinado caracter ou sequência	^*A*; ^*bE*
$	Corresponde a string que **termina** com certo conjunto de caractere	*A$*; ^bE$'
*	Verifica zero ou mais correspondências do caracterer à esquerda	*ma*n*
+	Verifica uma ou mais correspondências da ordem à esquerda	*ma+n*
?	Verifica zero ou uma correspondência de ordem à esquerda	*ma?n*
{}	Verifica repetições em uma string	*a{n,m}* onde n e m correspondem, respectivamente, o mínimo e o máximo
()	Verifica subpadrões	*(a|b|c)xz* combina qualquer string que corresponda a abc seguida de xz
\	Esta folga é usada para "escapar" de caracteres e matacaracteres	*$* torna o matacaracter *$* em um caracter comum
|	Verifica alternâncias	*a|b*

Algumas sequências especiais tornam alguns padrões mais fáceis e escrever:

SEQUENCIAS ESPECIAIS	
SEQUÊNCIAS ESPECIAIS	*VALOR*
\A	Verifica se uma string **começa** com determinado conjunto de caracteres
\b	Corresponde aos caracteres que estão no início ou final
\B	É o oposto de b
\d	Corresponde a qualquer dígito decimal. Equivale a [0-9]
\D	Corresponde a qualquer dígito não decimal. Equivale a [^0-9]
\s	Corresponde a uma string que contenha caracter de espaço e branco
\S	Corresponde a string que não correspondente ao espaço em branco
\W	Verifica presença de qualquer caracter não alpha-numérico
\Z	Corresponde a caracteres específicos no final de uma string

Sigamos para alguns casos de uso dos metacaract:

```
import re

string = 'Ola! Eu tenho 26 anos e carrego 2 bolsas'
#pattern = '[aeiou]' # ['a', 'u', 'e', 'o', 'a', 'o', 'e', 'a', 'e', 'o', 'o', 'a']
#pattern = '[^aeiou]'#['O', 'l', '!', ' ', 'E', ' ', 't', 'n', 'h', ' ', '2', '3', '
↪', 'n', 's', ' ', ' ', 'c', 'r', 'r', 'g', ' ', '2', ' ', 'b', 'l', 's', 's']
#pattern = '[a-c]' # ['a', 'a', 'c', 'a', 'b', 'a']
#pattern = '.....' # ['Ola! ', 'Eu te', 'nho 2', '3 ano', 's e c', 'arreg', 'o 2 b',
↪'olsas']

result = re.findall(pattern, string)
```

(continues on next page)

(continued from previous page)

```
print(result)
```

```
import re

string = 'Ola! Eu tenho 26 anos e carrego 2 bolsas'
pattern = 'sasxvdv$' # ['sas']
pattern = '[sas$]' # ['a', 'a', 's', 'a', 's', 'a', 's']
pattern = 'sasuie #' # []

result = re.findall(pattern, string)
print(result)
```

```
import re

string = "The rain in Spain falls maizly in the plain!"
pattern = 'ain*' #['ain', 'ain', 'ai', 'ain']
pattern = 'ai*n' #['ain', 'ain', 'ain']
pattern = 'a*in' #['ain', 'in', 'ain', 'in', 'ain']
pattern = 'ainhehe' #[]

result = re.findall(pattern, string)
print(result)
```

```
import re

string = "The rain in Spain falls maizly in the plain!"
#pattern = 'ain+' #['ain', 'ain', 'ain']
#pattern = 'ai+n' #['ain', 'ain', 'ain']
#pattern = 'a+in' # ['ain', 'ain', 'ain']
#pattern = a+isdsf

result = re.findall(pattern, string)
print(result)
```

```
import re

string = "aaaaaah! The rain in Spaain falls maizly in the plaaaaain!"
pattern = 'a{1,5}' #['aaaaa', 'a', 'a', 'aa', 'a', 'a', 'aaaaa']
pattern = 'a{5}' #['aaaaa', 'aaaaa']

result = re.findall(pattern, string)
print(result)
```

```
import re

string = "The rain in Spain falls maizly izamly zamily in the plain!"
#pattern = '(maiz)ly' #['maiz'] pois existe apenas uma correspondencia dessa ordem␣
↪seguida de `ly`
#pattern = '(m|a|i|z)ly' #['z', 'm', 'i']

result = re.findall(pattern, string)
print(result)
```

```
import re
```

(continues on next page)

(continued from previous page)

```
string = "The rain in Spain falls maizly izamly zamily in the plain!"
pattern = 'a|i' #['a', 'i', 'i', 'a', 'i', 'a', 'a', 'i', 'i', 'a', 'a', 'i', 'i', 'a
↪', 'i']
#pattern = 'e|a|i' #['e', 'a', 'i', 'i', 'a', 'i', 'a', 'a', 'i', 'i', 'a', 'a', 'i',
↪'i', 'e', 'a', 'i']


result = re.findall(pattern, string)
print(result)
```

```
import re

string = "The rain in Spain falls maizly 9 izamly zamily in the plain!"
pattern = '\AThe'
pattern = '\Arain'
pattern = '\d' # ['9']
pattern =  '\D' #  ['T', 'h', 'e', ' ', 'r', 'a', 'i', 'n', ' ', 'i', 'n', ' ', 'S',
↪'p', 'a', 'i', 'n', ' ', 'f', 'a', 'l', 'l', 's', ' ', 'm', 'a', 'i', 'z', 'l', 'y',
↪ ' ', ' ', 'i', 'z', 'a', 'm', 'l', 'y', ' ', 'z', 'a', 'm', 'i', 'l', 'y', ' ', 'i
↪', 'n', ' ', 't', 'h', 'e', ' ', 'p', 'l', 'a', 'i', 'n', '!']

result = re.findall(pattern, string)
print(result)
```

```
import re

string = "Therain"
pattern = \s #[]

result = re.findall(pattern, string)
print(result)

string_2 = "The rain"
pattern_2 = '\S' # ['T', 'h', 'e', 'r', 'a', 'i', 'n']

result_2 = re.findall(pattern_2, string_2)
print(result_2)
```

```
import re

string = "The rain * ^%$"
pattern = '\W' # [' ', ' ', '*', ' ', '^', '%', '$']

result = re.findall(pattern, string)
print(result)
```

```
import re

string = "The rain"
pattern = 'rain\Z' # ['rain']

result = re.findall(pattern, string)
print(result)
```

Vamos para os métodos do módulo:

- **re.findall()**

Retorna uma lista de strings contendo todas as correspondências.

```
import re

    string = 'Ola! Eu tenho 26 anos e carrego 2 bolsas'
    #pattern = '[aeiou]' # ['a', 'u', 'e', 'o', 'a', 'o', 'e', 'a', 'e', 'o', 'o', 'a
↪']
    #pattern = '[^aeiou]'#['O', 'l', '!', ' ', 'E', ' ', 't', 'n', 'h', ' ', '2', '3',
↪ ' ', 'n', 's', ' ', ' ', 'c', 'r', 'r', 'g', ' ', '2', ' ', 'b', 'l', 's', 's']
    #pattern = '[a-c]' # ['a', 'a', 'c', 'a', 'b', 'a']
    #pattern = '.....' # ['Ola! ', 'Eu te', 'nho 2', '3 ano', 's e c', 'arreg', 'o 2 b
↪', 'olsas']

    result = re.findall(pattern, string)
    print(result)
```

- **re.split()**

Divide a string onde há correspondência e retorna uma lista de strings onde as divisões ocorreram.

```
import re

string = "The rain 15 pain 20 spain"
pattern = '\d' # ['rain']

result = re.split(pattern, string) # ['The rain ', '', ' pain ', '', ' spain']
#result = re.split(pattern, string, 2) # ['The rain ', '', ' pain 20 spain']
print(result)
```

Tip: No casos onde o padrão não é encontrado, o split retorna um lista contendo a string original

Tip: O método split() suporta o argumento maxsplit que retornará o número máximo de divisões que ocorrerão.

- **re.sub**

```
#sintaxe do método
re.sub(pattern, replace, string)
```

Retorna uma string em que todas as ocorrências correspondentes são subtituídas pelo conteúdo da variável substituta.

```
import re

string = "The rain 15 pain 20 spain"
pattern = '\s+' # ['rain']
replace = ''

result = re.sub(pattern,replace, string) # Therain15pain20spain
#result = re.sub(pattern,replace, string,2) # este quarto argumento permite ao método␣
↪substituir apenas nas "n" primeiras ocorrências
print(result)
```

É possível também passar a contagem como um quarto parâmetros. Seu default é 0, retornando todas as ocorrências.

- **re.subn()**

É semelhante ao método anterior,porém retorna uma tupla de 2 itens: (nova string, número de substituições)

```
import re

string = "The rain 15 pain 20 spain"
pattern = '\s+' # ['rain']
replace = ''

result = re.subn(pattern,replace, string)
print(result) # ('Therain15pain20spain', 5)
```

- **re.search()**

O método search() procura um padrão em uma string. Ao encontrar o primeiro lugar, faz-se uma correspondência com a string. Se bem sucedido retorna um objeto de correspondências, do contrário retorna None.

```
import re

string = "The rain 15 pain 20 spain"
pattern = 'rain' # ['rain']


result = re.search(pattern,string)
print(result) # <re.Match object; span=(4, 8), match='rain'>
```

Expressão Regular no doc python: **'Doc_Python Re'_**

Referências

1. Paradigma da Programação
2. Programação Procedural
3. Programação Orientada a Objeto
4. Linguagens de programação
5. Variável
6. Estrutura de Dado
7. Operadores Python
8. Condicionais Python
9. Loop Python
10. Função
11. Classe

Tutorial Vitollino

6.1.2 Tour da Plataforma SuperPython

Olá! Seja muito bem-vindo ao Tour da plataforma SuperPython.

Contextualizando, a plataforma Superpython é um ambiente de desenvolvimento (IDE), orgulhosamente open source, planejado para possibilitar a programação na linguagem Python. Esta plataforma é um braço direto do Programa

Superpython que se consagra como um projeto interdisciplinar de criação e desenvolvimentos de Games por crianças, jovens e adultos.

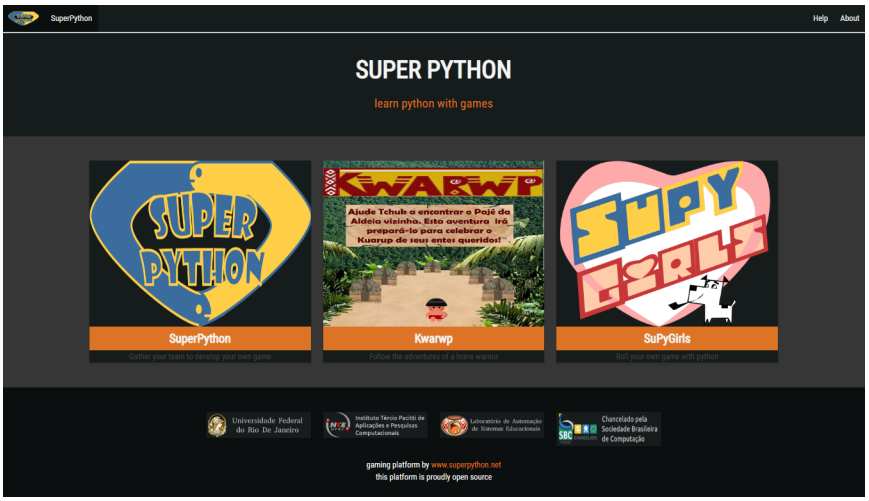
A plataforma se mantém de pé graças ao prof.Carlo que foi o idealizador e segue atualizando a plataforma sempre!! *Ode eterno aos santos!*

O presente tutorial é **unicamente** um guia de acessos da plataforma.

Acesse: Plataforma SuperPython

Página Inicial da plataforma

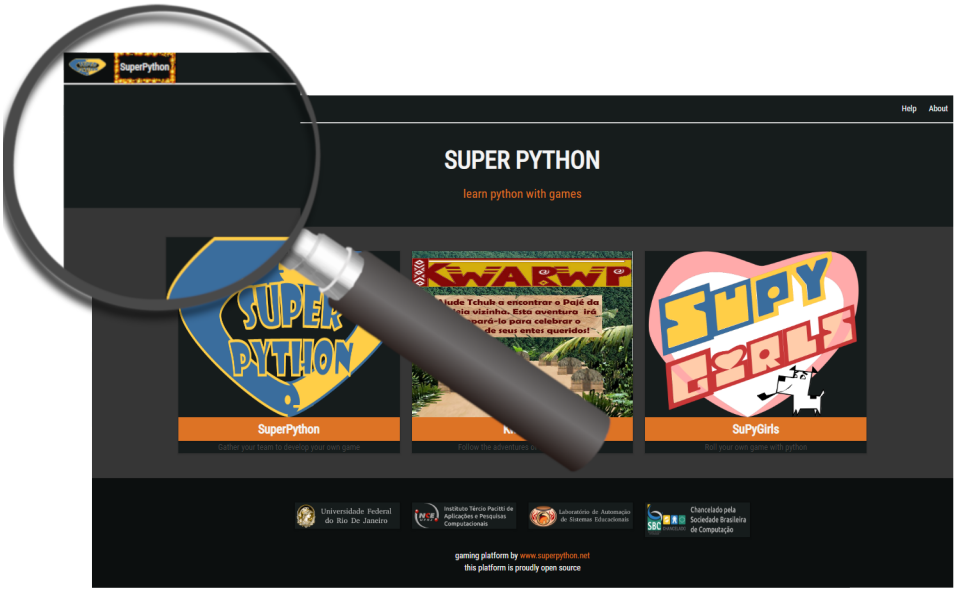
Esta é a apresentação atual da plataforma:



Retornando para a página principal

Sempre que **estiver** na plataforma pode voltar à página principal clicando no botão ressaltado na lupa.

Note: Em alguns momentos ele prega peças. Cuidado!



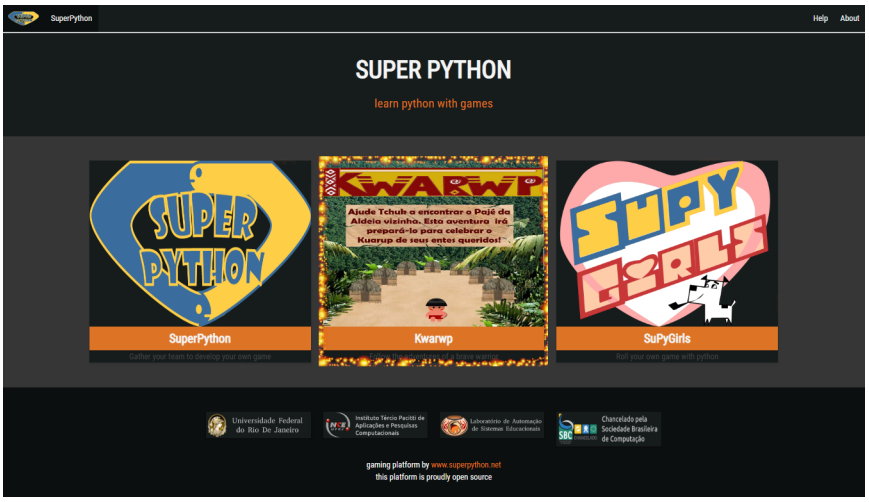
Acessando os Tutoriais da Plataforma

Nossa plataforma tem alguns tutoriais e Documentos! Deixe a curiosidade fluir e divirta-se!

Detalhando a página Inicial

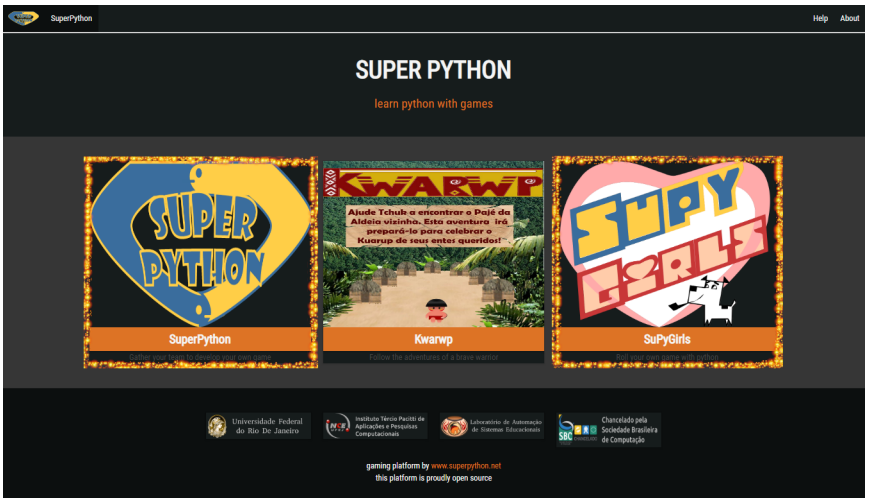
Dando um zoom out na página inicial é possível ver três botões alinhados:

- **O botão central** dá acesso ao game Kwarwp



- **Os botões extremos** dão acesso às salas de programação. :D

O clique nestes botões direcionam para o "salão principal".



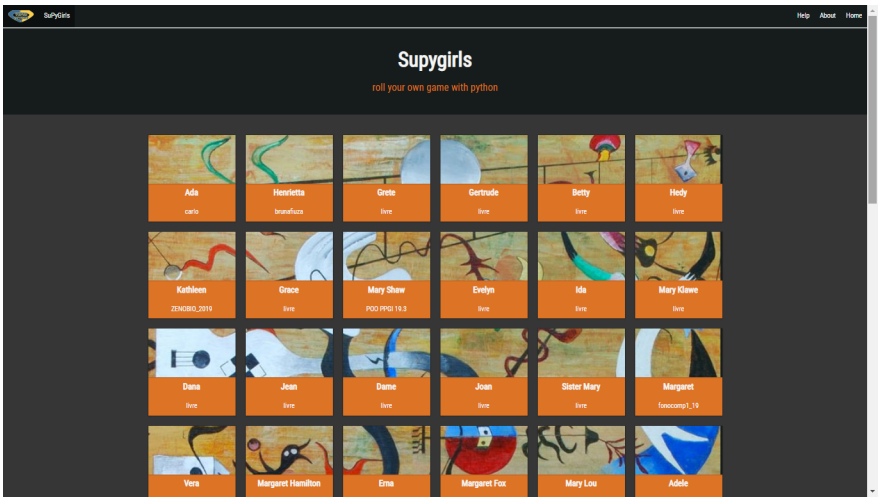
Acessando o Salão Principal de programação

Bem-vindo ao salão principal! Seu clique foi bem sucedido e agora você pode escolher o *pacote* correspondente ao projeto da qual você participa! É chamado pacote pois cada um dos 'quadrados' comporta, no mínimo, 40 sub-salas prontas para você codar!

Clique na sala com nome respectivo ao seu projeto e 'Be Happy!'

Note: Esqueceu o nome da sua sala? Observe que abaixo dos nomes em negrito aparecem nomes de outras pessoas ou nomes característicos de um projeto. Tente procurar pelo nome do seu professor ou projeto!! Se ainda tiver problemas contate o seu tutor ou prof.Carlo. Com certeza te salvarão!

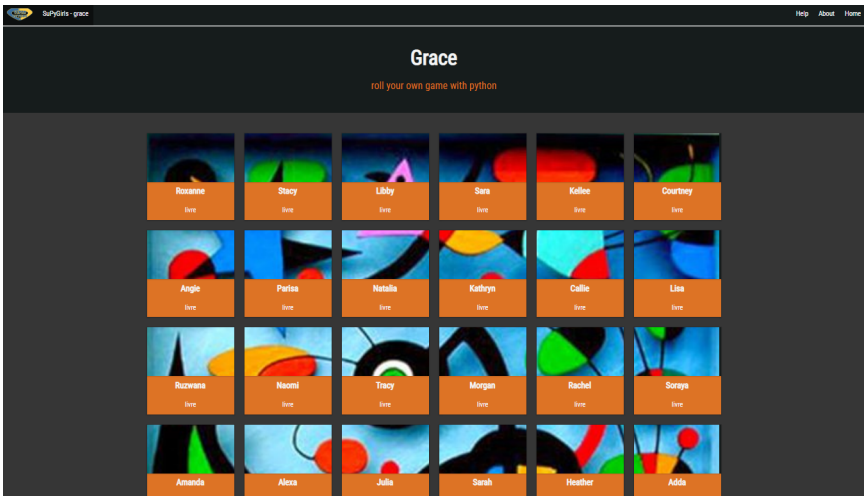
P.S: Você reconhece os grandes nomes que batizam as salas da Plataforma? Fica a dica! ;)



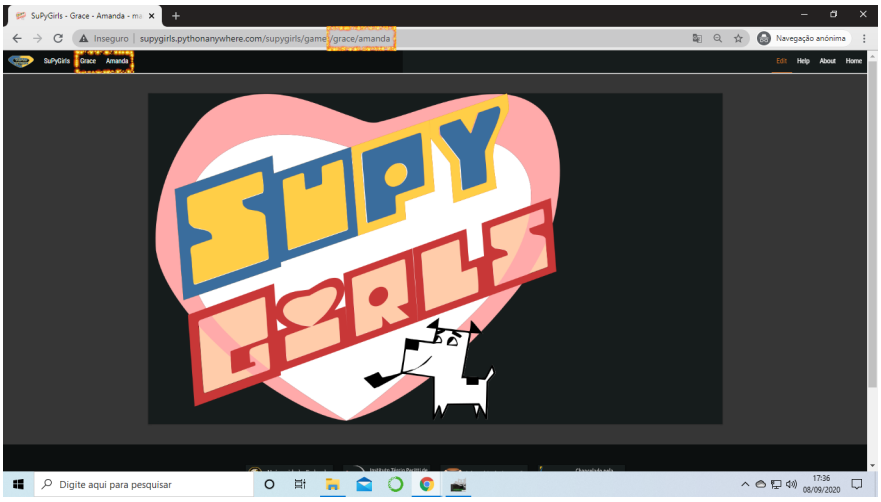
Acessando a sala pessoal de programação

O clique **em qualquer uma** das salas da imagem anterior, redireciona para uma página **idêntica a esta**. Agora você está vendo os módulos do pacote que você selecionou! Yuhaa!

Note: Cuidado para não acessar as salas pessoais de outro projeto!



E ao clicar em qualquer uma das salas você será redirecionado para sua IDE propriamente dita:

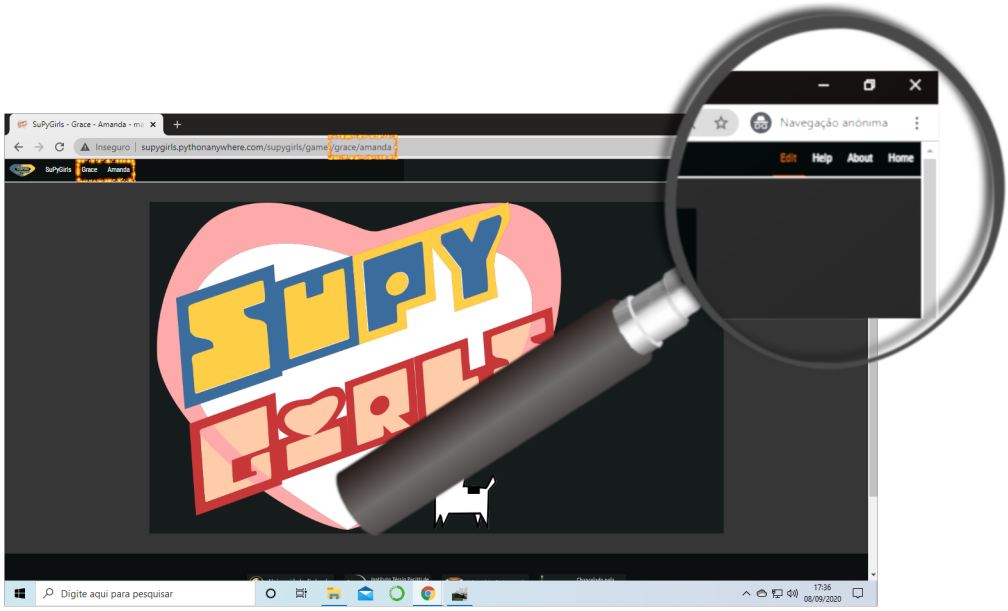


Observe a URL: supygirls.pythonanywhere.com/supygirls/game/**GRACE/AMANDA**

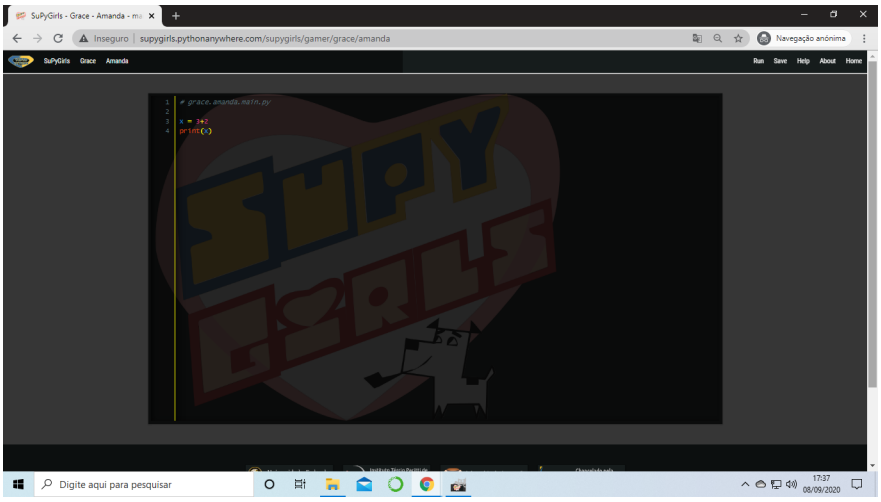
Esta url te indica, respectivamente, qual o pacote e módulo que você está trabalhando no momento.

Abrindo o Interpretador Python

Para acessa o interpretador basta apenas clicar no botão "**edit**".



Que te trará pra cá:

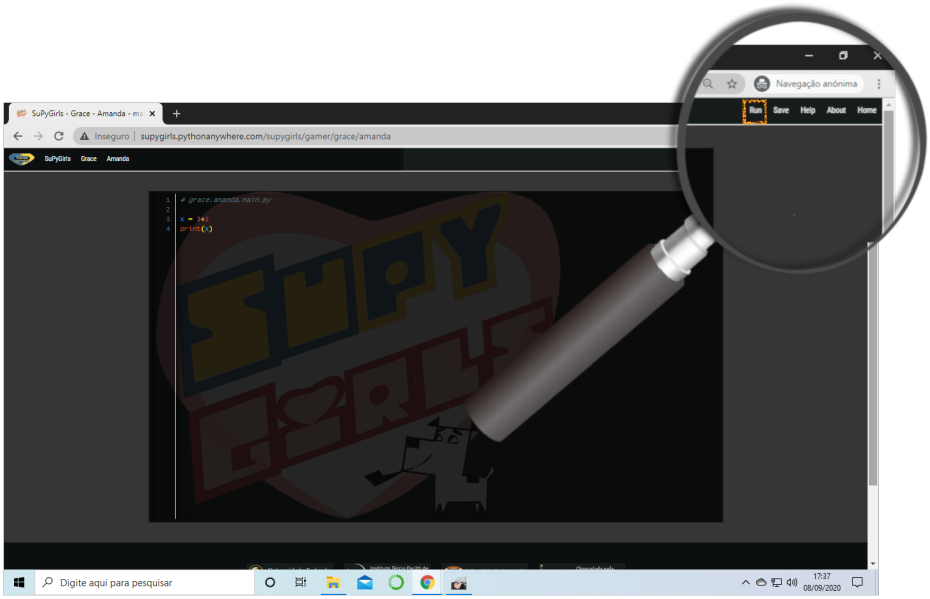


Observe! Já há uma amostra de código.

Este é o espaço onde você pode desenvolver.

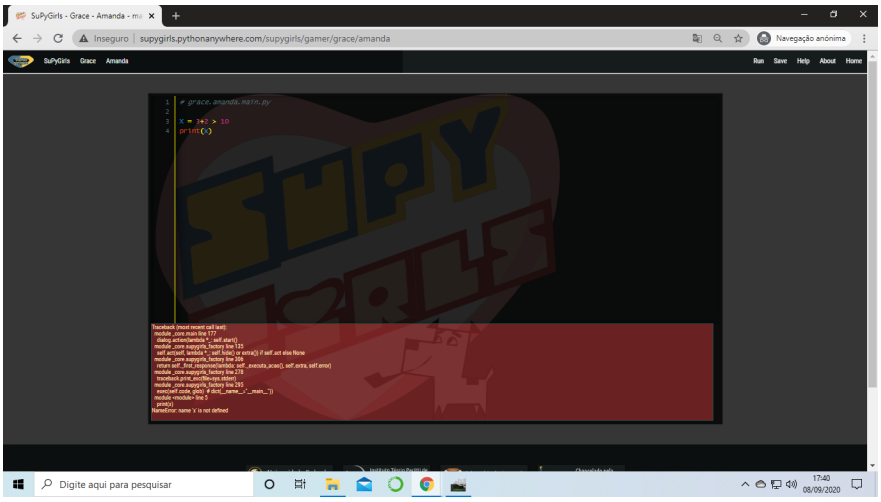
Startando o Interpretador

Para ver sua obra-prima por completo basta clicar em run:

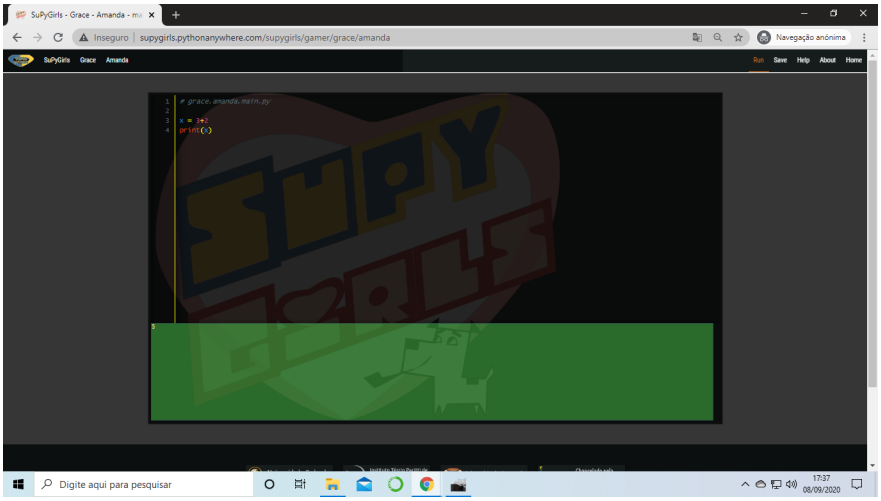


RESULTADOS POSSÍVEIS:

1. O interpretador trará respostas quando houver erro:

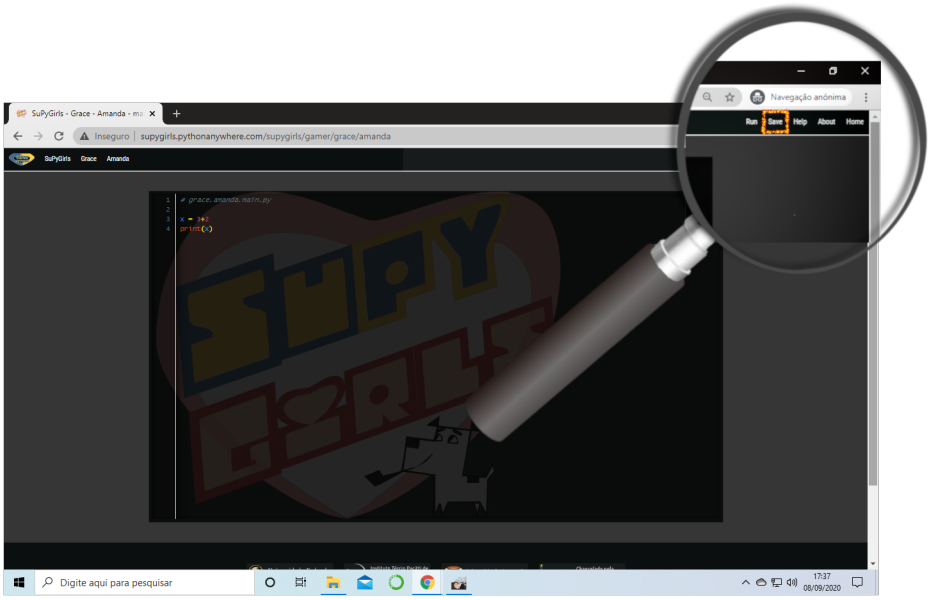


1. O interpretador trará respostas quando houver acerto:



Tip: Caso o interpretador não responda, ou apresente respostas antigas, não se desespere:

F5 NELEEE!!!!

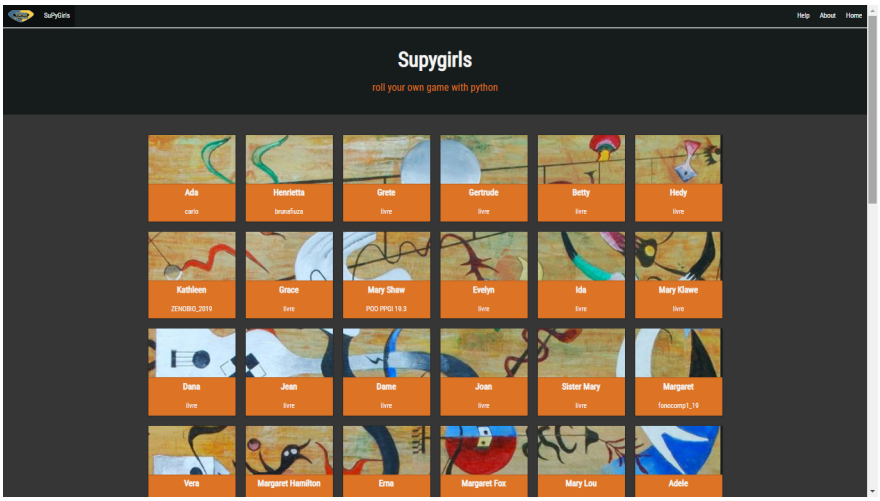


Warning: NEVER. MAI. NONCH NIE. Ποτ. .

NUNCA Saia da sala sem salvar o seu trabalho!

MÃO NA MASSAAAA, DIGO, NO TECLADOOOOOOO!!

6.1.3 Tutorial Vitollino



Warning: É vitoLLino com dois *LL's*

SUMÁRIO

1. *IMPORTANDO O VITOLLINO*

2. *IMPORTANDO MÓDULOS (SALAS)*
3. *STYLE*
4. *CENA*
5. *SALA*
6. *LABIRINTO*
7. *ELEMENTO*
8. *TEXTO (PopUp)*
9. *BOTÃO*
10. *MÚLTIPLA-ESCOLHA*
11. *INVENTÁRIO*
12. *MÚSICA*
13. *CÓDIGO*
14. *PORTAL*
15. *DROPPER*
16. *DROPPABLE*

IMPORTANDO O VITOLLINO

Para utilizar os recursos do vitollino é necessário, primeiramente, importá-lo para o módulo que está trabalhando.

```
"""É análogo ao caminho _spy/vitollino/main.py """
from _spy.vitollino.main  import Classe_Desejada, Classe_Desejada2
```

Outra forma de também importar é:

```
"""A abreviação do nome da classe pode auxiliar na organização e clareza do código␣
↪posteriormente ;)"""
from _spy.vitollino.main import  Classe_Desejada as abreviação_qualquer
```

IMPORTANDO MÓDULOS (SALAS)

```
""" Exemplo from cenas.imix import Inicial"""
 from  nome_do_pacote.nome_do_módulo import Classe_Desejada, funcao_Desejada



 """Essa linha impede cruzamentos indesejados entre os nomes dos repositórios."""
 if __name__ == "__main__":
     classe_principal()
```

See also:

Justificativa extensa da linha if __name__ == “__main__”

STYLE

O é utilizado para regular a altura e a largura da imagem que será mostrada.

```
from _spy.vitollino.main import STYLE


STYLE["width"] = 900 # width = 300 (default)
STYLE["heigth"] = "900px" # min-height = "300px"
```

CENA

A cena é uma tela com possibilidade de clique à esquerda,direita e centro.

```
from _spy.vitollino.main import Cena
"""Importa a classe Cena do Vitollino"""

IMAGEM_QUALQUER = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
IMAGEM_ESQUERDA = "string_correspondente_a_url_e_extensao_da_image" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
IMAGEM_DIREITA = "string_correspondente_a_url_e_extensao_da_image" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
IMAGEM_MEIO = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif

nome_da_cena_direita = Cena(IMAGEM_DIREITA)
nome_da_cena_esquerda = Cena(IMAGEM_ESQUERDA)
nome_da_cena = Cena(IMAGEM_QUALQUER, # Parâmetro obrigatório
                esquerda=nome_da_cena_esquerda, # default = NADA = SalaCenaNula()
                direita=nome_da_cena_direita,  # default = NADA = SalaCenaNula()
                meio=Cena(IMAGEM_MEIO)) # default = NADA = SalaCenaNula()
nome_da_cena_esquerda.esquerda = nome_da_cena

nome_da_cena.vai()
```

SALA

A sala é a formação de um ambiente formado de 4 cenas posicionadas em norte, sul, leste e oeste.

```
from _spy.vitollino.main import Cena, Sala
"""A Sala é uma COLEÇÃO de cenas organizadas nos pontos cadeais norte, sul, leste e␣
↪oeste
"""


IMAGEM_NORTE= "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
IMAGEM_LESTE = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
IMAGEM_OESTE = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
IMAGEM_SUL = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
```

(continues on next page)

(continued from previous page)

```
nome_da_cena_norte = Cena(IMAGEM_NORTE)
nome_da_cena_sul = Cena(IMAGEM_SUL)
nome_da_cena_leste = Cena(IMAGEM_LESTE)
nome_da_cena_oeste = Cena(IMAGEM_OESTE)

""" Bem como na composição na Cena, a ausencia de Cena em algum dos pontos cardeais␣
↪direciona para a SalaCenaNula()"""
nome_da_sala = Sala(n=nome_da_cena_norte, s=nome_da_cena_sul, l=nome_da_cena_leste,␣
↪o=nome_da_cena_oeste)

nome_da_sala.norte.vai() # A primeira Cena a ser visualizada
#nome_da_sala.sul.vai()
#nome_da_sala.leste.vai()
#nome_da_sala.oeste.vai()
```

LABIRINTO

O Labirinto é um conjunto de SALAS ligadas.

```
from _spy.vitollino.main import Cena, Sala, Labirinto
"""O Labirinto é uma coleção de Salas
"""

IMAGEM_NORTE= "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
IMAGEM_LESTE = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
IMAGEM_OESTE = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
IMAGEM_SUL = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif

IMAGEM2_NORTE= "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
IMAGEM2_LESTE = "string_correspondente_a_url_e_extensao_da_image" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
IMAGEM2_OESTE = "string_correspondente_a_url_e_extensao_da_image" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
IMAGEM2_SUL = "string_correspondente_a_url_e_extensao_da_image" # Extensões aceitas:␣
↪png, jpg, jpeg e gif

"""Cria as cenas da primeira sala"""
nome_da_cena1_norte = Cena(IMAGEM_NORTE)
nome_da_cena1_sul = Cena(IMAGEM_SUL)
nome_da_cena1_leste = Cena(IMAGEM_LESTE)
nome_da_cena1_oeste = Cena(IMAGEM_OESTE)

"""Cria a sala com a primeira leva de cenas"""
nome_da_sala1 = Sala(n=nome_da_cena_norte, s=nome_da_cena_sul, l=nome_da_cena_leste,␣
↪o=nome_da_cena_oeste)

"""Cria as cenas da segunda sala"""
nome_da_cena2_norte = Cena(IMAGEM2_NORTE)
nome_da_cena2_sul = Cena(IMAGEM2_SUL)
```

(continues on next page)

(continued from previous page)

```
nome_da_cena2_leste = Cena(IMAGEM2_LESTE)
nome_da_cena2_oeste = Cena(IMAGEM2_OESTE)

"""Cria a sala com as segunda leva de cenas"""
nome_da_sala2 = Sala(n=nome_da_cena2_norte, s=nome_da_cena2_sul, l=nome_da_cena2_
↪leste, o=nome_da_cena2_oeste)

"""Gera o Labirinto"""
resulta_labirito=Labirinto(c=nome_da_sala1,n=nome_da_sala2)
"""Inicia o labirinto referenciando a Sala e a cena"""
resulta_labirinto.centro.norte.vai()
```

ELEMENTO

O elemento é um objeto estático colocado em alguma parte da cena. Pode ser inserido no inventário.

Warning: Só é possível colocar elemento se houver alguma cena que acomode-a.

```
from _spy.vitollino.main import Cena, Elemento
""" O elemento é um objeto passível de ser colocado em alguma cena.
"""
MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif

nome_da_cena = Cena(MINHA_CENA)
nome_do_elemento = Elemento(MEU_ELEMENTO, tit="título_do_elemento",
                            style=dict(height=60,widht=60, left=600, top=20), # ou␣
↪x=eixo_x, y=eixo_y, w=largura, h=altura
                            cena = nome_da_cena)
```

TEXTO (PopUp)

É uma mensagem que aparecerá na tela.

- Texto associado a abertura da Cena

```
from _spy.vitollino.main import Cena, Elemento, Texto
""" O objeto é o elemento clicável de alguma cena.
"""
MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif

nome_da_cena = Cena(FUNDO)
nome_da_cena.vai()
texto_ = Texto(nome_da_cena, txt = "Mensagem desejada")
texto_.vai()
```

- Texto subordina aparecimento de Elemento

```
from _spy.vitollino.main import Cena, Elemento, Texto
""" O objeto é o elemento clicável de alguma cena.
"""
MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif

def chama_elemento(*args):
    nome_do_elemento = Elemento(LIVRO, tit="título_do_elemento",
                                style=dict(height=60,widht=60, left=600, top=20)) #␣
↪ou x=eixo_x, y=eixo_y, w=largura, h=altura
    nome_do_elemento.entra(nome_da_cena)

nome_da_cena = Cena(FUNDO)
nome_da_cena.vai()
texto_ = Texto(nome_da_cena, txt = "Mensagem desejada", foi = funcao_do_elemento) # o␣
↪método foi() esconde o popup
texto_.vai()
```

BOTÃO

O botão é um elemento visualizado como portal. Criando um botão é possível associar o clique a algum acontecimento.

Existe algumas formas de criar um botão:

- Associando ao método `vai()` da classe Elemento

```
from _spy.vitollino.main import Cena, Elemento
""" O botão é o elementoclicável de alguma cena.
"""
MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif

def funcao_de_acao_do_botao(event = None):
    #Funcao chamada no clique
    print("Você clicou no botão!") # evento associado ao clique: mensagem, cena, sala,
↪módulo...

nome_da_cena = Cena(MINHA_CENA)
nome_da_cena.vai() # instancia a cena
nome_do_elemento = Elemento(MEU_ELEMENTO, tit="título_do_elemento",
                            style=dict(height=60,widht=60, left=600, top=20), # ou␣
↪x=eixo_x, y=eixo_y, w=largura, h=altura
                            cena = nome_da_cena,
                            vai = funcao_de_acao_do_botao)
```

- Associando ao evento do browser

```
from _spy.vitollino.main import Cena, Elemento, Texto
""" O botão é o elemento clicável de alguma cena.
"""
MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
```

(continues on next page)

(continued from previous page)

```
def funcao_de_acao_do_botao(event = None):
    #Função chamada no clique resultará na chamada de um texto
    texto_surpresa = Texto(nome_da_cena, txt ="Mensagem que você deseja passar!")
    texto_surpresa.vai()

nome_da_cena = Cena(MINHA_CENA)
nome_da_cena.vai()
nome_do_elemento = Elemento(MEU_ELEMENTO, tit="título_do_elemento",
                            style=dict(height=60,widht=60, left=600, top=20), # ou␣
↪x=eixo_x, y=eixo_y, w=largura, h=altura
                            cena = nome_da_cena)

nome_do_elemento.elt.bind("click", funcao_de_acao_do_botao)
```

MÚLTIPLA-ESCOLHA

A múltipla escolha é implementada usando a classe Texto do Vitollino. Funciona como um popup onde o jogador pode selecionar algo (opção)

```
 from _spy.vitollino.main import Cena, Texto
 """ A multipla escolha é um popup com opções
 """

 MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
 MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif

 def resultado(opcao_escolhida):
     # O novo popupque será gerado quando o foi() do texto forchamado
     dicionario = dict(A="Você clicou no A", B="Você clicou no B") # dicionário que␣
↪guarda a devolutiva da opção escolhida
     devolutiva = Texto(nome_da_cena, txt=dicionario[opcao_escolhida])
     devolutiva.vai()


 nome_da_cena = Cena(MINHA_CENA)
 nome_da_cena.vai()


 pergunta = Texto(nome_da_cena, txt = "Seu enunciado aqui", foi = resultado, A=
↪"resposta", B= "resposta")
pergunta.vai()
```

INVENTÁRIO

Inventário é um espaço onde os elementos encontrados podem ser guardados. Há dois modos de criar um inventário:

- Objeto **não** resgatável

```
from _spy.vitollino.main import Cena, Elemento
from _spy.vitollino.main import INVENTARIO as inv
"""O inventário funciona como um depósito de elementos não resgatáveis
```

(continues on next page)

(continued from previous page)

```
"""

MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif


inv.inicia() # comando que starta o inventário
nome_da_cena = Cena(MINHA_CENA)
nome_do_elemento = Elemento(MEU_ELEMENTO, tit="título_do_elemento",
                            style=dict(height=60,widht=60, left=600, top=20), # ou ,
↪x=eixo_x, y=eixo_y, w=largura, h=altura,
                            cena = nome_da_cena,
                            vai = self.coloca_no_inventario)

coloca_no_inventario = lambda *_: inv.bota(nome_do_elemento, True) #testar

def coloca_no_inventário(self, *_):
   """Gera um função que será resgatada no vai() do elemento para associar o clique à␣
↪entrada no inventário"""
   inv.bota(nome_do_elemento, True)
```

- Objeto Resgatável

É possível resgatar o Elemento construindo uma classe que tenha o método de resgate

```
from _spy.vitollino.main import Cena, Elemento
from _spy.vitollino.main import INVENTARIO as inv

MEU_ELEMENTO = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões␣
↪aceitas: png, jpg, jpeg e gif
MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif

class Item_herdado(Elemento):
    """Construção de uma classe que herde de Elemento
    """
    def bota(self, *_):
        """Aciona estado de inv.bota = True para que eventual clique devolva o␣
↪Elemento para a cena"""
        inv.bota(self, True)
        #self.vai=lambda*_:self.resgata(x=x,y=y,w=w,h=h)
        """Método vai do Elemento atrelado ao evento de reposicionamento, onde o␣
↪memento especifica os argumentos pedidos pelo método resgata."""
        self.vai=lambda*_:self.resgata(*self.memento)

    def resgata(self,x,y,w,h):
        """Método para resgate do Elemento no inventário"""
        self.x,self.y,self.w,self.h= x,y,w,h
        """Retira Elemento atrelado ao título do inventário"""
        inv.tira(self.tit)
        """Coloca Elemento na cena"""
        self.entra(inv.cena)
        """Aciona estado de inv.bota = False para que eventual clique devolva o␣
↪Elemento para o inventário"""
        self.vai=self.bota
```

(continues on next page)

(continued from previous page)

```
    def mementor(self,memento):
        """Permite que o style do elemento a ser recolocado na tela seja especificado
↪"""
        self.memento=memento


class Main():

    def __init__(self):
        inv.inicia()
        self.minha_cena=Cena(MINHA_CENA)

        self.meu_elemento=Item_herdado(MEU_ELEMENTO, tit="nome_do_meu_elemento",
↪style=dict(height=60,widht=60, left=100, top=100),cena=self.minha_cena)
        self.meu_elemento.mementor((110,150,200,"200px"))
        self.meu_elemento.vai=self.meu_elemento.bota

        self.minha_cena.vai()

if __name__ == "__main__":
    Main()
```

MÚSICA

```
from _spy.vitollino.main import Cena, Elemento

MINHA_CENA = "string_correspondente_a_url_e_extensao_da_imagem" # Extensões aceitas:␣
↪png, jpg, jpeg e gif
MINHA_MUSICA = "string_correspondente_a_url_e_extensao_da_musica" # Extensões␣
↪aceitas: mp3, mp4

nome_da_cena = Cena(MINHA_CENA)
nome_da_cena.vai()

nome_da_musica = Musica(MINHA_MUSICA, loop = True, autoplay = True)
```

CÓDIGO

PORTAL

DROPPER

DROPPABLE

6.1.4 Instalação de Softwares Recomendados

Este setor compreende as etapas de instalação de softwares recomendados para o estudo da programação.

Note: Para que os programas funcionem com *eficiência* será necessário:

REQUISITOS MÍNIMOS		
	MÍNIMO	RECOMENDADO
MEMÓRIA RAM	1GB	8GB
ESPAÇO HD	4 GB	SSD
TELA	1024x768	

Selecione o sistema operacional de preferência:

Note: O Tutorial do sistema operacional Ubuntu **INCLUI** a instalação do SO.

Sistema Operacional Linux

See also:

Este documento é uma adaptação dos seguintes tutoriais: ../intro_comp/InstlProgrm

Selecione o Tópico de sua Necessidade:

INSTALAÇÃO do LINUX

See also:

Este documento é uma adaptação do seguinte tutorial existente: *INSTALAÇÃO DE PROGRAMAS*

SUMÁRIO

INFORMAÇÕES GERAIS

O Ubuntu é uma - de muitas - distribuição do sistema operacional (SO) Linux produzida pela empresa Canonical.

É uma distribuição gratuita e muito popular entre programadores pois possui uma estrutura de gestão, embora pouco intuitiva ao primeiro olhar, que entrega ao usuário a administração do sistema em troca de poucos comandos no console.

Vamos colocar na ponta do lápis:

VANTAGENS E DESVANTAGENS		
	LINUX	WINDOWS
SIST. OPERACIONAL	GRATUITO	PAGO
ATUALIZAÇÃO DO SISTEMA	SIM	SIM
SISTEMAS DIVERSIFICADOS	SIM*	NÃO
SEGURANÇA	SEGURO	ANTI-VÍRUS
DRIVERS GERAIS	SIM	SIM
DRIVERS "ESPECIAIS"	EXTERNOS*	SIM
JOGOS	PRÓPRIOS	SIM
ADMINISTRAÇÃO	TOTAL	PARCIAL
ATUALIZAÇÕES AUTOMÁTICAS	SIM	SIM

*O linux tem sistemas diversos que podem ser escolhidos visando as possibilidades do computador. Por exemplo, a distribuição Lubuntu visa computadores com mínimo 2gb de memória e processadores antigos.

*Quanto ao drivers especiais, há algumas dificuldades no uso de algumas placas de vídeo, monitores não vga e afins. É possível fazê-los funcionar na maioria das vezes, porém requer um esforço de pesquisa por drivers na internet.

Sem mais delongas...

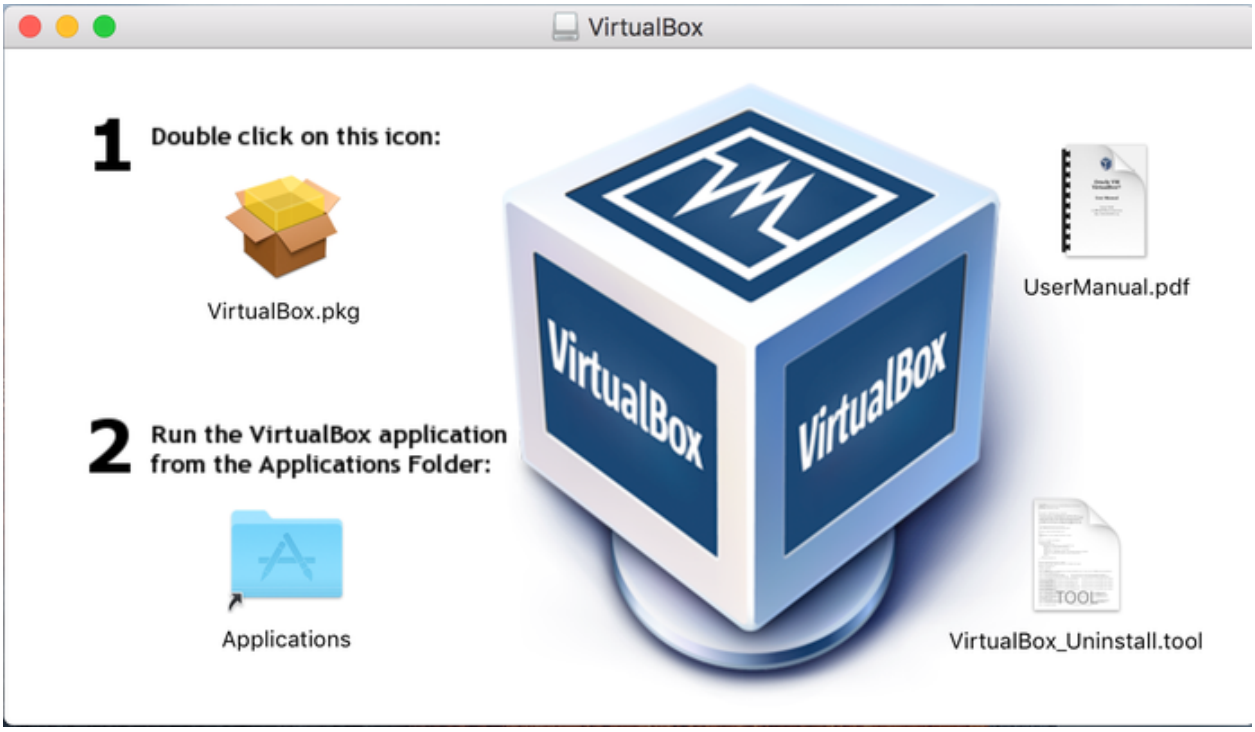
INSTALAÇÃO DA MÁQUINA VIRTUAL

A máquina virtual é um sistema que permite o gerenciamento de um novo sistema operacional ou software. O interessante no uso de uma máquina virtual é a possibilidade de estender seu computador sem danificar seu sistema principal.

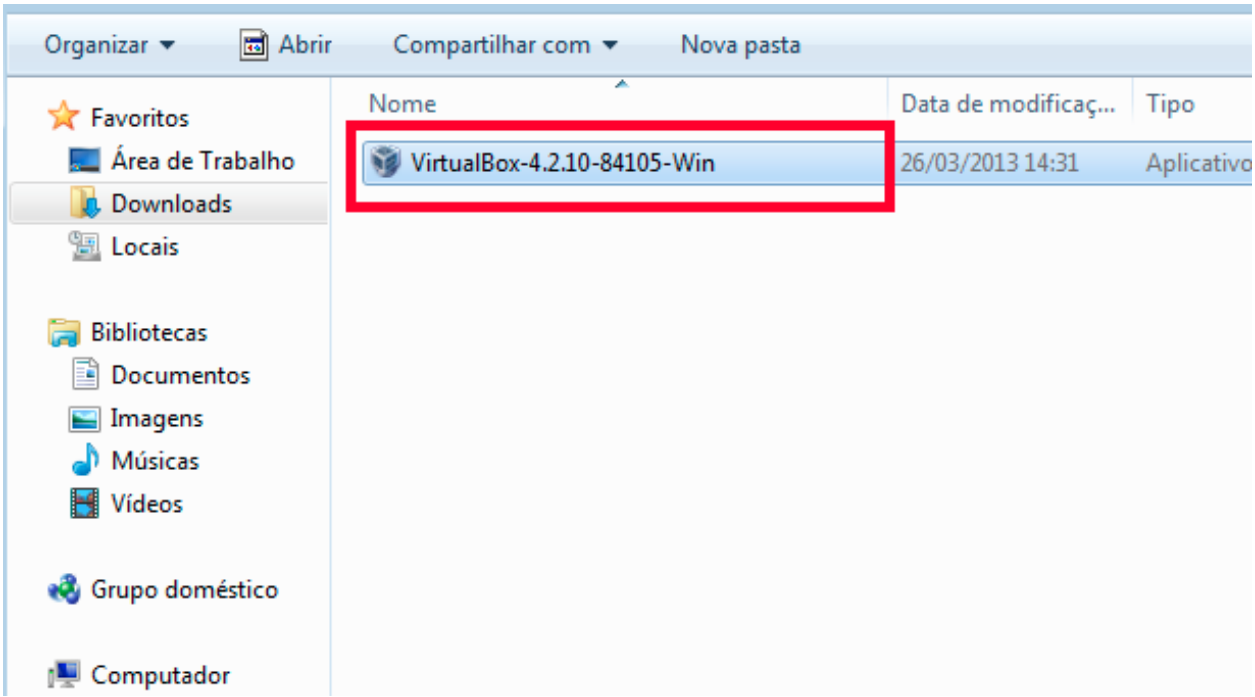
Em outras palavras, no uso do novo sistema operacional, você roda o Linux dentro do Windows, o windows dentro do mac, o windows dentro do Linux...

Vamos ao passoa passo:

1. Baixe o software Oracle VM VirtualBoX Link para download: Download VM VirtualBoX



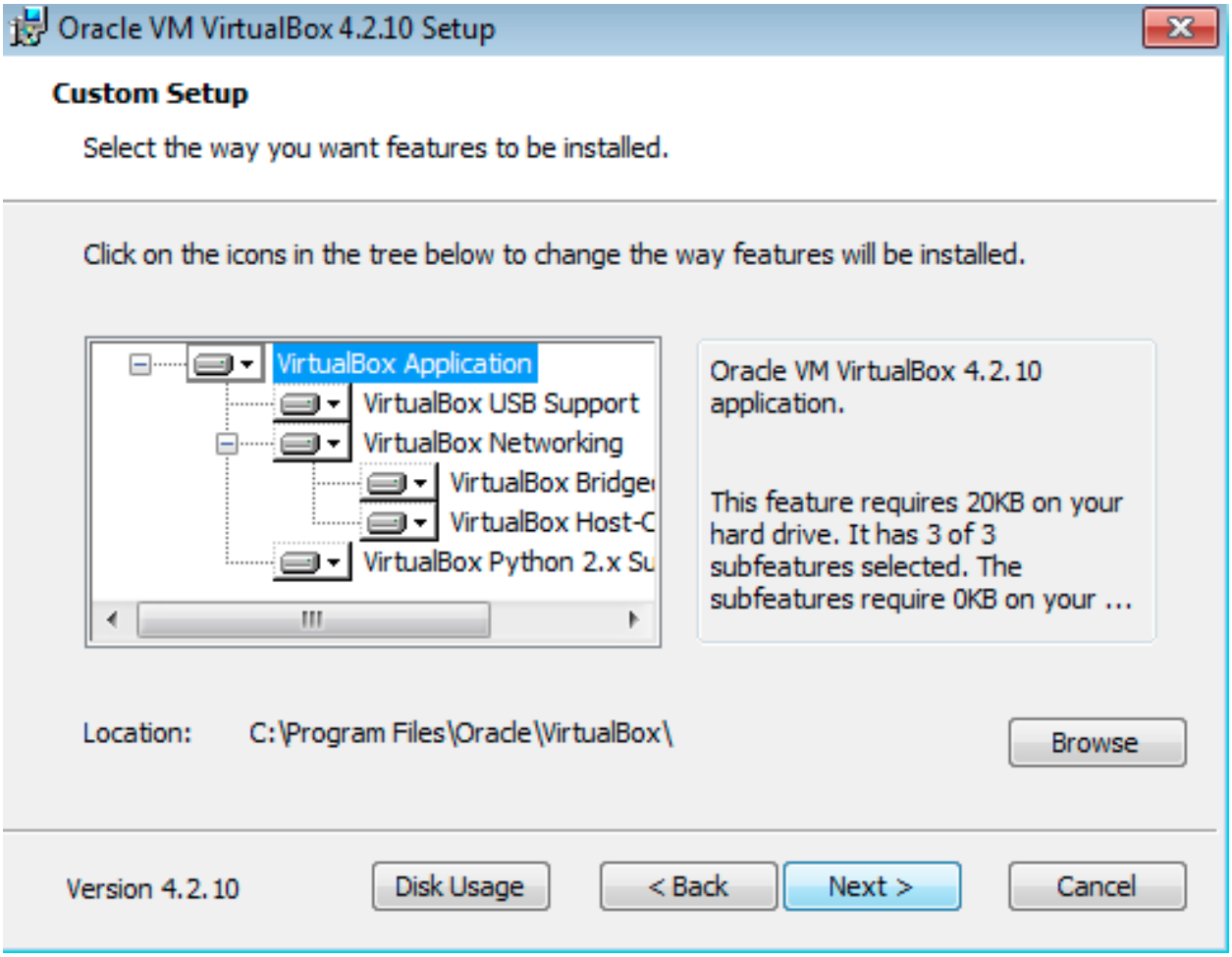
2. Clique no ícone do VirtualBox na sua pasta de downloads.



3. Abrirá uma aba que indica o início do seu processo de instalação.

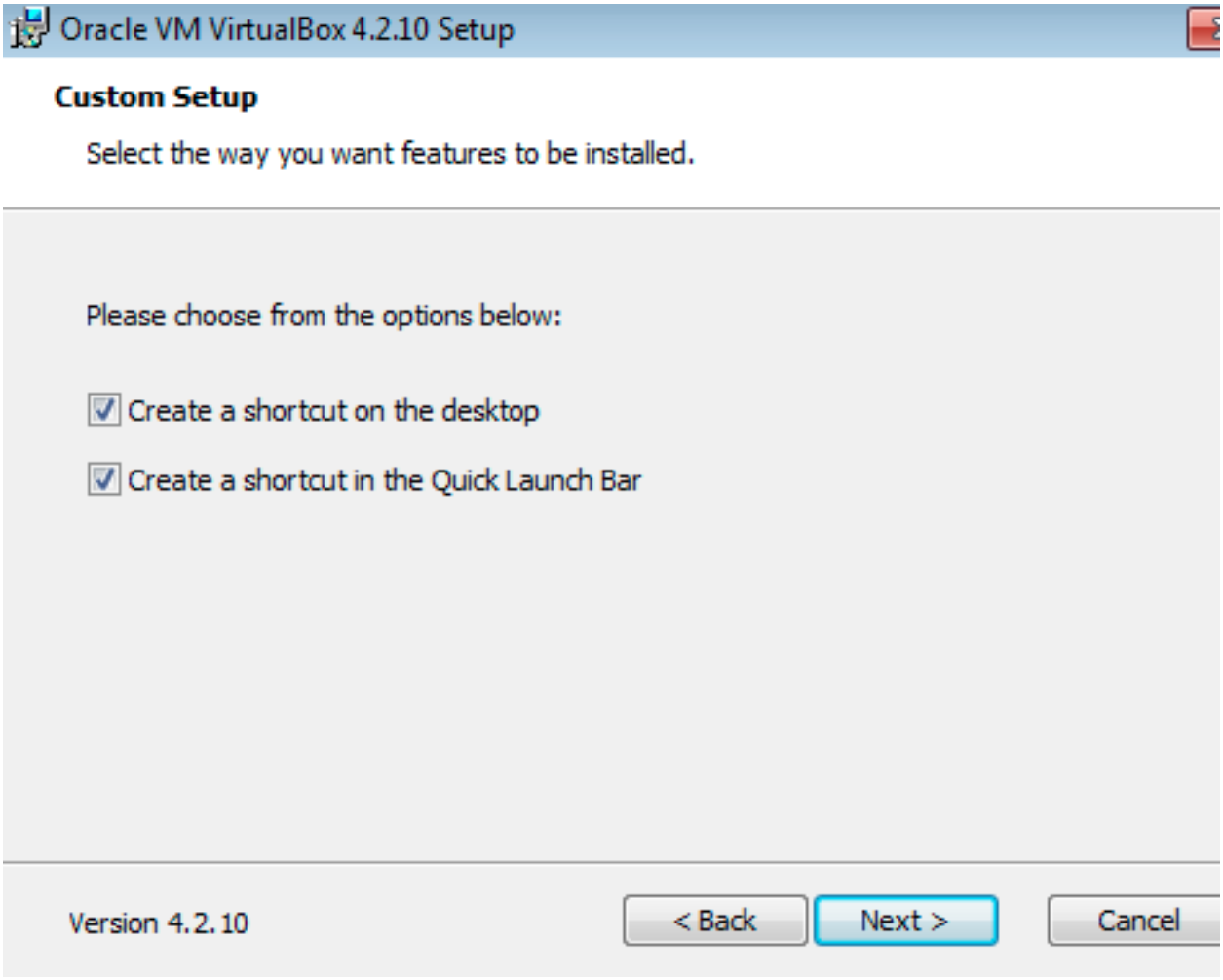
Clique em NEXT.

4. A próxima aba indica o local em que o programa será salvo em seu computador. Mantenha as informações selecionadas ou mude a seu gosto e clique em NEXT.

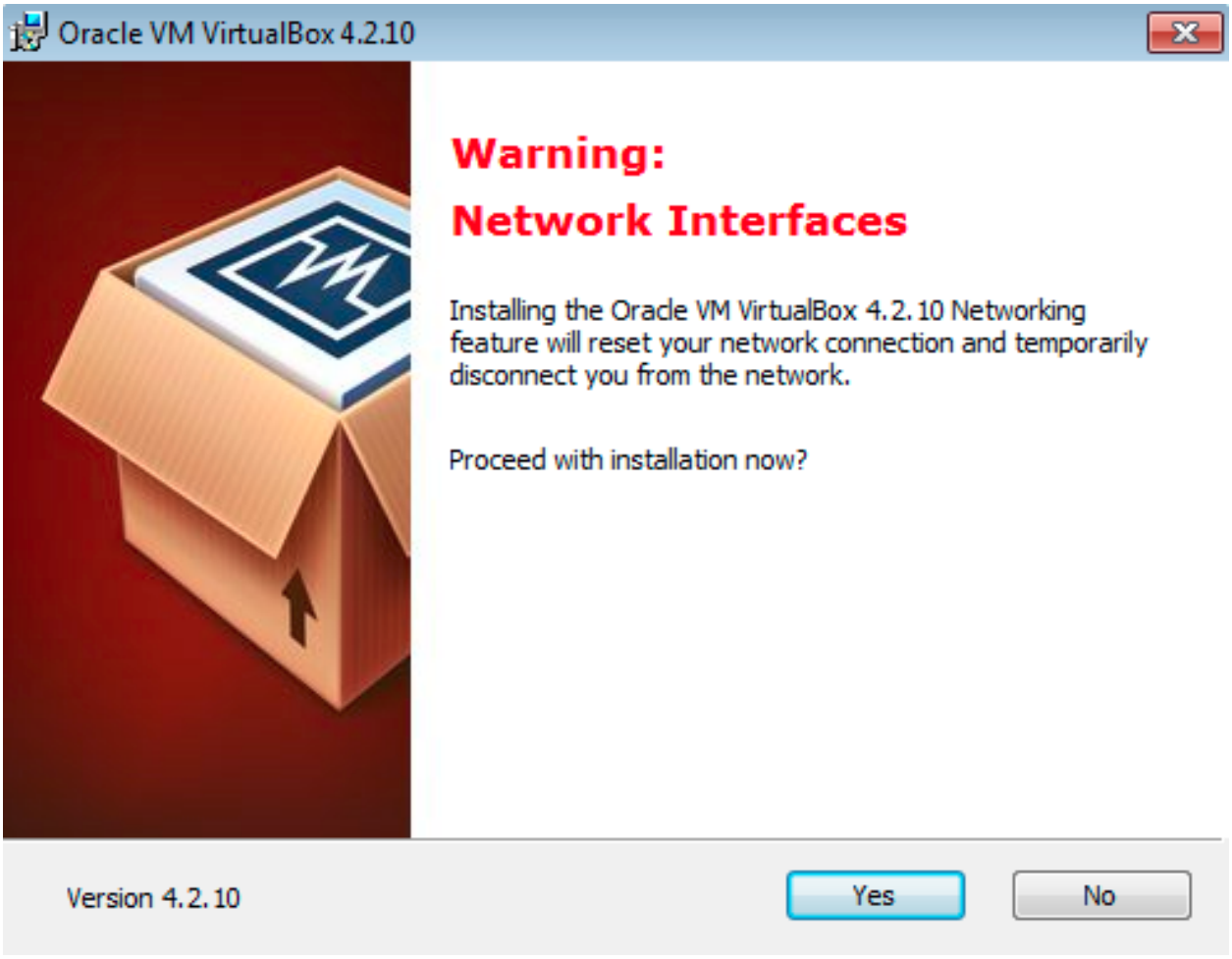


5. Faça suas escolhas e clique em NEXT.

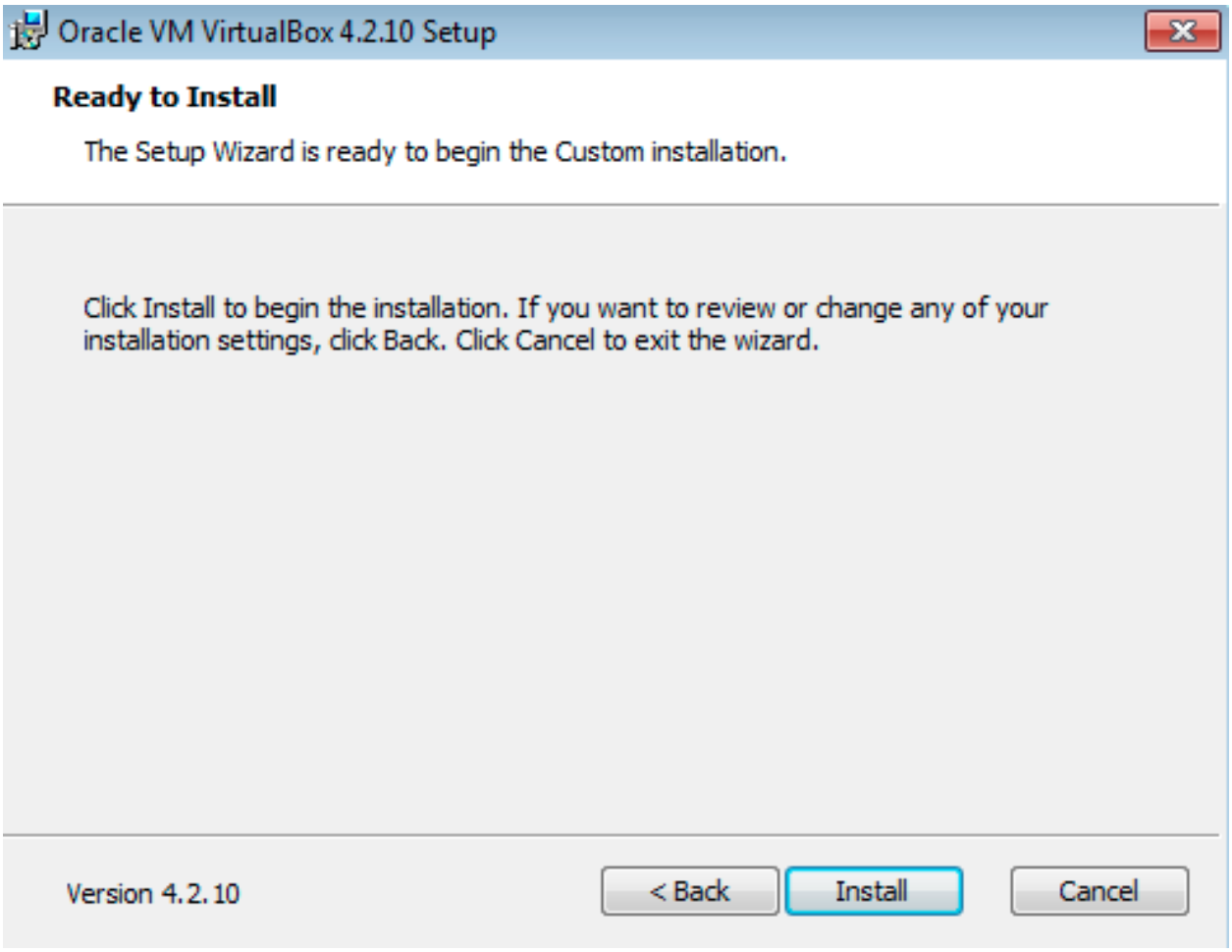
A aba abaixo indica se você quer um atalho para o seu Desktop (primeiro quadrado selecionado) e para sua barra de ferramentas (segundo quadrado selecionado).



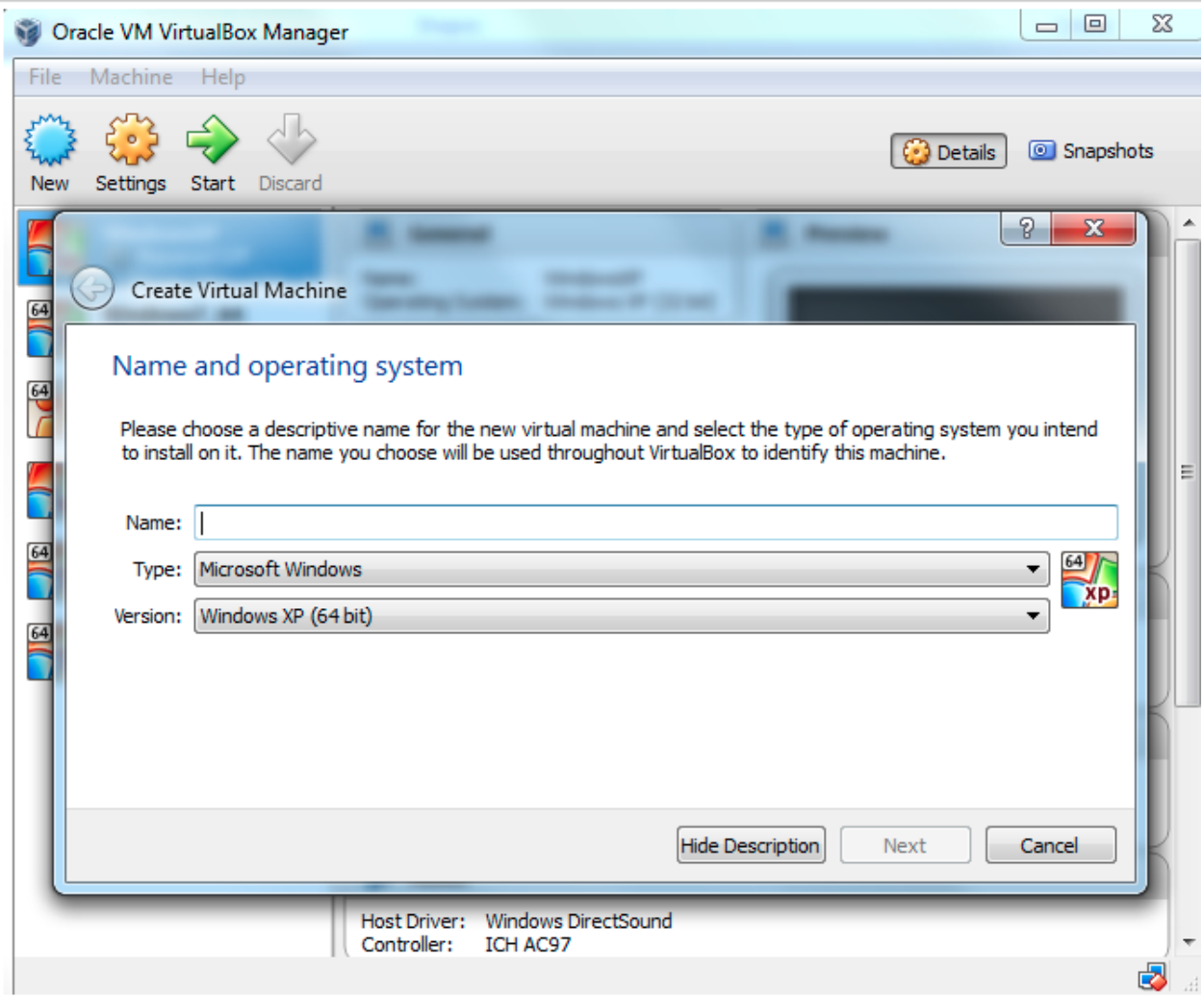
6. Essa é uma aba de atenção. Indica que para prosseguir com a instalção você será disconectado da internet. Salve o que for preciso e clique em NEXT.



7. Finalmente, estamos prontos para iniciar o real processo de instalção. Clique em NEXT.



8. Sua instalção está completa. Clique em FINISH.



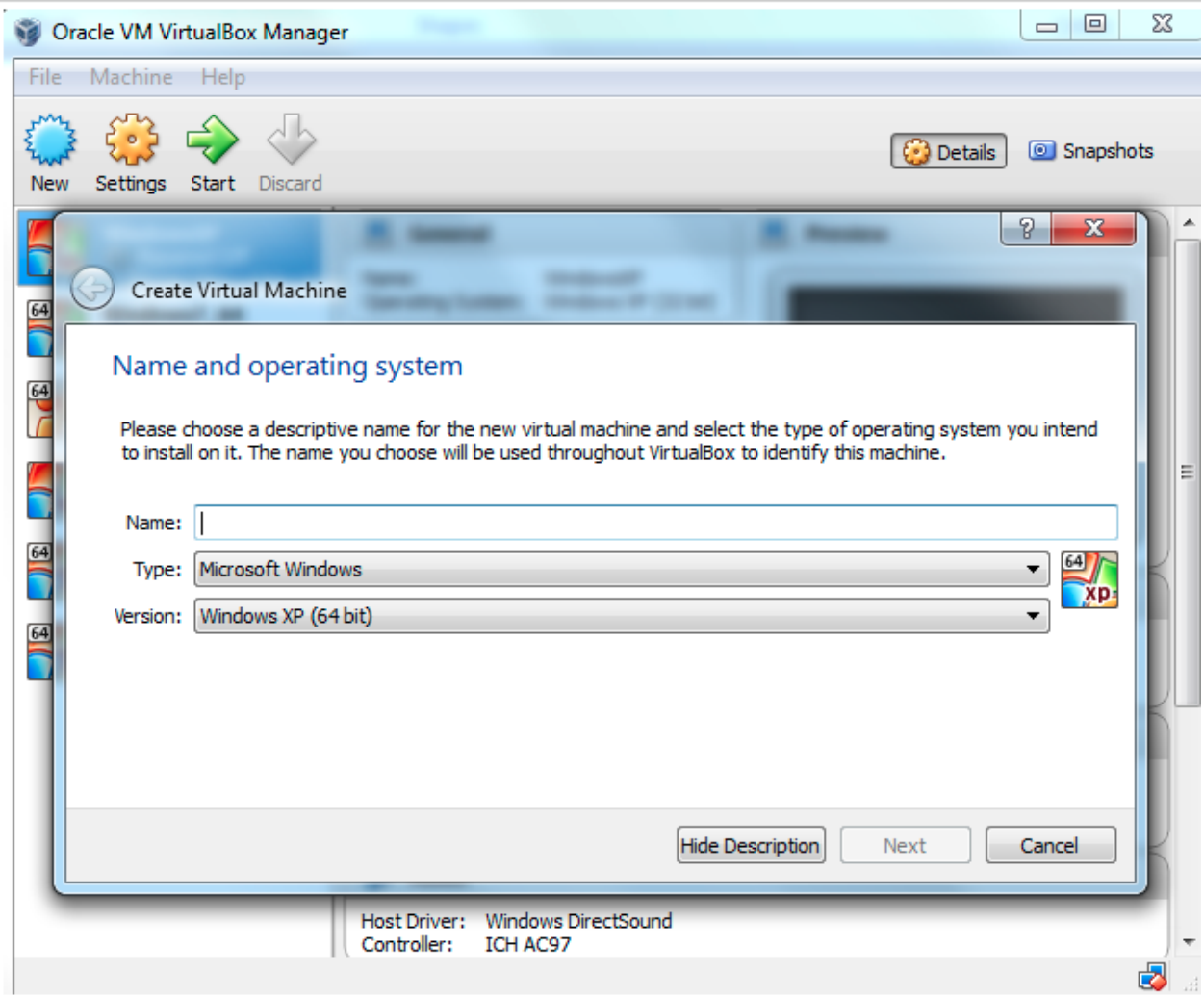
INSTALAÇÃO DO LINUX NA MÁQUINA VIRTUAL

1. Selecione abaixo um Linux compatível com o seu computador:

DISTRIBUIÇÕES LINUX			
	REQUISITO	64BIT	32BIT
LUBUNTU	700MBRAM;10GB HD	L64	L32
UBUNTU 20.04	4GBRAM;25GB HD	U64	
Elementary OS		E64	

Note: Existem diversas distribuições além das disponibilizadas acima. É só dar um google.

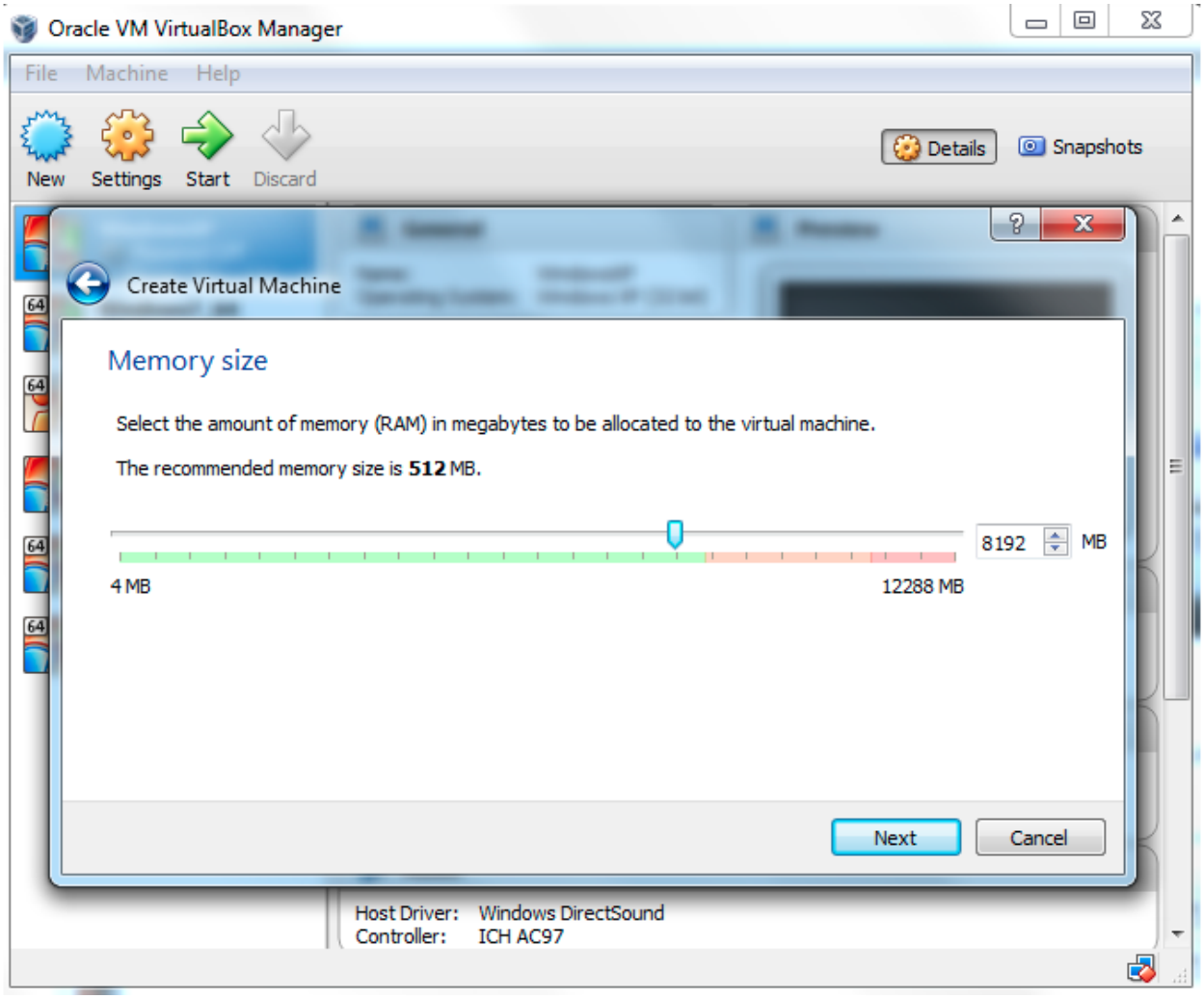
2. Abrir o Oracle VM VirtualBox Gerenciador e **clicar em Creat Virtual Machine**.



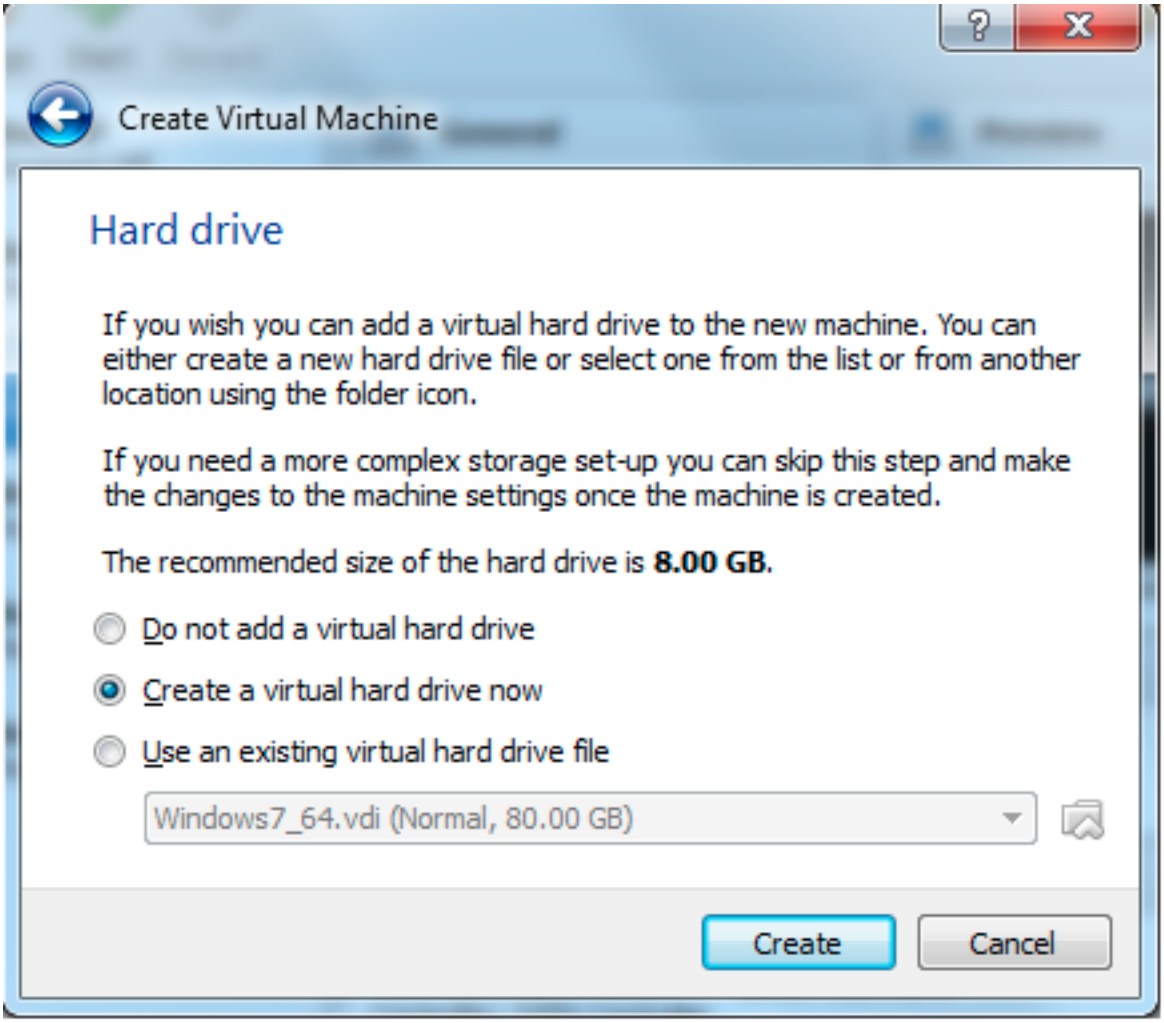
Preencha e clique em NEXT:

- Name: *Seu nome*
- Type: Linux
- Version: *Nome da sua versão*

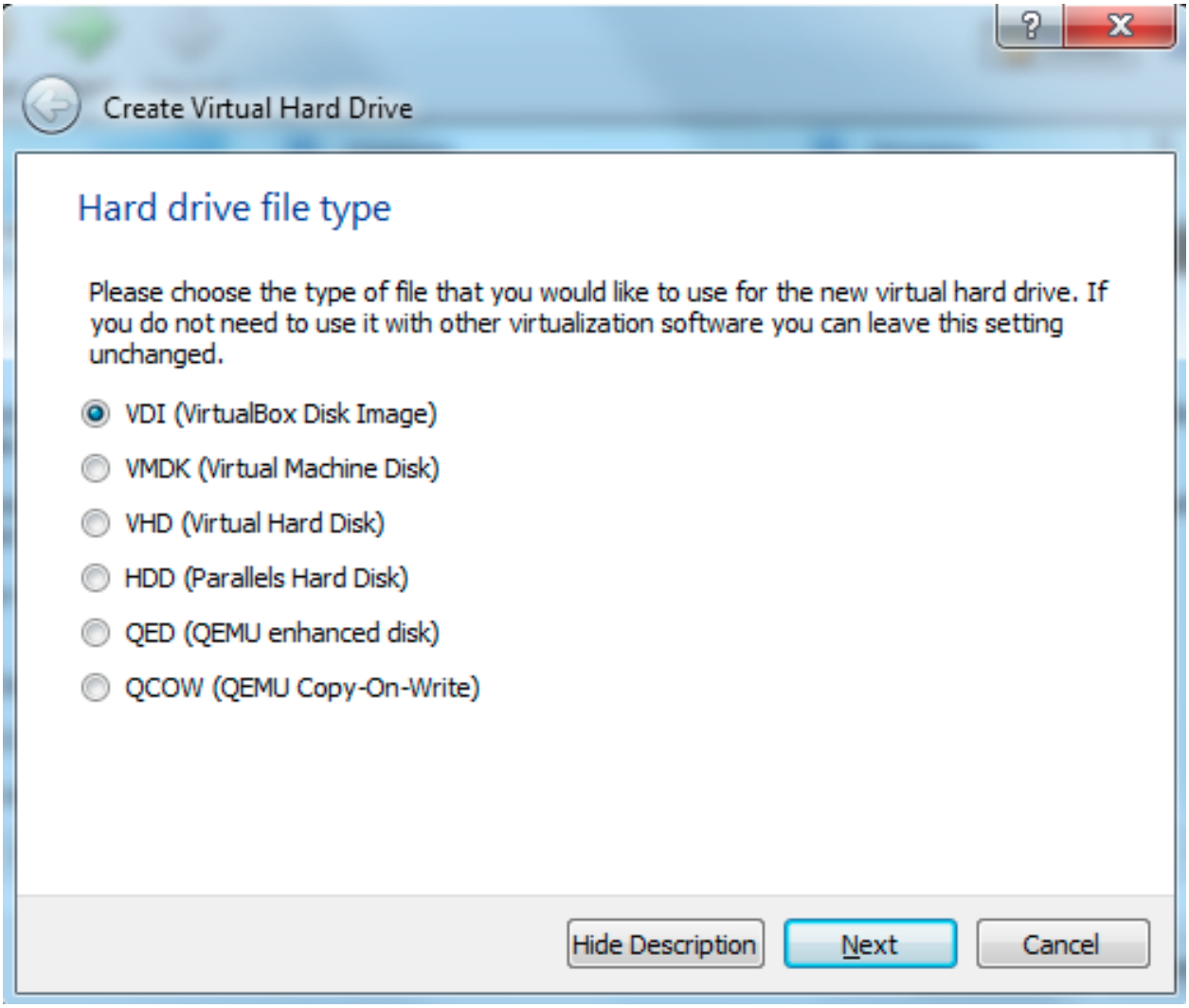
3. Selecione tamanho da memória: 2048 e clique em NEXT



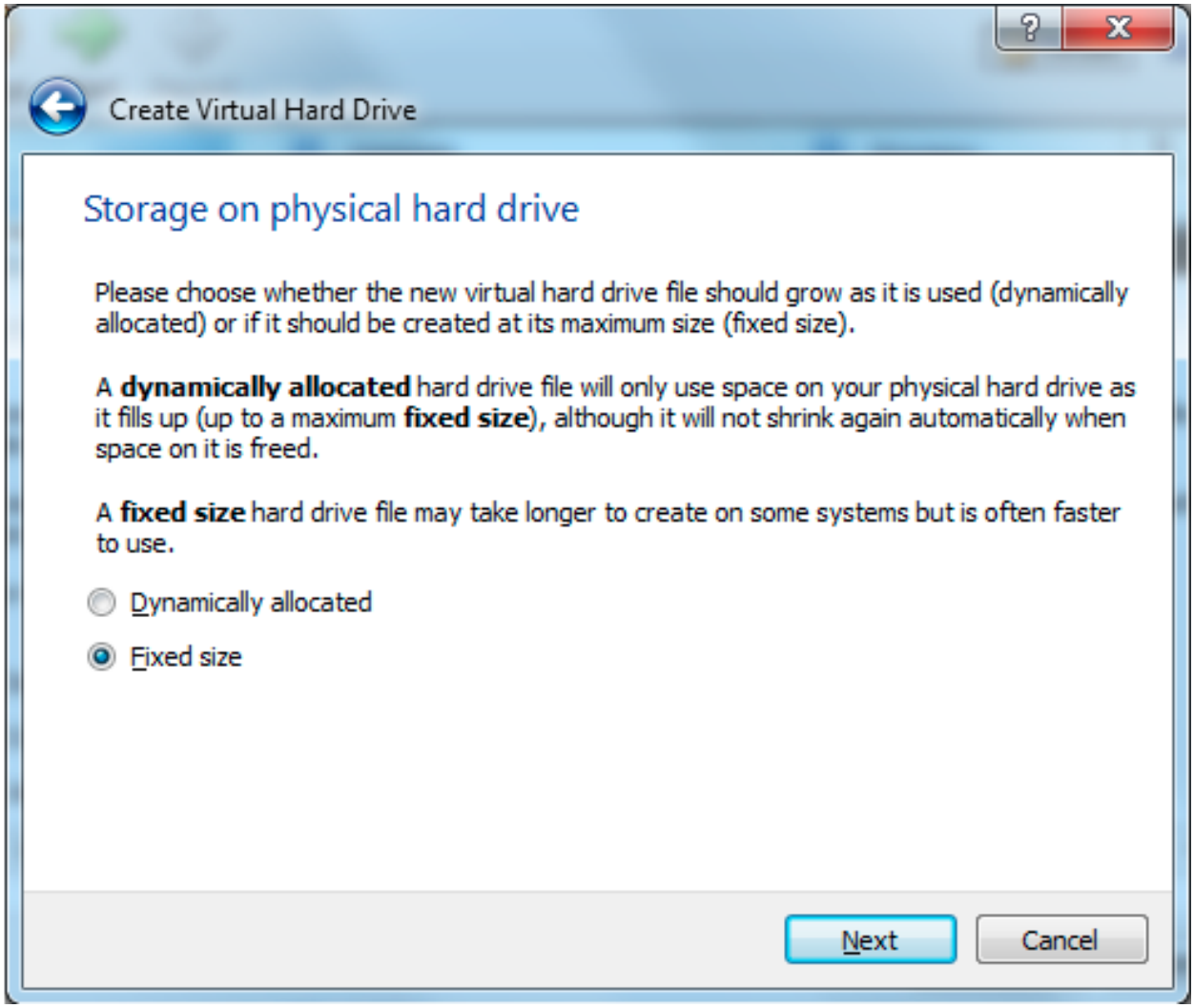
4. Clique em create



5. Selecione VDI (VIRTUALBOX Disk Image)

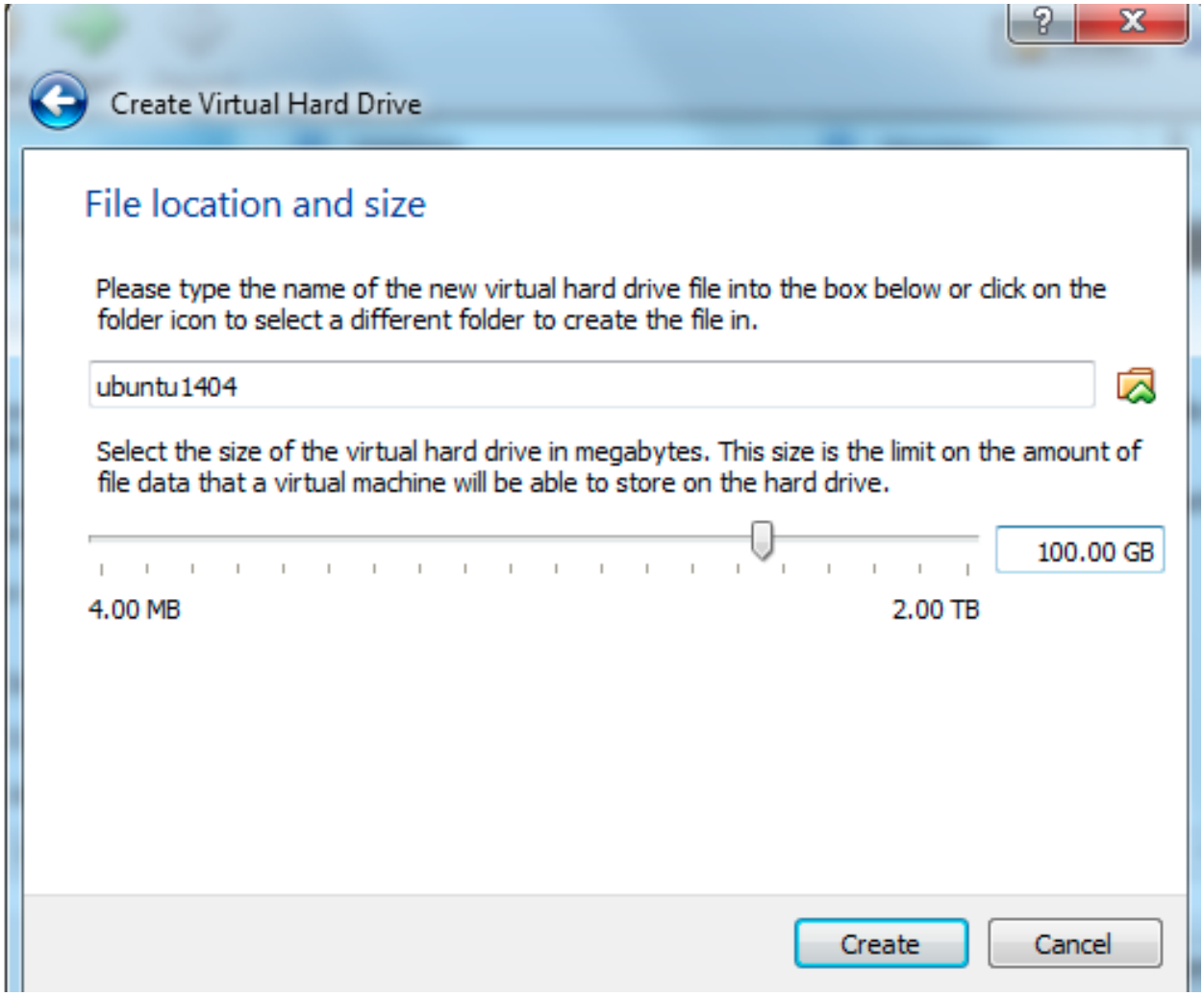


6. **SELECIONE DINAMICALLY ALLOCATED**

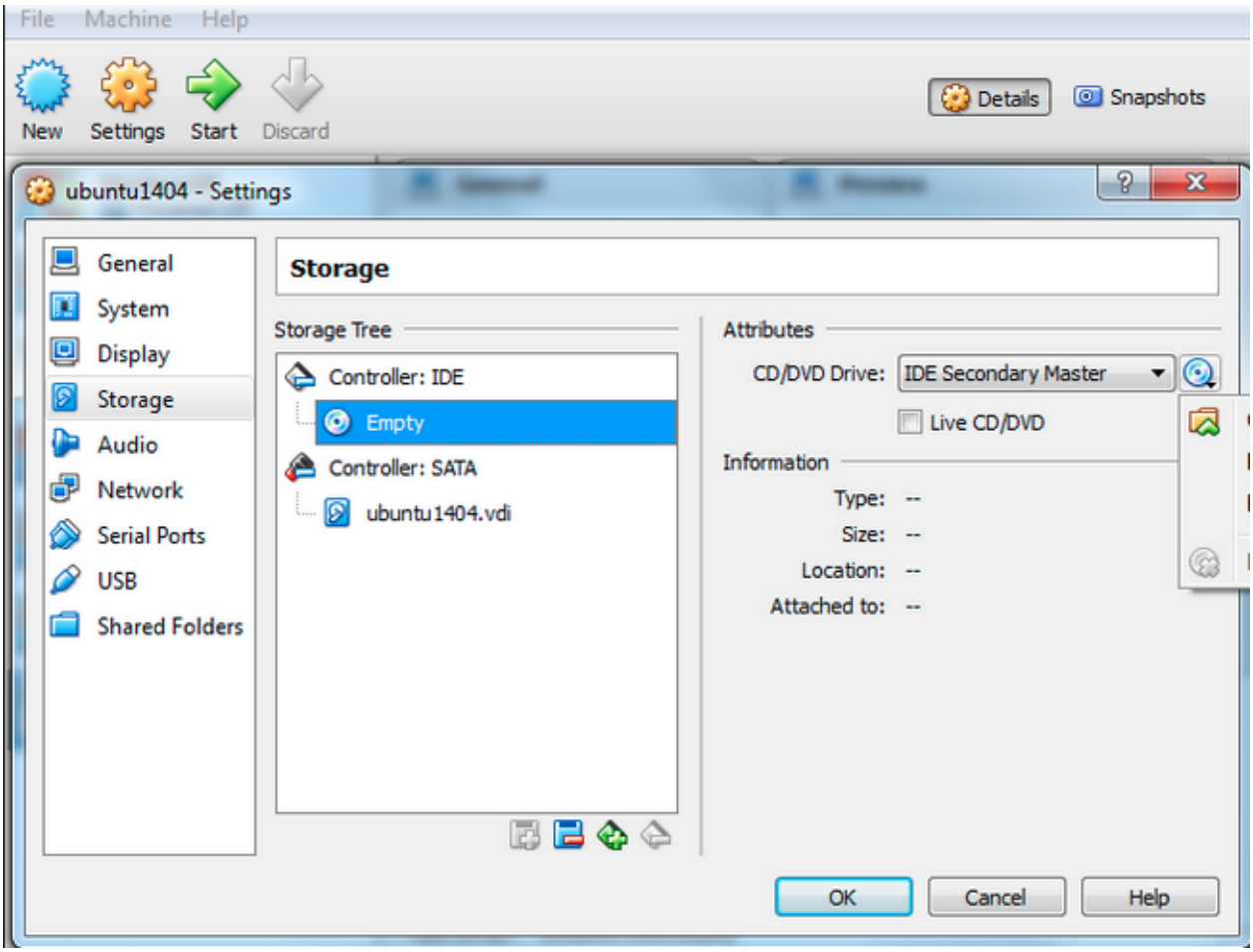


7. Localização e tamanho do arquivo.

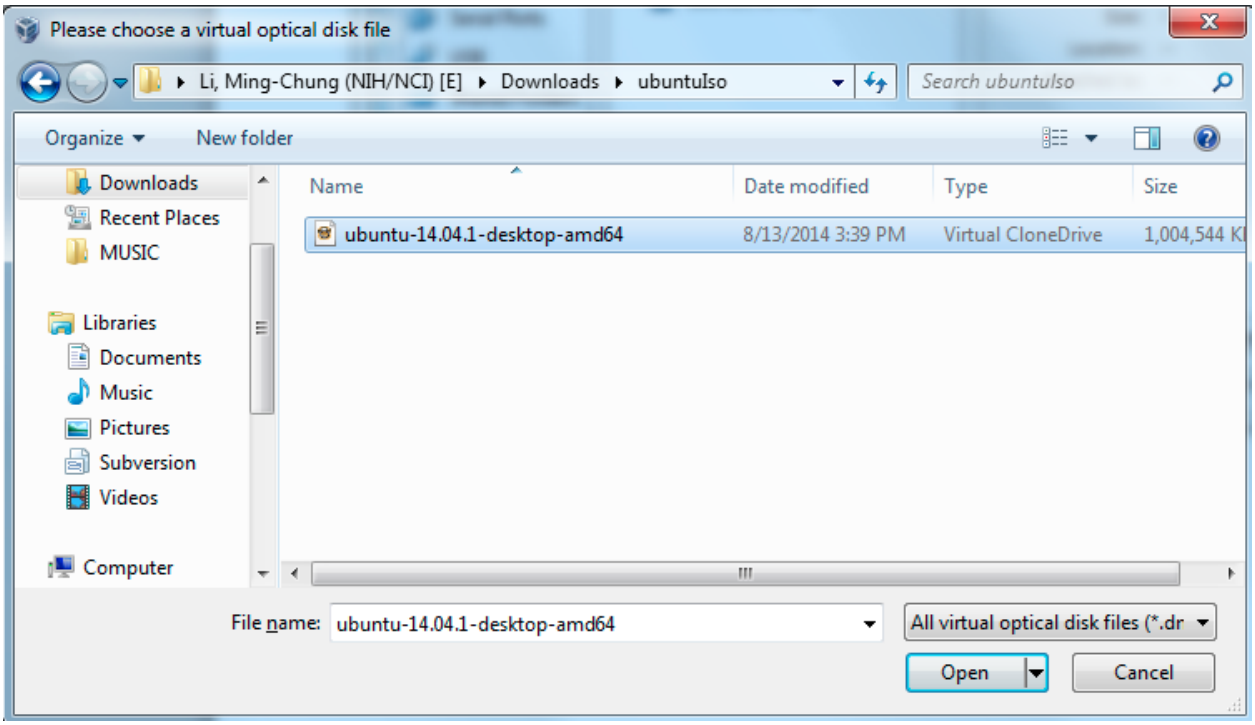
Warning: O tamanho do arquivo dependerá do espaço disponível no HD do seu computador.



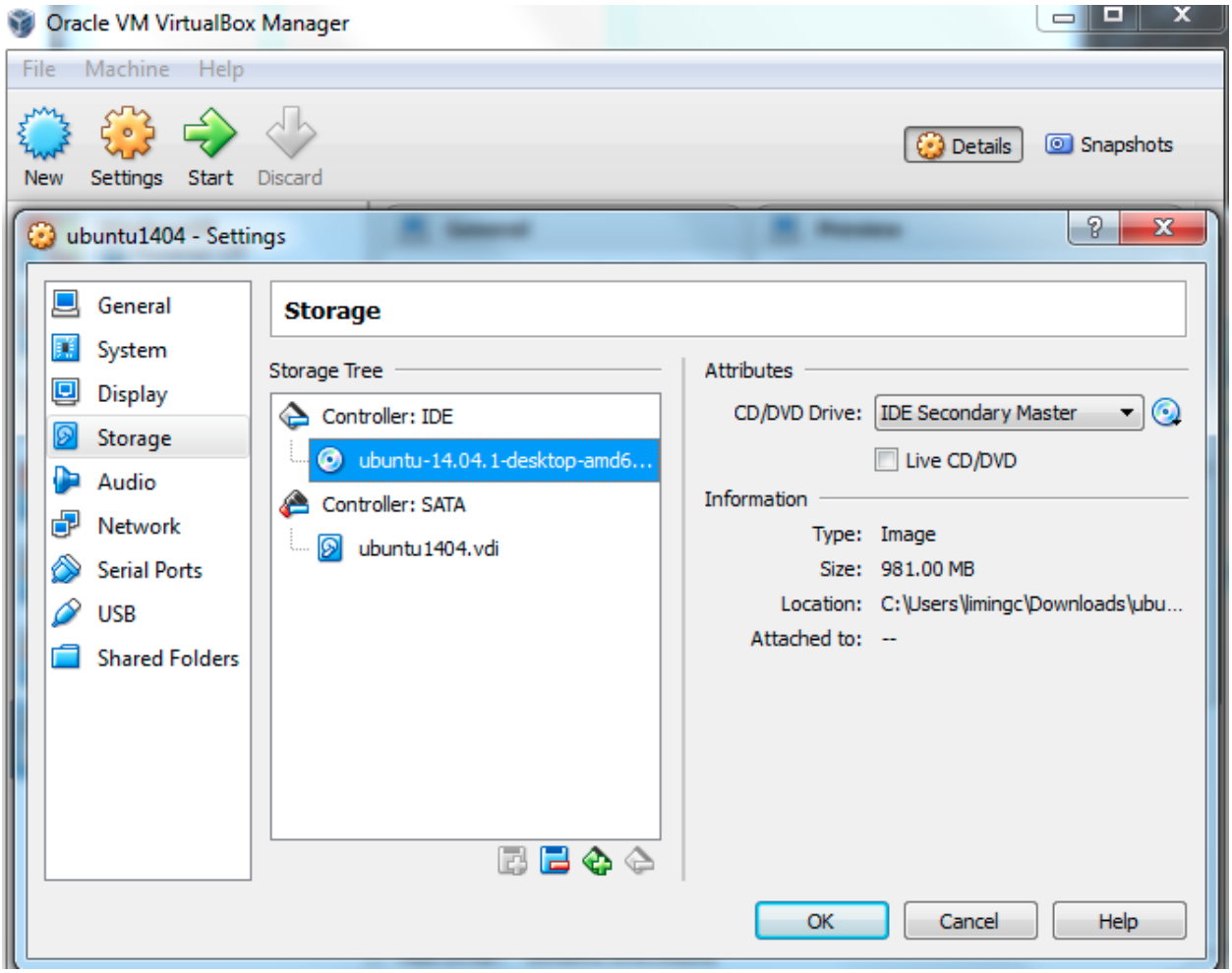
8. no campo à direita, o Armazenamento estará, por padrão *empty*, clique e selecione **Ide secundário Master**.



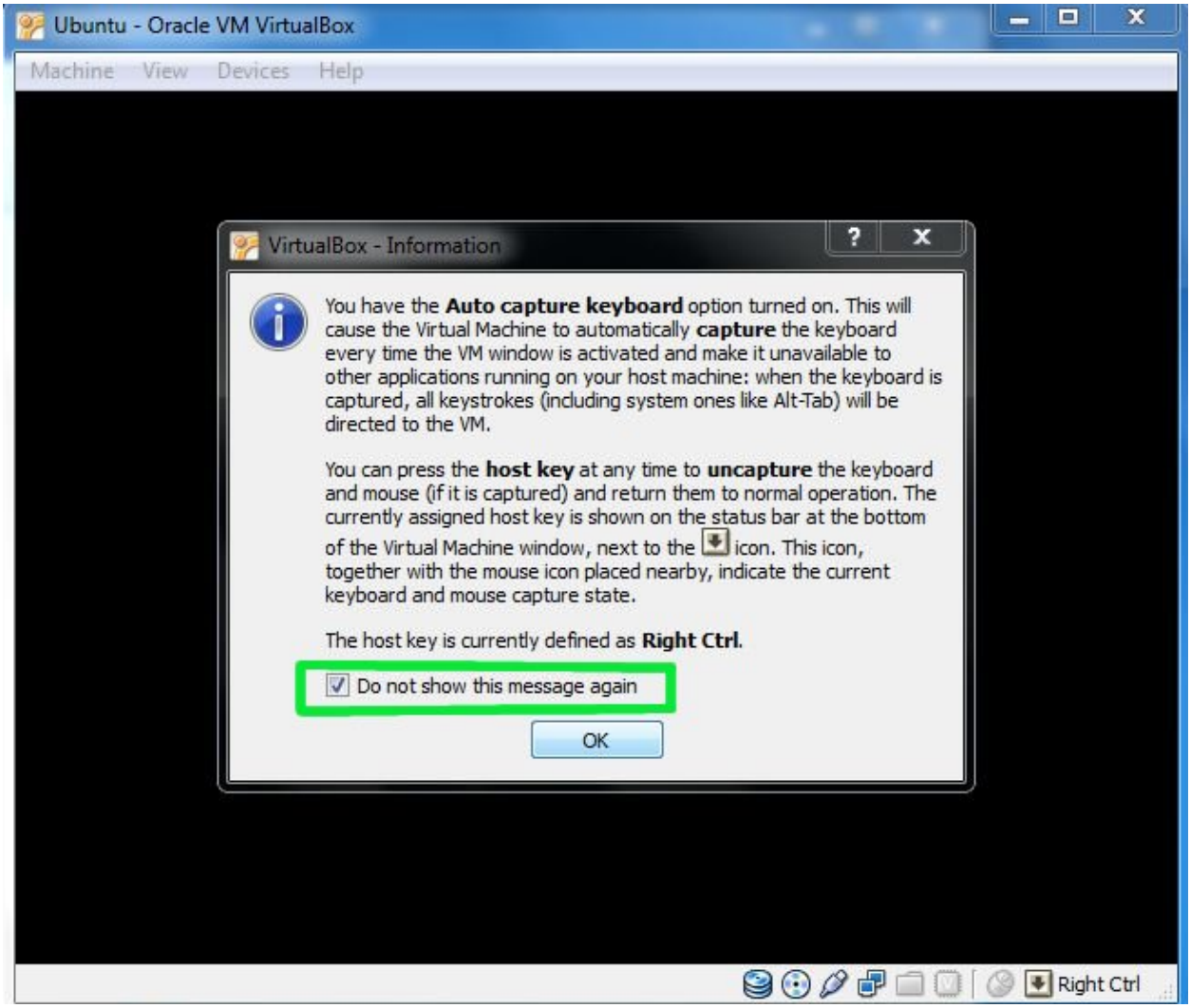
9. Clique no cd azul próximo à seta.
10. Selecione o arquivo do que você baixou para o seu computador.



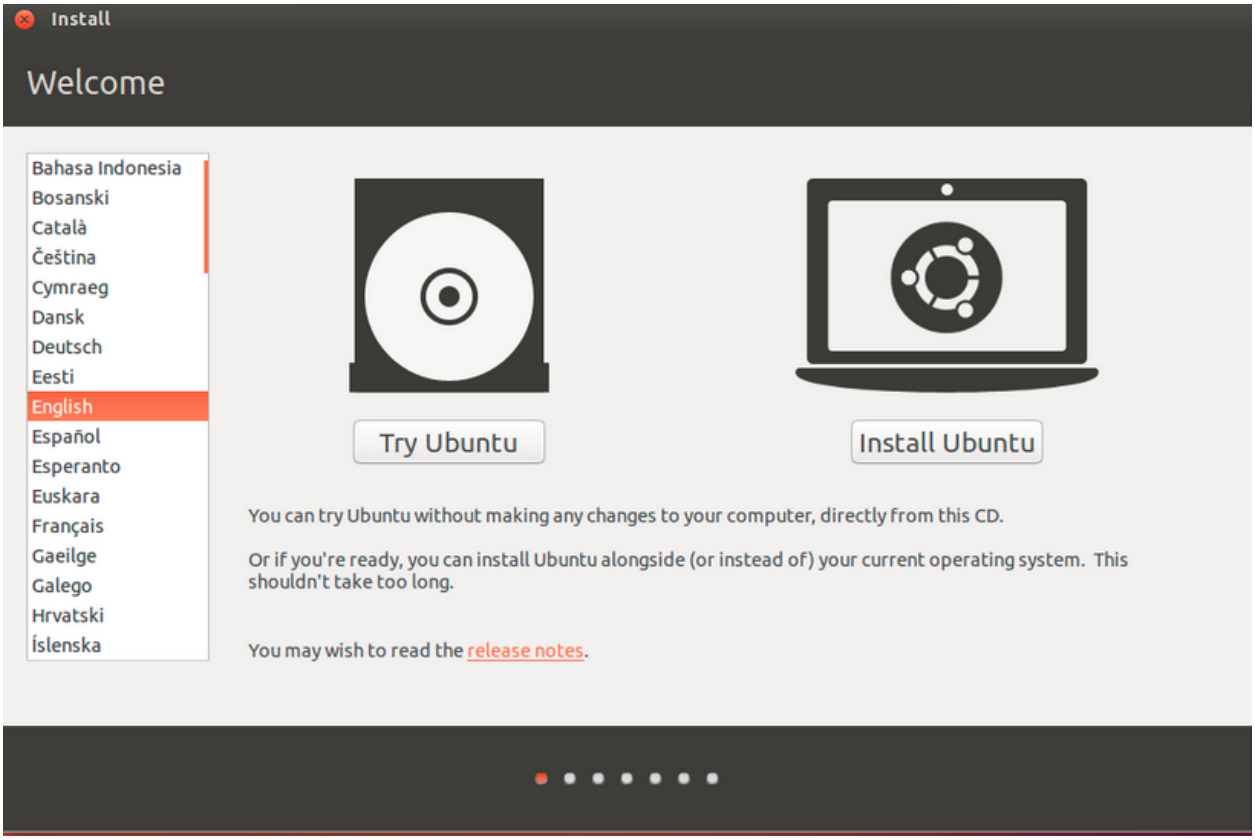
11. Observe se na imagem de CD aparece o do seu arquivo. Se sim, clique seta verde.



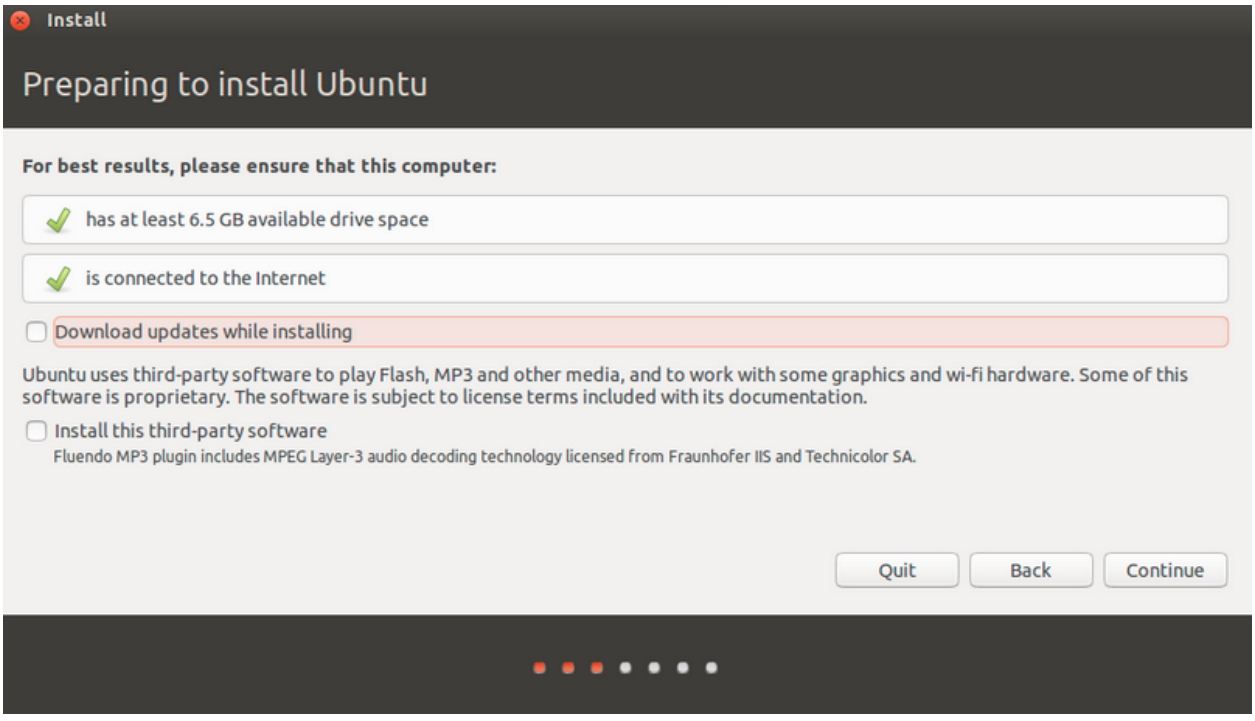
12. Clique em ok.



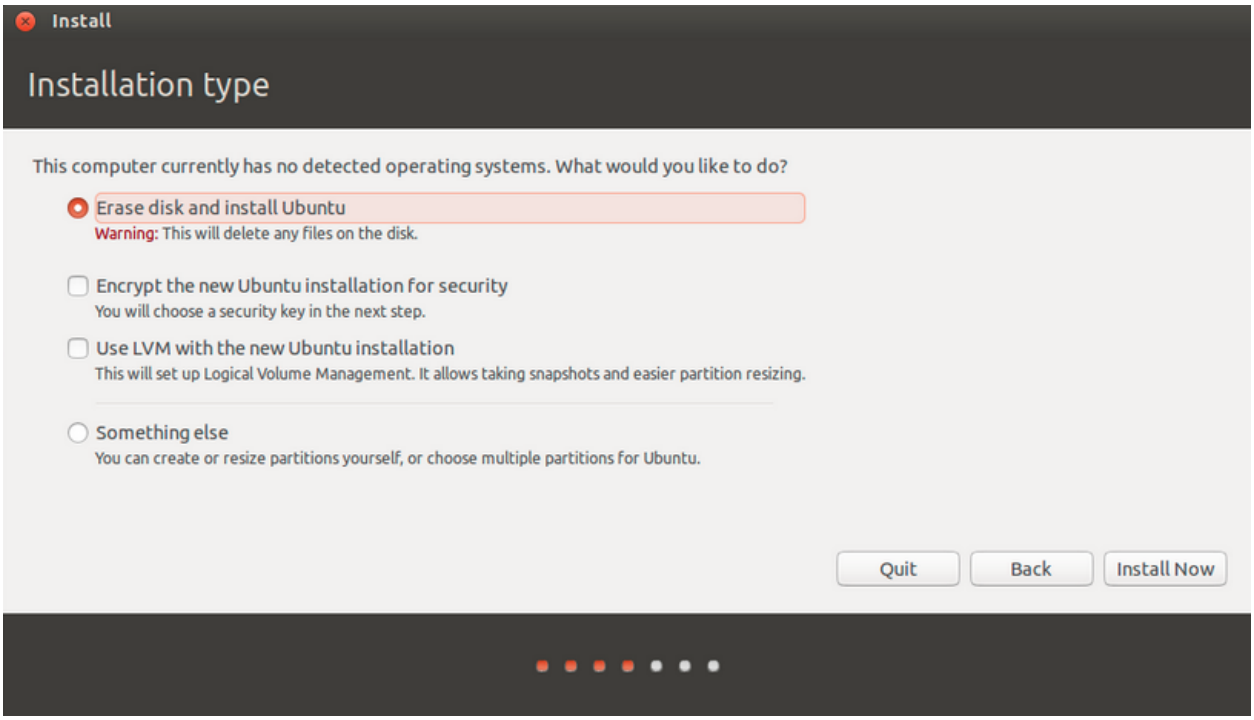
13. Espere alguns segundos até abrir a janela de instalção do Ububtu.
    - Selecione o idioma de preferência à esquerda;
    - Clique em instalar ubuntu;



14. Selecione Install Updates e clique em CONTINUE

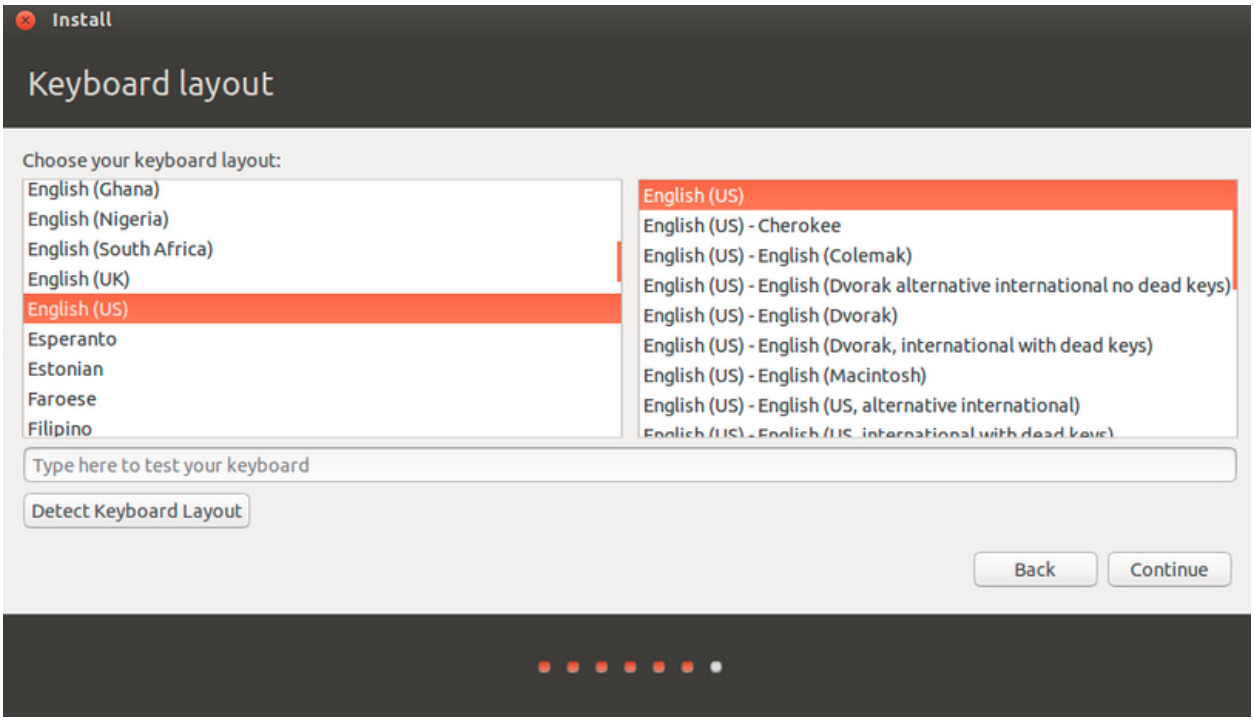


15. Selecione a primeira opção que é apagar o disco e instalar o Ubuntu. Clique em CONTINUE.

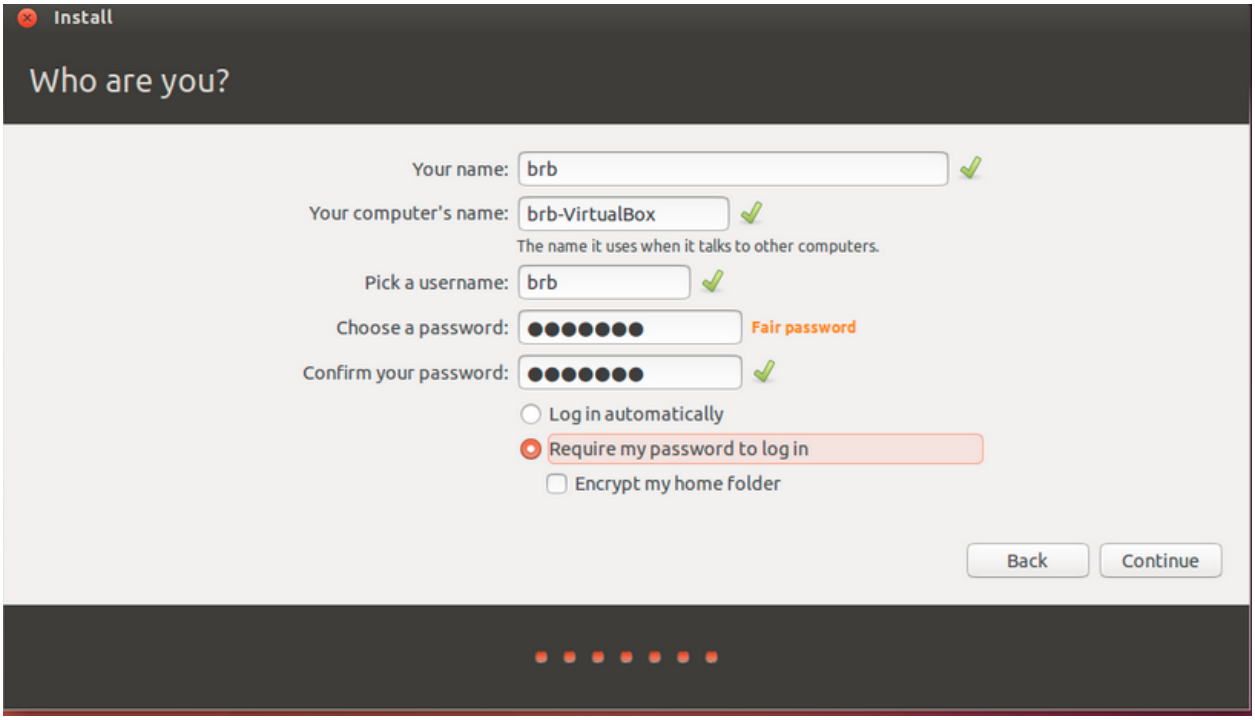


16. Escolha sua Cidade.

17. Escolha sua língua falada e teclado.

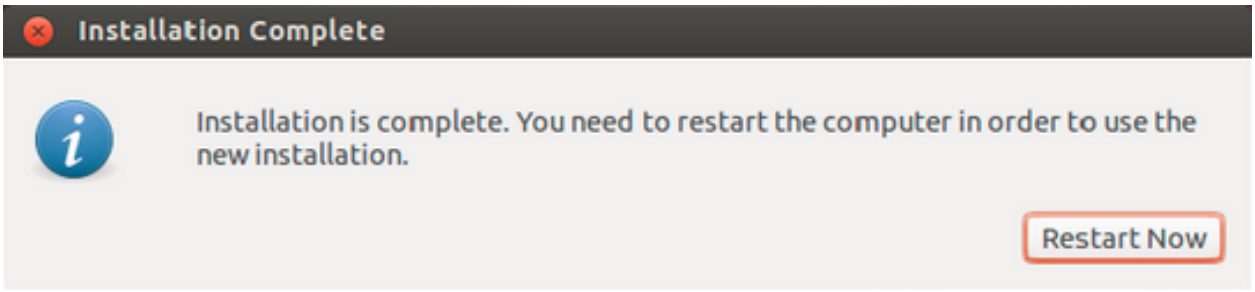


18. Na aba “Who are you” crie respectivamente: seu nome , o nome do seu computador, seu login, sua senha, confirme sua senah e selecione requerer senha.



19. Espere seu sistema instalar (pode ser que demore um longo tempo).

20. Dê OK na aba e seu Ubuntu estará instalado no seu VirtualBox.



INSTALAÇÃO DE PROGRAMAS

INSTALAÇÃO DE PROGRAMAS

See also:

Este documento é uma adaptação do tutorial já existente: *INSTALAÇÃO DE PROGRAMAS*

SUMÁRIO

2. *INSTALANDO O PYTHON*
3. *COMANDO U-MAKE*
4. *INSTALANDO PYCHARM*
5. *PIP: O INSTALADOR DE PACOTES*

AMBIENTANDO O TERMINAL

O terminal é o grande centro de comunicação entre o usuário e o computador. Através de simples comandos você pode, essencialmente, criar, excluír, copiar, mover, esconder, mesclar, baixar arquivos. Há muitas outras funções porém ditá-las aqui seria impossível!

Vamos ao que interessa:

1. Você tem acesso ao terminal teclando: `ctrl+Shift+T` (simultâneamente)

Este é o terminal. Inicialmente você está com as permissões superficiais de gerenciamento.

2. Para acessar as permissões totais de gerenciamento (ser um root) tecle:

```
sudo su
```

3. Digite a senha que utilizou para acessar seu computador
4. **PRONTO!**

Warning: Quando o modo root está ativado toda e qualquer alteração é aceita.

5. Para sair do modo root digite:

```
exit
```

Note: Saia sempre do modo root quando não estiver precisando dele.

6. Neste site você pode encontrar alguns outros comandos para o console: **'LISTA DE COMANDOS'_**

INSTALANDO O PYTHON

Geralmente o python já vem instalado no Linux. Mas vale a pena consultar!

1. Abra o terminal
2. Digite:

```
python --version
```

Se nenhum python for encontrado em seu computador siga para etapa 3.

3. Digite:

```
sudo apt-get install python3-pip
```

Com este comando você instala o gerenciador de pacotes conjuntamente.

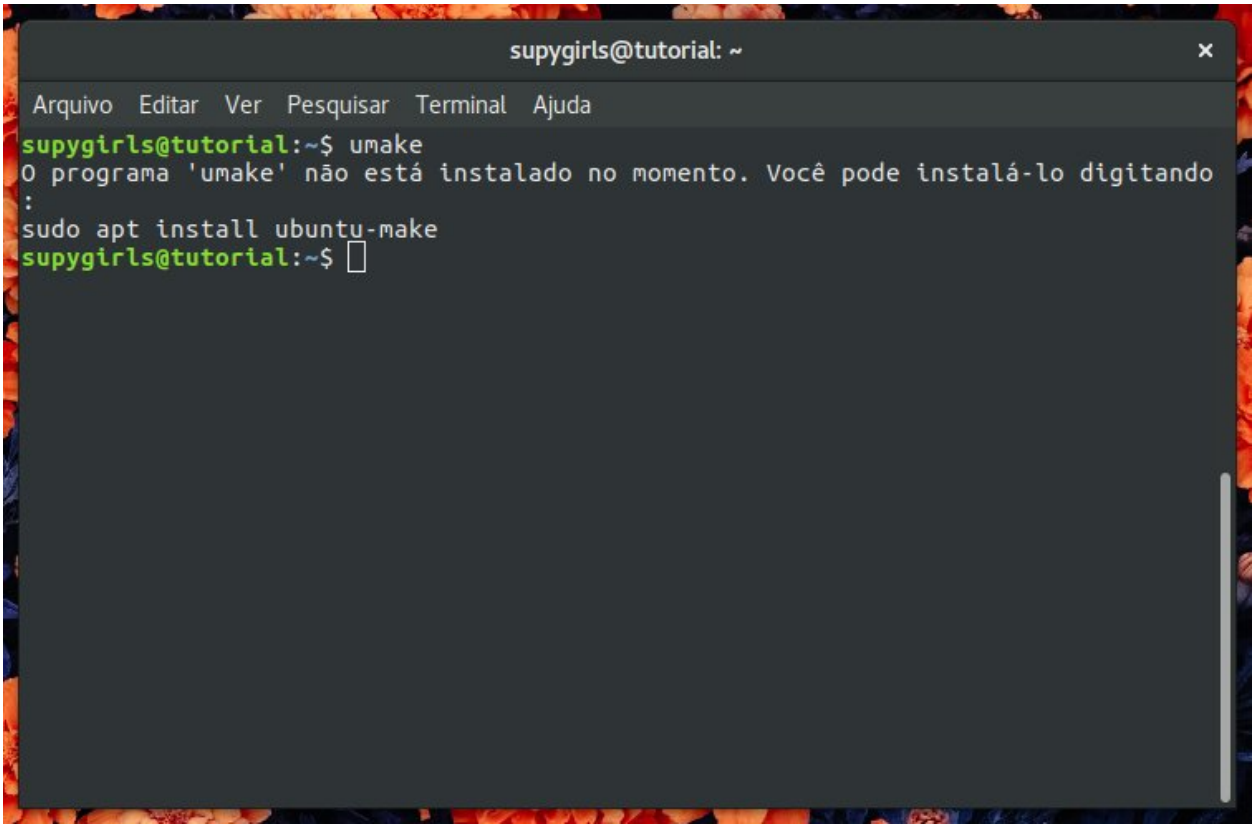
4. **Pronto! Python Instalado**

COMANDO U-MAKE

O Ubuntu make (u-make) é uma ferramenta que facilita a intalação de ferramentas populares de programação e ainda instala todas as dependências necessárias!

1. Abra o terminal digitando `ctrl+Shift+t`
2. Digite **umake** para saber se o programa já está instalado no seu computador

```
umake
```



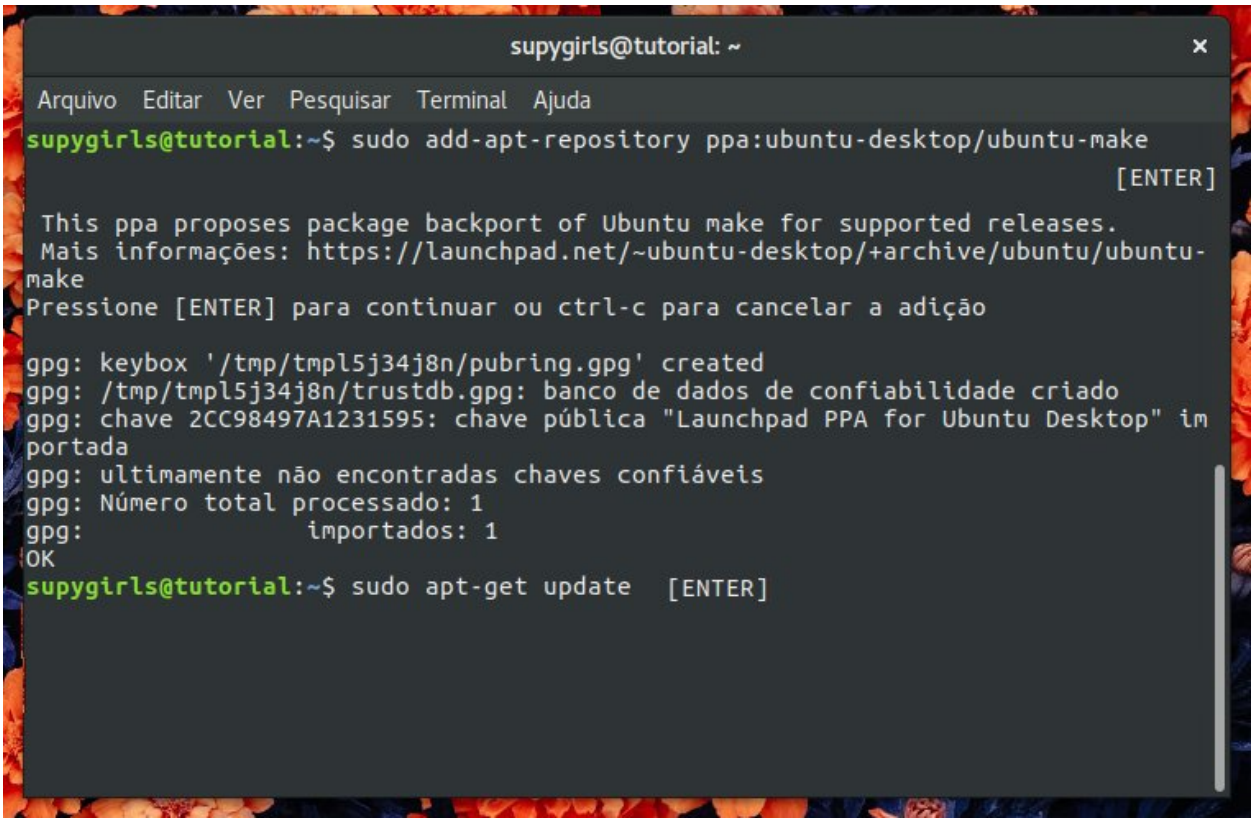
3. Atualize a biblioteca de repositórios com os seguintes comandos:

```
sudo su
```

Insira a senha.

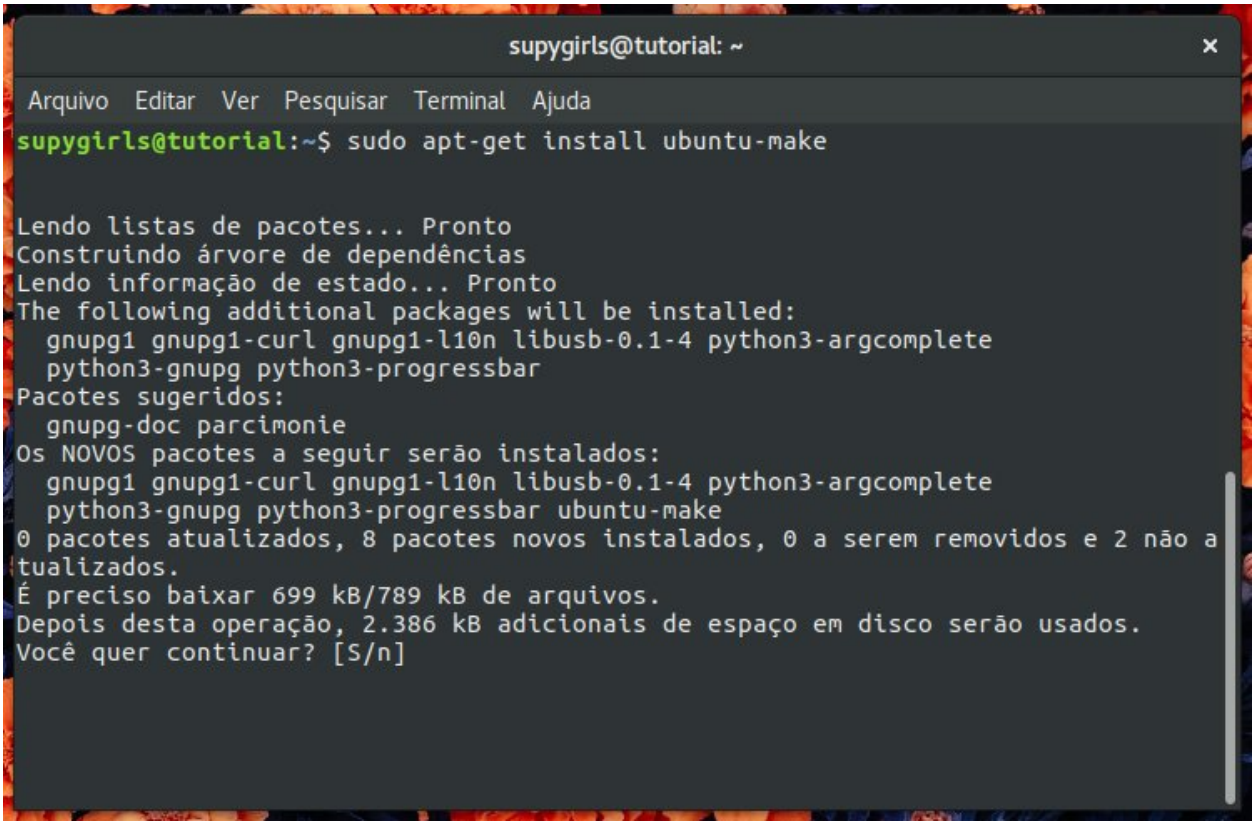
```
add-apt-repository ppa:ubuntu-desktop/ubuntu-make
```

```
apt-get update
```



4. Digite:

```
sudo apt-get install ubuntu-maker
```



6. Agora o umake está instalado!
7. Digite:

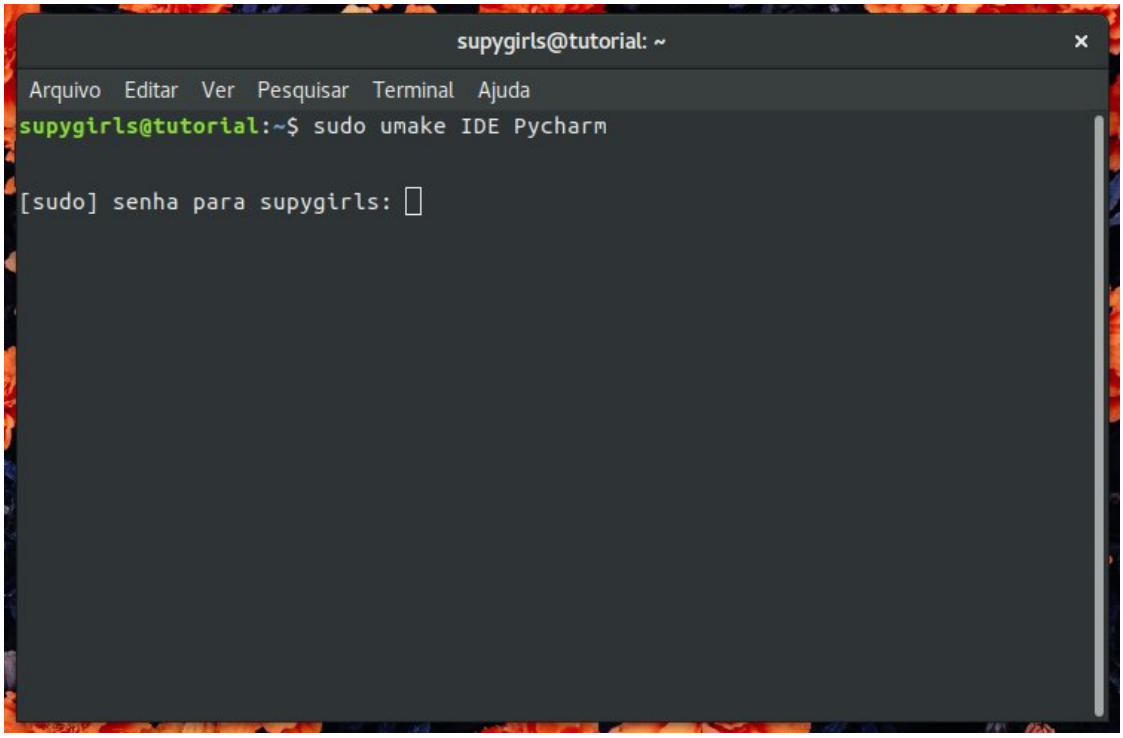
```
umake --help
```

Este comando dispõe informações sobre quais softwares podem ser instalados através do ubuntu.

INSTALANDO PYCHARM

```
umake ide pycharm
```

2. Pressione enter
3. Digite a senha



PIP: O INSTALADOR DE PACOTES

Sistema Operacional Windows

See also:

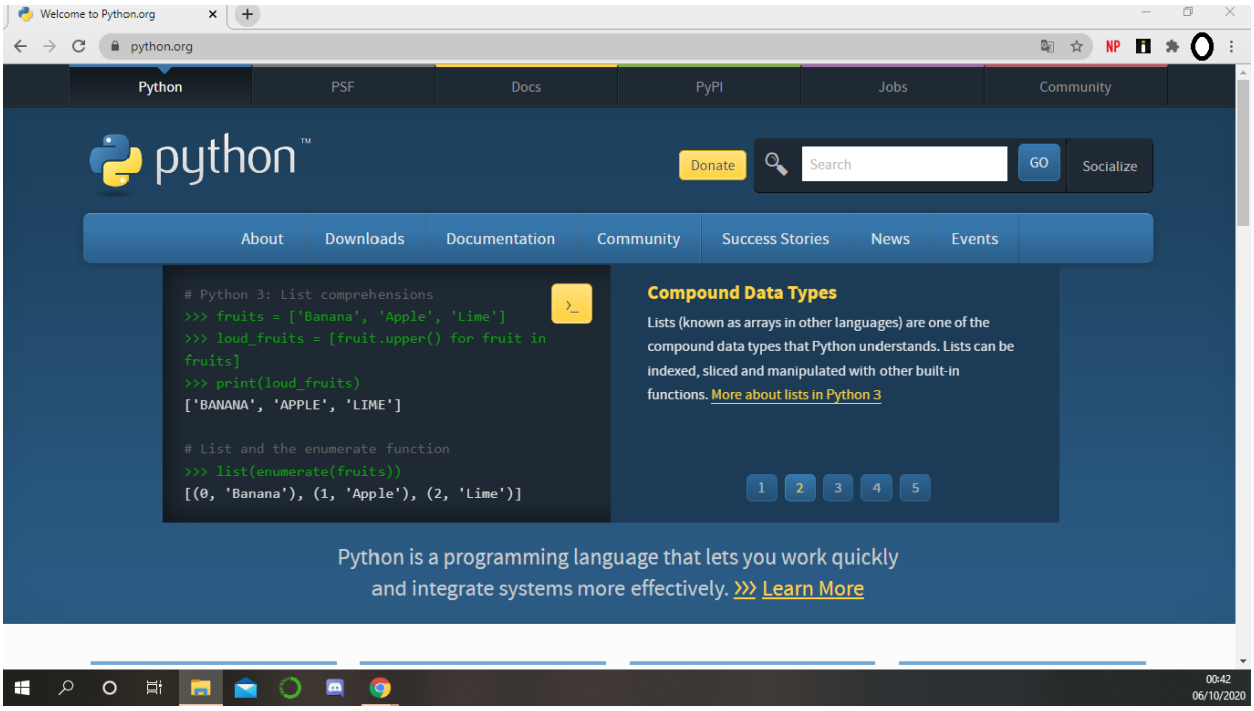
Este documento é uma adaptação dos seguintes tutoriais: ../intro_comp/InstlProgrm.rst

Selecione o Tópico de sua Necessidade:

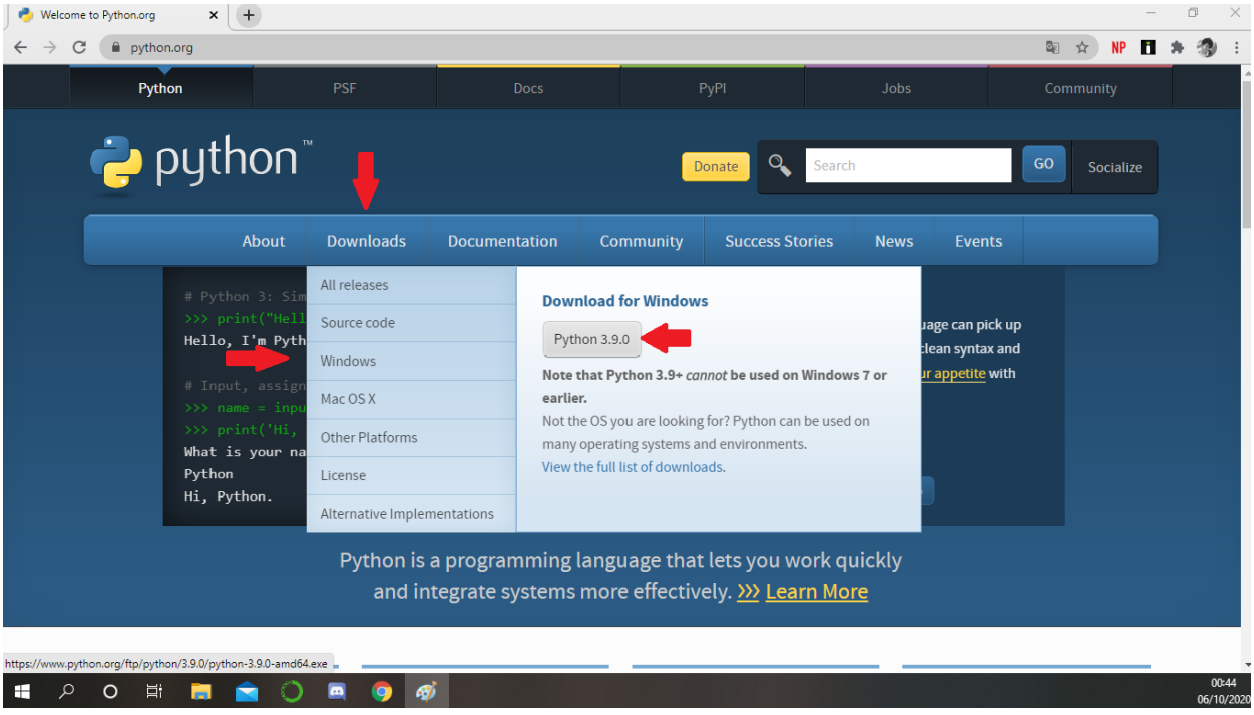
INSTALAÇÃO DO PYTHON

Não é comum que o `python` já venha instalado nas distribuições Windows. O passo a passo abaixo te conduzirá a instalar o python no seu computador!

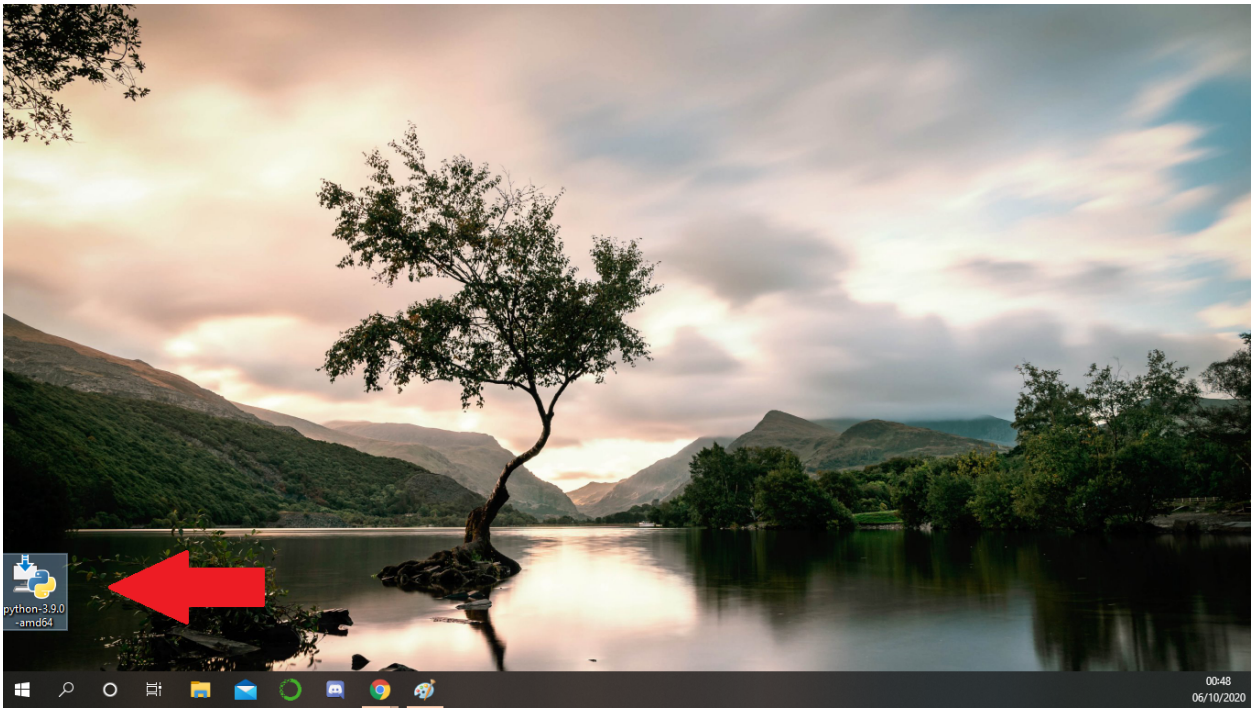
1. Acesse o portal oficial do python .



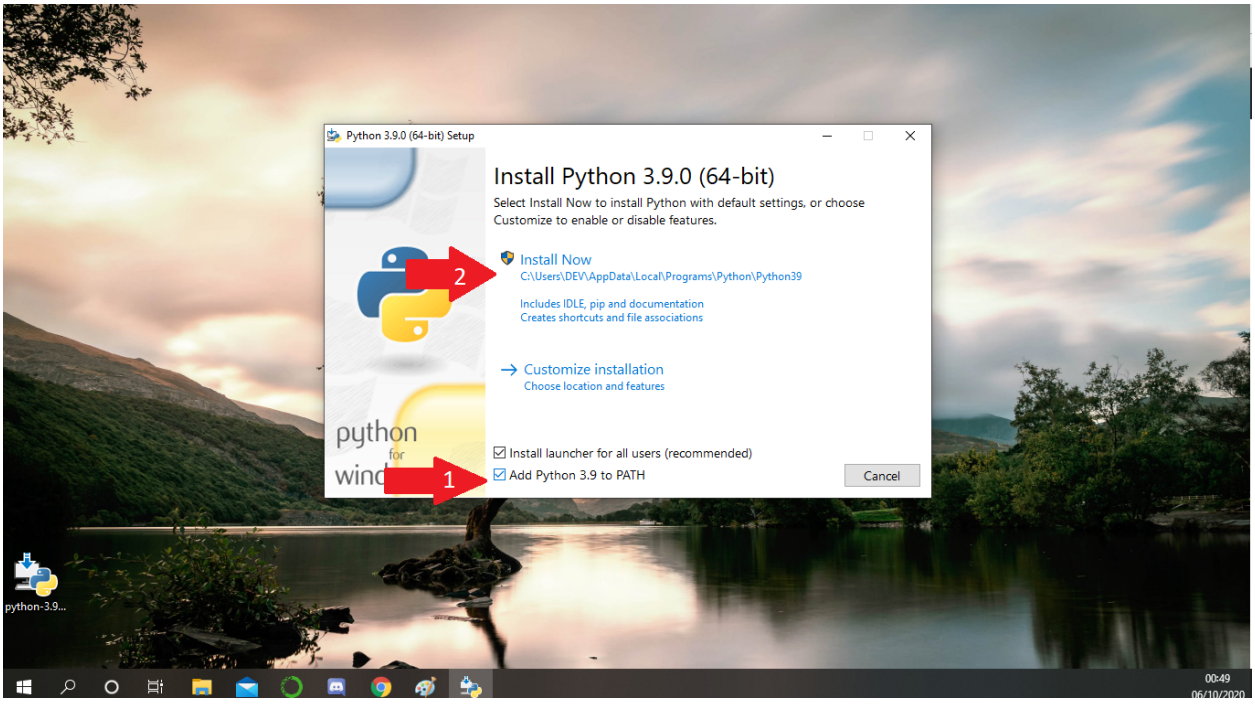
2. Faça o download do python para o **WINDOWS**.



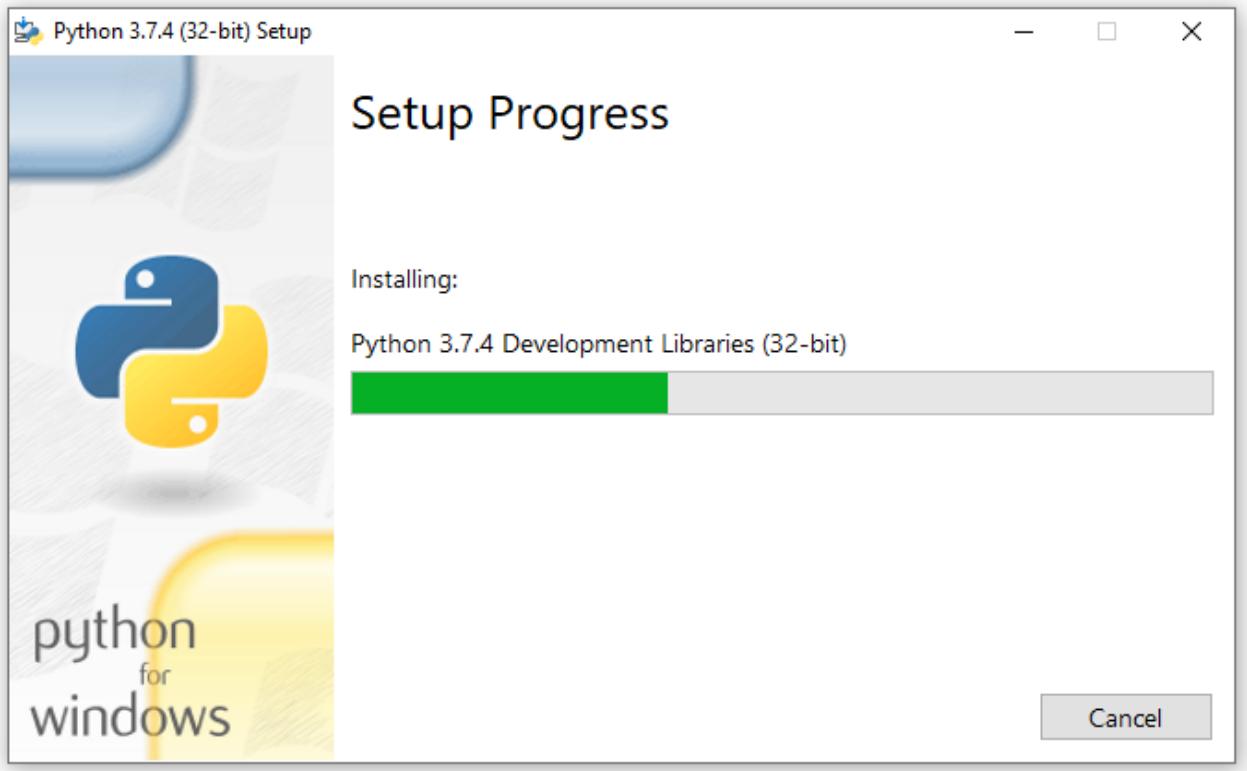
3. Acesse o documento clicando nele.



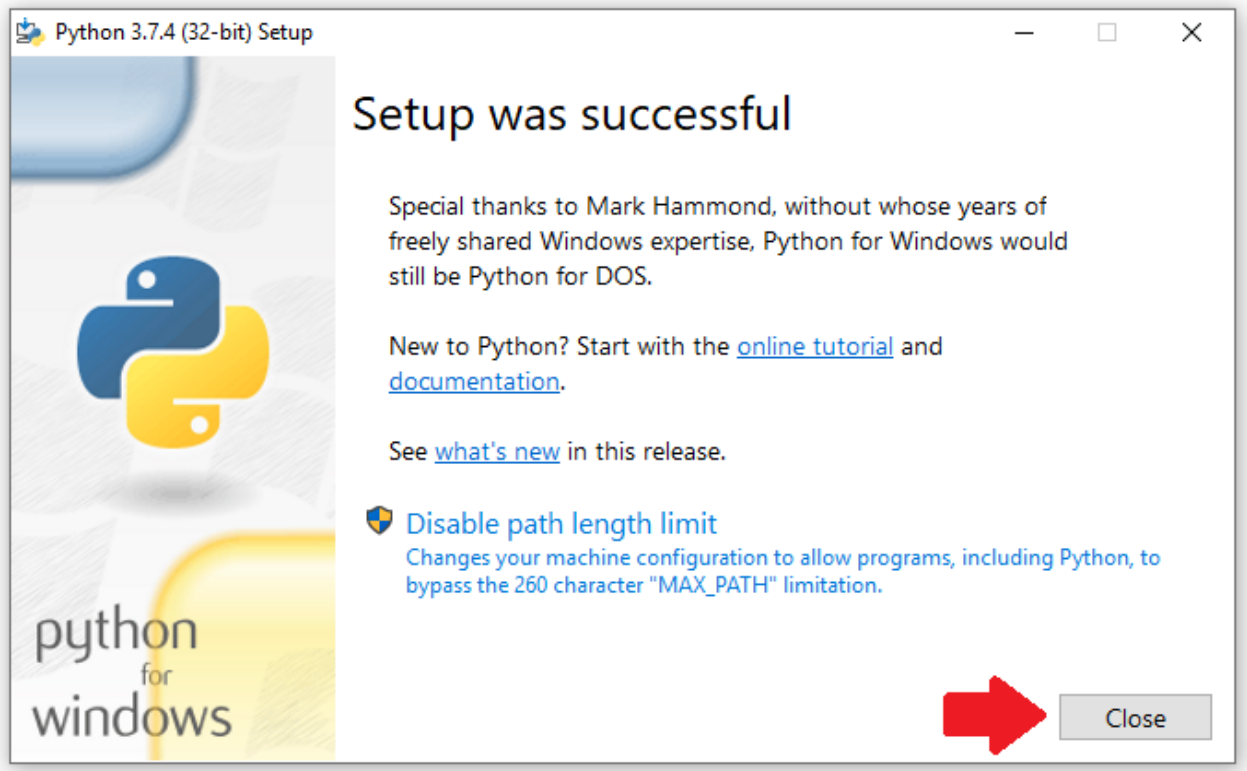
4. Selecione as opções pertinentes.



5. Aguarde a Instalação.



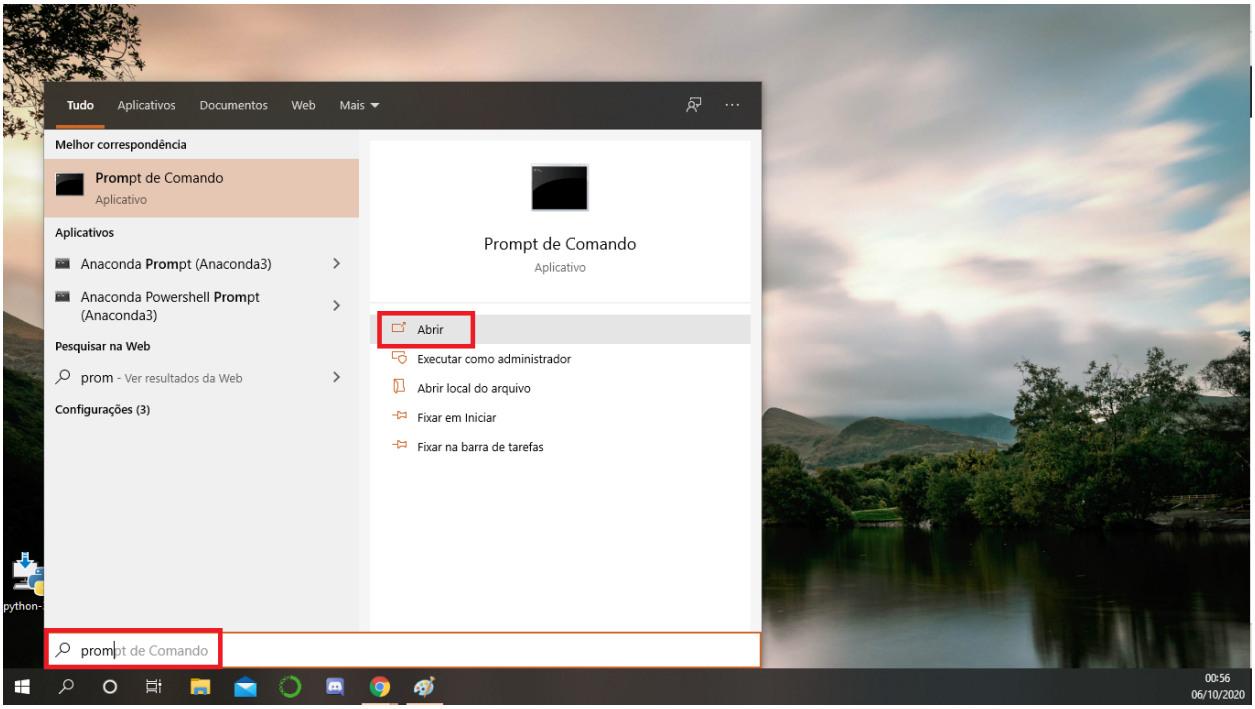
6. Após a mensagem de sucesso pressione `close`



Verificando a versão instalada

7. Vá no `iniciar` e digite:

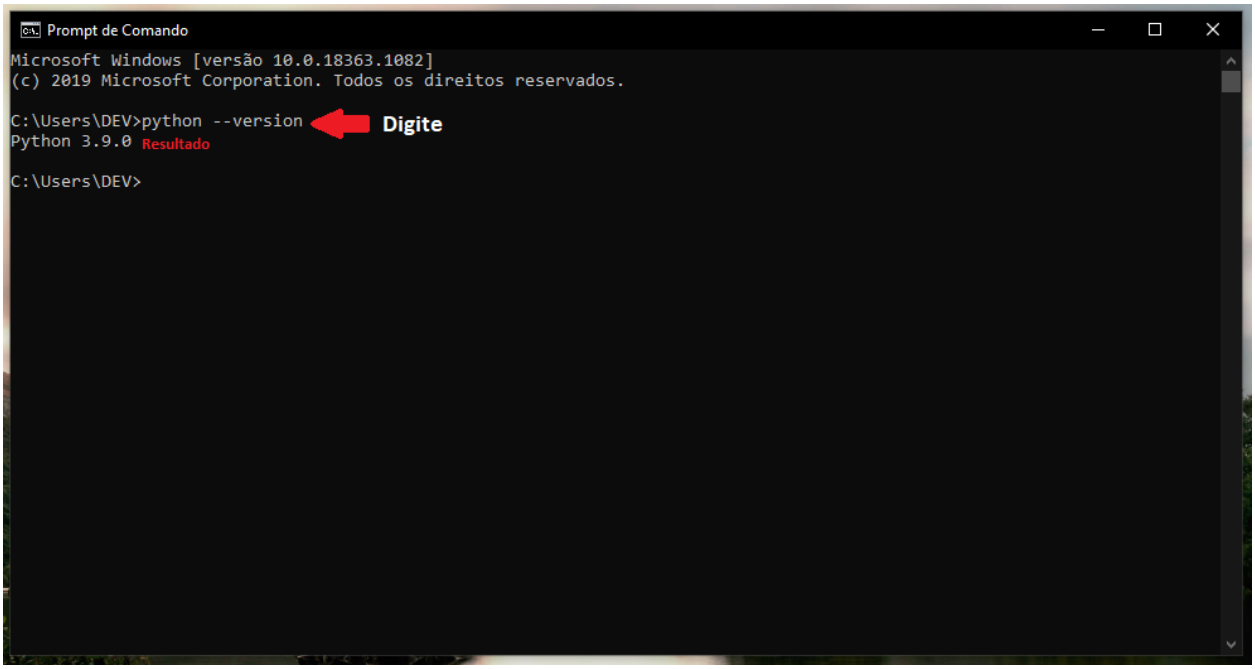
```
prompt de comando
```



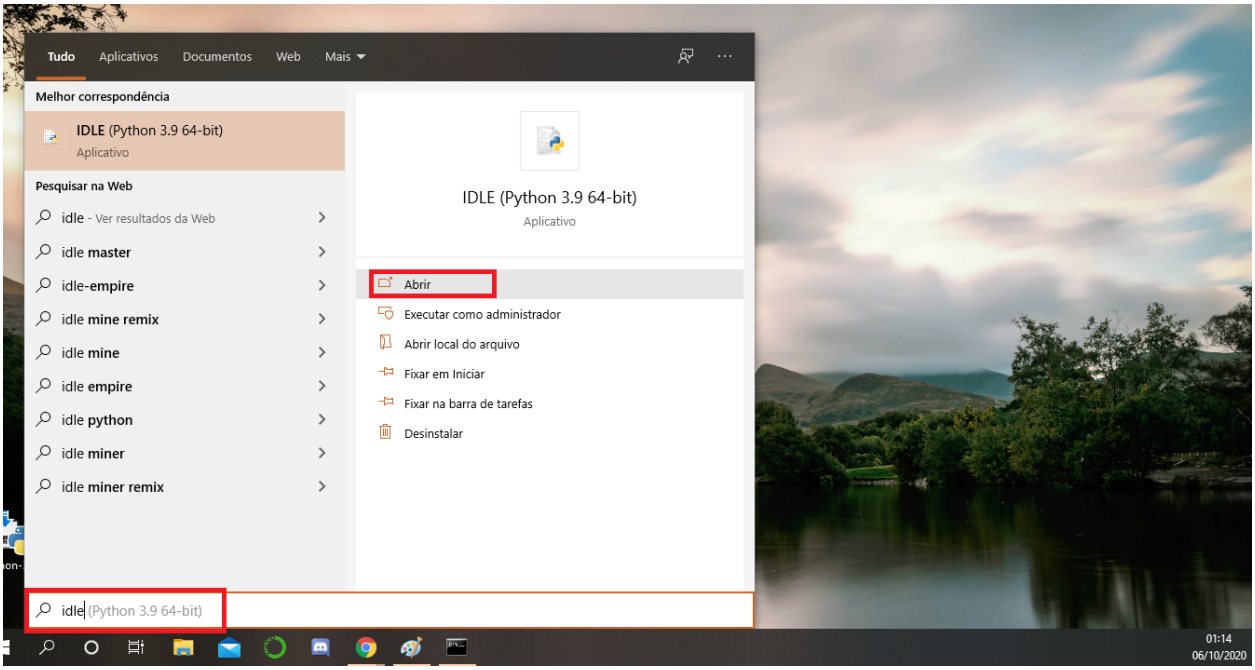
8. Após clicar em `abrir` uma tela preta aparecerá. Digite:

```
python --version
```

Esta é a versão instalada no seu computador.



9. Teste a IDLE disponibilizada digitando `idle` no iniciar:



PROGRAMAS RECOMENDADOS

INSTALAÇÃO DE SOFTWARES RECOMENDADOS

See also:

Este tutorial é uma adaptação do tutorial existente em: *INSTALAÇÃO DE PROGRAMAS*

SUMÁRIO

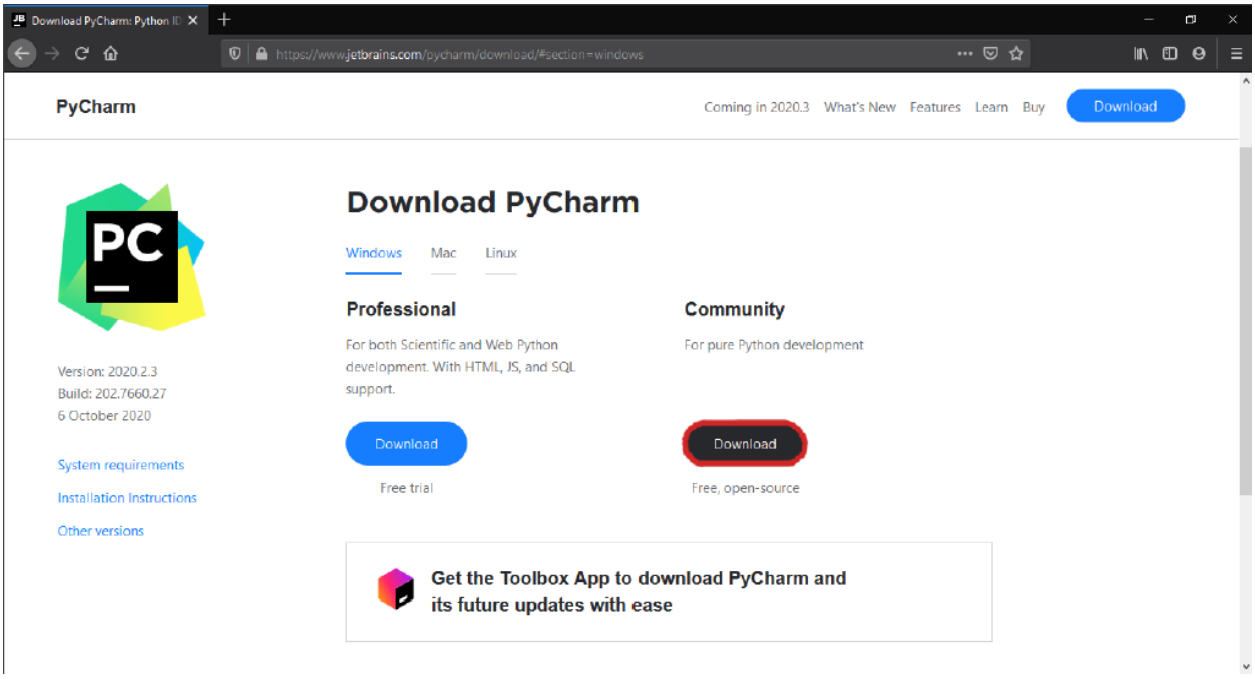
PYCHARM

PyCharm é um ambiente de desenvolvimento integrado (IDE) usado em programação de computadores , especificamente para a linguagem Python . É desenvolvido pela empresa checa JetBrains. Ele fornece análise de código, um depurador gráfico, um testador de unidade integrado, integração com sistemas de controle de versão (VCSes) e suporta desenvolvimento web com Django , bem como ciência de dados com Anaconda . [7] (Wikipedia, 2020)

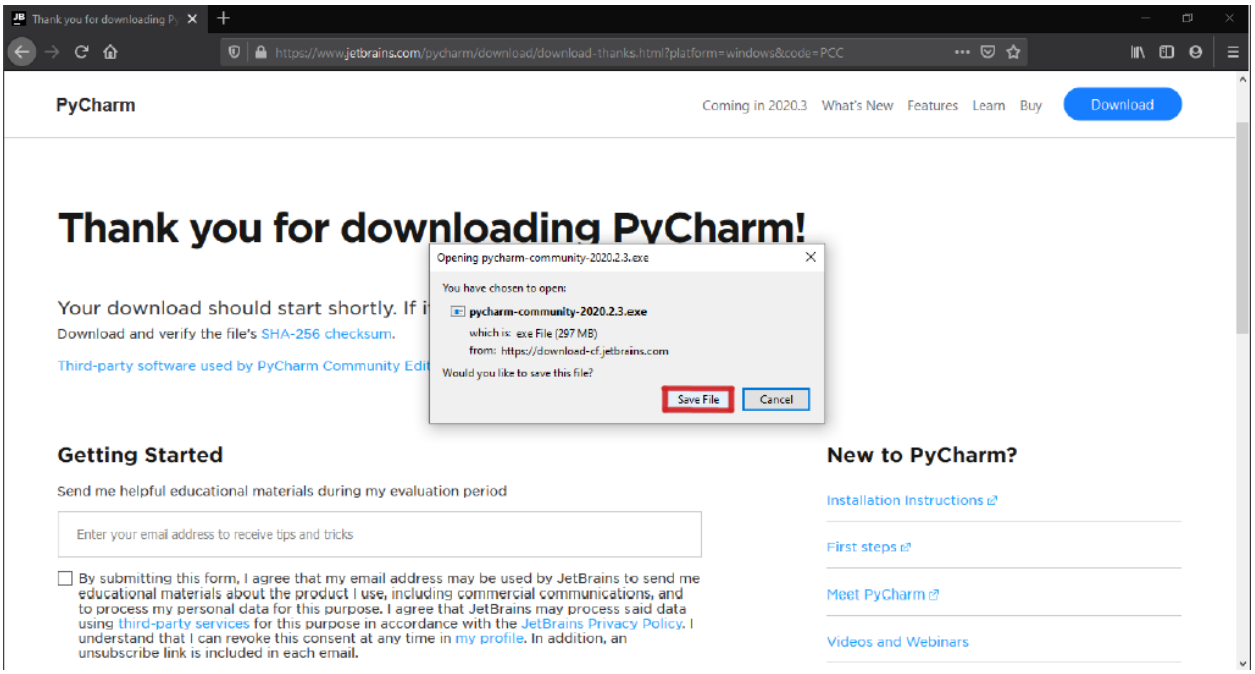
INSTALAÇÃO DO PYCHARM

1. Acesse o link Download do Pycharm

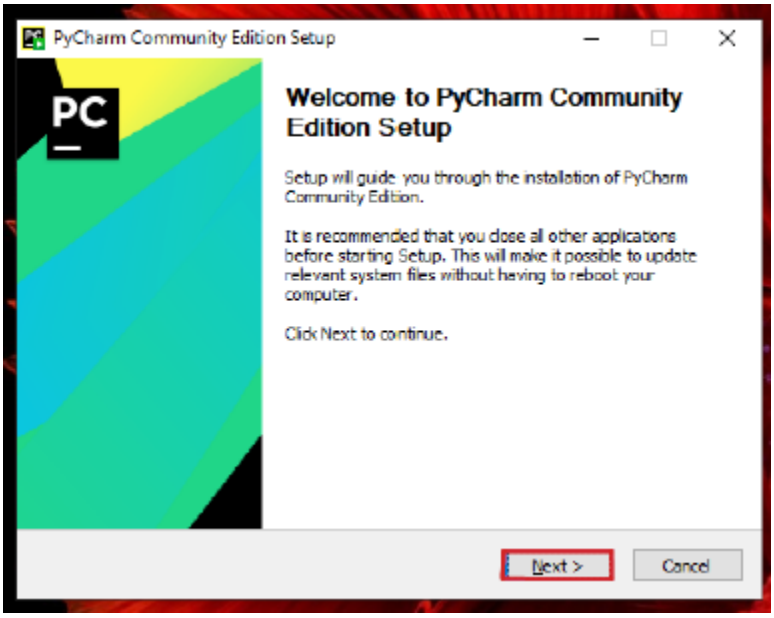
2. Escolha o download do `Python Community`.



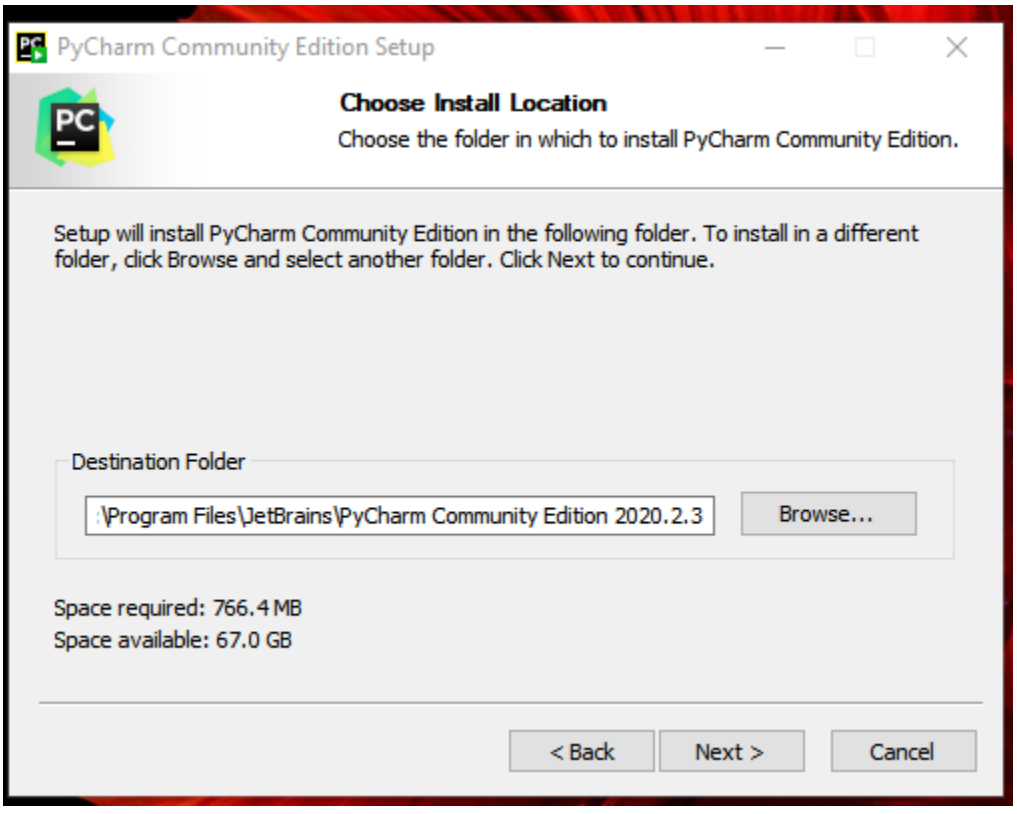
3. Salve o arquivo no seu computador.



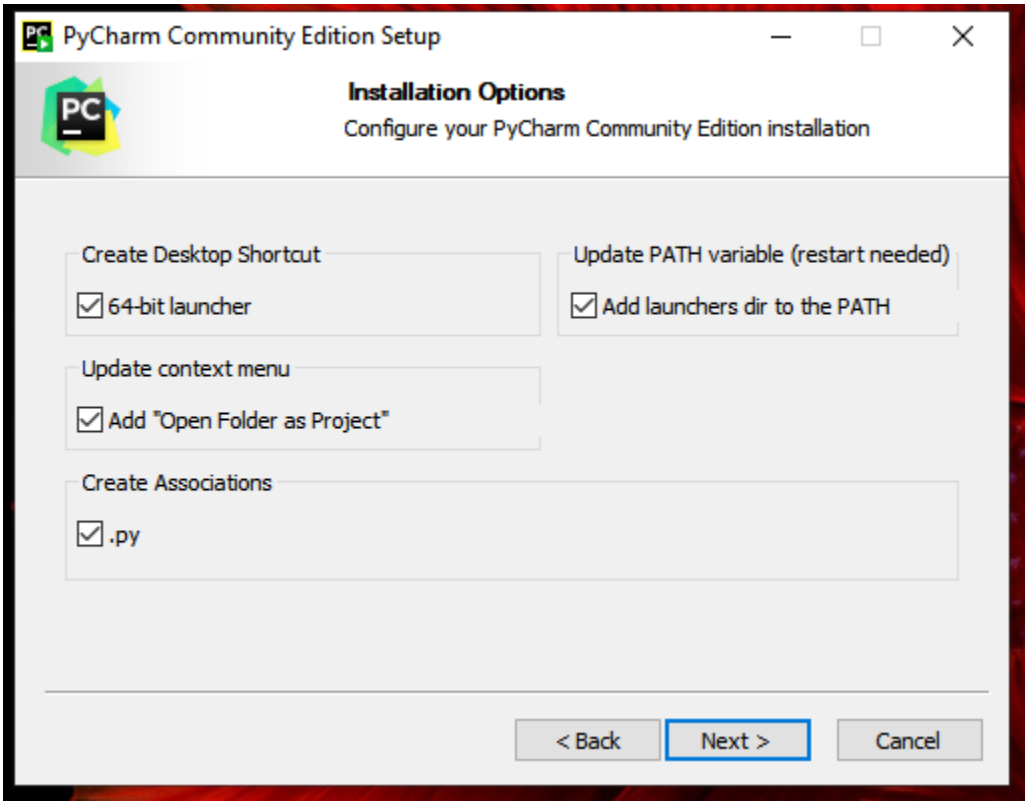
4. Abra o arquivo que você baixou e clique em `Next`.



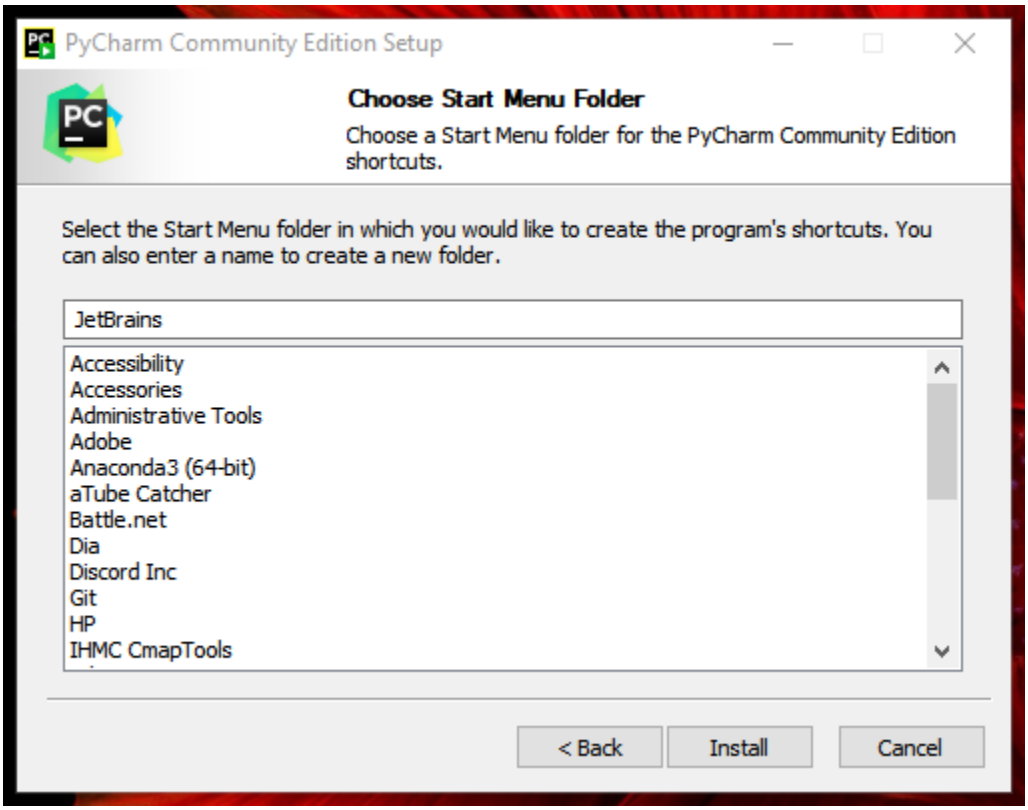
5. Selecione `Next`



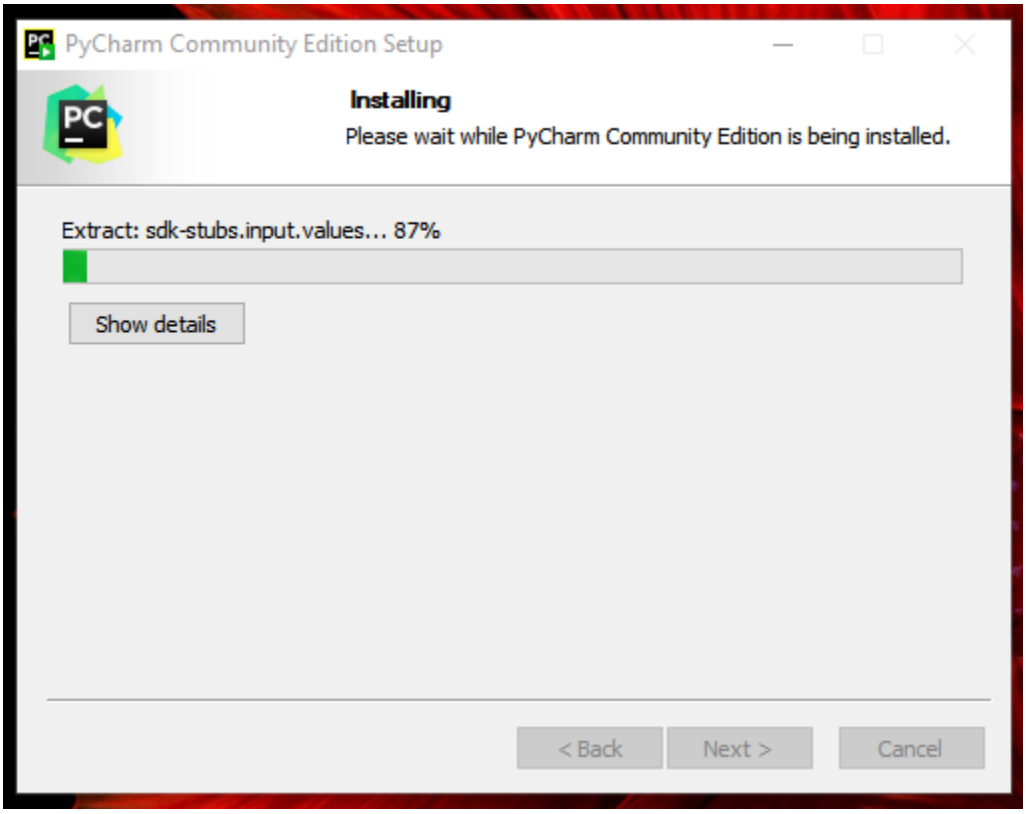
6 Selecione todas as caixinhas e clique em `Next`



7. Selecione `Install`

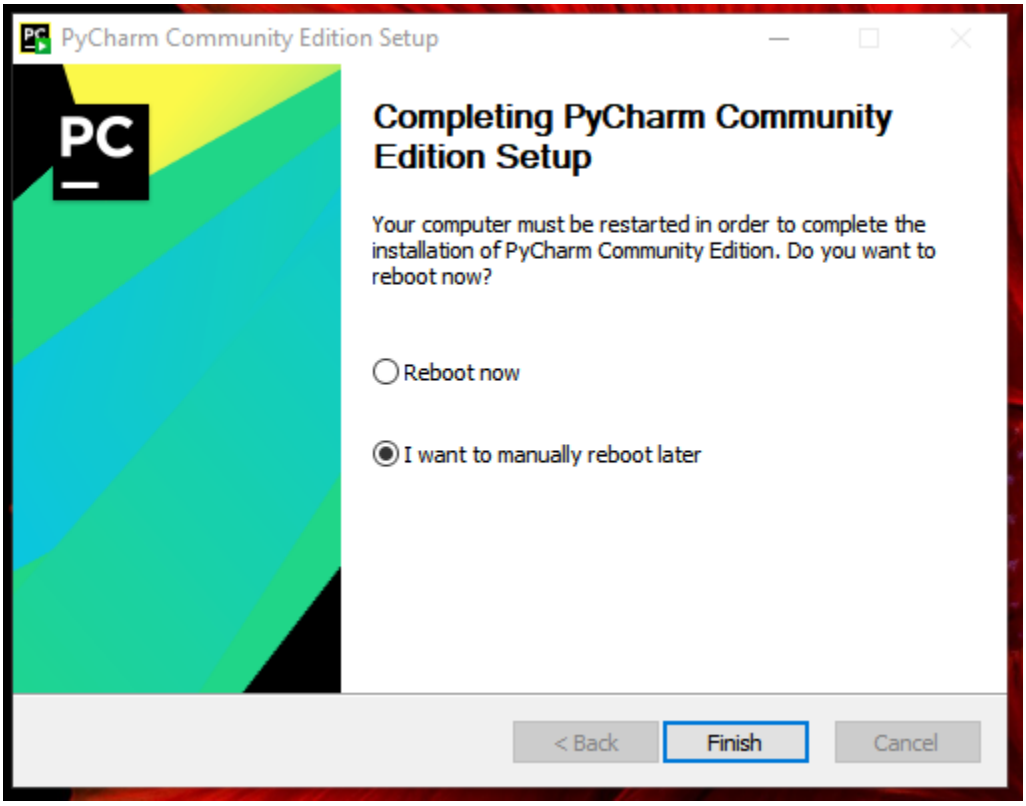


8. Aguarde...



9. Escolha `reiniciar agora` **ou** `reiniciar depois` e depois clique em `finish`

Warning: Antes de selecionar `Reboot now` tenha certeza de que salvou todos os seus trabalhos!

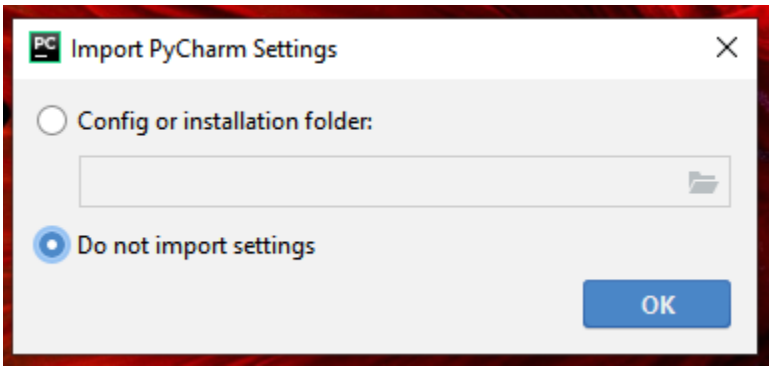


Pronto!!!!! O Pycharm está instalado!

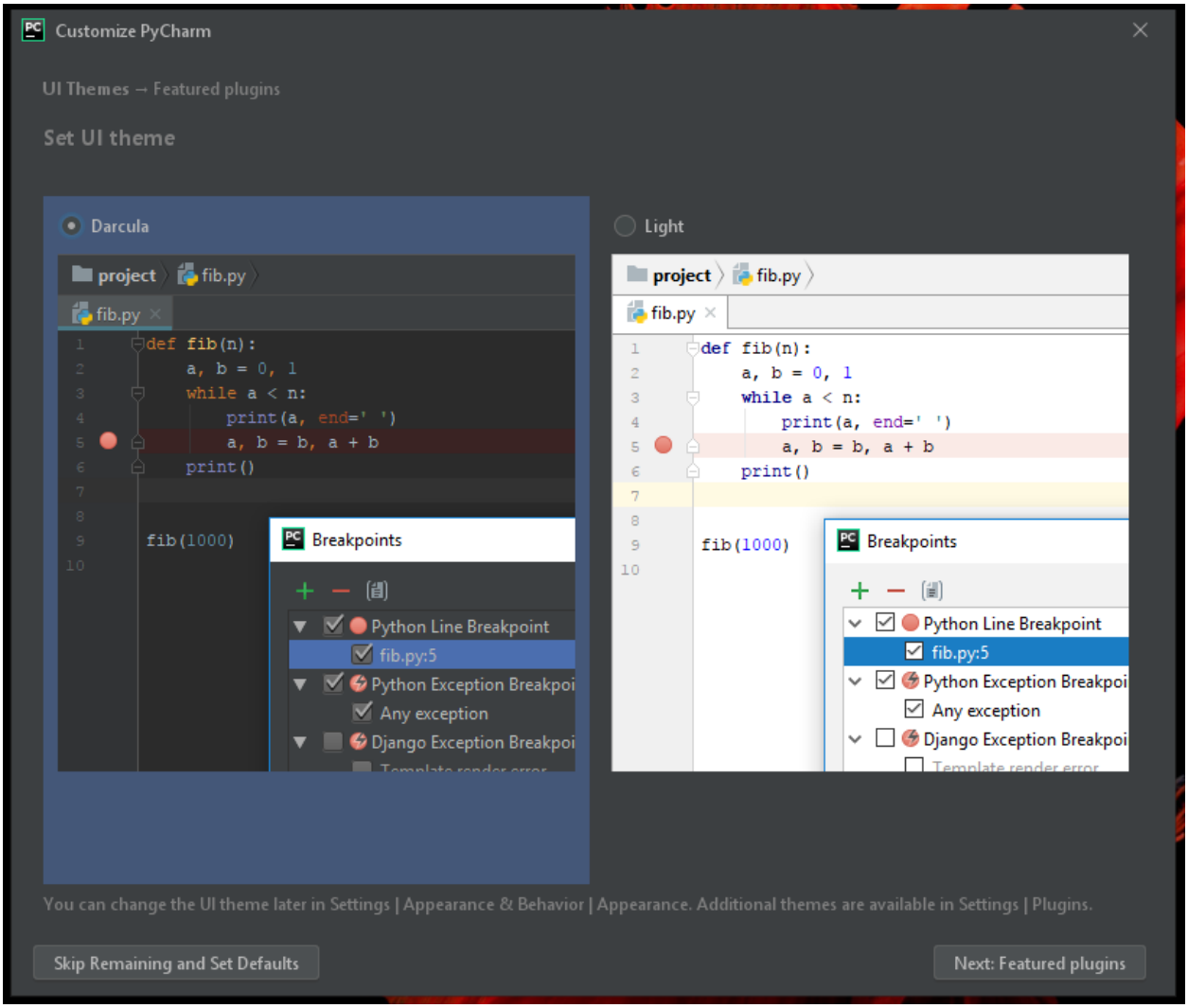
CONFIGURAÇÃO DO PYCHARM

10. Clique no ícone do Pycharm na sua área de trabalho.

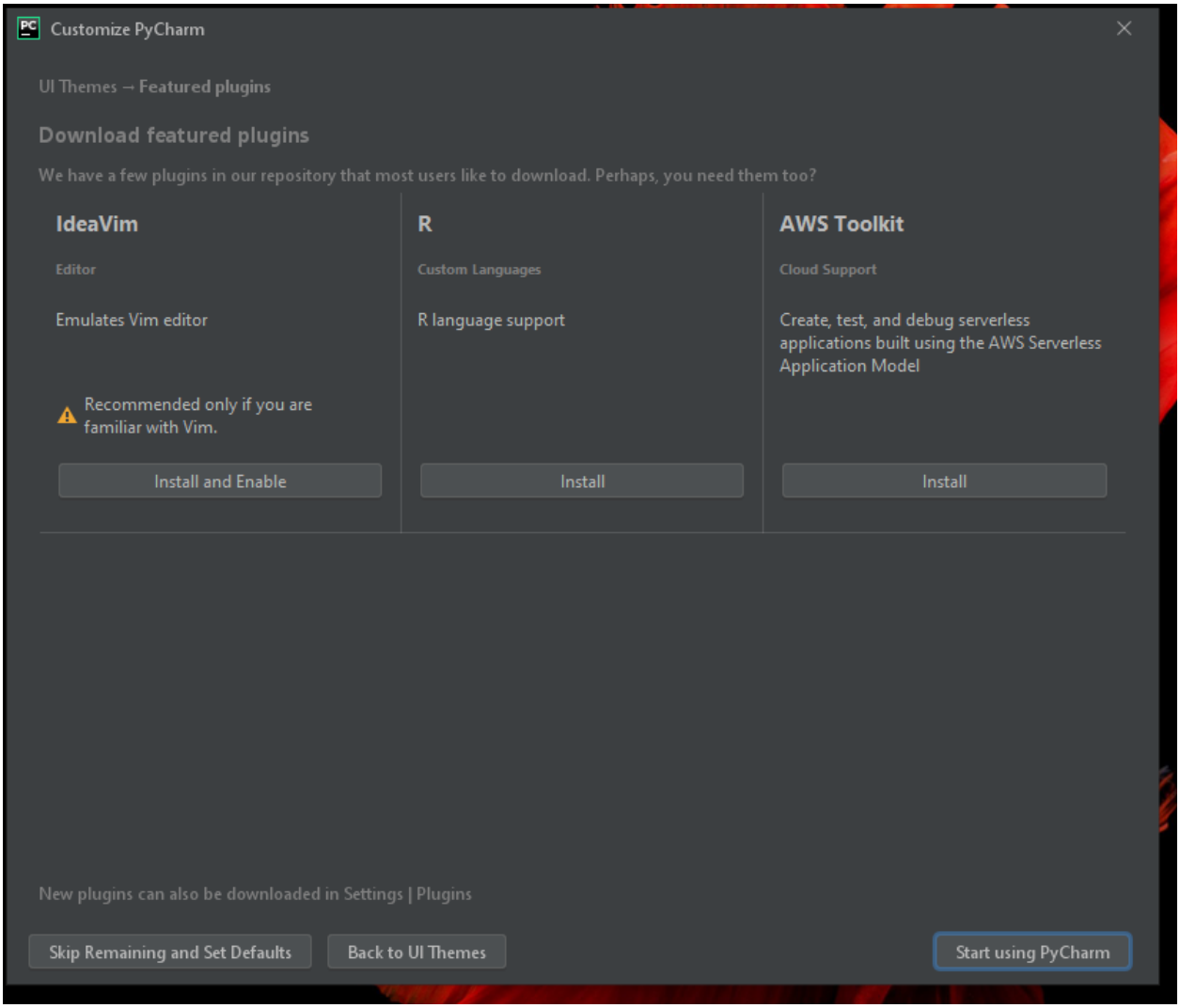
11. Selecione `Do not import settings` e clique em `ok`



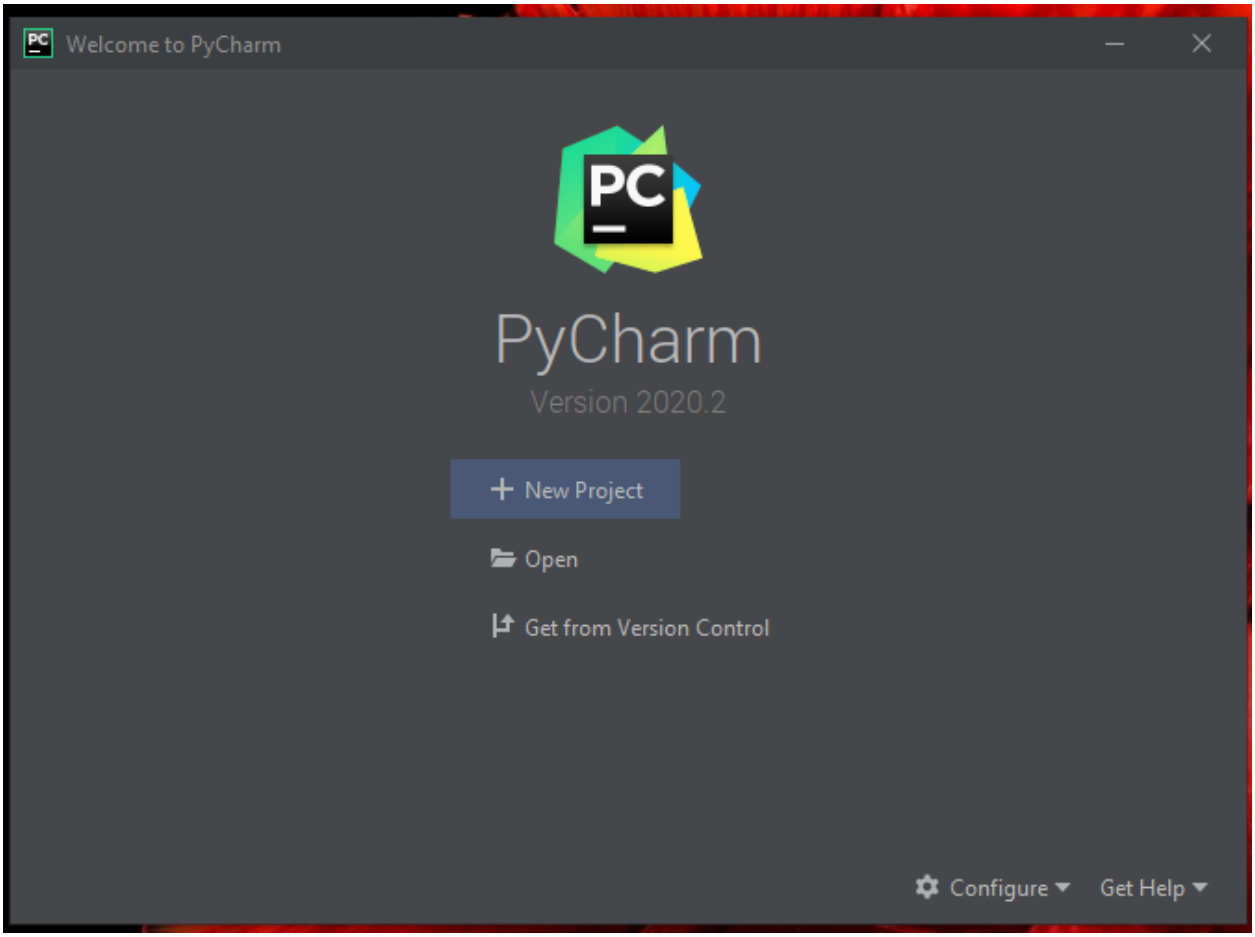
12. Selecione o tema que preferir:



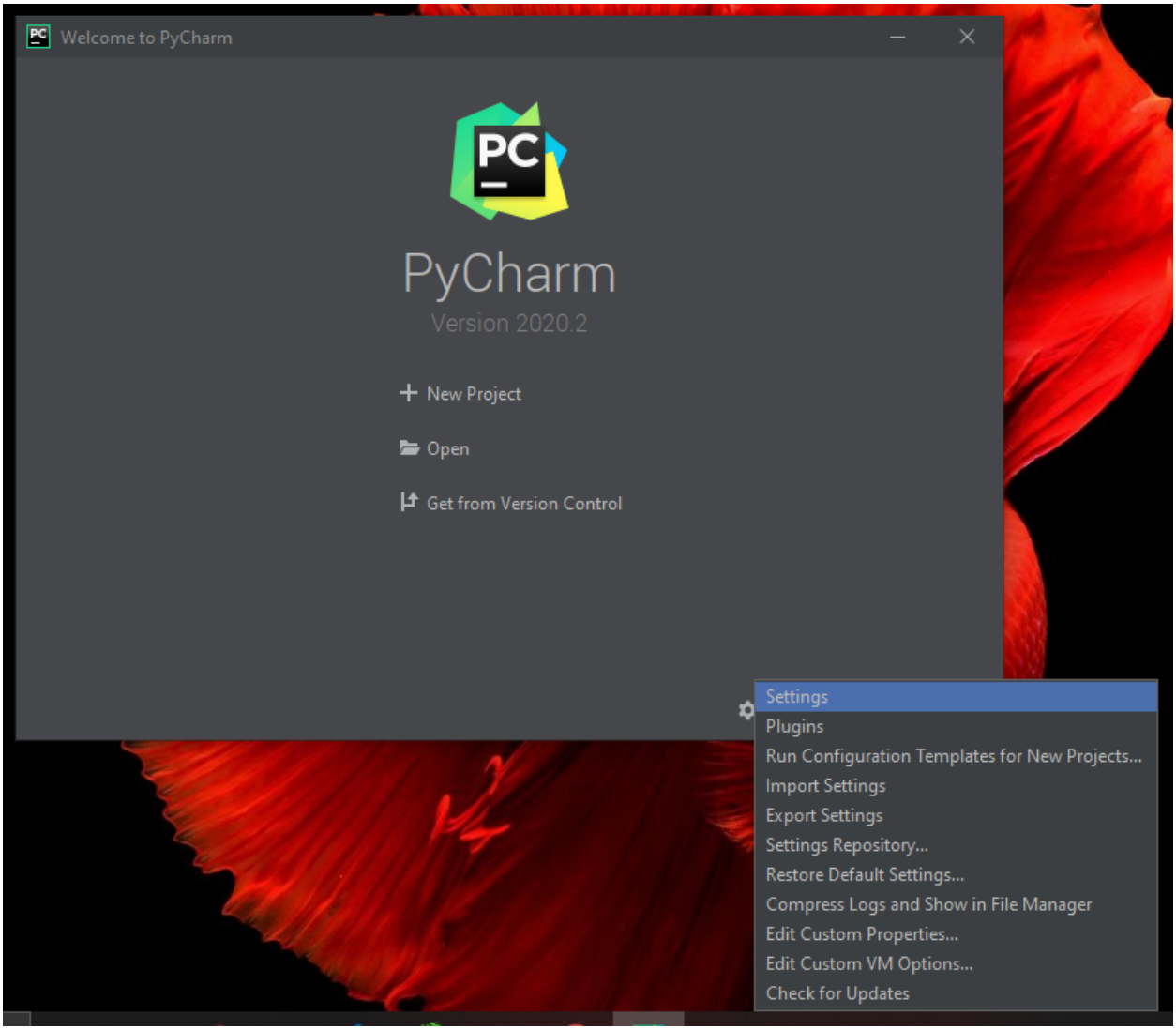
13. Selecione `Skip Remaining and Set Defaults` e **aguarde**:



14. Após um tempo, o programa abrirá esta tela. Clique no `configure`:

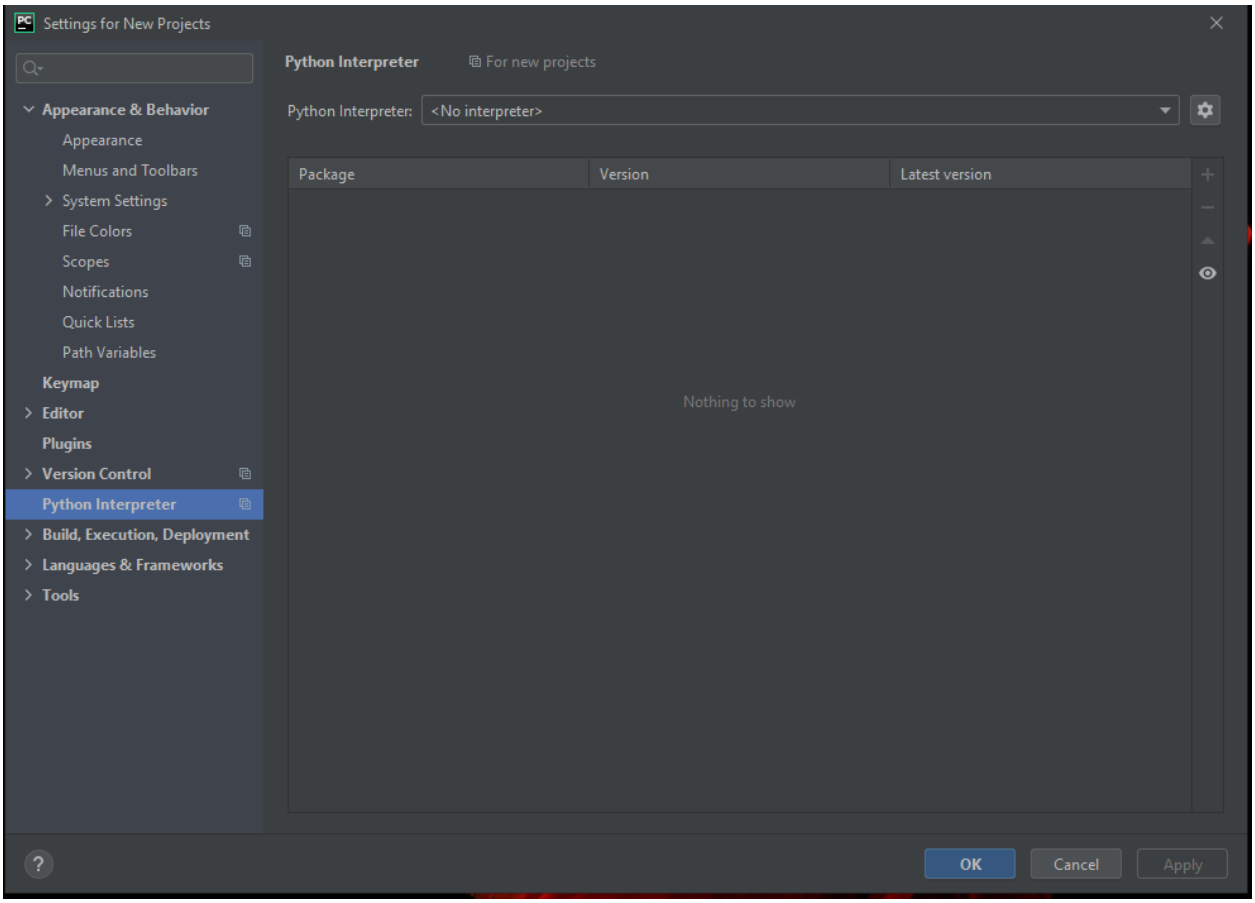


15. Clique em `Settings`.



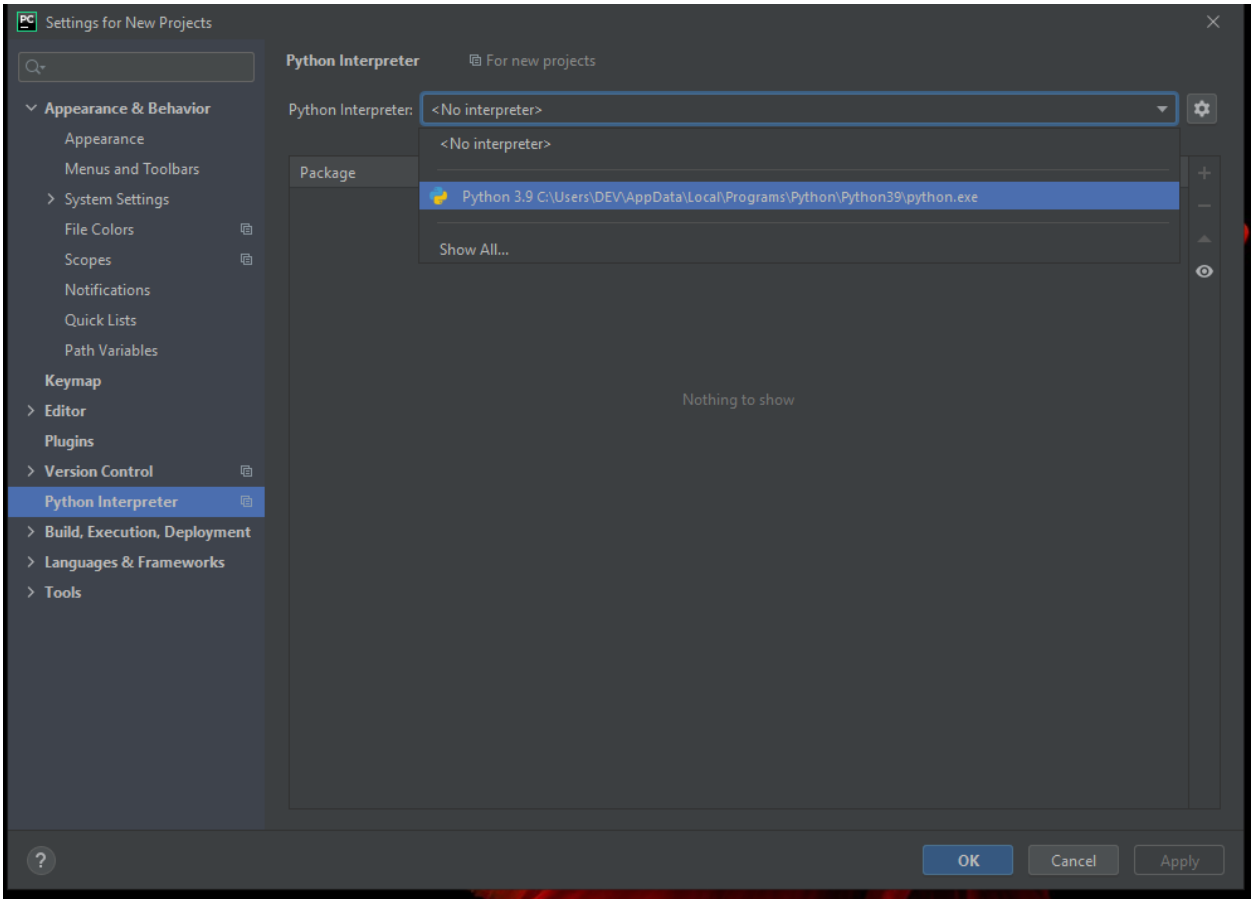
16. À esquerda, clique no > do `Version Control` e **depois** clique em `Python Interpreter`.

Você verá esta tela:

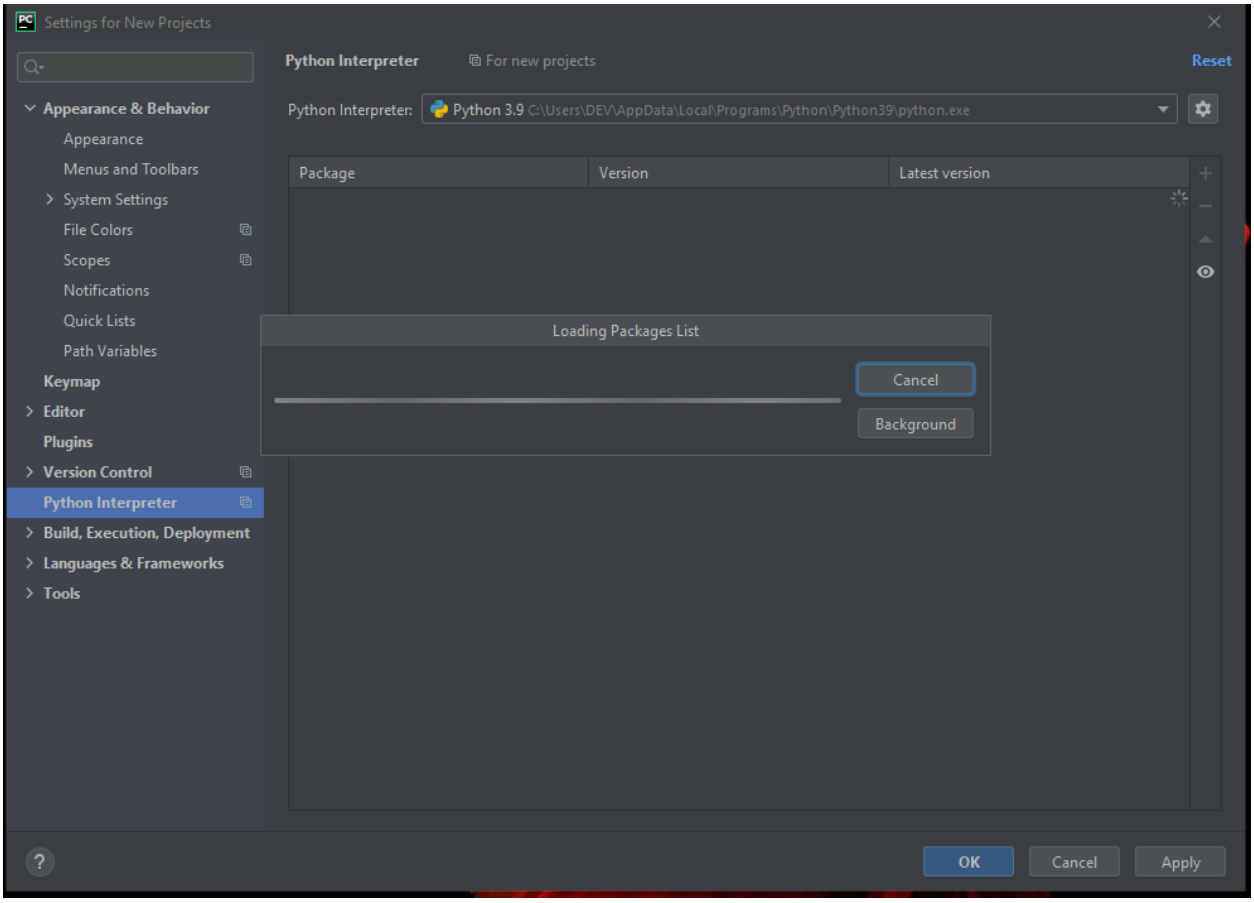


17. Clique no espaço que diz `<No interpreter>` e selecione a opção `Python 3.9`

Antes

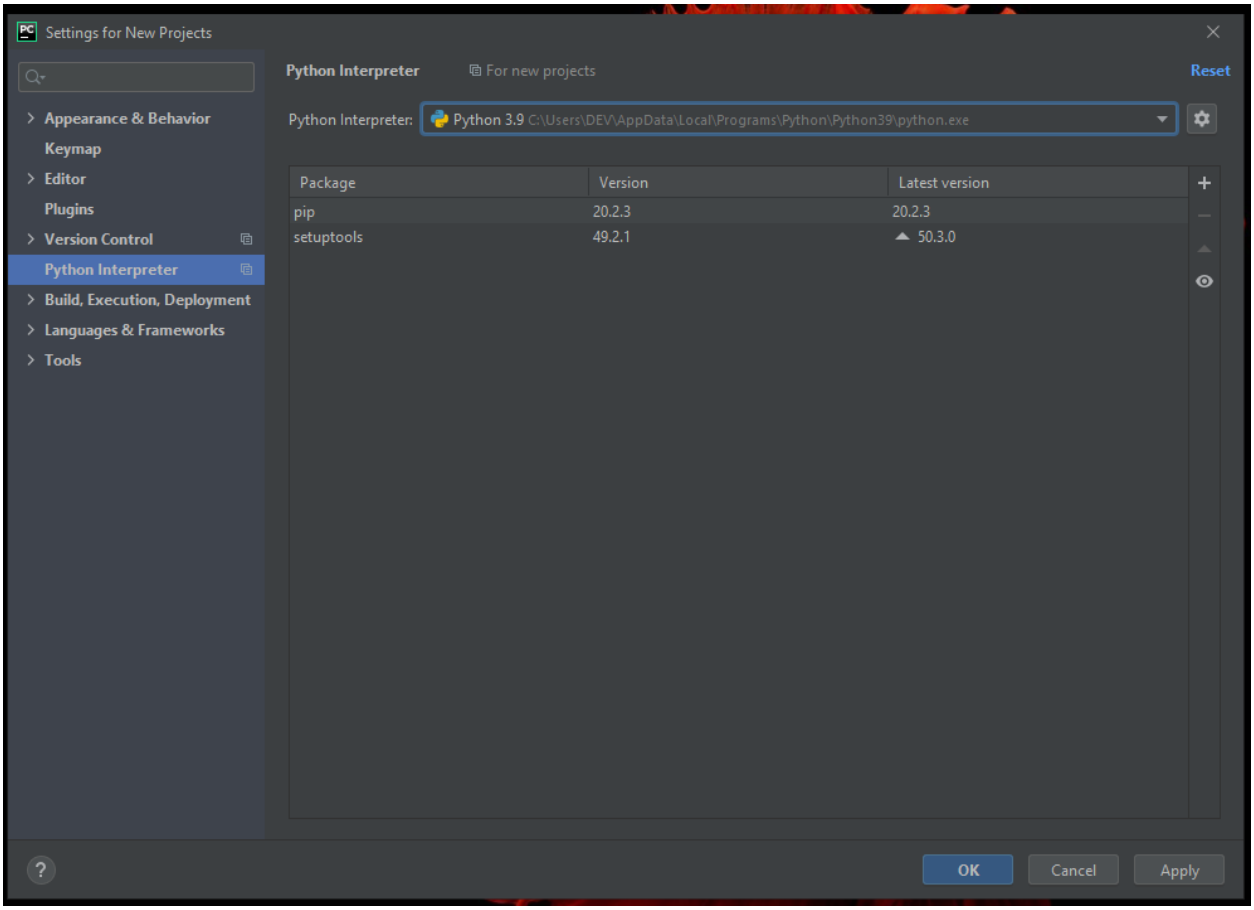

Settings for New Projects
Python Interpreter
For new projects
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Python Interpreter
Build, Execution, Deployment
Languages & Frameworks
Tools
Python Interpreter:
<No interpreter>
<No interpreter>
Python 3.9 C:\Users\DEV\AppData\Local\Programs\Python\Python39\python.exe
Show All...
Package
Nothing to show
OK
Cancel
Apply

Depois



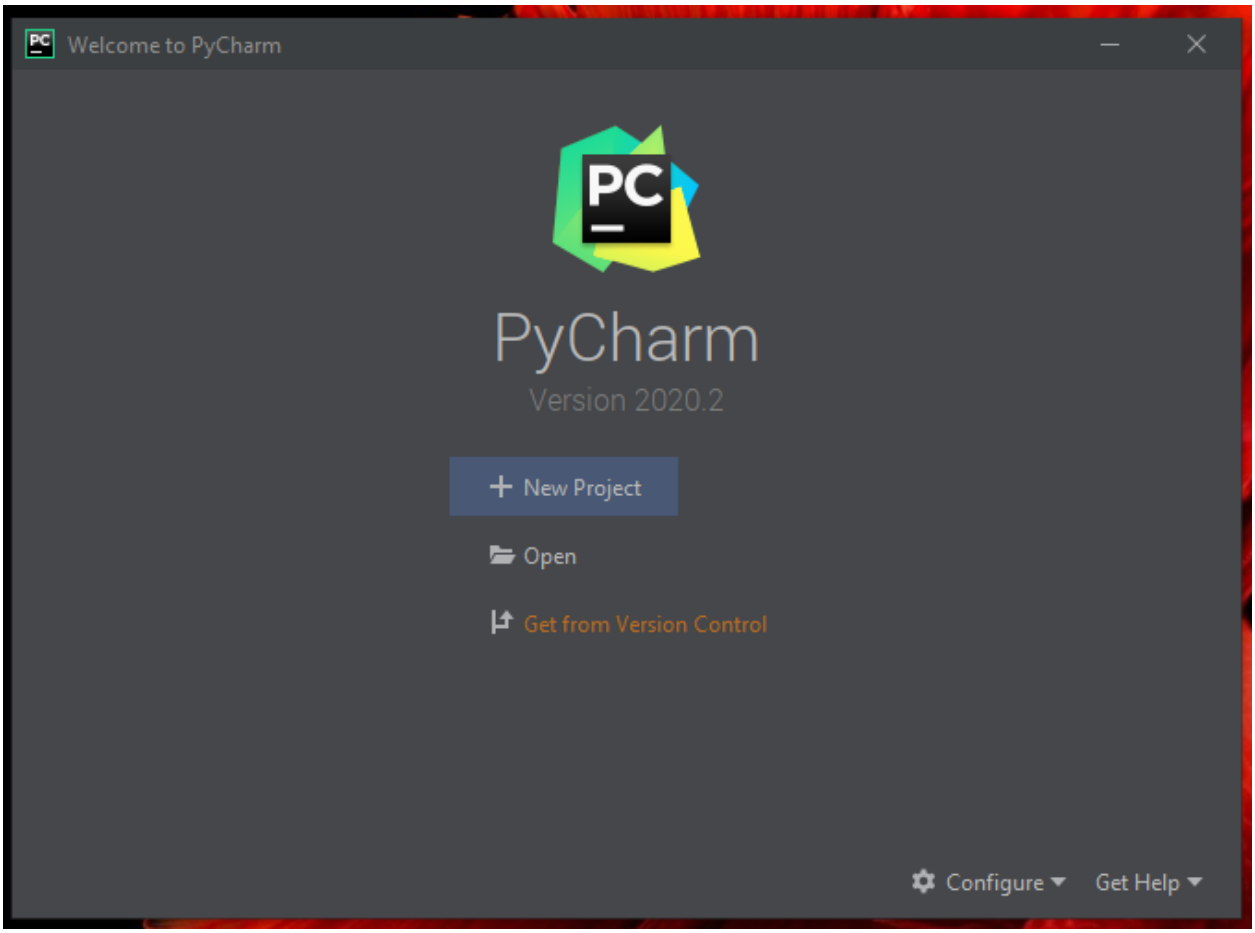
Note: Esta etapa só foi possível pois instalamos previamente o python na máquina! :D

18. Agora clique em `Apply` e depois em `ok`:

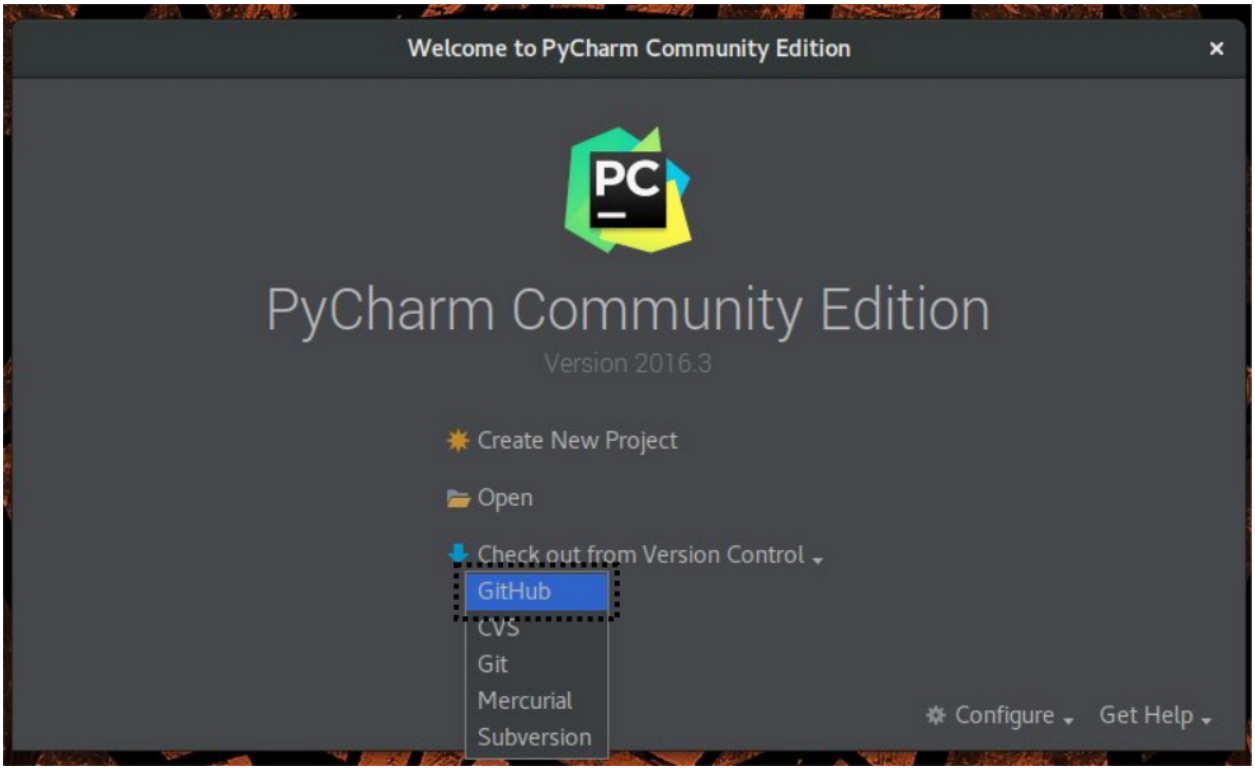


IMPORTANDO PROJETOS

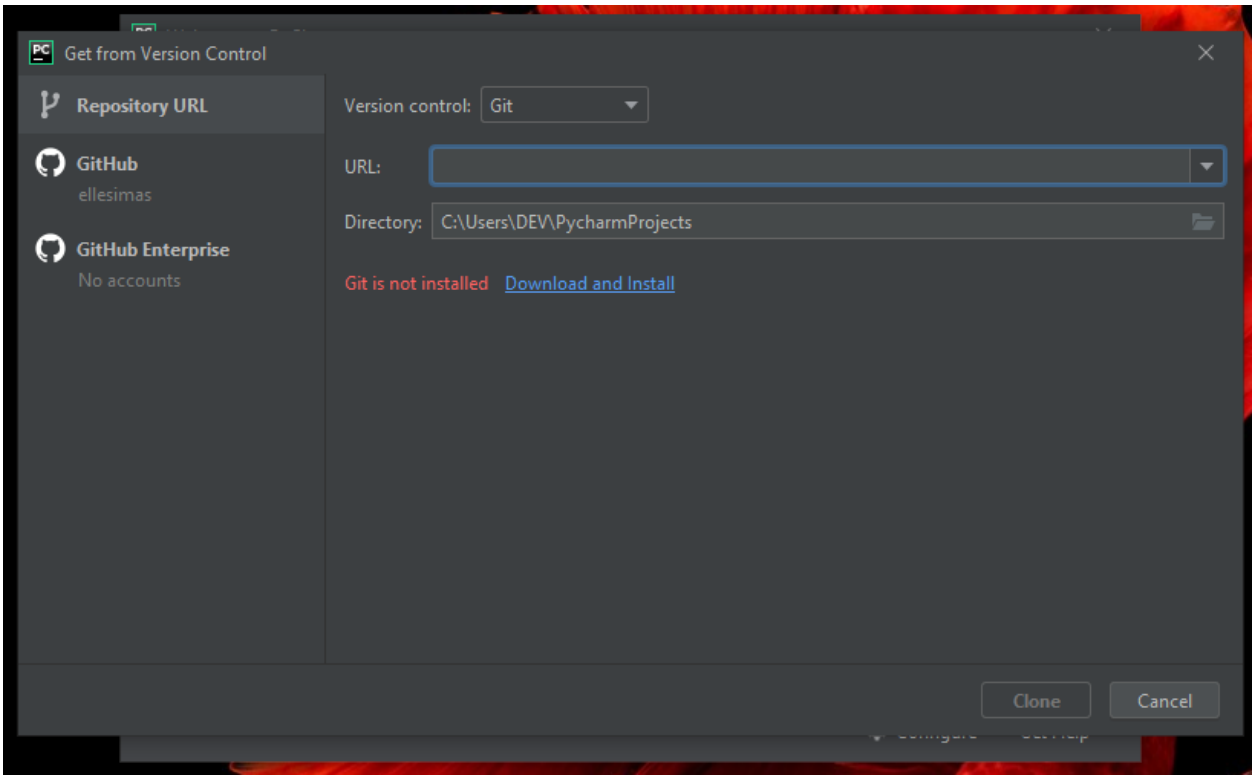
19. Você deve estar vendo esta tela agora:



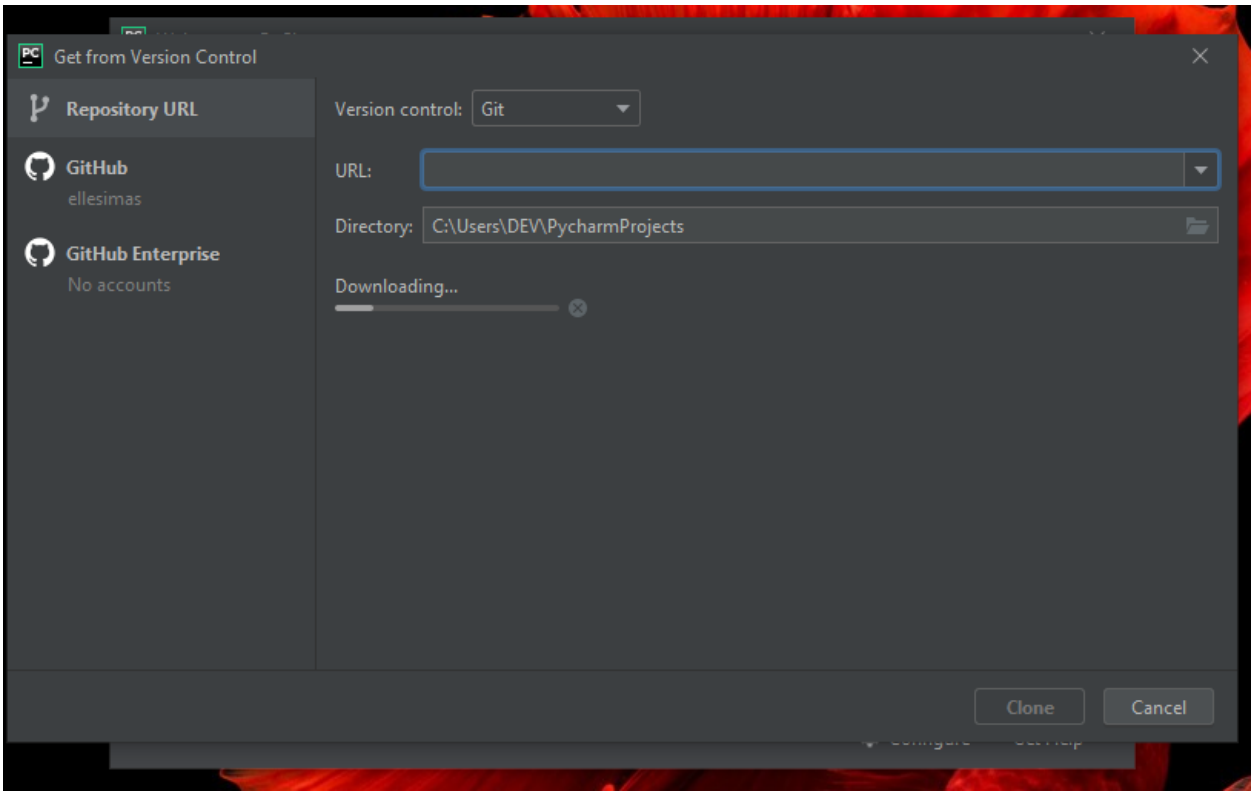
20. Clique em `Get from Version Control` e **depois** em `GitHub`



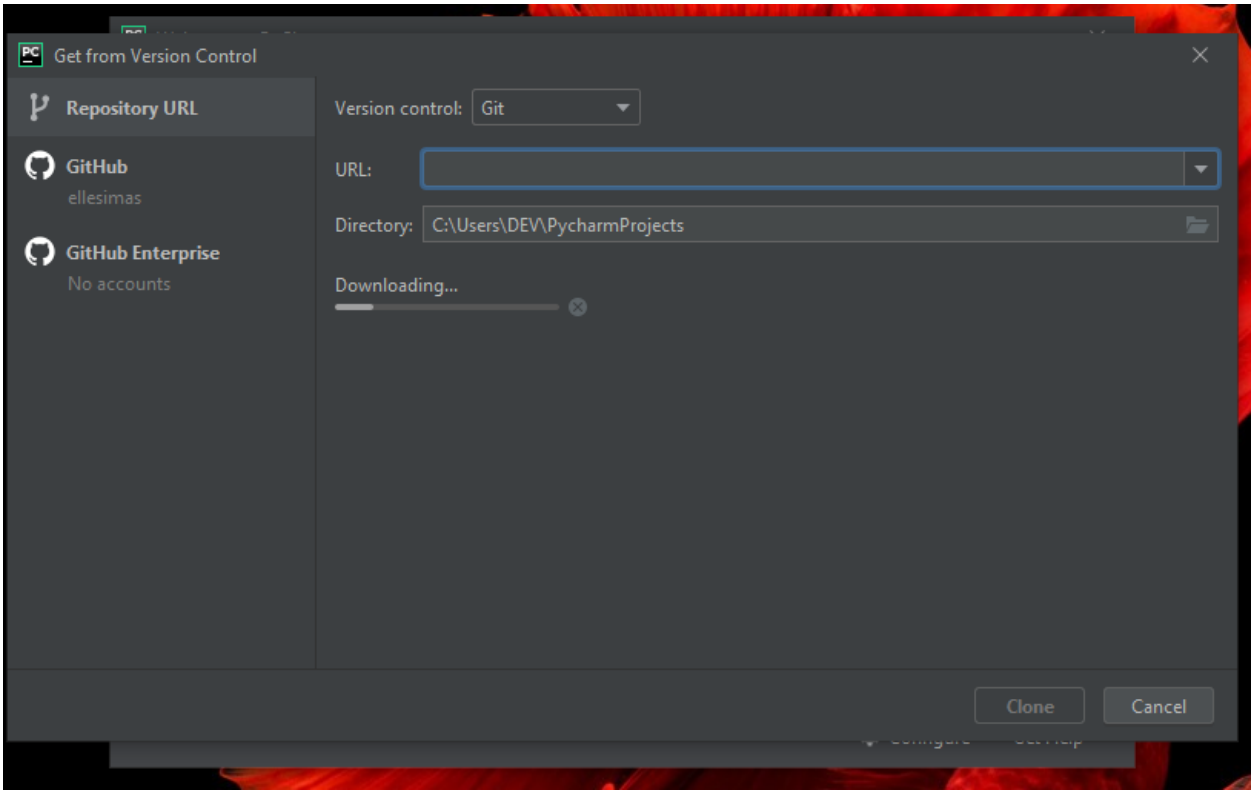
21. Nesta tela clique em `Download and Install`.



depois:

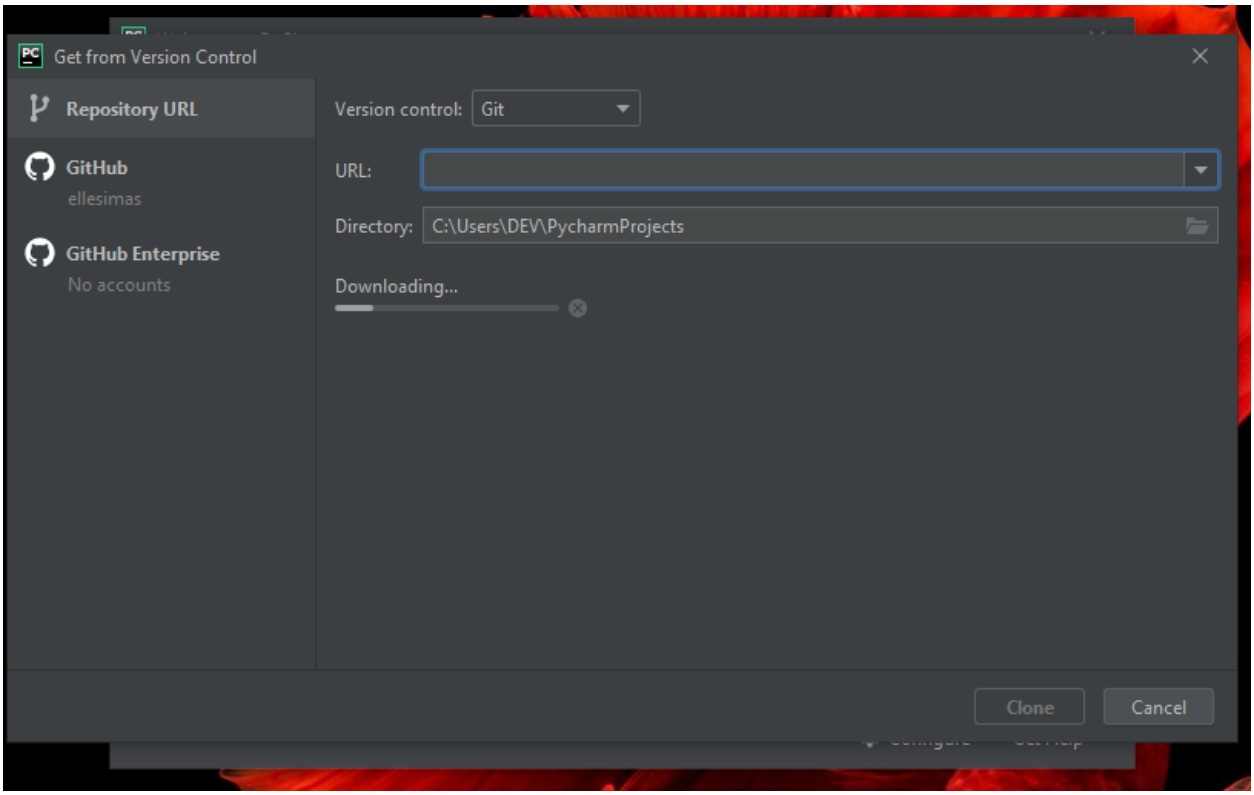


22. Nesta tela há duas possibilidades: *clonar projetos através do link* e *clonar projetos da conta github*

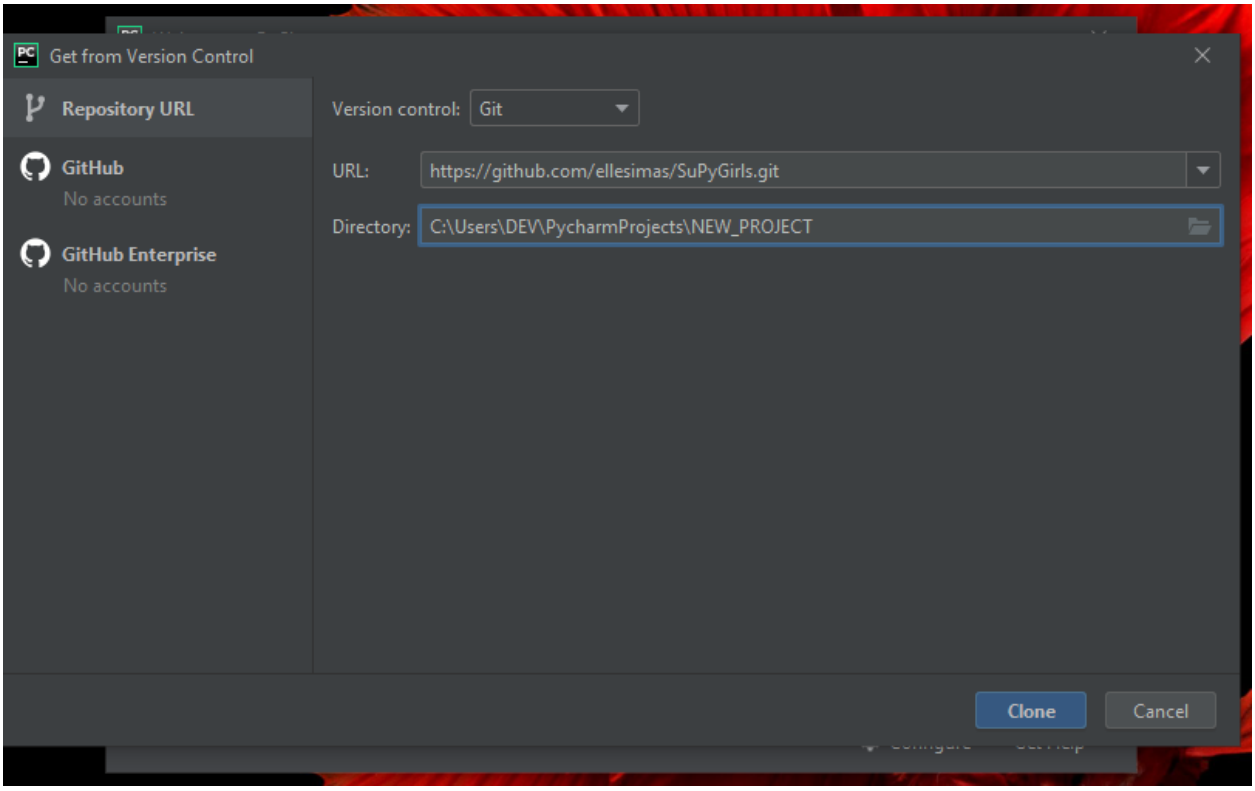


CLONANDO PROJETOS ATRAVÉS DO LINK

1. Espere a conclusão do download do git.



2. No espaço `URL` insira o link do repositório que você deseja clonar:



3. No espaço `Directory` dê um nome ao seu novo projeto (clone) alterando **a última parte do caminho**.

Exemplo:

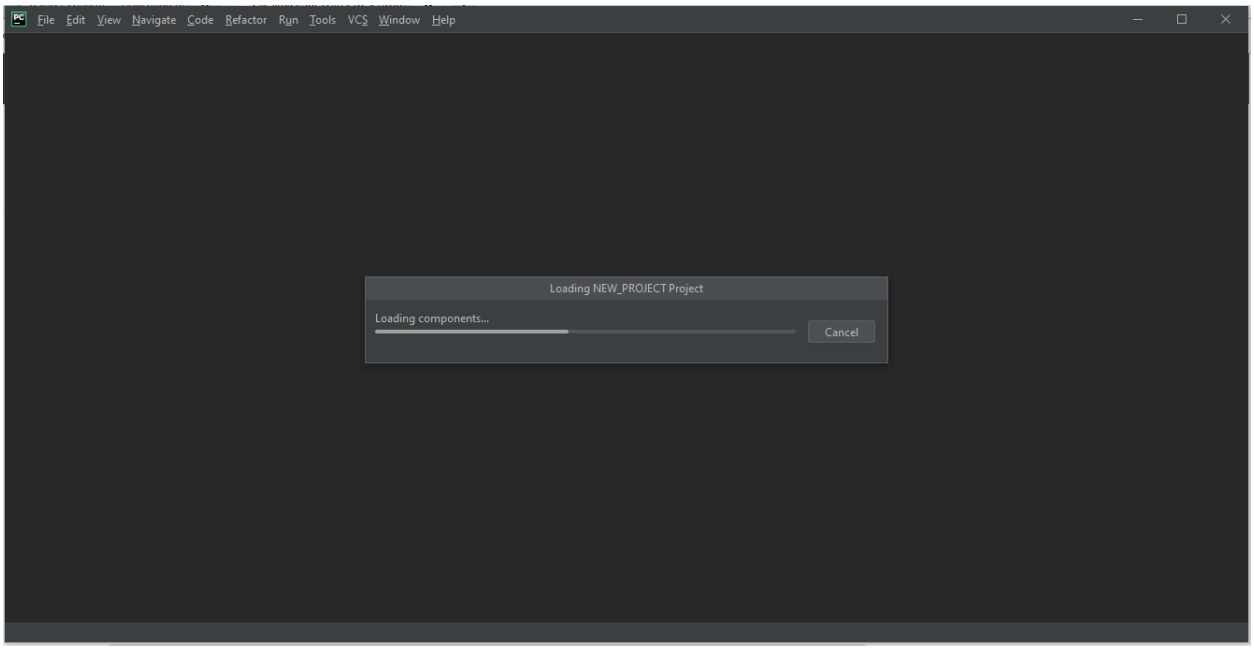
```
C:\Users\DEV\PycharmProjects\NEW_PROJECT # este é o caminho atual
```

Eu posso alterar para:

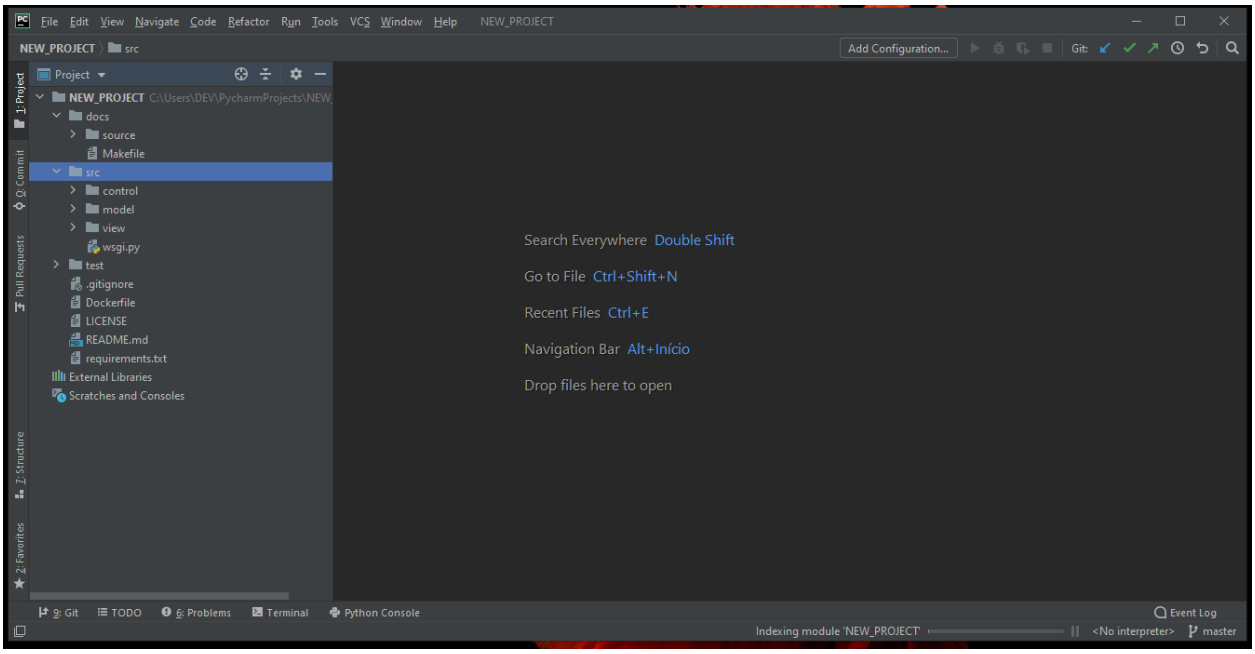
```
C:\Users\DEV\PycharmProjects\Meu_Novo_Clone # este é o caminho com outro nome
```

4. Clique em `Clone` e verá esta tela:

5. Posteriormente verá esta:

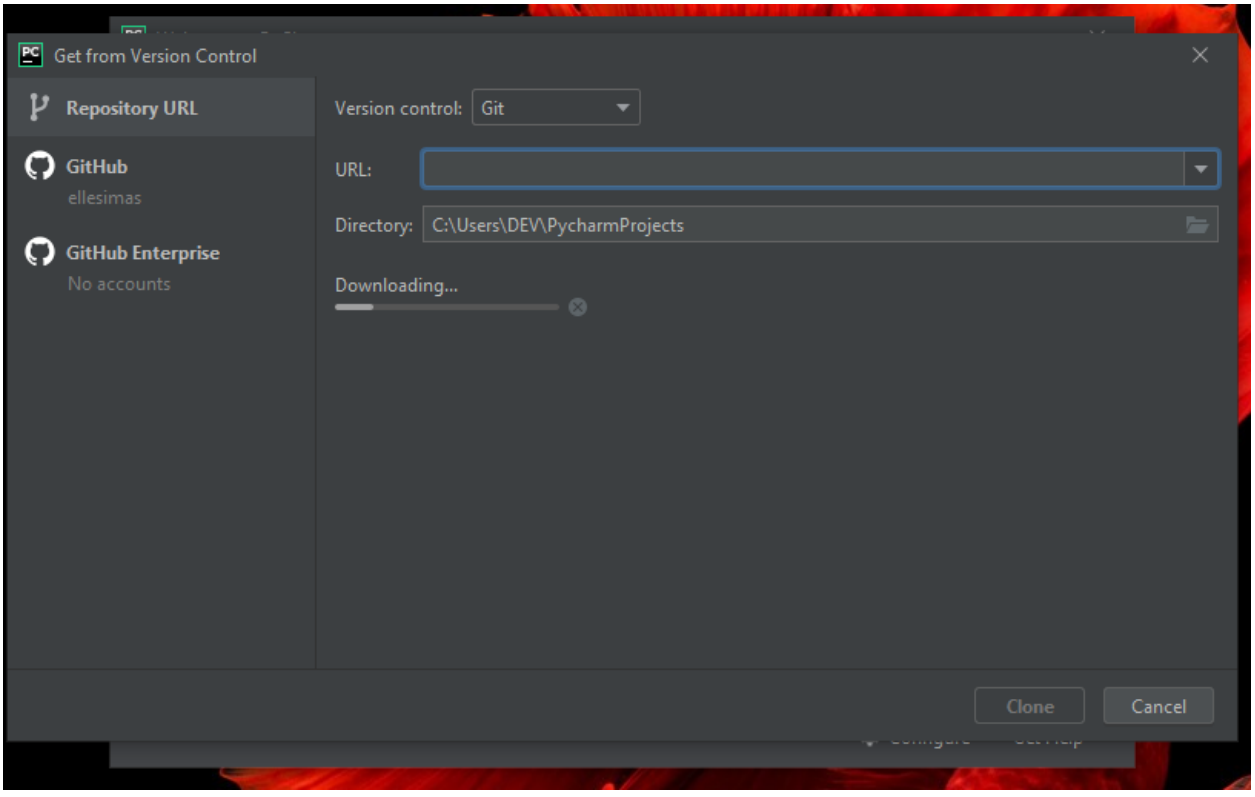


6. E então **TCHARAAAAAM!!!!!** Pycharm pronto para o uso!

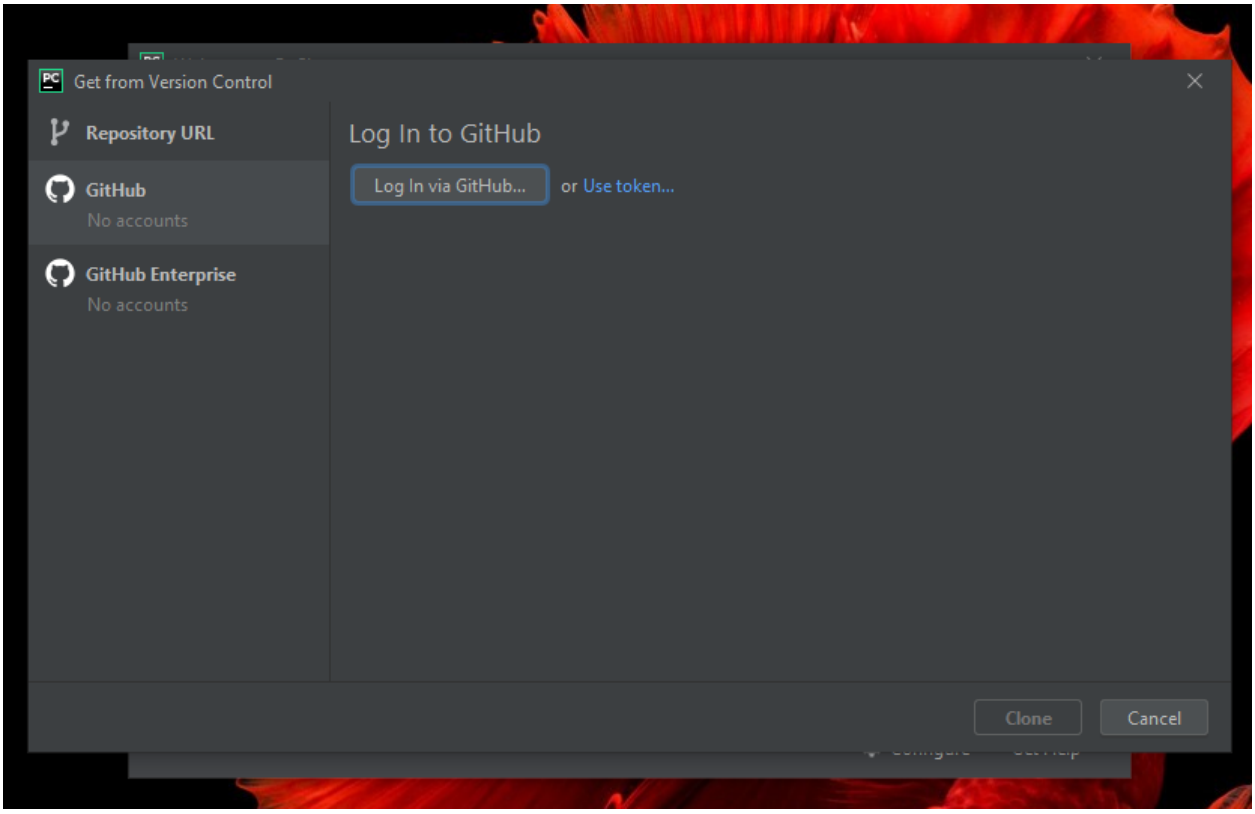


CLONANDO PROJETOS ATRAVÉS DA CONTA

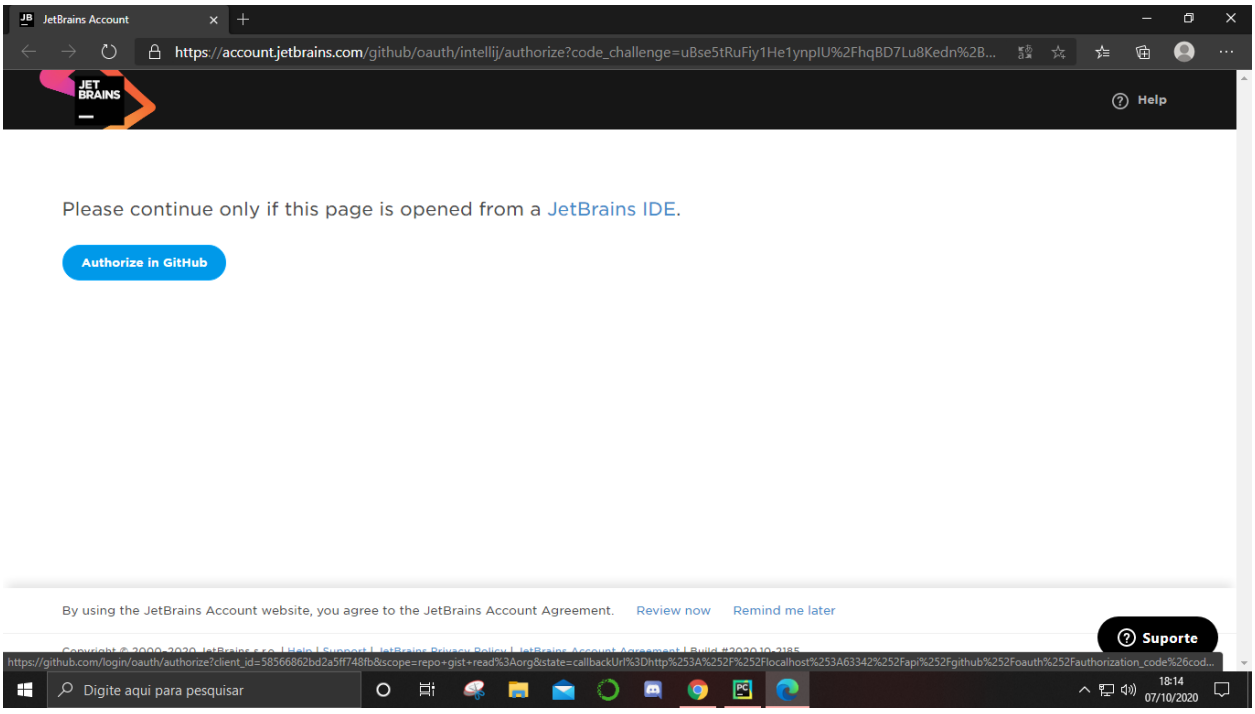
1. Clique no `GitHub` à esquerda.



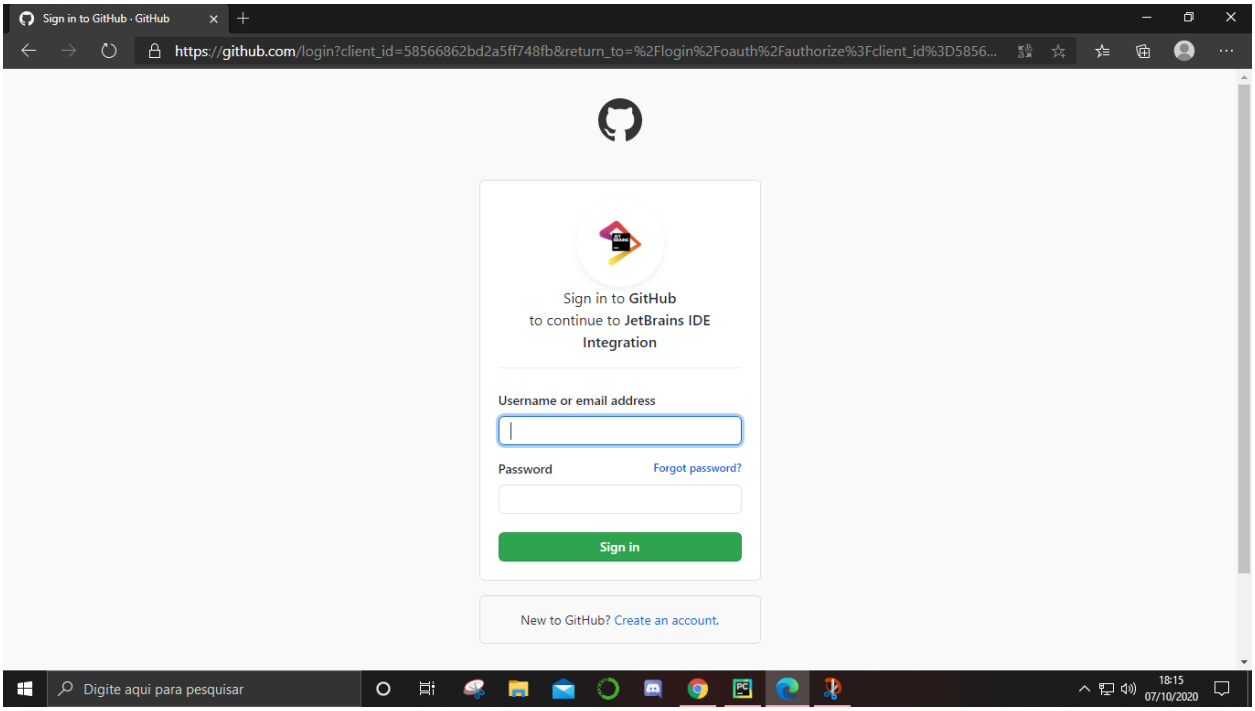
2. Clique em `Log In via GitHub`



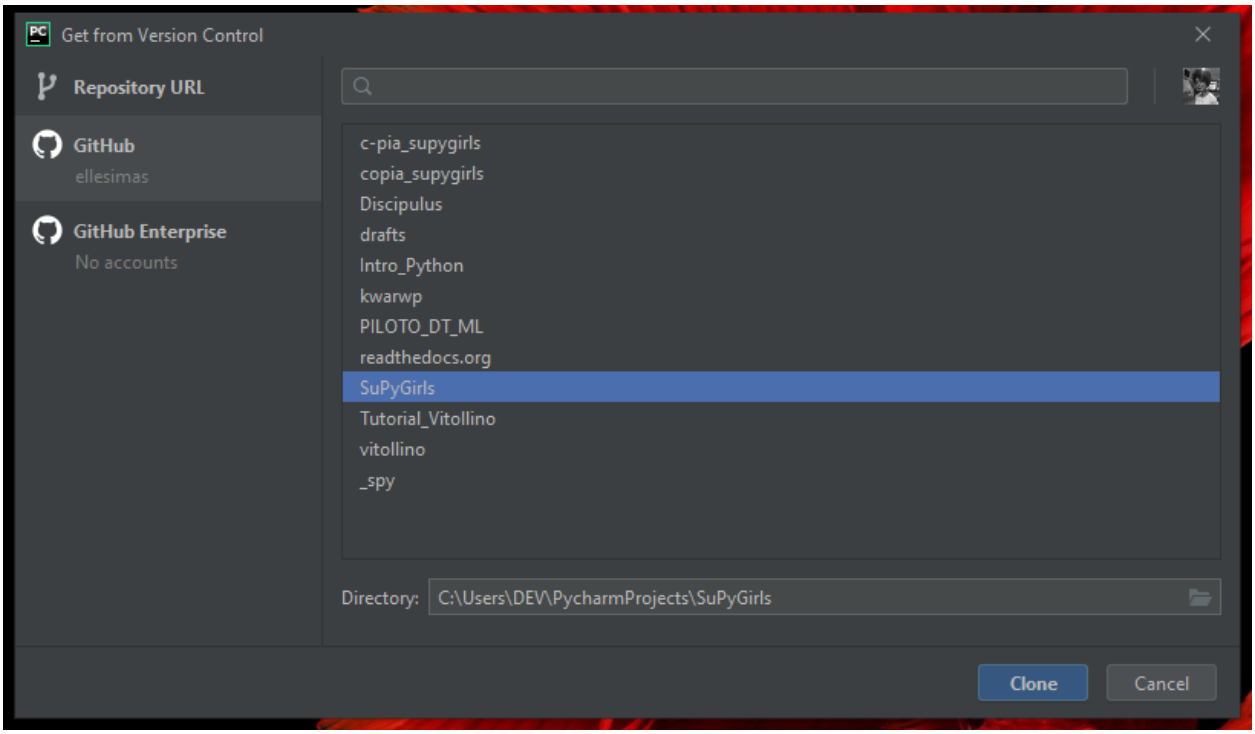
3. Autorize o víclulo entre o Pycharm e o GitHub



4. Adicione seu login e senha

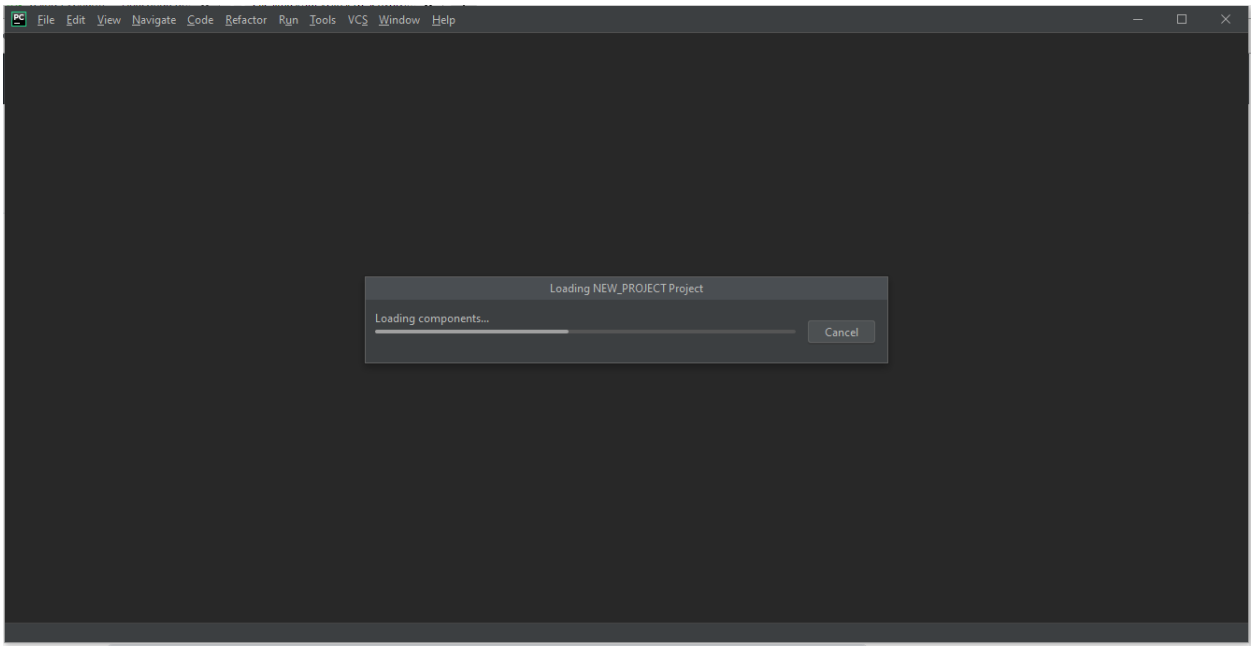


5. Volte para o Pycharm e selecione o Repositório que deseja:

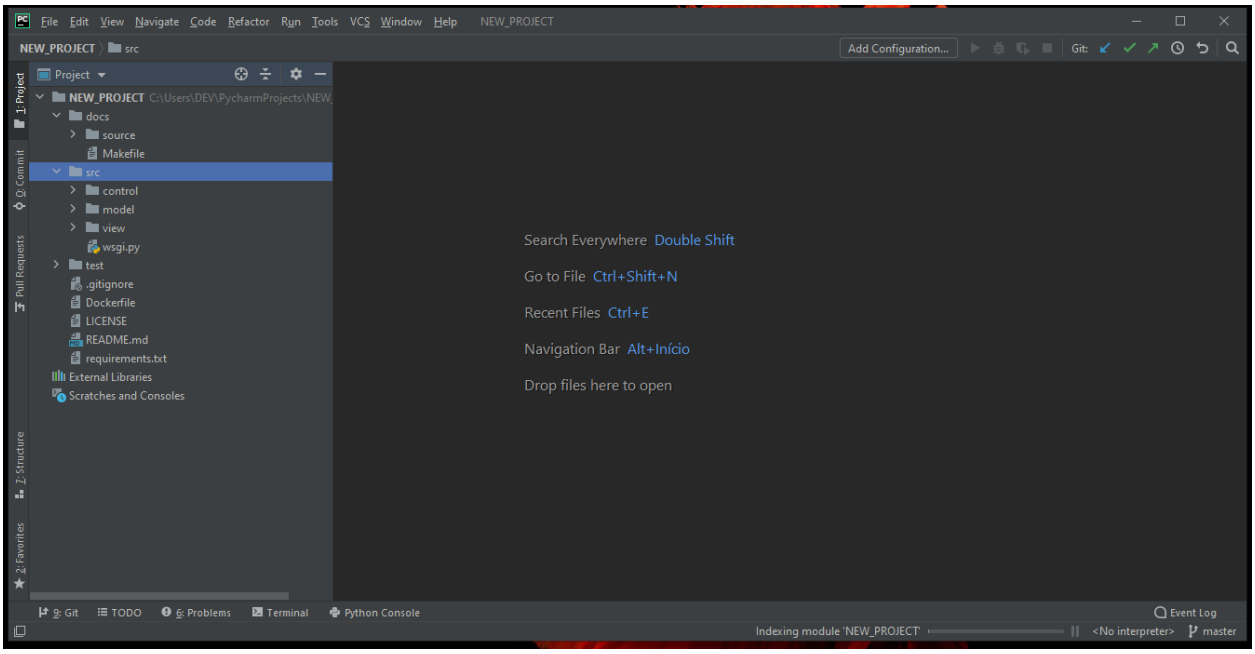


6. Aguarde o carregamento. . .

7. Posteriormente verá esta:



8. E então **TCHARAAAAAM!!!!!** Pycharm pronto para o uso!

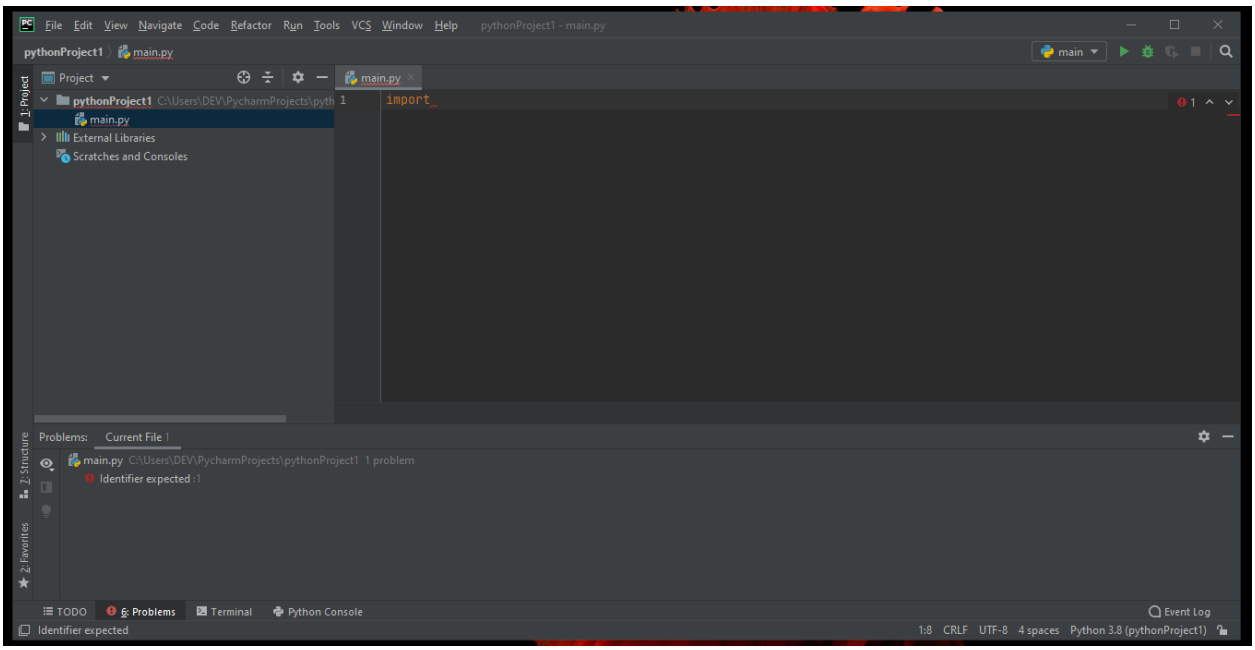


IMPORTANDO BIBLIOTECAS NO PYCHARM

As bibliotecas são repositórios/módulos que guardam códigos que podem ser reutilizados posteriormente.

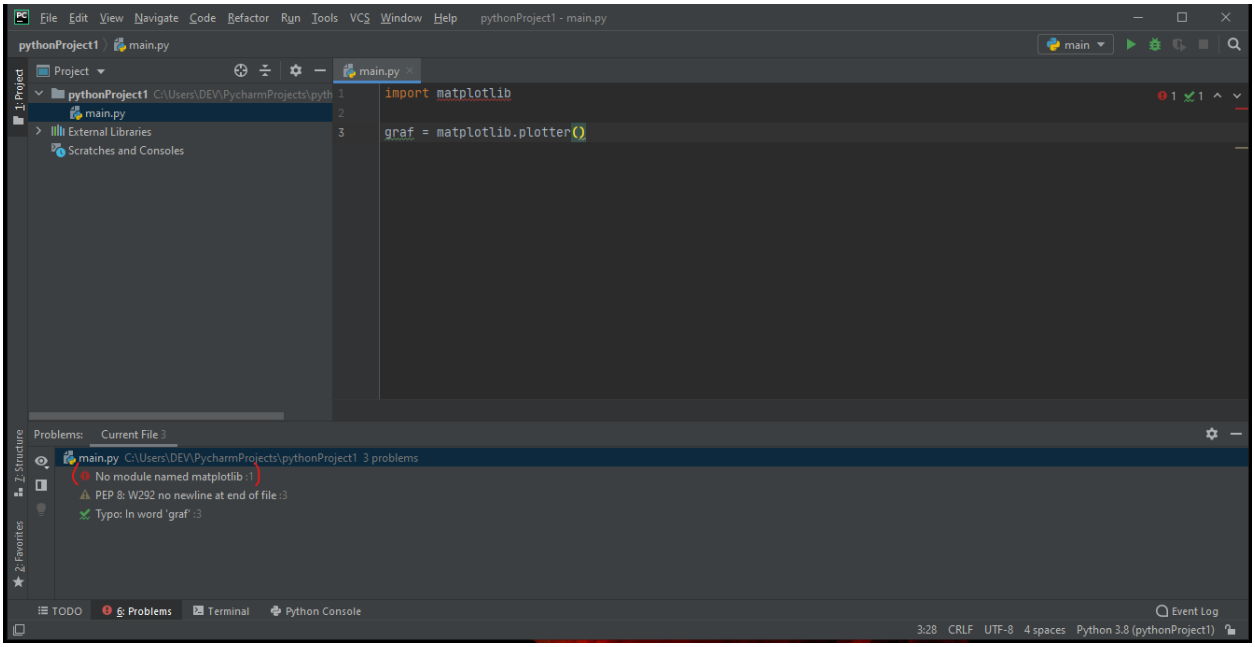
1. Abra algum projeto no Pycharm.
2. **Verifique se a biblioteca já está instalada** digitando `import nome_da_biblioteca`

```
# vamos testar a existência da biblioteca matplotlib
import matplotlib
```

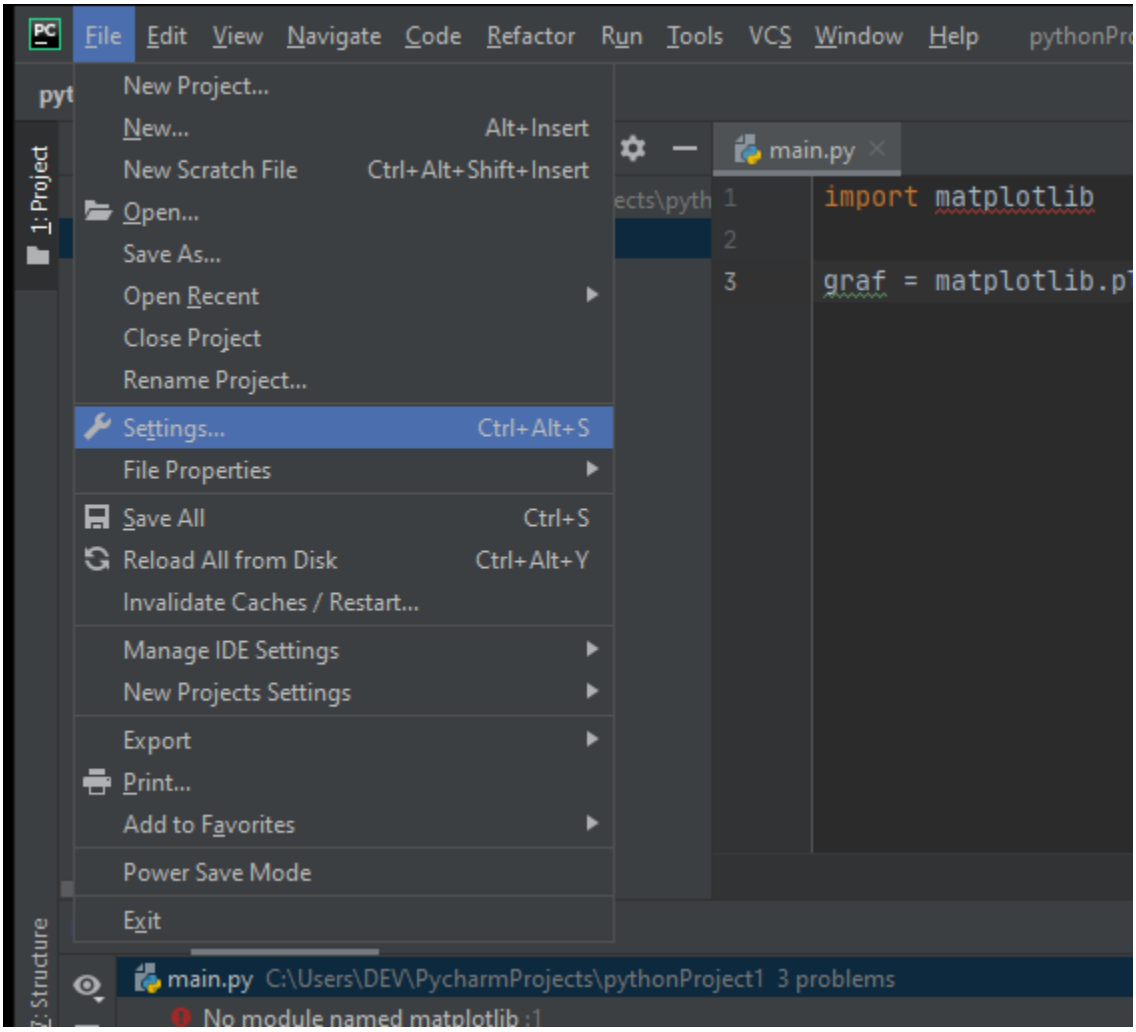


3. Observando embaixo é possível ver a mensagem `No module named matplotlib`, ou seja, o programa

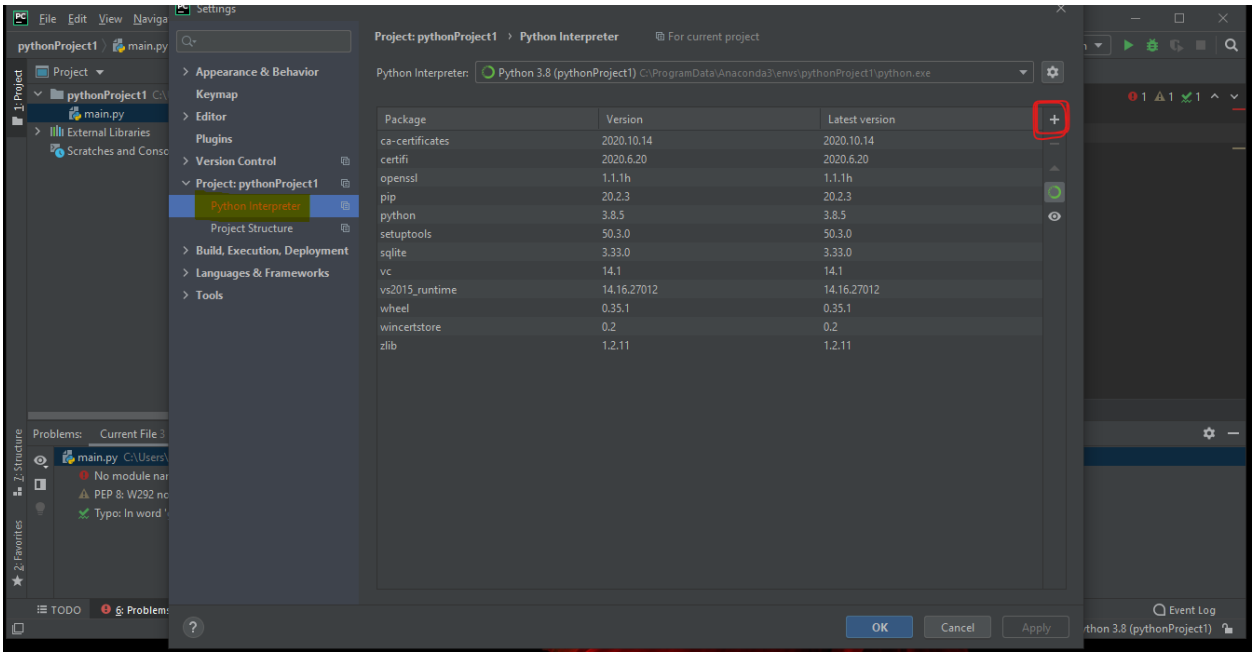
está reclamando que não há nenhum módulo instalado com esse nome.



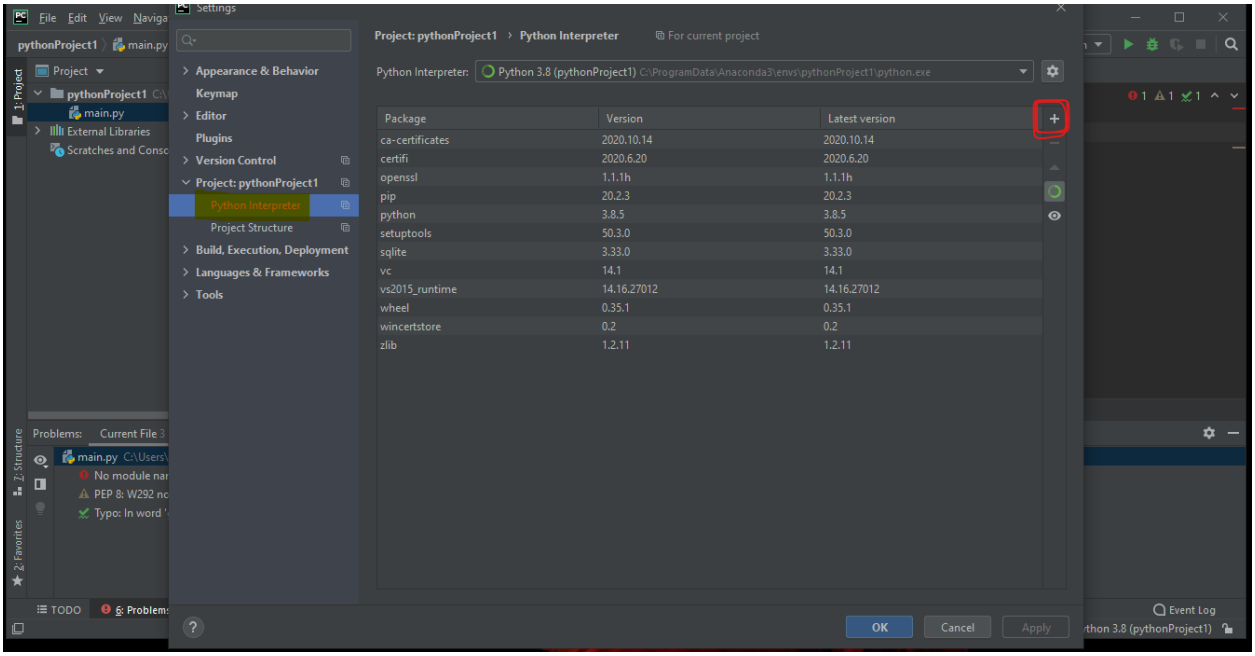
4. Na barra superior clique em `File` e **depois** em `settings`



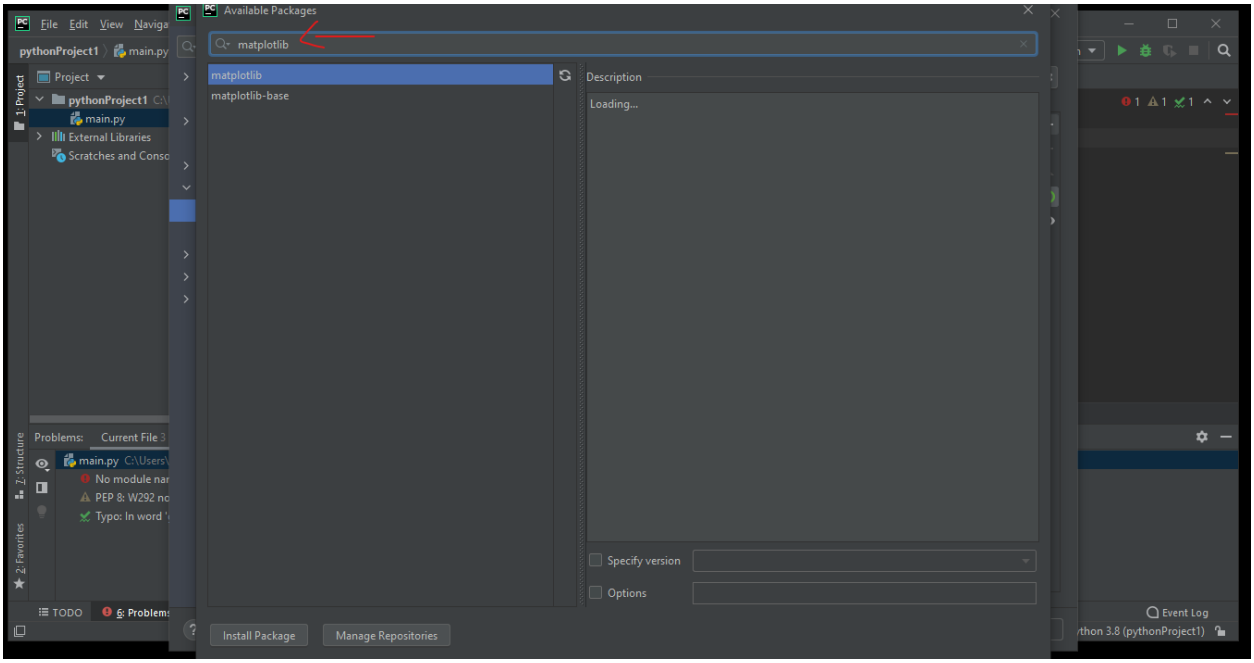
5. Clique em `Project:  [...]` e depois em `Python Interpreter`



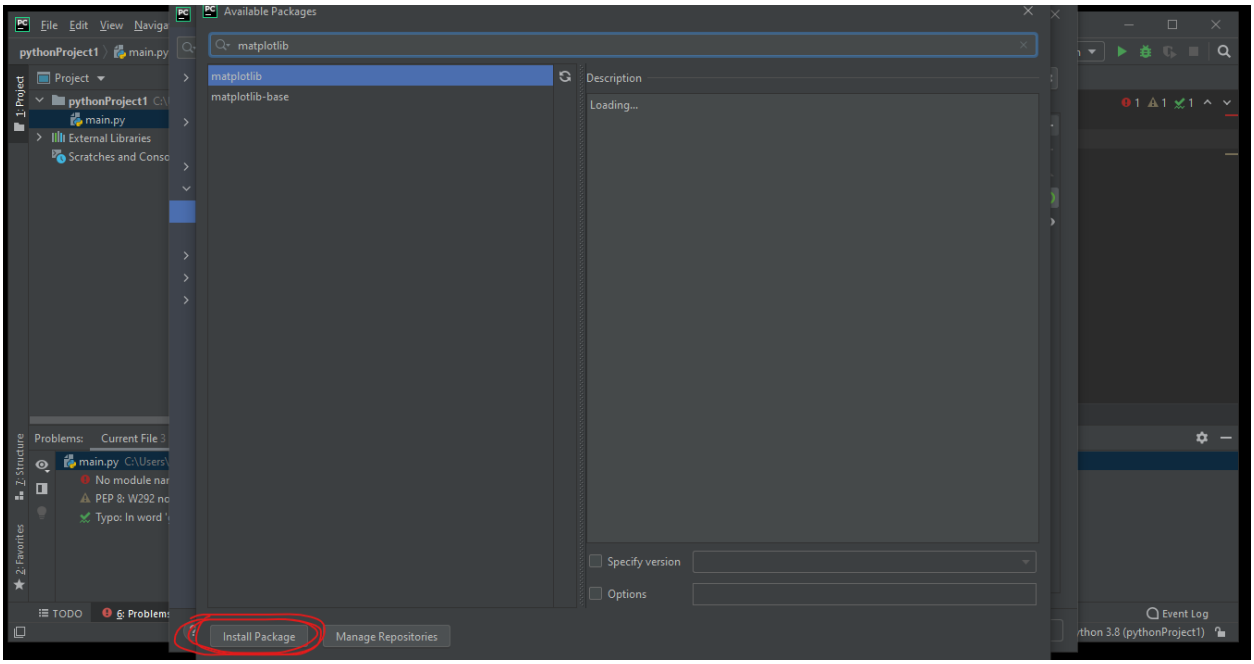
6. Clique no + ressaltado à direita



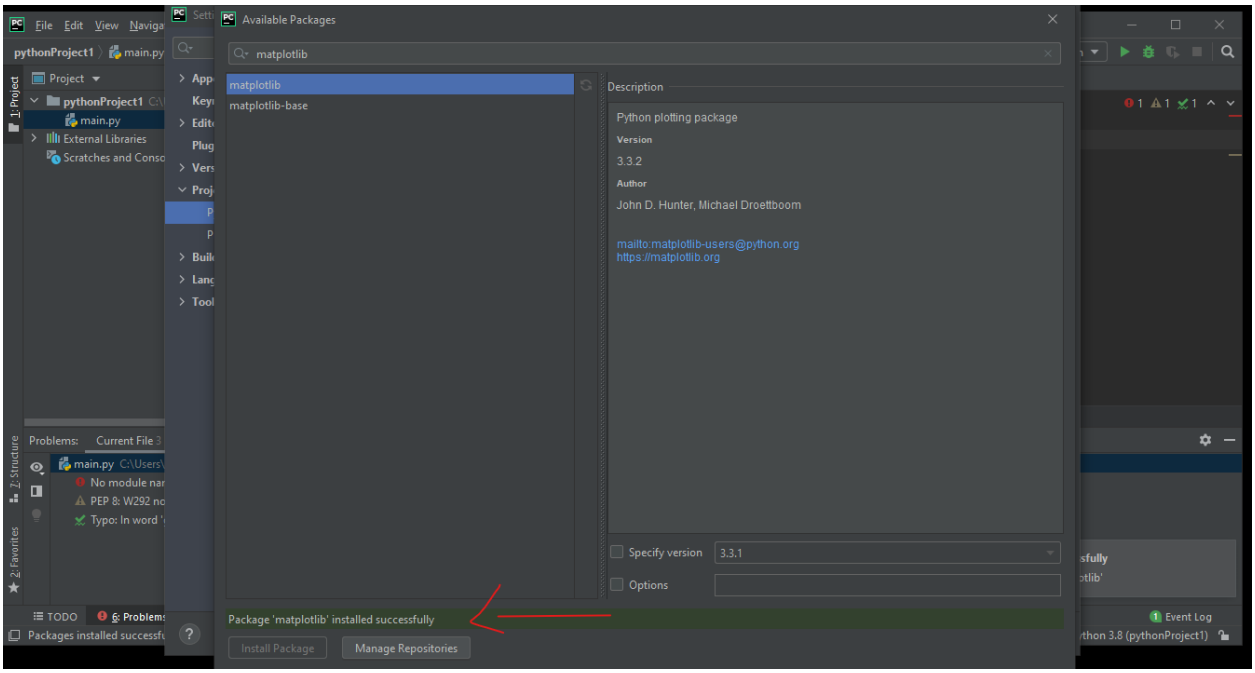
7. Escreva o nome da biblioteca que deseja no espaço:



8. Clique em `Install Package` na parte inferior da tela e aguarde (pode demorar)

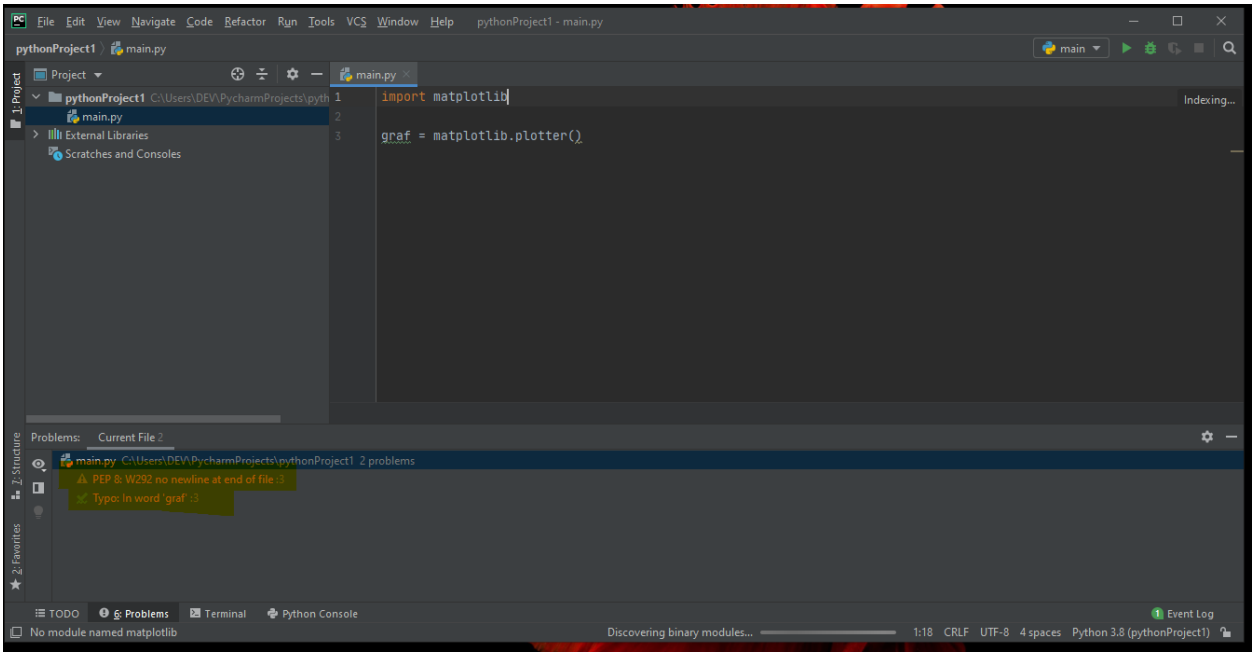


8. A mensagem `Package 'matplotlib' installed successfully` confirma a instalação do pacote.



Feche as telinhas.

9. Voltando a tela inicial é possível observar que o console **não** reclama mais da ausência do módulo:



SPYDER

ANACONDA

Tutorial Author: Emanuelle M. P. Simas;

CHAPTER 7

Indices and tables

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index