LINGUAGEM DE PROGRAMAÇÃO APLICADA

AULA 1

Prof. Osmar Tenorio Pereira Dias Junior

CONVERSA INICIAL

Lógica de programação aplicada

Nesta etapa, serão trabalhados conceitos intermediários em programação. Veremos alguns itens relacionados com os pacotes de funções disponíveis na linguagem Python, como podemos incluir tais pacotes em programas escritos em Python e como eles podem ser utilizados. De forma mais específica, veremos as bibliotecas NumPy e Plotly, que servem para manejar matrizes numéricas e descrição de gráficos baseados nessas matrizes. Teremos uma visão sobre o significado das matrizes numéricas, com uma explanação dos conjuntos dados possíveis. Faremos alguns exemplos de usos dessas bibliotecas de programação com vistas a demonstrar sua utilidade, e como esse processo é realizado.

Depois, mostraremos os primeiros passos no ambiente de repositório de arquivos Git, essencial para o controle de versões de um projeto de software. Isso é extremamente útil e amplamente utilizado pelos desenvolvedores do mundo todo, e a evolução dos códigos de programação não atingiria o atual grau de desenvolvimento sem essa ferramenta. É natural que, durante a construção de um projeto de programa computacional, existam encaminhamentos diferentes, experimentações de diferentes estratégias de programação e evolução da execução do projeto ao longo do desenvolvimento. Dessa forma, é inevitável a necessidade de algum sistema que permita o registro de todos os encaminhamentos dados durante o processo de desenvolvimento. Isso precisa ser feito com os registros de datas e autores, sabendo qual parte do programa está em qual arquivo, tanto quando um trabalho sendo feito por um programador quanto por uma equipe. No caso de vários programadores, a situação é mais crítica, pois seria impossível a perfeita sincronização das tarefas sem um controle muito perfeito desses elementos.

Com o Git local, veremos o GitHub, que é a extensão do trabalho em um computador pessoal para uma rede que pode chegar a ser mundial, inclusive. Isso porque as mesmas lógicas inseridas no sistema local são disponibilizadas aos colaboradores do projeto pela internet – na *nuvem*, como é chamada essa forma de armazenamento.

TEMA 1 – BIBLIOTECAS DO PYTHON

Bibliotecas são conjuntos de módulos e pacotes que fornecem funcionalidades específicas para facilitar o desenvolvimento de software em diversas áreas. São compostas por um conjunto de funções, classes e métodos que podem ser utilizados por desenvolvedores para realizar tarefas específicas sem a necessidade de reescrever códigos do zero. As bibliotecas desempenham um papel crucial na expansão das capacidades da linguagem e na criação de uma comunidade de desenvolvedores que compartilham e contribuem com soluções para problemas comuns.

Algumas bibliotecas mais populares em Python:

- NumPy: Uma biblioteca para computação numérica que fornece suporte a arrays e matrizes multidimensionais, bem como funções matemáticas para operações eficientes em dados numéricos.
- Pandas: Uma biblioteca para manipulação e análise de dados, oferecendo estruturas de dados poderosas como DataFrame para trabalhar com dados tabulares.
- Matplotlib: Uma biblioteca para criação de gráficos e visualizações estáticas em Python.
- Plotly: Uma biblioteca para criação de gráficos interativos e dashboards.
- TKInter: biblioteca nativa no Python para geração de janelas com diversas possibilidades de elementos para interação, como botões, rótulos, campos para inserção de dados entre outros.
- Scikit-learn: Uma biblioteca para aprendizado de máquina que fornece ferramentas simples e eficientes para análise de dados e modelagem estatística.
- Requests: Uma biblioteca para realizar requisições HTTP de forma fácil e eficiente.
- Django e Flask: Bibliotecas para desenvolvimento web, facilitando a criação de aplicativos e websites.
- TensorFlow e PyTorch: Bibliotecas para aprendizado de máquina e deep learning.
- PyGame: Biblioteca para auxílio na construção de jogos em Python.

Para utilizar uma biblioteca em Python, é necessário instalá-la, geralmente utilizando um gerenciador de pacotes como o pip. Depois de instalada, você pode importar a biblioteca no seu código Python para começar a utilizar suas funcionalidades.

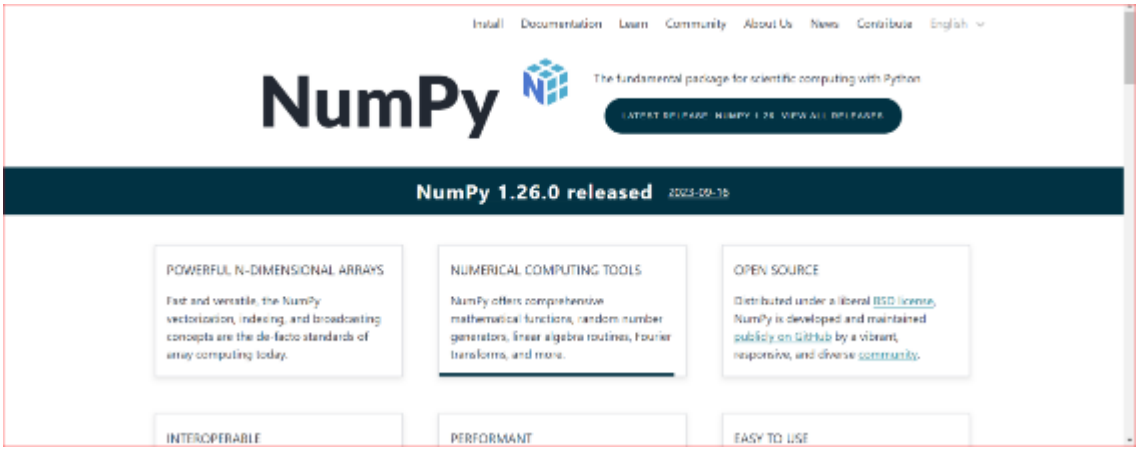
Os pacotes possuem manuais, embora sejam criados por pessoas ou empresas que não são, necessariamente, da python.org. Ex.: processamento de dados, visualização de gráficos, comunicação em rede, aprendizado de máquina, manipulação de arquivos, desenvolvimento de jogos

TEMA 2 – UTILIZANDO BIBLIOTECAS

2.1 Pacote Numpy

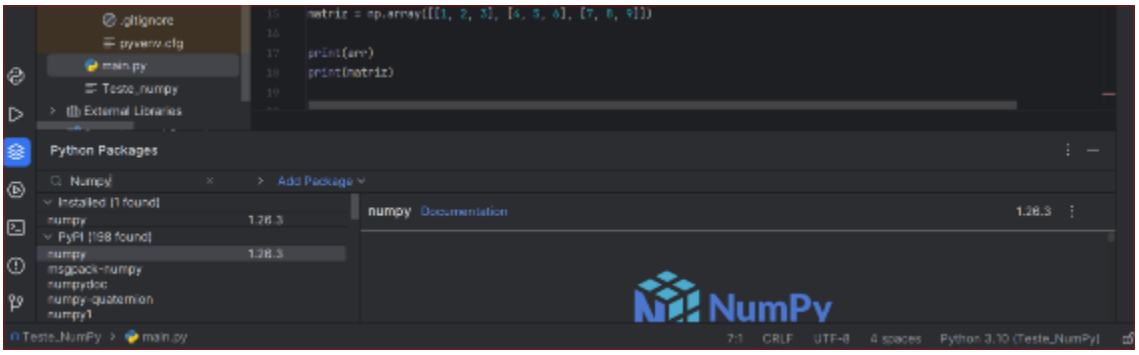
Um biblioteca bastante útil e amplamente utilizada é a Numpy. Na verdade, é um conjunto de bibliotecas direcionado para vários aspectos relacionados com conjuntos de dados. Podem ser aspectos matemáticos, estatísticos, estabelecimento de correlações entre conjuntos de dados, entre outros. A Figura 1 mostra a página inicial do site do fabricante desse pacote (2024).

Figura 1 – Site oficial do pacote NumPy



A instalação do pacote NumPy é necessária para o uso de suas funcionalidades. No Pycharm podemos instalar por meio do instalador de pacotes, no lado esquerdo na tela desse ambiente de desenvolvimento (Figura 2).

Figura 2 – Caminho para instalação de pacotes no Pycharm



Uma vez instalado o pacote, podemos utilizar suas funções. Para isso, é necessário importar essa biblioteca no código onde ela será utilizada, devendo-se colocar a seguinte linha de comando: **import numpy as Np**.

Na Figura 3, há um exemplo simples sobre como usar um dos comandos possíveis da biblioteca NumPy.

Figura 3 – Código exemplo com algumas funcionalidades do NumPy

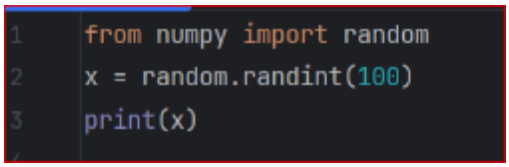
```
#Importando a biblioteca NumPy
import numpy as np

# Criando um array unidimensional
arr1d = np.array([1, 2, 3, 4, 5])
print("Array 1D:")
print(arr1d)

# Criando um array bidimensional (matriz)
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("\nArray 2D (Matriz):")
print(arr2d)
```

Na programação em Python, via de regra, as bibliotecas são importadas dessa forma, invocando o comando **import** e inserindo a biblioteca requerida. Algumas vezes, podemos importar somente uma parte da biblioteca. Alguns pacotes são conjuntos de bibliotecas e podemos importar somente uma delas ou todas. Para tal, usamos uma linha de comando parecida com a seguinte:

Figura 4 – A função “random” sendo importada do pacote NumPy



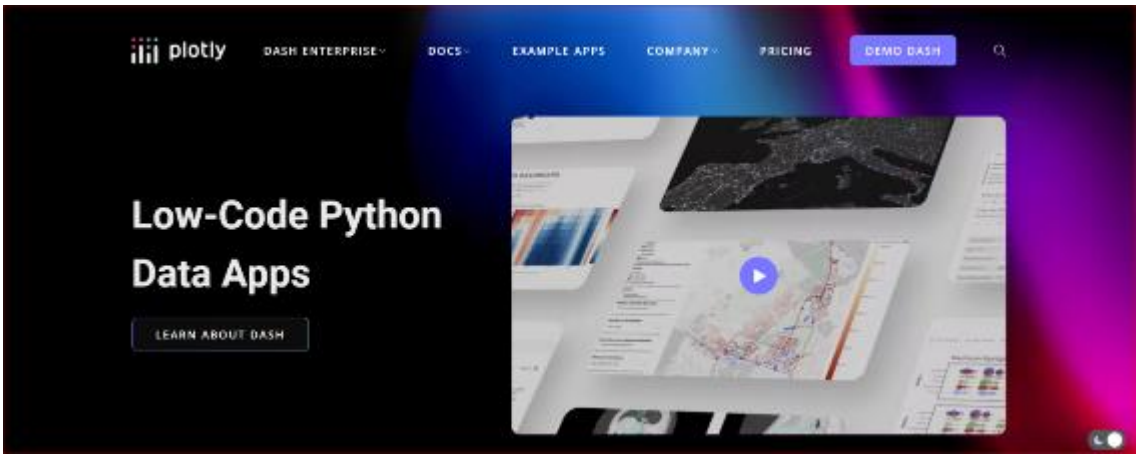
Na Figura 4, vemos um pequeno programa para gerar um número aleatório entre 0 e 100 . Nesse caso, importamos somente a função “random” da biblioteca NumPy. De modo geral, isso pode ser feito com todas as bibliotecas, quando importamos somente a função que será utilizada. Isso pode trazer um benefício de tornar o tamanho do programa menor quando da geração do arquivo executável, por exemplo. Mas quando usarmos várias das funções disponíveis na biblioteca, o melhor é importá-la inteira.

Para usar bibliotecas nos programas em Python é necessário instalar as bibliotecas que não estão originalmente incluídas no pacote do interpretador. Algumas bibliotecas já estão integradas ao interpretador Python e não precisam ser instaladas. Um exemplo é a biblioteca TKInter, já presente, porém, ela precisa ser importada, uma vez que não funciona diretamente. Isso tornaria os arquivos executáveis desnecessariamente grandes.

2.2 Pacote Plotly

O pacote Plotly é voltado para os aspectos gráficos possíveis a partir de conjuntos de dados. Ao procurarmos informações sobre o pacote Plotly, encontramos a página do fabricante com as opções referentes ao conjunto de funcionalidades oferecidas (Figura 5). Embora haja uma opção paga para esse pacote, podemos usar as funcionalidades no ambiente de desenvolvimento Python de forma livre e gratuita. O que é cobrado é um conjunto de outras funcionalidades que permitem recursos muito mais avançados e direcionados ao uso comercial e empresarial, que não fazem parte do escopo de nosso estudo aqui.

Figura 5 – Página inicial do pacote Plotly



Plotly é uma biblioteca gráfica interativa para Python que permite a criação de visualizações de dados interativas e de alta qualidade. Com o Plotly, podemos criar gráficos estáticos, gráficos interativos, dashboards (páginas com dados e gráficos informativos sobre um determinado assunto) e visualizações 3D. O pacote Plotly suporta uma variedade de tipos de gráficos, como scatter plots (gráficos de dispersão), gráficos de barras, gráficos de linha, gráficos de dispersão 3D, entre outros.

Algumas características e pontos destacados do Plotly incluem:

- Interatividade: Os gráficos criados com o Plotly são interativos, o que significa que você pode adicionar recursos como zoom, pan, dicas de ferramentas (tooltips) e outras interações, tornando a exploração dos dados mais dinâmica.
- Exportação e compartilhamento: Os gráficos Plotly podem ser exportados em vários formatos, como imagens estáticas ou arquivos interativos em HTML. Isso facilita o compartilhamento dos resultados com outras pessoas.
- Dashboards interativos: Além dos gráficos, o Plotly é uma parte fundamental do Dash, uma estrutura para criar dashboards interativos em Python. Com o Dash, você pode criar aplicativos web interativos com visualizações de dados e controles interativos.
- Comunidade ativa: O Plotly possui uma comunidade ativa, e a documentação é abrangente, o que facilita o aprendizado e o uso da biblioteca.

Para começar a usar o Plotly em Python, geralmente você precisa instalá-lo usando o gerenciador de pacotes e importá-lo no início da codificação que estiver desenvolvendo.

2.3 Comparação entre o uso de array e lista

No Python existe a **lista**, que trabalha com conjuntos de dados heterogêneos, e a biblioteca do NumPy apresenta também **arrays**, que podem trabalhar com conjuntos de dados de um só tipo, ou seja, homogêneos. A principal diferença entre elas é que todos os elementos de um array ou matriz devem ser do mesmo tipo de dado, e as listas admitem tipos diferentes em seus registros.

O módulo array fornece uma implementação mais compacta de conjuntos de dados do que as listas. As listas podem conter elementos de vários tipos diferentes. Para uso de array é necessário a importação do módulo array. Observe o exemplo na Figura 6.

Figura 6 – Exemplo de uso de array e lista

```
# Uso de array requer a importação do módulo array
from array import array
meu_array = array( _typecode: 'i',  _initializer: [1, 2, 3, 4, 5])

# Para uso de lista não é necessário importar nenhuma biblioteca
minha_lista = [1, 2, 3, "quatro", 5.0]
```

As arrays são mais eficientes em termos de espaço de armazenamento e desempenho, mas têm a limitação de que todos os elementos precisam ser do mesmo tipo. Em muitos casos, as listas são mais comumente usadas em Python devido a sua flexibilidade e funcionalidades adicionais.

No pacote NumPy existe uma biblioteca, principal nesse pacote, que é a ndarray. As principais características do ndarray são :

- Estrutura N-dimensional: O ndarray é uma matriz multidimensional que pode ter qualquer número de dimensões. Pode ser unidimensional (vetor), bidimensional (matriz), tridimensional ou ainda maior.
- Elementos homogêneos: Todos os elementos em um ndarray devem ser do mesmo tipo de dado. Isso significa que todos os elementos em uma

única matriz são do mesmo tipo, o que permite operações eficientes em grandes conjuntos de dados.

- Indexação eficiente: O ndarray suporta uma variedade de métodos eficientes para acessar, modificar e manipular seus elementos. Isso inclui fatiamento (slicing), indexação booleana, entre outros.
- Operações numéricas: NumPy fornece funções e operadores que podem ser aplicados a ndarrays de maneira eficiente, o que é crucial para aplicações científicas e computação numérica.
- Broadcasting: NumPy suporta broadcasting, o que permite que operações sejam realizadas em ndarrays de diferentes formas e tamanhos, facilitando a escrita de código mais conciso e eficiente.

Para utilizar o ndarray, primeiro, é necessário importar a biblioteca NumPy e, em seguida, criar um array. Vemos um exemplo de implementação desses comandos na Figura 7.

Figura 7 – Exemplo de implementação do ndarray no NumPy

```
import numpy as np

# Criando um ndarray unidimensional
arr = np.array([1, 2, 3, 4, 5])

# Criando um ndarray bidimensional
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(arr)
print(matriz)
```

Esse exemplo gera a saída no console mostrada na Figura 8.

Figura 8 – Saída de console do código usando ndarray do NumPy

```
[1 2 3 4 5]
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Process finished with exit code 0
```

Na primeira linha, vemos a array de uma dimensão com os valores [1, 2, 3, 4, 5]. Na sequência, vemos a array de duas dimensões, que forma uma matriz bidimensional, [[1 2 3], [4 5 6], [7 8 9]]. Nessa matriz, temos a primeira

linha com os valores 1, 2 e 3, na segunda linha os valores 4, 5 e 6 e, na terceira linha, os valores 7, 8 e 9.

O ndarray é amplamente utilizado em áreas como ciência de dados, aprendizado de máquina, processamento de imagens, simulações numéricas e muitos outras situações em que operações eficientes em grandes conjuntos de dados são necessárias.

Podemos fazer uma comparação entre os tempos para manejar uma lista e uma matriz, ambas com muitos dados. Para isso, vamos utilizar um código que registra os tempos de início e de fim do processo de preenchimento de uma lista e uma array com numpy. O código para essa experiência está na Figura 9.

Figura 9 – Código para comparar os tempos de preenchimento de uma lista e uma array

```
# Comparar desempenho
import numpy
import time
start_time = time.time()
lista = [0] * 1000000000
end_time = time.time()
elapsed_time = end_time - start_time
print(f'O tempo para criação de uma lista com 1 bilhão de zeros foi
{elapsed_time}')

start_time = time.time()
ndarray = numpy.zeros(1000000000)
end_time = time.time()
elapsed_time = end_time - start_time
print(f'O tempo para criação de uma ARRAY com 1 bilhão de zeros foi
{elapsed_time}')
```

O código da Figura 10 nos retorna o seguinte no console do Pycharm:

Figura 10 – Saída do console no Pycharm

```
O tempo para criação de uma lista com 1 bilhão de zeros foi 11.780919551849365
O tempo para criação de uma ARRAY com 1 bilhão de zeros foi 0.2524073123931885
```

Como podemos ver na saída do console mostrada na Figura 10, a montagem de uma lista com um bilhão de zeros levou 11,78 segundos para ser completada, enquanto a montagem de uma matriz com o mesmo um bilhão de zeros levou somente 0,25 segundos, o que é uma diferença muito significativa.

Outra forma é inserindo o tipo de dado, usando o formato utf-8 na experiência. O código fica como apresentado na Figura 11:

Figura 11 – código para criar uma matriz usando números apenas com 8 bits

```
import numpy
import time
start_time = time.time()
ndarray = numpy.zeros( shape: 1000000000, dtype='uint8')
end_time = time.time()
elapsed_time = end_time - start_time
print(f'O tempo para criação de uma ARRAY com 1 bilhão de zeros foi {elapsed_time}')
```

E a saída para essa situação é mostrada na Figura 12:

Figura 12 – Saída do console do Pycharm para criação de uma matriz com números simples com 8 bits cada

```
O tempo para criação de uma ARRAY com 1 bilhão de zeros foi 0.0

Process finished with exit code 0
```

Podemos observar que, ao montar a matriz de um bilhão de zeros, mas agora restringindo o tamanho em bits de cada dado, ganhamos mais performance ainda, uma vez que antes a matriz estava sendo gerada com dados em 64 bits cada e, no segundo caso, os números são de 8 bits. Isso significa que podemos utilizar estratégias para aumentar a rapidez com que um programa pode rodar, com algum tipo de estratégia e algum tio de algoritmo diferente. Esse tipo de dado em matrizes é muito utilizado na composição de gráficos e imagens nos programas, e tal ganho de performance é extremamente importante na construção de jogos, para que as montagens das telas não segurem o jogo como um todo. Esse tipo de análise é feita pelos construtores das *game engines* (“mecanismos de jogo”), de forma que se consiga altos índices de desempenho inclusive com gráficos mais elaborados.

2.4 Pacote PyGame

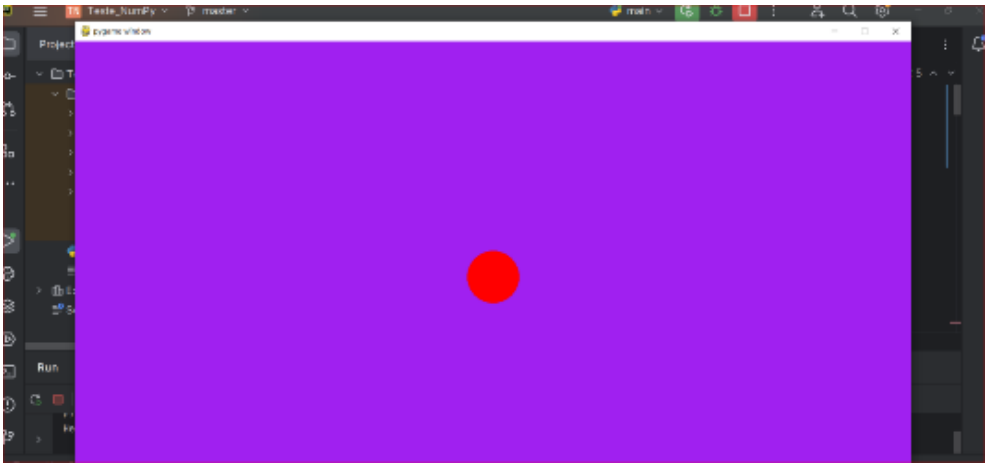
O pacote PyGame é um conjunto de bibliotecas e funções apropriadas para a construção de jogos. O conjunto possui vários comandos para as construções das telas, opções de movimentações de figuras, manejo de imagens e sons, entre outras tantas funcionalidades voltadas à criação de jogos.

Figura 13 – Página inicial do pacote PyGAme

No site oficial do Pygame (Figura 13), encontramos, também, exemplos e tutoriais sobre como utilizar os diversos comandos existentes nesse pacote.

A Figura 14 mostra uma tela na qual uma bola vermelha pode ser movida em um fundo roxo.

Figura 14 – Exemplo disponível na página do PyGame



Na página inicial, podemos ver dois exemplos básicos e funcionais, bastante simples, mas muito ilustrativos sobre como a biblioteca pode funcionar: há a montagem de uma tela roxa no fundo, em uma primeira camada de trás para frente, e uma bola vermelha em uma camada à frente. Nesse código, temos a possibilidade de mover essa bola utilizando as teclas.

Figura 15 – Teclas usadas no exemplo inicial do PyGame

	w = acima ↑	
a = esquerda ←		d = direita →
	s = abaixo ↓	

Figura 16 – Código exemplo da esfera com controle de posição

```
# Example file showing a circle moving on screen
import pygame

# pygame setup
pygame.init()
screen = pygame.display.set_mode((1280, 720))
clock = pygame.time.Clock()
running = True
dt = 0

player_pos = pygame.Vector2(screen.get_width() / 2, screen.get_height() / 2)

while running:
    # poll for events
    # pygame.QUIT event means the user clicked X to close your window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # fill the screen with a color to wipe away anything from last frame
    screen.fill("purple")

    pygame.draw.circle(screen, "red", player_pos, 40)

    keys = pygame.key.get_pressed()
    if keys[pygame.K_w]:
        player_pos.y -= 300 * dt
    if keys[pygame.K_s]:
        player_pos.y += 300 * dt
    if keys[pygame.K_a]:
        player_pos.x -= 300 * dt
    if keys[pygame.K_d]:
        player_pos.x += 300 * dt

    # flip() the display to put your work on screen
    pygame.display.flip()

    # limits FPS to 60
    # dt is delta time in seconds since last frame, used for framerate-
    # independent physics.
    dt = clock.tick(60) / 1000

pygame.quit()
```

No código da Figura 16, podemos alterar algumas características de modo a verificar que diferentes efeitos obtemos. Podemos alterar a cor do fundo, mudando o nome da cor no comando screen.fill("purple") para “green” (verde) ou “gray" (cinza). Poderíamos mudar as teclas que movimentam o círculo ou a cor do próprio círculo. No comando pygame.draw.circle(screen, "red", player_pos, 40) podemos mudar a cor da bola ou seu tamanho. Pode ser nas cores “grey” (cinza), “blue” (azul), “yellow” (amarela), ou qualquer outra.

Esses exemplos e explicações apresentados são extremamente úteis para nosso aprendizado e aprofundamento no uso do pacote. No site do PyGame existem tutoriais sobre como criar jogos do começo ao fim, e até instruções avançadas. Existe a possibilidade de comprar um *e-book* por U$ 3,00 (três dólares). Esse livro contém diversas possibilidades para aprofundamento no uso da biblioteca. Embora não seja obrigatório nem

imprescindível, pois conseguimos um bom grau de usabilidade dos comandos sem o livro, ele pode ser muito interessante para conhecermos técnicas mais apuradas de seu manejo.

2.5 Pacote TKInter

Há vários pacotes naturalmente integrados ao Python e será sempre necessário importar essas bibliotecas. No entanto, esses pacotes integrados ao interpretador do Python não precisam ser instalados. A grande maioria das bibliotecas possuem um manual ou documentação, ou seja, um site onde estão explicados todos os comandos que podem ser utilizados e exemplos de implementação. Isso torna muito acessível a utilização das bibliotecas e a imensa possibilidade de funcionalidades que podem ser implementadas na programação em Python. Nessas documentações, existem vários exemplos, todos os comandos possíveis do pacote estão disponíveis e, às vezes, há cursos breves para o desenvolvedor aprender a utilizar plenamente o conjunto de instruções do pacote.

No site python.org estão as documentações das bibliotecas integradas ao núcleo do Python. Nesse ambiente, há exemplos que podem servir como pontos de partida para o desenvolvedor aprofundar seu conhecimento em relação ao uso da biblioteca. Um exemplo é a biblioteca TKInter, com uma parte de sua documentação na Figura 17, usada para construir janelas de interface do programa que estivermos construindo.

Figura 17 – Manual do TKinter dentro do site python.org

Na Figura 18, há um exemplo de código “Olá Mundo” do manual do tkinter.

Figura 18 – Um primeiro exercício usando o TKInter no Python

Nesse exemplo a pequena janela exibe uma “label” na primeira linha com a frase “Hello World” , um botão com a palavra “Quit” e, na segunda linha, a frase “Teste linha 2 coluna 0 !”. A segunda linha foi inserida por ocasião desse exemplo, baseado na estrutura mostrada na documentação.

Outro exemplo, um pouco mais complexo, é o de um relógio em tempo real usando uma janela modelada pelo TKInter (Figura 19).

Figura 19 – Janela TKInter com um relógio em tempo real

Na Figura 20, é mostrado o código em Python que faz o relógio ser atualizado uma vez a cada mil milisegundos, que é o mesmo que um segundo. Além disso, o código forma a janela com um espaço para exibir os valores do horário em letras brancas com um fundo preto. Todas essas características podem ser alteradas nos parâmetros passados para os métodos apropriados de cada aspecto da janela.

Figura 20 – Código Python para o relógio em tempo real e a janela TKInter

```
import tkinter as tk
from time import strftime

def atualizar_tempo():
    tempo_agora = strftime('%H:%M:%S %p')
   rotulo_tempo['text'] = tempo_agora
    root.after(1000, atualizar_tempo)  # Atualiza a cada 1000
milissegundos (1 segundo)

# Criar a janela principal
root = tk.Tk()
root.title("Relógio em Tempo Real")

# Criar o rótulo para exibir o tempo
rotulo_tempo = tk.Label(root, font=('calibri', 40, 'bold'),
background='black', foreground='white')
rotulo_tempo.pack(anchor='center')

# Chamar a função para atualizar o tempo
atualizar_tempo()

# Iniciar o loop da interface gráfica
root.mainloop()
```

TEMA 3 – INTRODUÇÃO AO GIT

O Git é um VCS (*version control system*), ou sistema de controle de versão distribuído e amplamente utilizado para rastrear alterações no código fonte durante o desenvolvimento de um projeto de software. Foi criado por Linus Torvalds em 2005, o mesmo criador do kernel Linux (o núcleo do sistema operacional de código aberto Linux). O Git é projetado para ser eficiente, flexível e escalável, sendo utilizado por desenvolvedores individuais e equipes de programadores.

Algumas características importantes do Git incluem:

- Controle de versão distribuído: Cada desenvolvedor possui uma cópia local do repositório Git completo. Isso permite que as alterações sejam feitas offline e posteriormente sincronizadas com outros repositórios.
- *Branching* e *merging*: O Git facilita a criação de ramos, chamados de *branches*, permitindo que os desenvolvedores trabalhem em diferentes funcionalidades ou correções de *bugs* simultaneamente. Esses ramos podem ser mesclados ou misturados, operação conhecida como ‘merge’, para incorporar as alterações de volta ao ramo principal.

- Histórico de alterações: O Git mantém um histórico completo de todas as alterações feitas no código ao longo do tempo. Isso possibilita rastrear quem fez o quê, quando e por quê.
- Repositórios remotos: Os repositórios Git podem ser hospedados em servidores remotos, como o GitHub, possibilitando a colaboração entre desenvolvedores que não precisam estar nem mesmo na mesma cidade.
- Desempenho: Git é conhecido por sua velocidade e eficiência no gerenciamento de grandes projetos com muitos arquivos e históricos extensos.

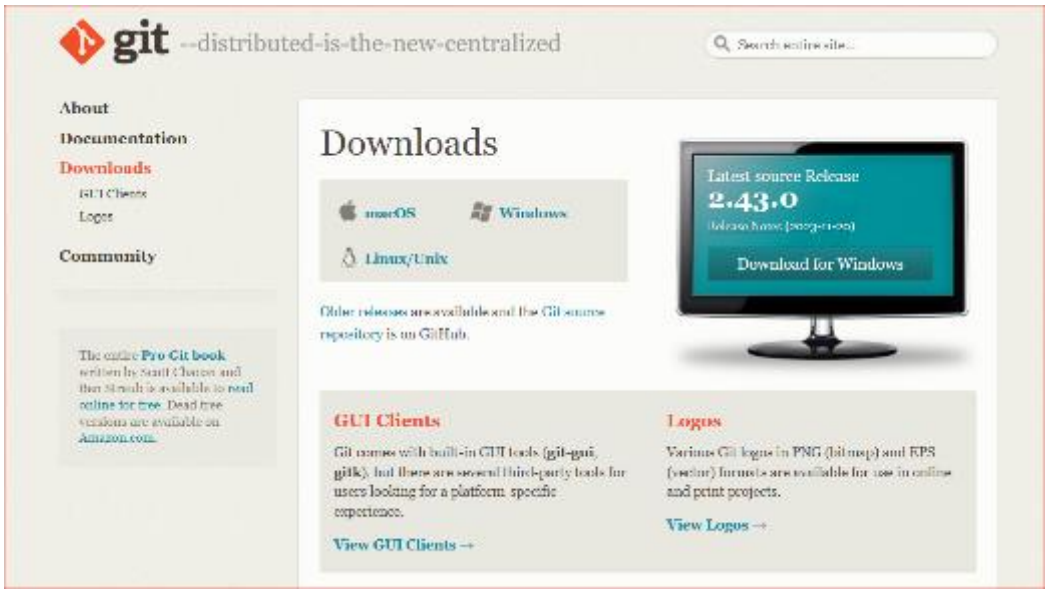
Ao utilizar o Git, os desenvolvedores podem colaborar de maneira eficaz, acompanhar o progresso do projeto, reverter alterações indesejadas e gerenciar diferentes versões do código-fonte de forma organizada.

3.1 Instalação e configuração inicial

A primeira ação necessária para o uso do Git é a instalação. Isso pode ser feito a partir do *download* do arquivo de instalação diretamente do site oficial do Git. A Figura 7 mostra o aspecto da página inicial no ano de 2024, momento da escrita deste material. Vale ressaltar a extrema importância de baixar o arquivo única e exclusivamente do site oficial. Por mais confiável que um site possa parecer, é altamente recomendável usar somente esta fonte, pois pode ocorrer que algum site estabeleça sua própria versão com adição ou supressão de funcionalidades que nem sempre são desejáveis. Podem ocorrer, ainda nesse sentido, alterações no software que dificultem seu uso e a devida coerência com a documentação oficial do fabricante do Git. Isso vale para a instalação de qualquer software: sempre baixar do site do fabricante do software, nunca de sites que oferecem *downloads* de cópias, pois podem conter códigos maliciosos em seus conjunto de arquivos.

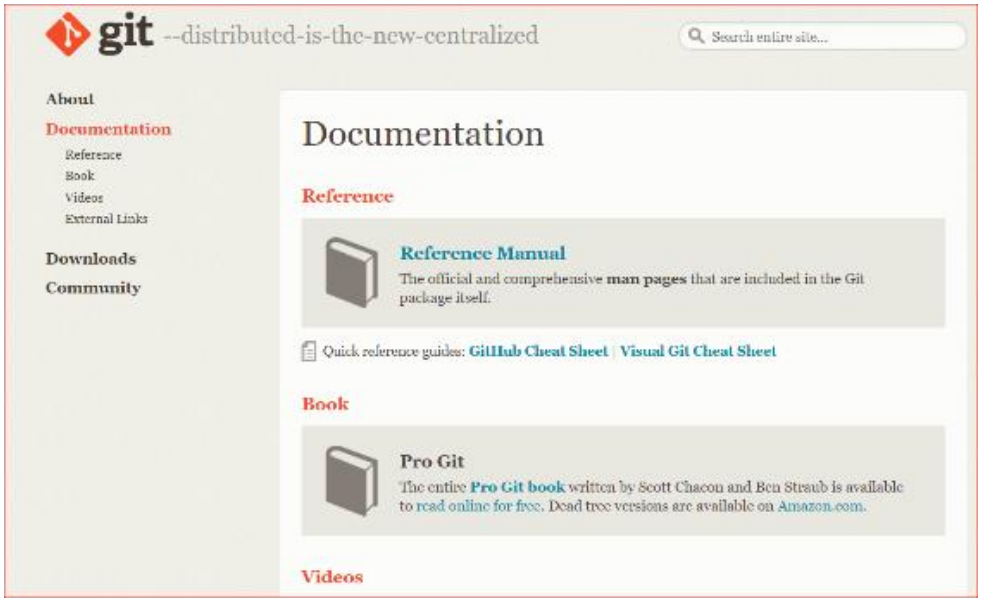
No site oficial do Git existem diversas informações para a perfeita instalação e configuração do sistema, bem como documentação sobre as funcionalidades e opções para o melhor uso deste (Figura 21).

Figura 21 – Aspecto da página inicial do Git



No site oficial do Git existe a documentação completa do sistema, além de livros e vídeos relacionados com esse ambiente. São informações que vão dos princípios mais básicos aos mais avançados. Uma parte do que está disponível no site do Git é mostrado na Figura 22.

Figura 22 – Página oficial do Git com as documentações sobre o sistema



O livro *Pro Git* é gratuito e pode ser baixado também diretamente do site oficial do Git. Esse *e-book* contém todos os detalhes sobre o Git e seu uso. Embora esteja em inglês, podemos utilizar um tradutor para alguns trechos, se necessário. É extremamente importante que o estudante busque uma

compreensão aprofundada para poder utilizar o sistema e assim dominar a sua lógica de trabalho.

3.1 Conceitos básicos do Git

De modo geral, podemos entender o Git como uma metodologia, pois a finalidade do uso desse sistema é muito direta e específica. Dessa forma, é necessário olharmos para alguns conceitos básicos presentes no Git. Os primeiros conceitos são detalhados a seguir.

- Repositório: A ideia mais básica do uso de um sistema de controle de versão é o repositório. Isso porque é diferente da ideia de diretório, pois o repositório será a pasta que contém tudo que for componente do projeto. Além disso, o repositório conterá as informações sobre as versões na medida em que ocorrerem. Isso significa que a pasta de repositório contém os arquivos e as informações sobre esses arquivos de forma encadeada logicamente, de modo que os colaboradores que tiverem acesso podem saber qual a história das partes ou do todo de um projeto. Isso inclui as partes que são usadas e as que não são, com registros sobre porque foram ou não utilizadas.
- *Commit*: O conceito de *commit* pode ser entendido como “deixar em guarda de’, ou seja, de depositar os arquivos relativos ao projeto como se fossem valores depositados em um banco. De certa forma, os arquivos de um projeto controlado por versão é um valor muito importante, e o controle de versão é fundamental para a evolução do sistema em construção e consumo por parte dos usuários quando estiver disponível para isso. Podemos, portanto, entender a operação de *commit* com o depósito do arquivo no repositório
- *Branch*: O *branch* pode ser pensado com um galho de uma árvore ou um tronco com os galhos crescendo ao longo do tempo. Esse conceito está relacionado com o fato de que um sistema computacional vai crescendo quando é desenvolvido, e evolui quando já está sendo utilizado pelos interessados. Podemos pensar, ainda, nas possibilidades de testes de módulos que não sejam utilizados na versão final, mas que existiram nos testes durante o desenvolvimento. Essas partes desenvolvidas e testadas podem não fazer parte da versão final, mas são importantes

para a história do desenvolvimento, podendo ser revisitadas e analisadas novamente tempos após a versão final estar sendo utilizada. Essa pode ser considerada como uma importante motivação para a estruturação de sistemas de controle de versão, pois os testes feitos no desenvolvimento são super importantes. Os *branches* são os caminhos percorridos no tempo do desenvolvimento do programa. Podemos decidir por criar mais do que um caminho para nosso desenvolvimento, de acordo com as experiências que desejarmos fazer em função desse desenvolvimento.

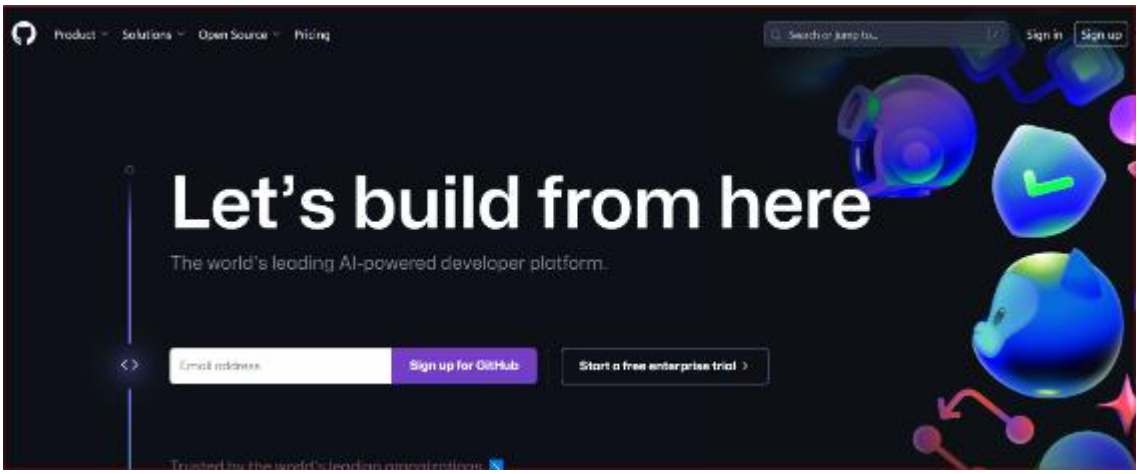
- *Merge*: É a junção de um pedaço do software com o corpo principal. Essa junção, no ambiente do Git, não é uma junção simples. O que ocorre é que as modificações são incorporadas de forma inteligente ao corpo principal do programa. Desse modo, não haverá repetições de partes de código nem acumulações desnecessárias.
- *Push*: é um comando para enviar uma *branch* para o GitHub, onde está o repositório compartilhado na internet e segue a mesma filosofia do Git, mas permite que todos os colaboradores do projeto depositem suas contribuições, com todas as informações das versões inclusas.

TEMA 4 – INTRODUÇÃO AO GITHUB

O GitHub é uma plataforma de desenvolvimento de software baseada em nuvem que oferece serviços de controle de versão usando o sistema Git. Criado em 2008, o GitHub se tornou fundamental para colaboração e gerenciamento de código-fonte em projetos de software. Ele permite que desenvolvedores colaborem em equipes distribuídas geograficamente, rastreiem alterações no código, gerenciem problemas, revisem códigos e coordenem o desenvolvimento de software de maneira eficiente.

Além disso, o GitHub oferece recursos como integração contínua, hospedagem de páginas da web e uma variedade de ferramentas para aprimorar o fluxo de trabalho de desenvolvimento. Sua interface amigável facilita a navegação pelos repositórios e o acesso a informações cruciais sobre um projeto. A página inicial do GitHub é mostrada na Figura 23.

Figura 23 – Página inicial do GitHub em 2024



O GitHub é amplamente utilizado por desenvolvedores, empresas e organizações de todos os tamanhos para desenvolver, colaborar e compartilhar código de maneira eficiente, promovendo a transparência e a inovação na comunidade de desenvolvimento de software.

A utilização do GitHub é igual à do Git, com a única diferença de que os diretórios ficam alocados na nuvem, ou seja, não ficam no computador do usuário. Por isso, há algumas configurações que se fazem necessárias para garantir o bom funcionamento e a segurança das informações.

Para utilizar o GitHub é necessário criar uma conta e estar devidamente logado a ela para poder criar e gerenciar os repositórios utilizados. Existem vários tutoriais que ajudam a utilizar o GitHub. Um deles é um “Hello World” oferecido quando entramos em nossa conta, mesmo antes da criação de qualquer repositório ou outra ação qualquer.

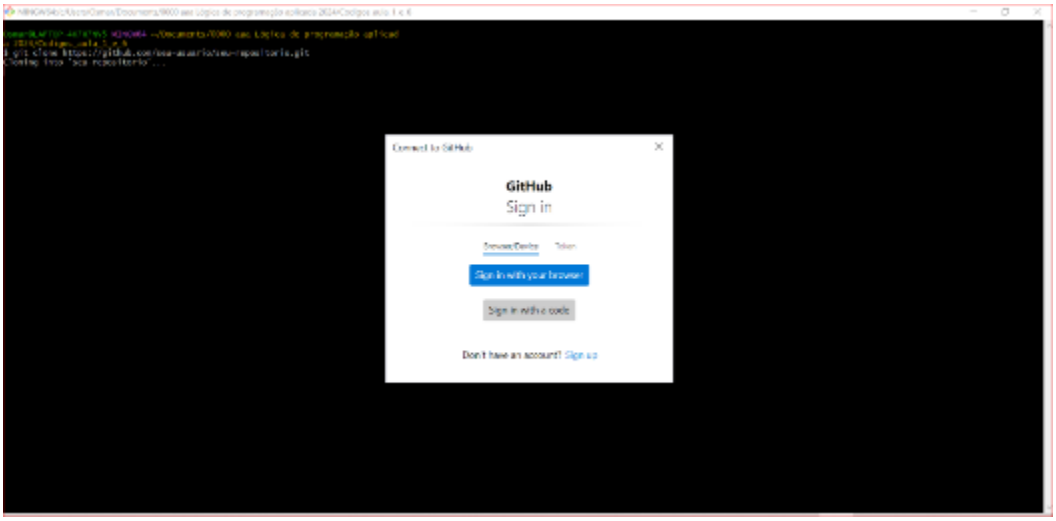
Figura 24 –Tutorial inicial na página de entrada do GitHub

Na Figura 24 há um tutorial inicial de apresentação para os primeiros contatos com as funcionalidades e filosofia de trabalho do GitHub. Na imagem mostramos que houve tradução do inglês para o português, utilizando o botão direito do mouse e clicando em “traduzir do inglês”. O vídeo de apresentação tem locução em inglês e tem legendas em espanhol. Apresenta somente uma apresentação sem detalhamento técnico ou operacional, mas é uma primeira vista sobre de que se trata o GitHub.

O fluxo para estabelecer um projeto alocado em GitHub com vários usuários (dois ou mais) pode ser implementado seguindo este fluxo:

1. Criar um repositório no GitHub:
   - Acesse o GitHub e clique em “New repository”.
   - Preencha as informações do repositório, como nome, descrição, e escolha se deseja inicializar com um README.
2. Clone o repositório criado localmente. Isso pode ser feito por meio do Git bash, que é uma tela para receber e executar linhas de comando próprias do Git: git clone[1] (Figura 25).

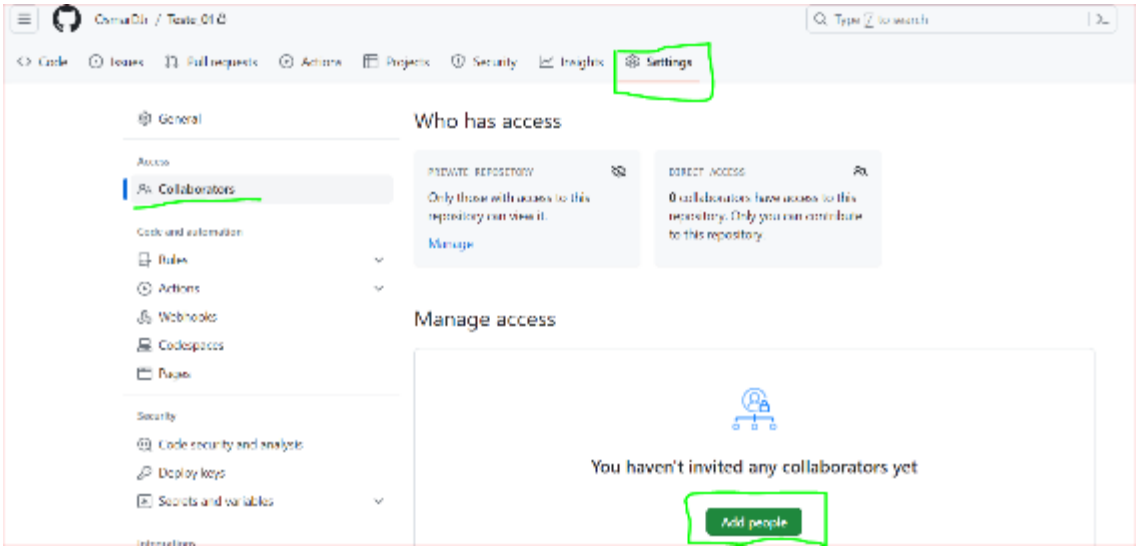
Figura 25 – Aspecto do Git bash (local) e a solicitação de login na conta Git



   Para que a clonagem possa acontecer é necessário que a conta no GitHub esteja criada e acessível, uma vez que o Git bash irá acionar o procedimento de login na conta e iniciar o processo de clonagem.
3. Configure colaboradores: Acesse as configurações do repositório no GitHub. Isso pode ser feito em “Manage access” e convide os colaboradores pelo nome de usuário ou e-mail (Figura 26).

[1] Disponível em: https://github.com/seu-usuario/seu-repositorio. Acesso em:29 fev. 2024.

Figura 26 – Aspecto da página do GitHub para adicionar colaboradores ao projeto



A adição de outros colaboradores é opcional, e pode ser feita a qualquer momento depois que o repositório é criado. Da mesma forma, pode-se excluir colaboradores a qualquer tempo.

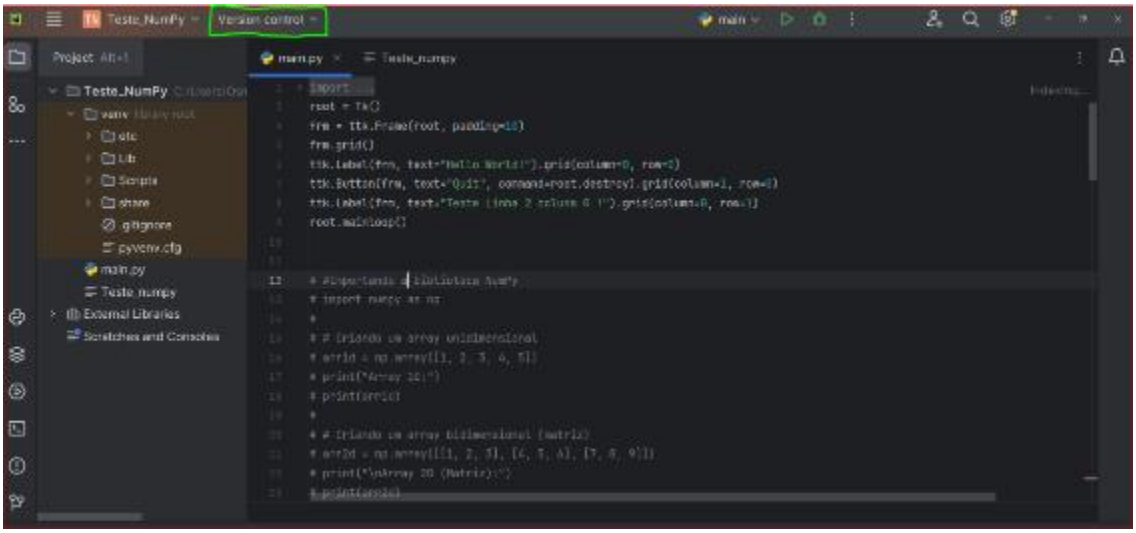
Outro aspecto que deve ser lembrado é que o repositório pode ser clonado mediante configuração prévia. Mas essa possibilidade abre um caminho para um conceito conhecido como "*fork*" que, em tradução literal, significa "garfo". Esse é o caso de quando fizermos um projeto baseado em algum outro projeto já existente – por exemplo, o sistema de controle de banco de dados MariaDB é um *fork* do MySQL, criado por um dos desenvolvedores desse último quando passou a ser comercializado, diferentemente do MariaDB, que é sustentado pela comunidade e continua sendo gratuito. O *fork* é a ideia de "dar uma garfada" em algum projeto que já existe para alterar alguma de suas características, ainda que seja jurídica. Não deve ser confundido com cópia, pirataria ou plágio. A diferença é que o *fork* possui as devidas autorizações para acontecer, enquanto as forma ilegais são clandestinas e não autorizadas.

TEMA 5 – VERSIONAMENTO COM PYCHARM

O ambiente de desenvolvimento Pycharm permite que os códigos desenvolvidos em Python sejam integrados no Git e GitHub de forma extremamente simples e direta. Existe uma aba "Version Control" que permite a

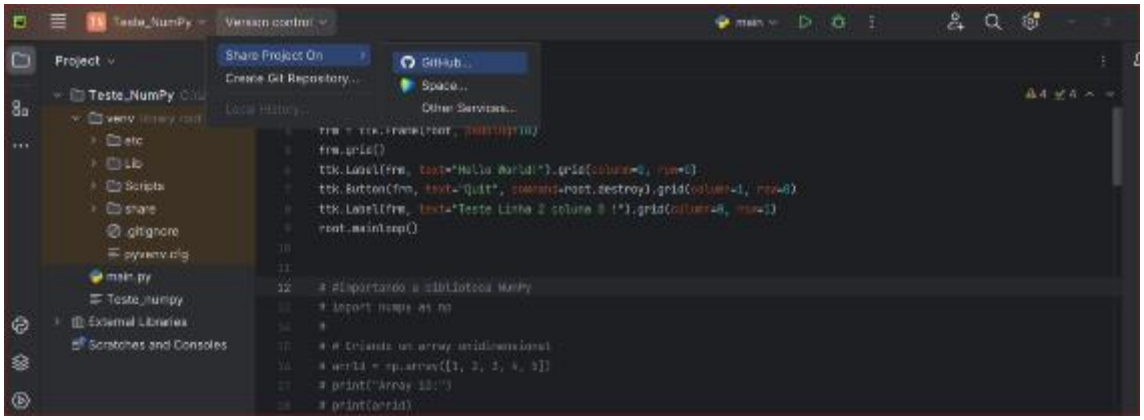
execução de um *commit* e *push* diretamente de dentro desse ambiente (Figura 27).

Figura 27 – Atalho para controle de versão no Pycharm



Nessa aba, pode-se criar um repositório no Git (local) ou fazer o ciclo de *push* diretamente no GitHub. No exemplo da Figura 13, o projeto chama-se Teste_Numpy e, ao clicar em ***share on*** GitHub, o Pycharm providencia para que o projeto crie um *branch master*, ou um “tronco mestre” de forma que, na sequência, poderão ser criados outros *branches*, *commits* e demais encaminhamentos diretamente de dentro do próprio Pycharm, como mostrado na Figura 28.

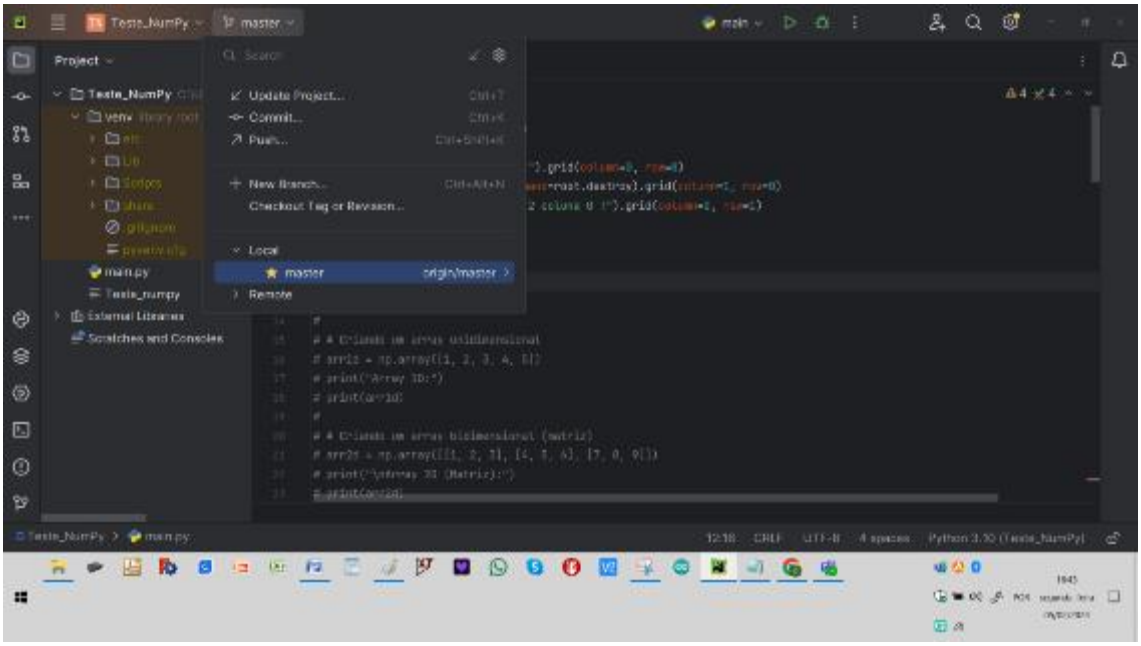
Figura 28 – Opções para compartilhamento e controle de versão



A Figura 29 mostra a disponibilidade dos comandos básicos do Git (*commit*, *branche*, *Update*, entre outros) agora disponíveis diretamente dentro do Pycharm. Isso significa que o desenvolvedor poderá trabalhar no Pycharm e

já colaborar diretamente no GitHub, tornando o trabalho incrivelmente prático e produtivo.

Figura 29 – Opções integradas para o projeto já integrado no repositório



FINALIZANDO

Nesta etapa, abordamos algumas bases necessárias para o aprofundamento no conhecimento da programação. Percorremos as bibliotecas e mostramos alguns exemplos. Fizemos uma introdução no pacote NumPy, que permite manipular elementos matemáticos com bastante eficiência e simplicidade. Essa biblioteca é usada, pelos mesmos motivos, em outra biblioteca que vimos, a PyGame. Esta última nos permitiu uma excelente entrada em uma das mais promissoras aplicações da programação: os jogos eletrônicos. Complementarmente conhecemos outra biblioteca que manipula elementos matemáticos, mas de forma gráfica: a Plotly. Esse pacote permite expressar em gráficos as sequências de dados que estiverem programadas, inclusive extensas séries de dados. Vimos ainda a biblioteca TKInter, que possibilita a execução de nossos programas em interfaces gráficas, conhecidas como janelas. Pudemos estudar, ainda que superficialmente, algumas características que podem ser programadas para termos uma interface atraente para o usuário.

Depois, tivemos um contato com as funcionalidades e com a filosofia de trabalho de um sistema de controle de versão que é o Git. Tivemos nossos

primeiros contatos com seus conceitos mais básicos, como repositório – o lugar onde os diretórios ficam armazenados com as demais informações de mudanças e os tempos dessas mudanças; o *commit*, que é a ação de carregar os arquivos ou mudanças em um software em desenvolvimento; e, o *branch* que é o caminho por onde o projeto está evoluindo, formando uma estrutura semelhante à de uma árvore, com seu tronco e galhos.

Com o Git, avançamos nas ideias de controle de versão de nossos projetos para o compartilhamento e possibilidade de contribuição por pessoas que não precisam estar no mesmo lugar – nem mesmo no mesmo país. Tivemos contato com o GitHub, o qual leva todas as funcionalidades do controle de versão local para um ambiente na internet, em nuvem. Uma vez que o repositório está alocado em nuvem, usuários de todas as partes, inclusive em grandes números, podem adicionar contribuições ao projeto de forma ordenada e segura para todos.

REFERÊNCIAS

BEHRMAN, K. R. **Fundamentos de Python para ciência de dados**. Porto Alegre: Bookman, 2022.

CHACON, S. STRAUB, B. **Pro Git.** Version 2.1.416. (S.l.): Apress, 2024. Disponível em: https://git-scm.com/book/en/v2. Acesso em: 29 fev. 2024.

FERREIRA, A. G. **Design patterns e gerência de configuração**: do projeto ao controle de versões. São Paulo: Saraiva, 2021.

LAMBERT, K. A. **Fundamentos de Python**: primeiros programas. Tradução Edson Furmankiewicz. 1. ed. São Paulo: Cengage Learning, 2022.

NUMPY. Disponível em: https://numpy.org/. Acesso em: 29 fev. 2024.

PLOTLY. Disponível em: https://plotly.com/. Acesso em: 29 fev. 2024.

PYGAME. Disponível em: https://www.pygame.org/news. Acesso em: 29 fev. 2024.

SHAW , Z. A. **Aprenda Python 3 do jeito certo**. Rio de Janeiro: Alta Books, 2019.

TKINTER. Disponível em: https://docs.python.org/pt-br/3/library/tkinter.html. Acesso em: 29 fev. 2024.