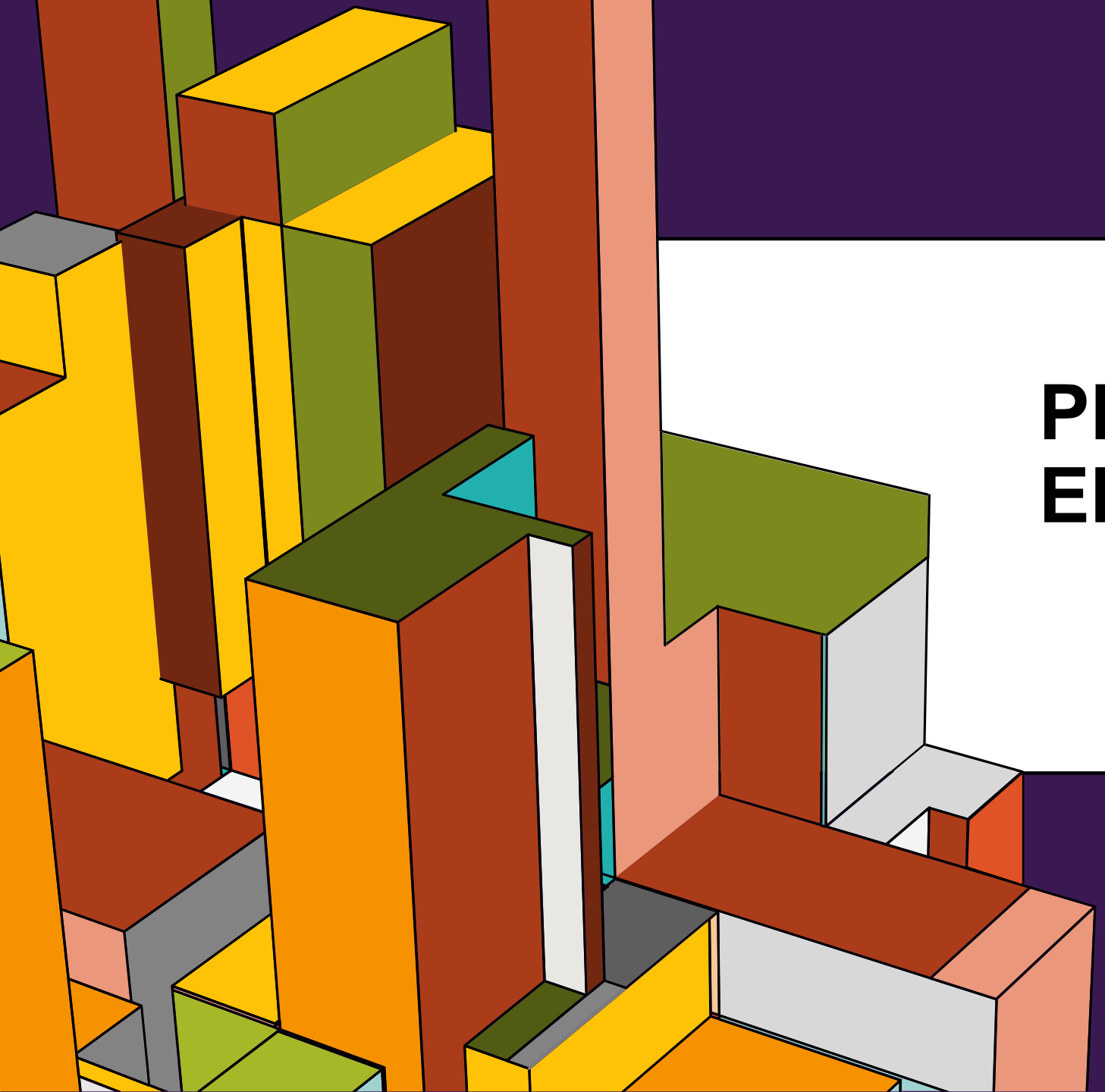


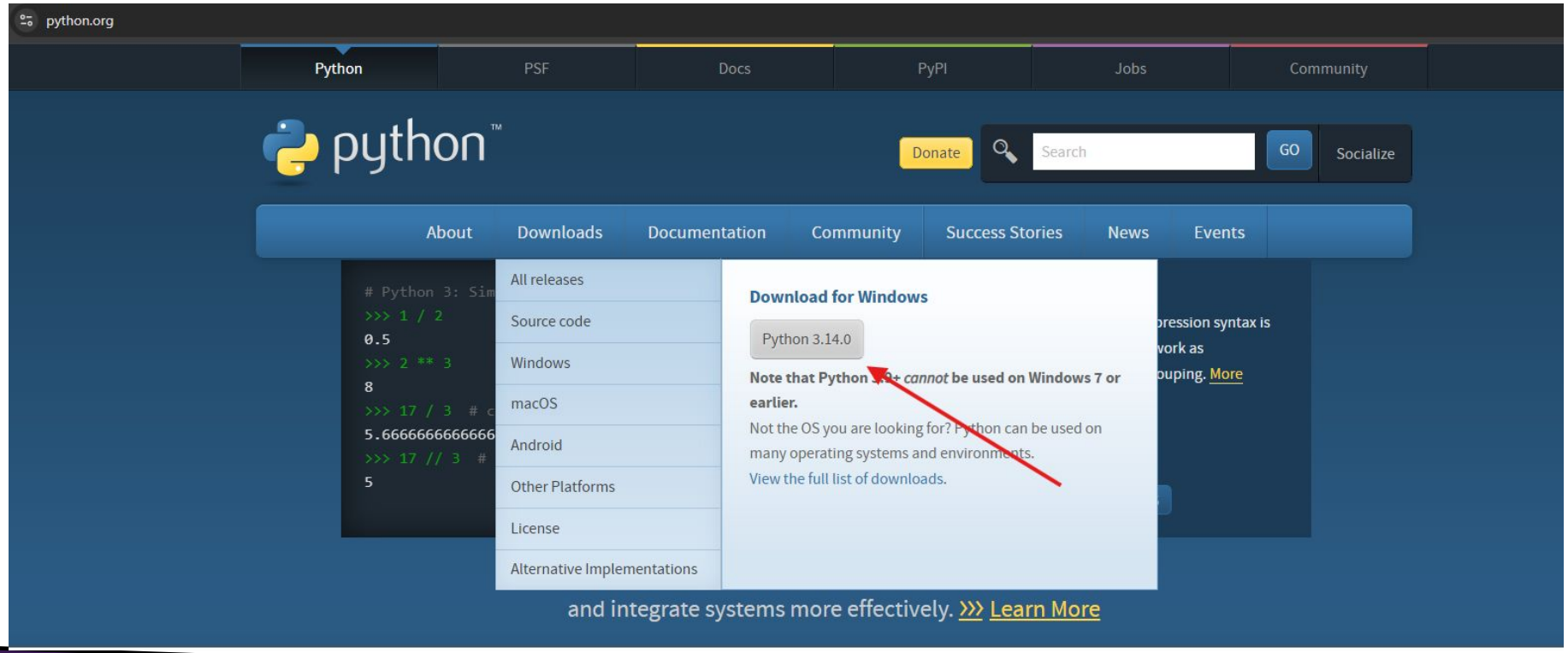
PROGRAMAÇÃO EM PYTHON

python

SENAI

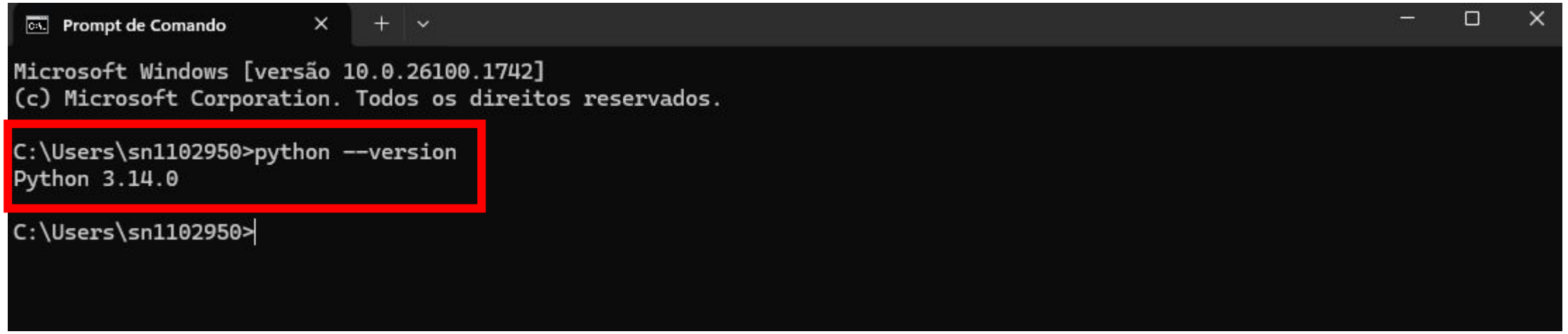
INSTALANDO O PYTHON

1. Faça download do Python na pagina oficial (www.python.org)
2. Execute o instalador e marque a opção Add Python 3.14.0 to PATH
3. Clique na opção Install Now



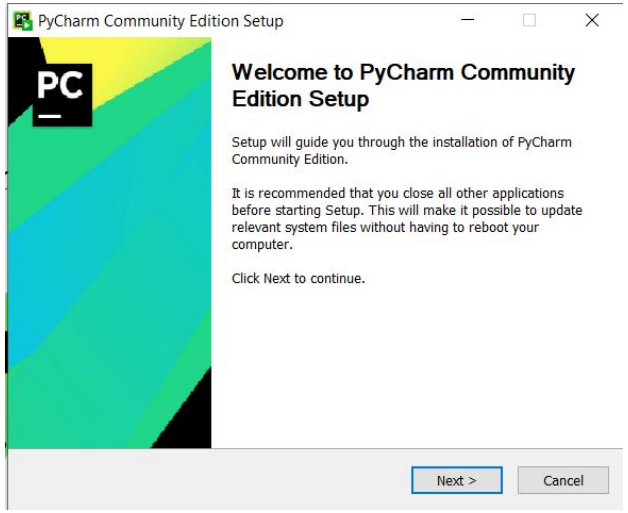
VERIFICANDO A INSTALAÇÃO

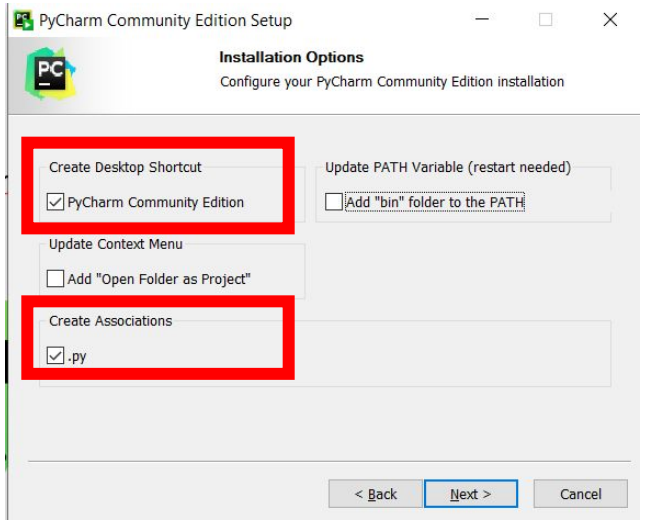
1. Abra o Prompt de comando
2. Digite o comando python --version



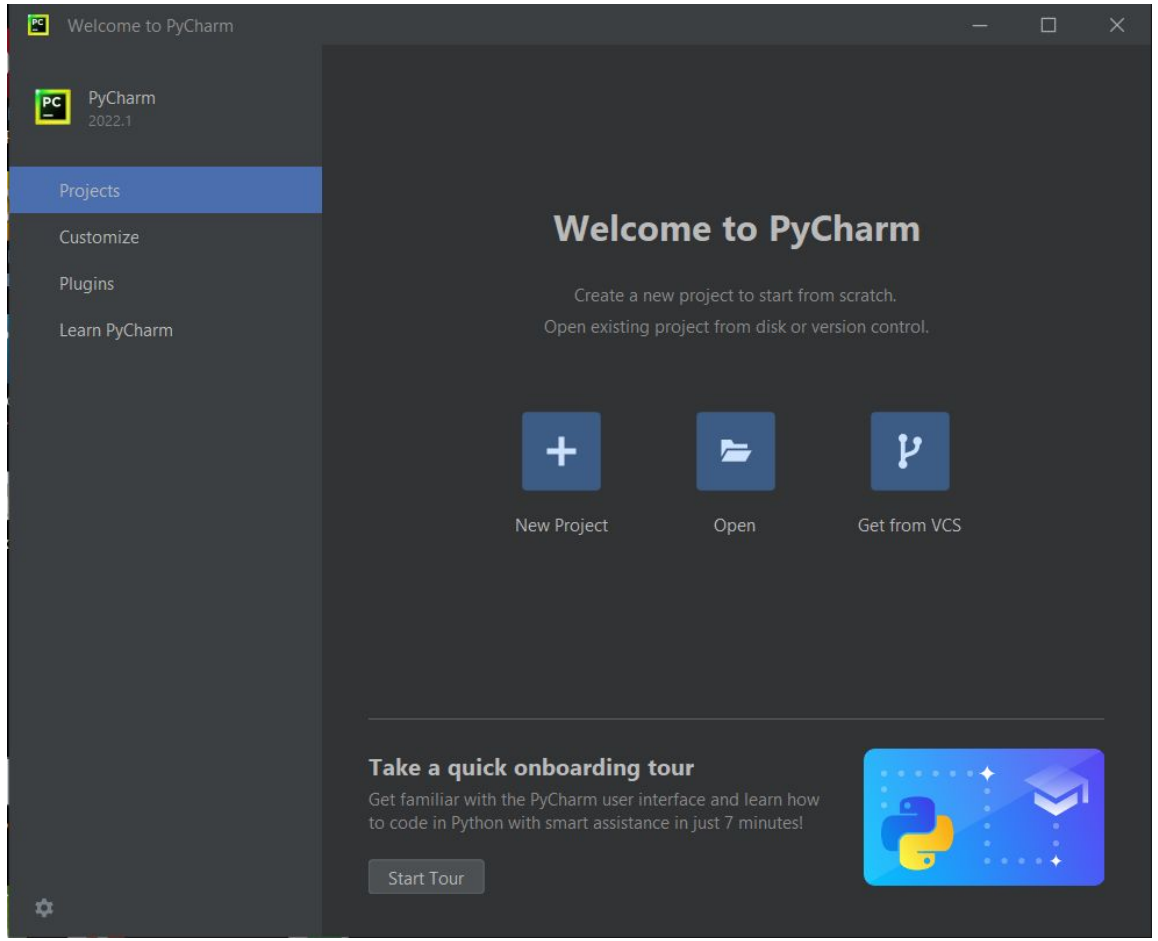
INSTALANDO A IDE PYCHARM

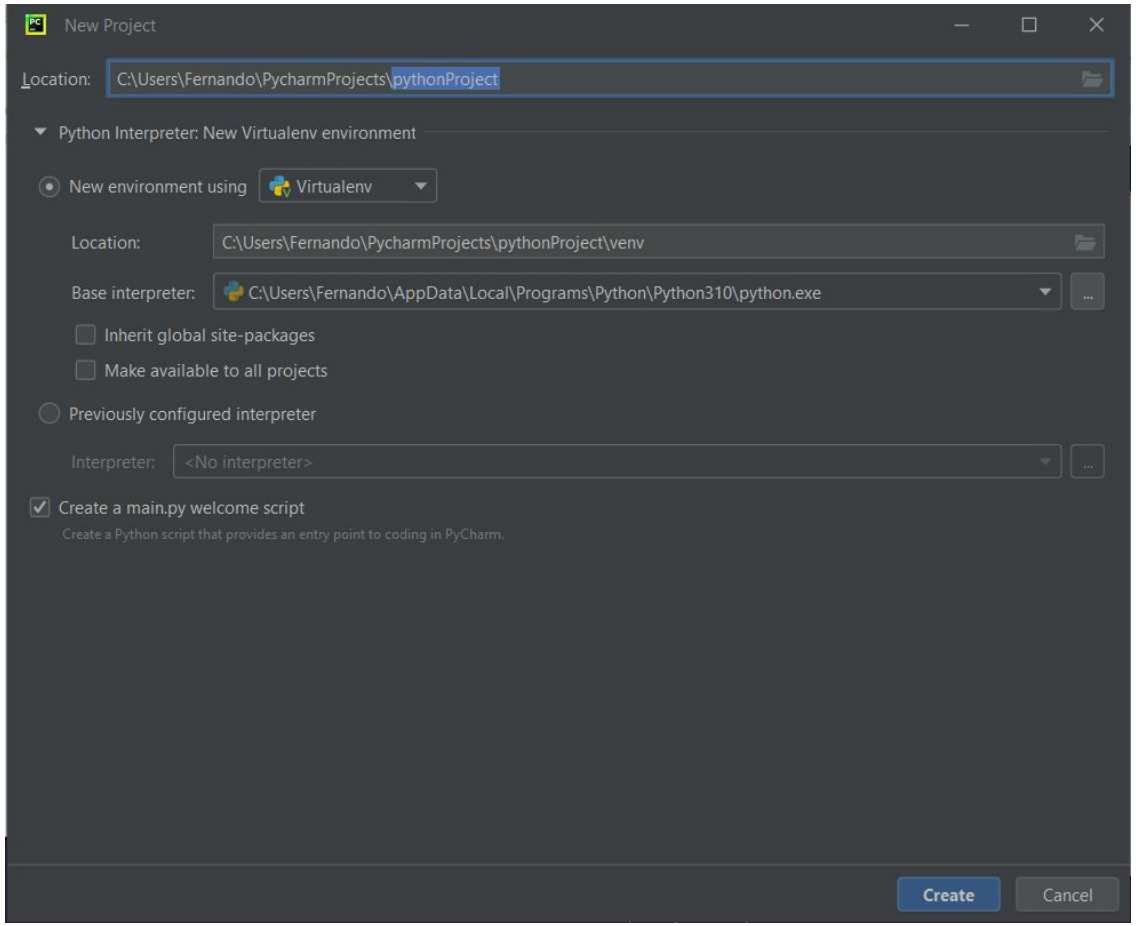
O PyCharm fornece complementação de código inteligente, inspeções de código, realce dinâmico de erros e correções rápidas, juntamente com melhorias de código automatizada e recursos de navegação avançados.



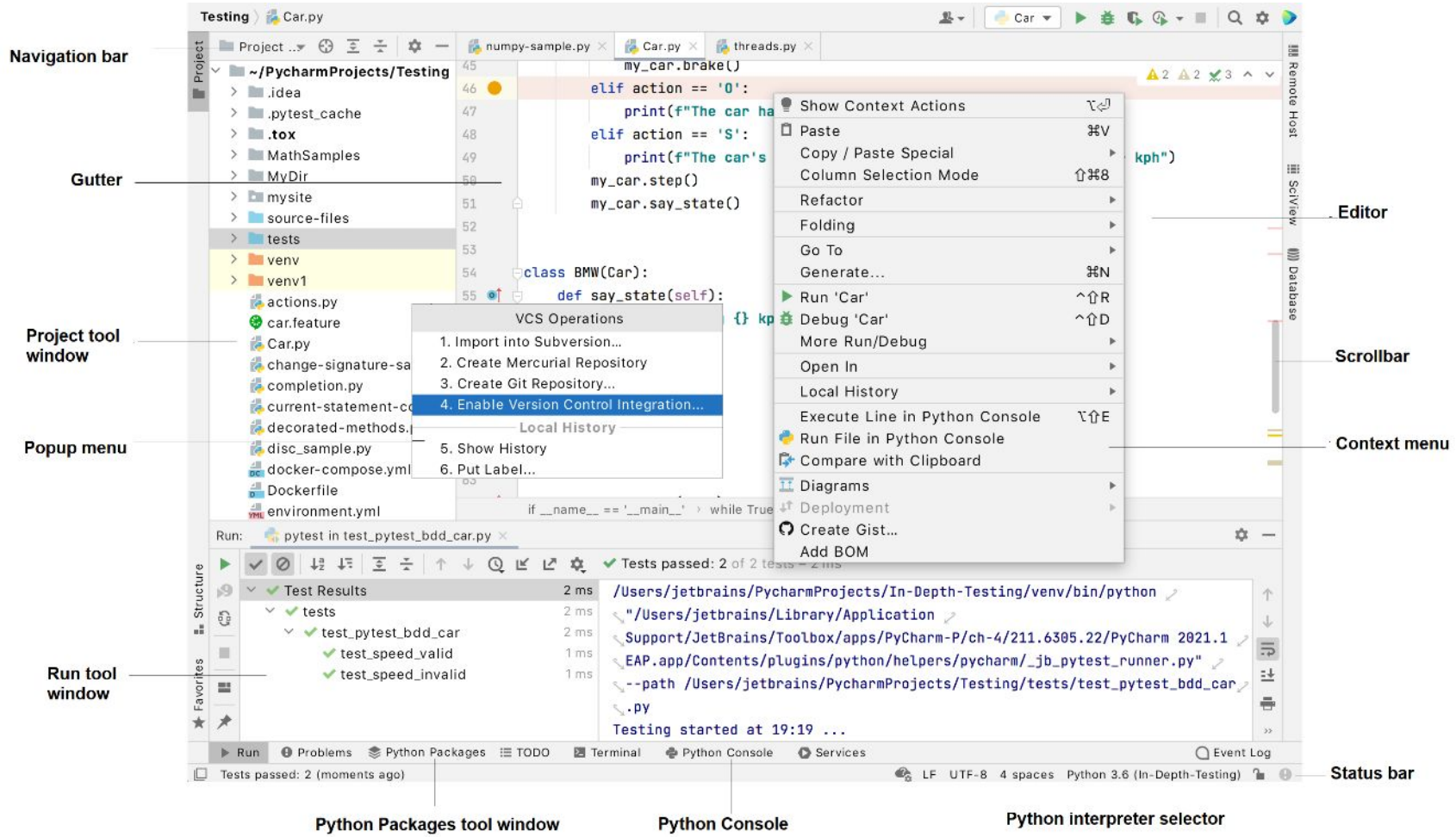


INICIANDO UM PROJETO NO PYCHARM

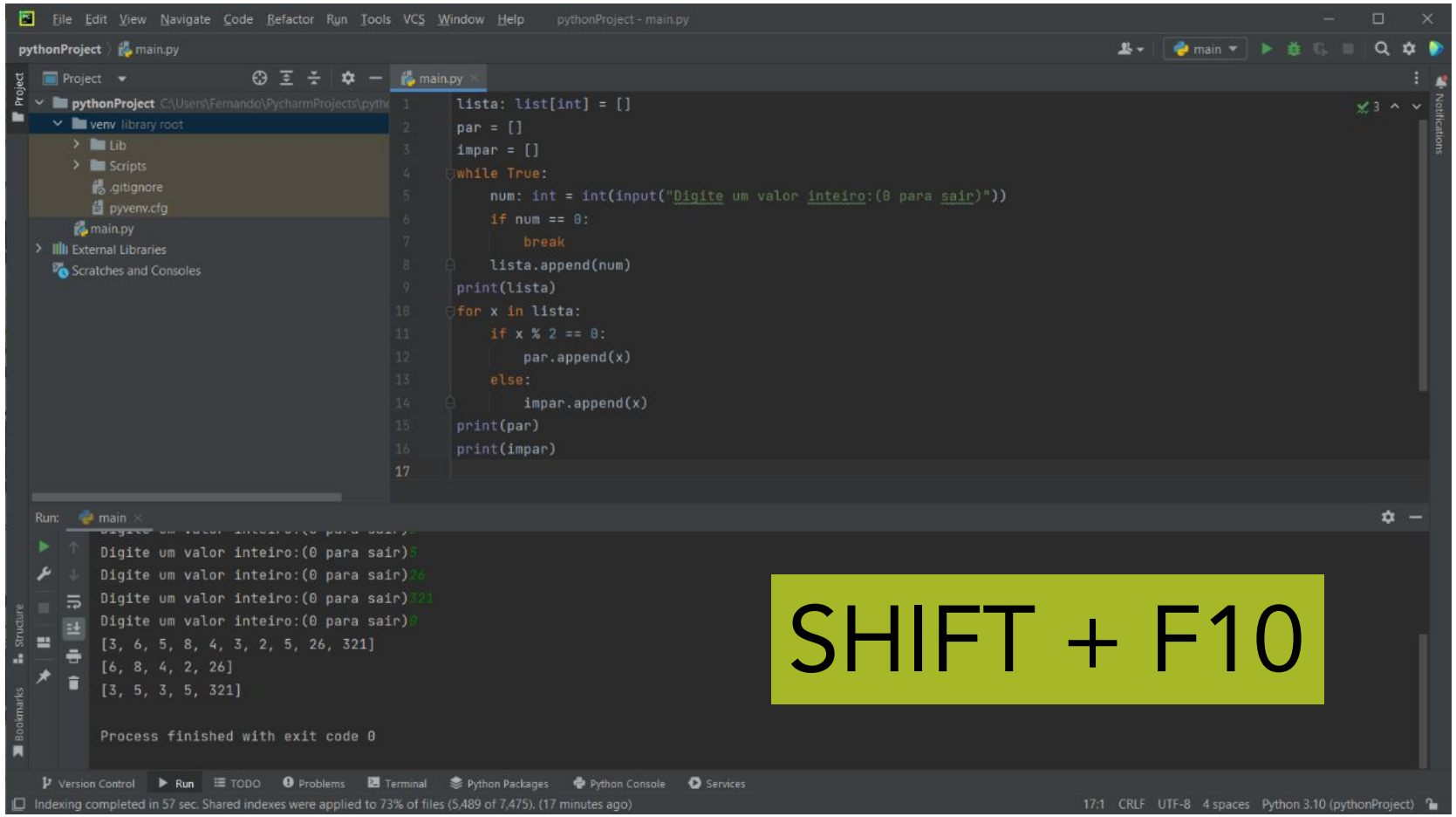




CONHECENDO O PYCHARM



RODANDO O PRIMEIRO PROJETO



FUNÇÕES

De acordo com Downey (2015), uma função é uma sequência nomeada de instruções que executa uma determinada operação, sendo criada a partir de um nome e de instruções que a compõe, possibilitando ser utilizada posteriormente.

Funções são blocos de códigos que possuem um nome e podem ou não receber parâmetros.

Para criar uma função usamos a instrução def.

EXEMPLOS

```
def nomeDaFuncao(parametro):
    print('Esta função recebeu ',
parametro, ' como parâmetro. ')
    print('O tipo deste parâmetro , é: ',
type(parametro))
    return '\nCurso de Python é top!'
print(nomeDaFuncao('string'))
```

O bloco de código da função não é executado enquanto a mesma não for chamada

EXEMPLO 1

```
def soma(n1,n2):
    soma = n1+n2
    return soma
def subtracao(n1,n2):
    sub = n1-n2
    return sub
print('Bem vindo a calculadora')
num1=int(input('Digite o primeiro numero: '))
num2=int(input('Digite o segundo numero: '))
op = int(input('Digite 1 para somar e 2 para subtrair'))
if op == 1:
    print(f'O valor de soma {num1} e {num2} é {soma(num1,num2)}')
elif op ==2:
    print(f'O valor da subtração {num1} e {num2} é
{subtracao(num1,num2)}')
else:
    print('Comando invalido!!')
```

FUNÇÕES RECURSIVAS

- Chamamos de função recursiva uma função que chama a si mesma.
- Recursividade pode ser um problema se gerar chamadas infinitas.
- O Python tem um limite de execuções recursivas por padrão igual a 1 000 ou seja, uma função recursiva não vai ficar rodando infinitamente.

UMA FUNÇÃO QUE CHAMA OUTRA FUNÇÃO

https://www.calculadora.app/matematica/fatorial/#google_vignette

```
def fatorial(numero):
    if type(numero) is not int:
        return '\nO número deve
ser um inteiro!'
    if numero ⩽ 0:
        return 1
    else:
        return numero *
fatorial(numero-1)
print(fatorial(8))
```

A partir do momento em que uma função é criada, é possível utilizá-la dentro de outras funções e, inclusive, dentro dela mesma.
Esse conceito é base para recursividade de funções. A seguir é mostrado um exemplo de uma função recursiva que calcula o fatorial de um número “n”.

VARIÁVEIS GLOBAIS E LOCAIS

- Ao usarmos funções, começamos a trabalhar com variáveis locais e globais. Uma variável local, declarada dentro de uma função, existe apenas dentro desta função, sendo inicializada a cada chamada à função.
- Desta forma, ela não é visível fora da função. Uma variável global é definida fora de uma função, podendo ser vista por todas as funções do módulo e por todos os módulos que importam o módulo que a definiu.

VARIÁVEIS GLOBAIS E LOCAIS

Variáveis locais

Variáveis globais

```python
def soma(n1, n2):
    soma1 = n1 + n2
    return soma1
def subtracao(n1, n2):
    sub = n1 - n2
    return sub
print('Bem vindo a calculadora')
num1 = int(input('Digite o primeiro numero: '))
num2 = int(input('Digite o segundo numero: '))
op = int(input('Digite 1 para somar e 2 para subtrair'))
if op == 1:
    print(f'O valor de soma {num1} e {num2} é {soma(num1, num2)}')
elif op == 2:
    print(f'O valor da subtração {num1} e {num2} é {subtracao(num1, num2)}')
else:
    print('Comando invalido!!')
```

VARIÁVEIS GLOBAIS E LOCAIS

O escopo de nomes em Python é mantido por meio de namespaces, que são dicionários que relacionam os nomes dos objetos (referências) e os objetos em si.

```
valor = 100.00

def calculo():
    global valor
    valor = 50.00
    print(f"Valor dentro da função:
{valor}")

print(f"Valor fora da função:
{valor}")
calculo()
print(f"Valor fora da função:
{valor}")
```

Variáveis globais

Os nomes ficam definidos em dois dicionários que podem ser consultados utilizando-se as funções locals() e globals(). Estes dicionários são atualizados em tempo de execução.

VARIÁVEIS GLOBAIS DEVEM SER UTILIZADAS O MÍNIMO POSSÍVEL, POIS DIFICULTAM A LEITURA E VIOLAM O ENCAPSULAMENTO DA FUNÇÃO.

UM BOM USO DE VARIÁVEIS GLOBAIS É UTILIZANDO AS MESMAS COMO CONSTANTES. SEMPRE QUE VÁRIAS FUNÇÕES PRECISEM DE UMA INFORMAÇÃO QUE É FIXA, PODEMOS CRIAR CONSTANTES GLOBAIS. NORMALMENTE CRIAMOS CONSTANTES EM MAIÚSCULAS.

PARÂMETROS DA FUNÇÃO

Parâmetros ou argumentos de funções são essenciais para que possamos chamar funções informando valores.

Desta forma, tornamos as funções flexíveis, recebendo valores diferentes a cada execução.

PARÂMETROS DA FUNÇÃO

Neste exemplo, a função soma() não solicita parâmetros, o retorno será sempre o mesmo, baseado nas variáveis valor1 e valor2, a menos que estas sejam alteradas no nível do código interno da função (variáveis locais).

```
def soma():
    valor1 = 10
    valor2 = 25
    return valor1 + valor2

total = soma()
print(f"O total é: {total}")
```

FUNÇÃO COM PARÂMETROS OPCIONAIS.

```
def soma(valor1, valor2,
imprime = False):
    resultado = valor1 +
valor2
    if imprime:
        print(f"Soma:
{resultado}")
    return resultado
total = soma(10, 84)
```

Já neste exemplo, a função soma possui três parâmetros que tornam o retorno mutável ao chamar a função. Os parâmetros são posicionais quando passamos somente os valores para eles (ex: *soma(3,4,True)*), ou podemos alterar as posições explicitamente (ex: *soma(valor2 = 4, valor1 = 3, imprime = True)*.

valor1 e valor2 são obrigatórios, imprime é opcional porque já foi definida como falso (se não digitarmos nada em sua posição, já entra como False).

FUNÇÃO COM PARÂMETROS OPCIONAIS.

```
def soma(valor1, valor2,
imprime = False):
    resultado = valor1 +
valor2
    if imprime:
        print(f"Soma:
{resultado}")
    return resultado
total = soma(10, 84)
```

Parâmetros opcionais não podem estar no início caso seja combinado com parâmetros obrigatórios.

Total recebe 94, porém, nada é impresso, porque o valor padrão para “imprime” é falso e não foi informado na chamada.

NOMEANDO PARÂMETROS

```
def retangulo(largura, altura,
caractere="*"):
    linha = caractere * largura
    for i in range(altura):
        print(linha)
retangulo(caractere="$",
altura=5, largura=15)
```

Quando informamos o nome do parâmetro, não importa a ordem em que passamos os mesmos.

FUNÇÕES COMO PARÂMETROS

```
def soma(num1, num2):
    return num1 + num2
def multiplicacao(num1, num2):
    return num1 * num2
def calcular(funcao, num1, num2):
    return funcao(num1, num2)
total_soma = calcular(soma, 10, 20)
total_multiplicacao = calcular(multiplicacao, 10,
20)
print(f"Total Soma: {total_soma}")
print(f"Total Multiplicação:
{total_multiplicacao}")
```

PARÂMETROS EMPACOTADOS EM LISTAS

```
lista = [10, 20, True]
def soma(valor1, valor2, imprime = False):
    resultado = valor1 + valor2
    if imprime:
        print(f"Soma: {resultado}")
    return resultado

soma(lista[0], lista[1], True)
soma(*lista)
```

O empacotamento de lista evita termos que informar individualmente os valores da lista pelo índice.

O asterisco indica que estamos desempacotando a lista utilizando seus valores como parâmetros da função.

DESEMPACOTAMENTO DE PARÂMETROS

```python
def soma(imprime, *valores):
    total = 0
    for valor in valores:
        total += valor
    if imprime:
        print(f"Soma: {total}")
    return total

soma(True, 10, 20, 30, 78)
soma(False, 10, 50)
```

Qual melhoria específica tivemos neste código em relação ao do slide anterior?

Verifica-se que os valores enviados por tupla são utilizados separadamente ao serem desempacotados com *valores (*args)

VALIDAÇÃO DE ENTRADA DE DADOS

- Podemos utilizar funções para criar uma infinidade de coisas, uma das coisas para a qual podemos criar funções é para validar a entrada de dados do usuário, reduzindo assim, possíveis erros no sistema.
- Imagine que você tenha que validar uma entrada de dados específica, em vez de ficar sempre escrevendo o código de validação, você pode criar uma função e chamá-la sempre que necessário.

EXEMPLO

Vamos supor que você precise solicitar que o usuário informe um número inteiro dentro de uma faixa. Você pode criar uma função que receberá os três números e fará esta validação.

O `isdigit()`método retorna True se todos os caracteres forem dígitos, caso contrário, False.

```
def validar_faixa(numero, inicio, fim):
    if numero.isdigit():
        if int(numero) < int(inicio) or int(numero) > int(fim):
            print(f"Valor inválido! Informe um número inteiro entre
{inicio} e {fim}.")
        else:
            return True
    else:
        print(f"Número inválido! Informe um número inteiro entre
{inicio} e {fim}.")
while True:
    resposta = input("Informe um número inteiro entre 1 e 10: ")
    if validar_faixa(resposta, 1, 10):
        break
```

EXEMPLO 2

- Vamos validar um limite máximo de caracteres de um texto informado pelo usuário.

```
def validar_tamanho(texto, maximo):
    if len(texto) > maximo:
        print(f"O texto deve conter no máximo {maximo}
caracteres!")
    else:
        return True
while True:
    texto = input("Informe um texto de no máximo 20
caracteres: ")
    if validar_tamanho(texto, 20):
        break
```

EXEMPLO 3

```
def valida_elementos(texto, lista):
    if texto.upper() in lista:
        return True
    else:
        print("Opção inválida, veja as opções
disponíveis:")
    for item in lista:
        print(item)

while True:
    lista = ["CARRO", "NAVIO", "ÔNIBUS", "AVIÃO"]
    resposta = input("Informe um meio de
transporte: ")
    if valida_elementos(resposta, lista):
        break
```

Vamos validar a entrada do usuário com base em uma lista fornecida verificando se o texto informado está contido na lista

MÓDULOS

- À medida que um programa aumenta e são adicionadas novas funções, fica impossível mantê-las em um único arquivo.
- É neste momento que entra o conceito de módulos.
- Todo arquivo .py é um módulo.
- Um programa pode conter diversos módulos.
- Um módulo pode usar outros módulos, desta forma aproveitamos códigos existentes.

EXEMPLO

operacoes.py

```
def media(*lista_numeros):
    total = 0
    for numero in
lista_numeros:
        total += numero
    resultado = total /
len(lista_numeros)
    return resultado
def soma(*lista_numeros):
    total = 0
    for numero in
lista_numeros:
        total += numero
    return total
```

programa.py

```
import operacoes

lista = []

while True:
    numero = int(input("Informe um número
int ou 0 para sair: "))
    if numero == 0:
        break
    lista.append(numero)
soma = operacoes.soma(*lista)
media = operacoes.media(*lista)
print(f"Soma: {soma}")
print(f"Média: {media}")
```

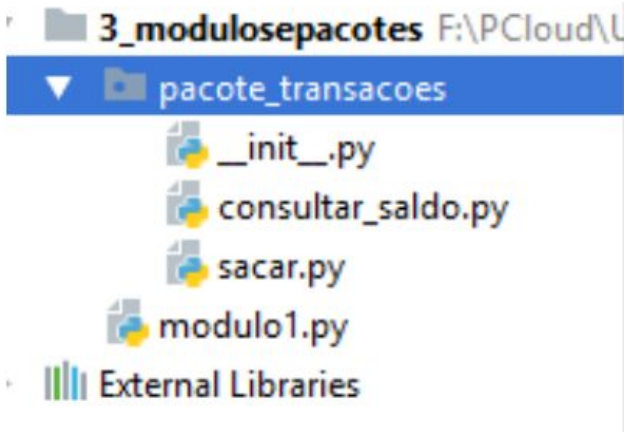
PACOTES

- Enquanto módulos são organizados em arquivos, pacotes ou packages são organizados em pastas identificadas com o arquivo __init__.py.
- Os pacotes funcionam como coleções para organizar módulos de forma hierárquica.
- É possível importar todos os módulos de um pacote, tirando os que começam com "__" (2 underscores) usando a declaração a seguir:

```
from nome_do_pacote import *
```

ARQUIVO _INIT_.PY

- O arquivo __init__.py indica ao Python que aquela pasta é um pacote importável. ode estar vazio, conter código de inicialização do pacote ou definir uma variável chamada __all__, que é uma lista de módulos que pertencem ao pacote e que serão importados quando usamos *.




```
__init__.py
__all__ =
['consultar_saldo','sac
ar']
```

IMPORTANDO PACOTES

O código abaixo importará todos os módulos de math, para importar apenas o necessário utilizamos from.

```
import math
print(math.sqrt(25))
```

O código abaixo importará o módulo sqrt do pacote math.

```
from math import sqrt
print(sqrt(25))
```

Observe que ao utilizar from package import item, o item pode ser um subpacote, submódulo, classe, função ou variável.

EXEMPLO

Vamos criar um pacote de módulos com a seguinte estrutura da pasta:

```
projeto/
├── programa.py
└── meu_pacote/
    ├── __init__.py
    ├── operacoes.py
    └── mensagens.py
```

📄 operacoes.py

```
def soma(a, b):
    return a + b
```

📄 mensagens.py

```
def ola():
    return "Olá, eu sou um módulo do pacote!"
```

EXEMPLO

📄 __init__.py
```
__all__ = ["operacoes", "mensagens"]
```

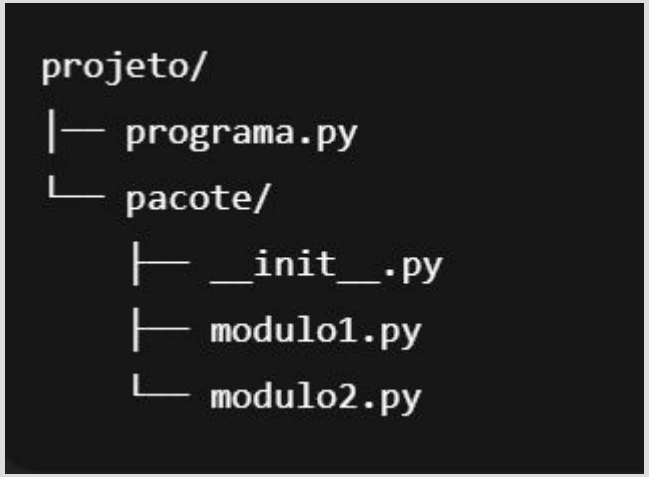
📄 programa.py (arquivo principal)
```
from meu_pacote import *

print(operacoes.soma(2, 3))
print(mensagens.ola())
```

<u>Resultado esperado:</u>

```
Olá, eu sou um módulo do pacote!
```

Sempre que estiver usando pacotes em Python, lembre:



👉 O arquivo que **executa** o programa deve ficar **fora do pacote**,
👉 e o pacote deve ter **init.py** para ser reconhecido pelo Python.

PACOTES - RESUMO DO EXEMPLO

Termo	Significa	Exemplo
Módulo	Um arquivo `.py`	`operacoes.py`
Pacote	Uma pasta com vários módulos e um `__init__.py`	`meu_pacote/`
`__init__.py`	Torna a pasta um pacote e controla o que será importado	`__all__ = [...]`
`from pacote import *`	Importa todos os módulos listados no `__all__`	`operacoes`, `mensagens`

EXERCÍCIOS

1. Faça um programa, com uma função interna que necessite de um argumento do tipo numérico decimal. A função retorna o valor de caractere ‘P’, se seu argumento for positivo, e ‘N’, se seu argumento for zero ou negativo.

2. Faça um programa com uma função IMPORTADA de um pacote externo chamada somaImposto. A função possui dois parâmetros formais: taxaImposto, que é a quantia de imposto sobre vendas expressa em porcentagem e custo, que é o custo de um item antes do imposto. A função “altera” o valor de custo para incluir o imposto sobre vendas.

EXERCÍCIOS

3. Faça um programa que utilize uma função chamada valorPagamento para calcular o valor devido de uma prestação. O programa deve solicitar ao usuário o valor da prestação e o número de dias de atraso, enviando essas informações para a função, que retornará o valor corrigido. O cálculo deve seguir estas regras: se não houver atraso, o valor permanece o mesmo; caso haja atraso, deve ser aplicada uma multa de 3% sobre o valor da prestação, além de juros de 0,1% ao dia. Após receber o valor calculado, o programa deve exibi-lo na tela e, em seguida, solicitar novamente um novo valor de prestação, repetindo o processo até que o usuário informe 0 como valor da prestação. Quando isso acontecer, o programa deve encerrar a execução e exibir um relatório com a quantidade de prestações pagas no dia e o total recebido.

EXERCÍCIO 1 - Gabarito

```
def verifica_numero(valor):
    # 1) Tentando converter para número
    try:
        numero = float(valor)
    except ValueError:
        return "Erro: valor digitado não é numérico."

    # 2) Lógica principal
    if numero > 0:
        return 'P'
    else:
        return 'N'


# Programa principal
entrada = input("Digite um número: ")
resultado = verifica_numero(entrada)
print(f"Resultado: {resultado}")
```

A função:

- tenta converter o argumento para número (float)
- se não conseguir, retorna uma mensagem de erro
- se conseguir, retorna 'P' ou 'N'

✔ try / except

Tenta executar um código.
Se der erro (ex.: converter "abc" para número), o programa não trava — o except captura.

EXERCÍCIO 2 - Gabarito

Arquivo do pacote: impostos.py (Crie este arquivo dentro de meu_pacote):

```
def somaImposto(taxaImposto, custo):
    taxa_decimal = taxaImposto / 100
    novo_custo = custo + (custo * taxa_decimal)
    return novo_custo
```

Estrutura de pastas:

```
exercicio2/
├── programa.py
└── meu_pacote/
    ├── __init__.py
    └── impostos.py
```

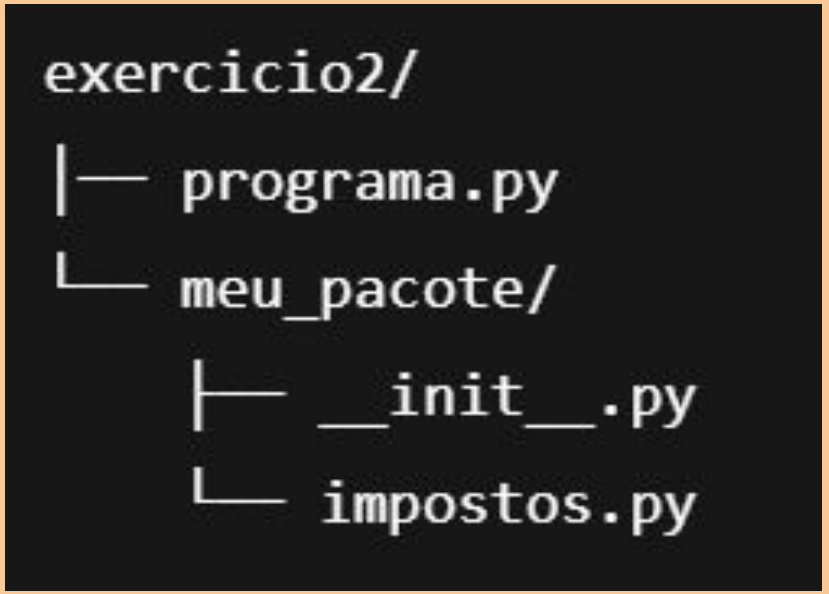
EXERCÍCIO 2 - Gabarito

Conteúdo do __init__.py (Crie este arquivo também dentro de meu_pacote):

```
from .impostos import somaImposto

__all__ = ["somaImposto"]
```

Estrutura de pastas:



EXERCÍCIO 2 - Gabarito

Arquivo principal: programa.py (Crie este arquivo FORA de meu_pacote):

```
from meu_pacote import somaImposto

taxa = float(input("Informe a taxa de imposto (%): "))
custo = float(input("Informe o custo do item: "))

resultado = somaImposto(taxa, custo)

print(f"Valor final com imposto: {resultado:.2f}")
```

Estrutura de pastas:

```
exercicio2/
├── programa.py
└── meu_pacote/
    ├── __init__.py
    └── impostos.py
```

EXERCÍCIO 3 - Gabarito

meu_pacote_pagamentos/pagamento.py:

```
def valorPagamento(prestacao, dias_atraso):
      try:
       valor = float(prestacao)
   except (TypeError, ValueError):
       raise ValueError("Prestação deve ser um número válido.")

   try:
       dias = int(dias_atraso)
   except (TypeError, ValueError):
       raise ValueError("Dias de atraso deve ser um inteiro válido.")

   if dias <= 0:
       return round(valor, 2)

   multa = 0.03 * valor              # 3%
   juros = 0.001 * valor * dias      # 0.1% ao dia = 0.001 * valor * dias
   total = valor + multa + juros
   return round(total, 2)
```

Estrutura de pastas:

```
exercicio3/
├── programa.py
└── meu_pacote_pagamentos/
    ├── __init__.py
    └── pagamento.py
```

EXERCÍCIO 3 - Gabarito

meu_pacote_pagamentos/__init__.py:

```
from .pagamento import valorPagamento

__all__ = ["valorPagamento"]
```

Estrutura de pastas:

```
exercicio3/
├── programa.py
└── meu_pacote_pagamentos/
    ├── __init__.py
    └── pagamento.py
```

EXERCÍCIO 3 - Gabarito

programa.py (arquivo principal):

```
from meu_pacote_pagamentos import valorPagamento

print("=== Sistema de Pagamento de Prestações ===")
total_prestacoes = 0
soma_total = 0.0

while True:
    entrada_valor = input("Informe o valor da prestação (0 para encerrar): ").strip()
    entrada_valor = entrada_valor.replace(",", ".")
    try:
        valor_prest = float(entrada_valor)
    except ValueError:
        print("Valor inválido. Tente novamente.")
        continue

    if valor_prest == 0:
        break

    entrada_dias = input("Informe o número de dias em atraso (0 se não houver): ").strip()
    try:
        dias_atraso = int(entrada_dias)
    except ValueError:
        print("Dias inválidos. Use um número inteiro. A entrada será reiniciada.")
        continue

    try:
        valor_a_pagar = valorPagamento(valor_prest, dias_atraso)
    except ValueError as e:
        print(f"Erro no cálculo: {e}")
        continue

    print(f"Valor a pagar: R$ {valor_a_pagar:.2f}")

    total_prestacoes += 1
    soma_total += valor_a_pagar

print("\n=== Relatório do dia ===")
print(f"Quantidade de prestações pagas: {total_prestacoes}")
print(f"Valor total recebido: R$ {soma_total:.2f}")
print("Encerrando o programa. Obrigado.")
```

Estrutura de pastas:

```
exercicio3/
├── programa.py
└── meu_pacote_pagamentos/
    ├── __init__.py
    └── pagamento.py
```