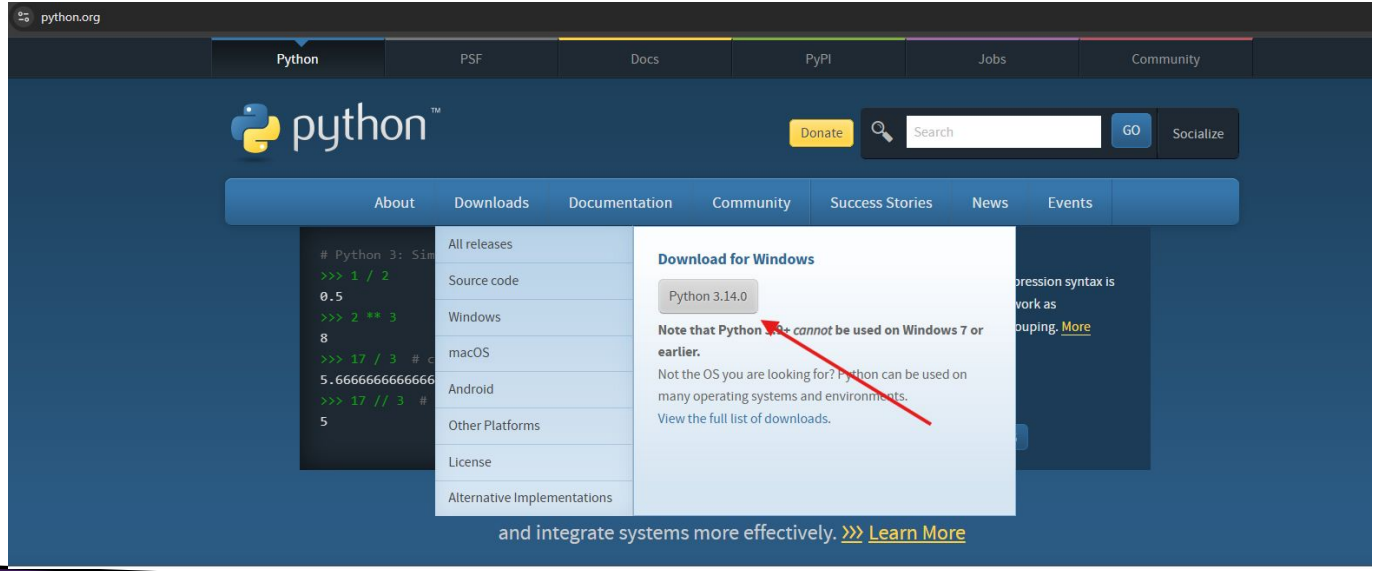
PROGRAMAÇÃO EM PYTHON

python

SENAI

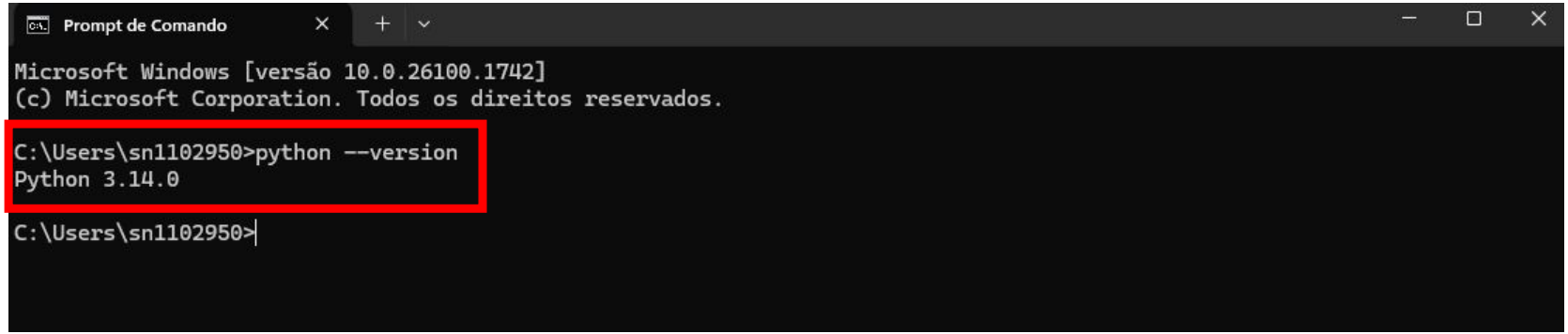
INSTALANDO O PYTHON

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



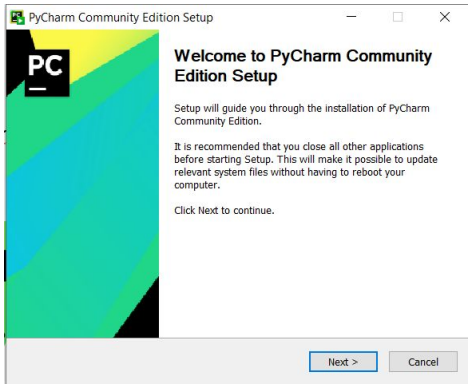
VERIFICANDO A INSTALAÇÃO

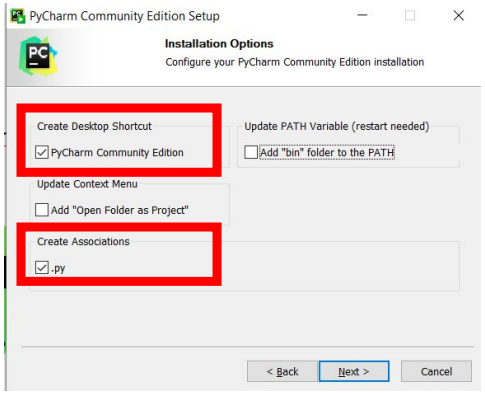
1. Abra o Prompt de comando
2. Digite o comando python --version



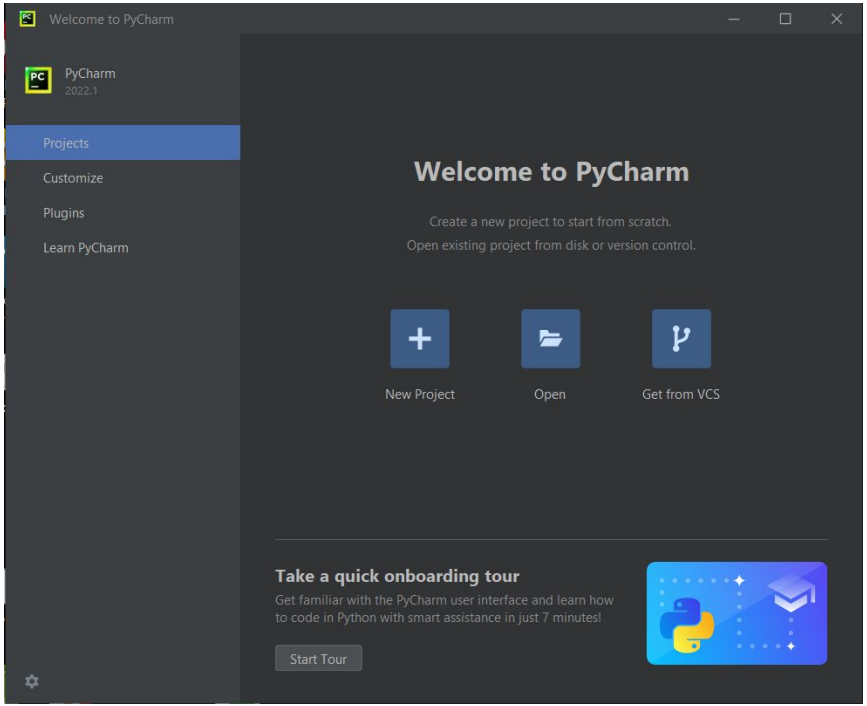
INSTALANDO A IDE PYCHARM

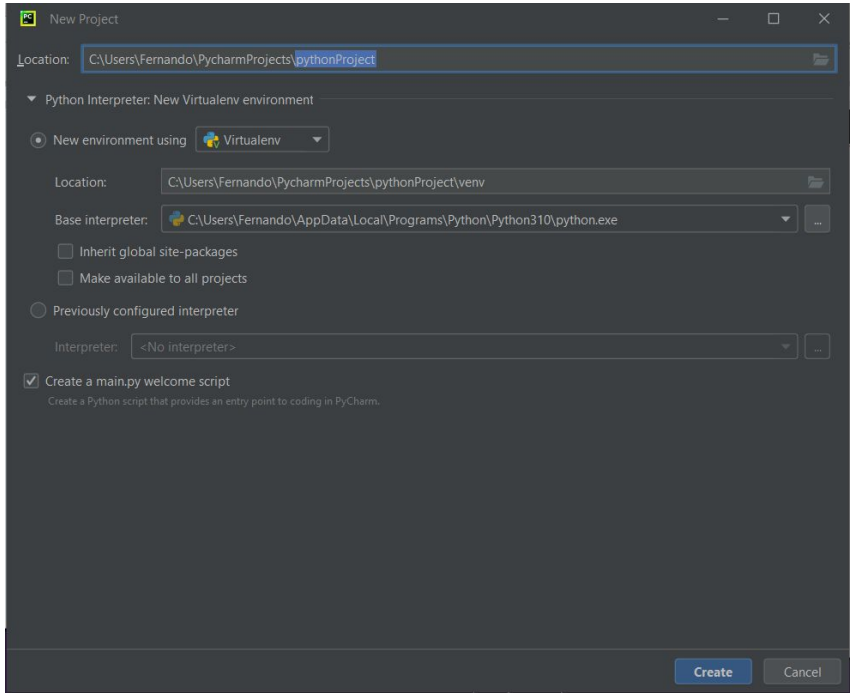
O PyCharm fornece complementação de código inteligente, inspeções de código, realce dinâmico de erros e correções rápidas, juntamente com melhorias de código automatizada e recursos de navegação avançados.



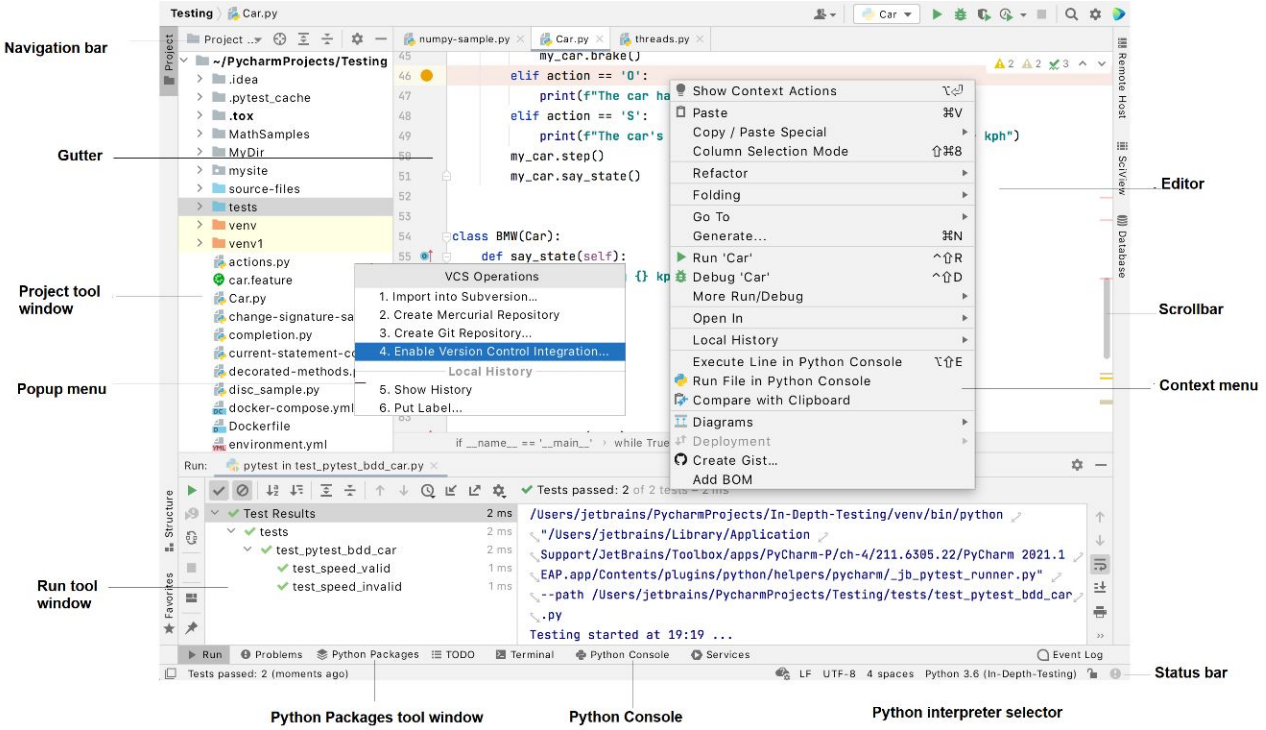


INICIANDO UM PROJETO NO PYCHARM

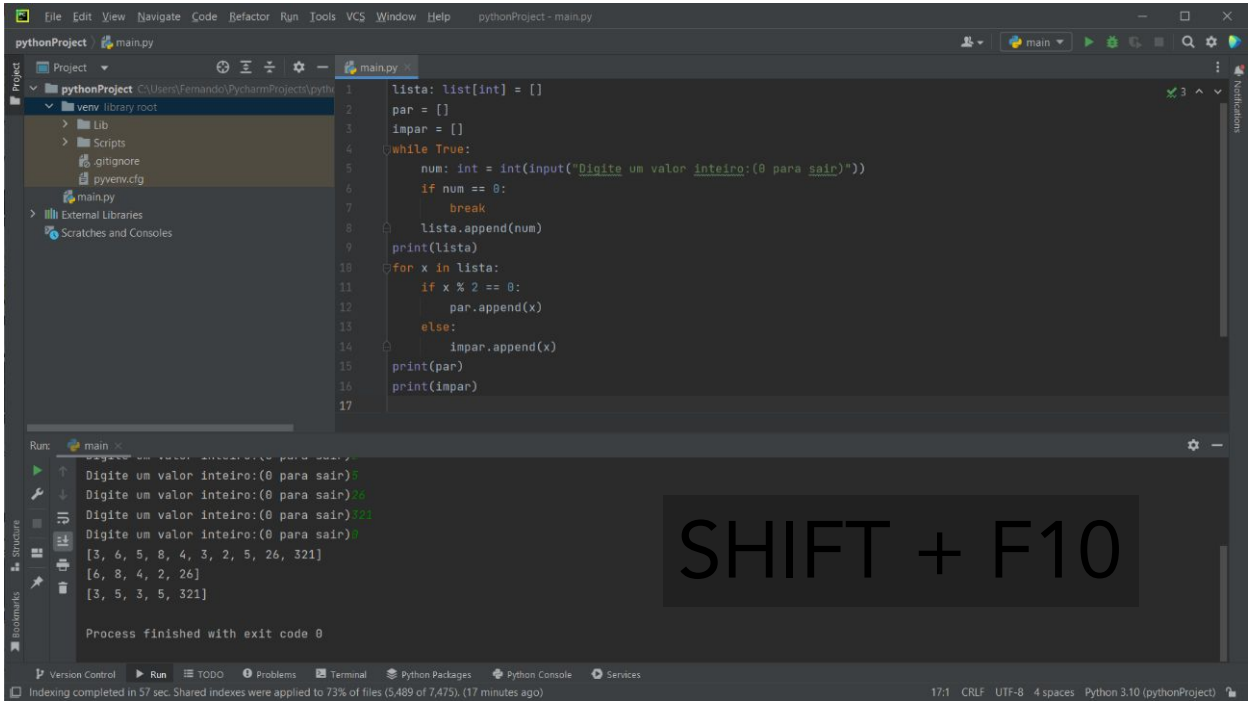




CONHECENDO O PYCHARM



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)