3º BIMESTRE – PROGRAMAÇÃO ORIENTADA A OBJETOS - PYTHON

Sumário

29 – CRIAÇÃO DE ROTINAS DE PROGRAMAS – def ____()

No python, assim como em outras linguagens podemos criar rotinas de programas. Estas podem ou não receber e retornar valores.

Veremos alguns exemplos de criação e uso de rotinas com/sem passagens de parâmetros, e com/sem retorno de valores.

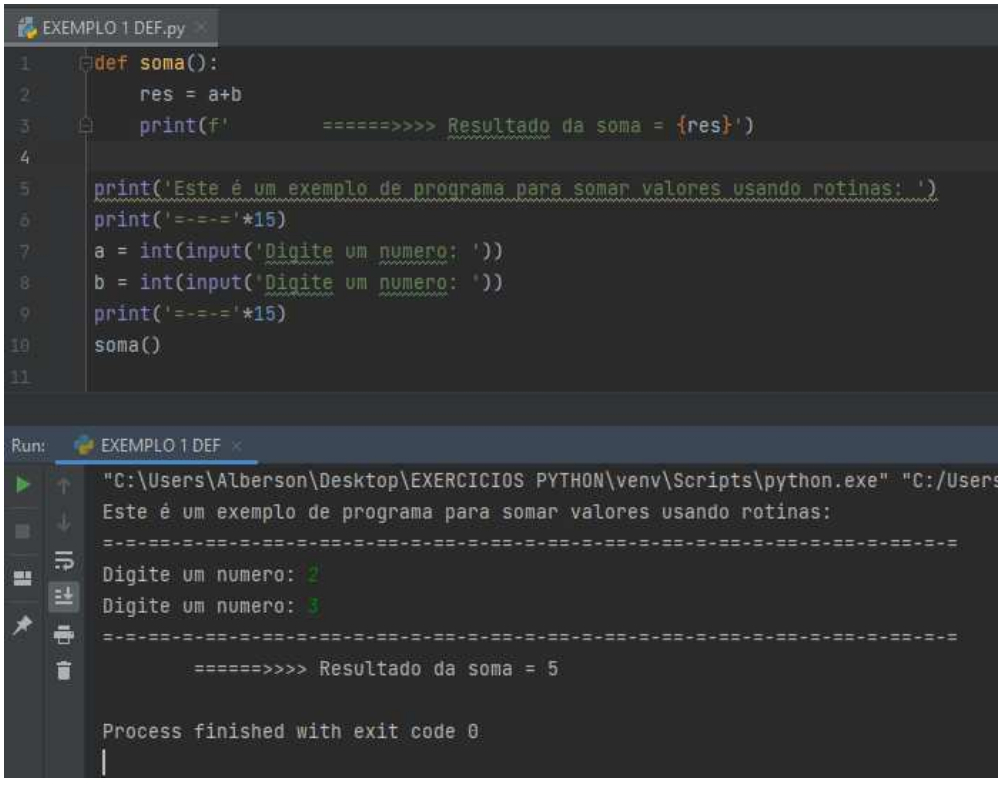
29.1 – CRIAÇÃO E USO DE ROTINAS SEM PASSAGEM DE PARÂMETROS E SEM RETORNO DE VALORES – (PROCEDIMENTOS)

Para criação de rotinas SEM passagens de parâmetros usaremos a seguinte sintaxe:

```
def <nome_rotina>():
    <comandos da rotina>
```

Vamos aos exemplos:

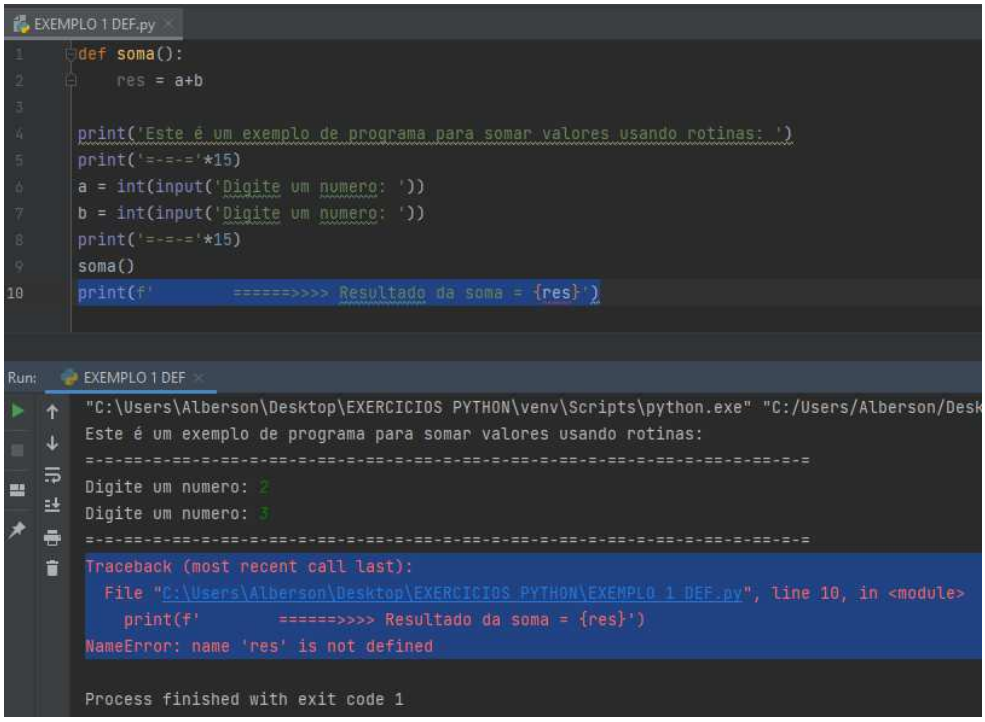
Exemplo 1:



Neste exemplo:

a) Criei a rotina chamada def soma()
b) Esta rotina somará os dois valores armazenados em ***a*** e ***b,*** e mostrará o resultado para o usuário
c) A variável ***res*** foi criada dentro da rotina, ou seja, é uma variável **LOCAL** e por este motivo não pode ser usada fora dela.
d) O comando escrito na linha 10 colocou a rotina em execução

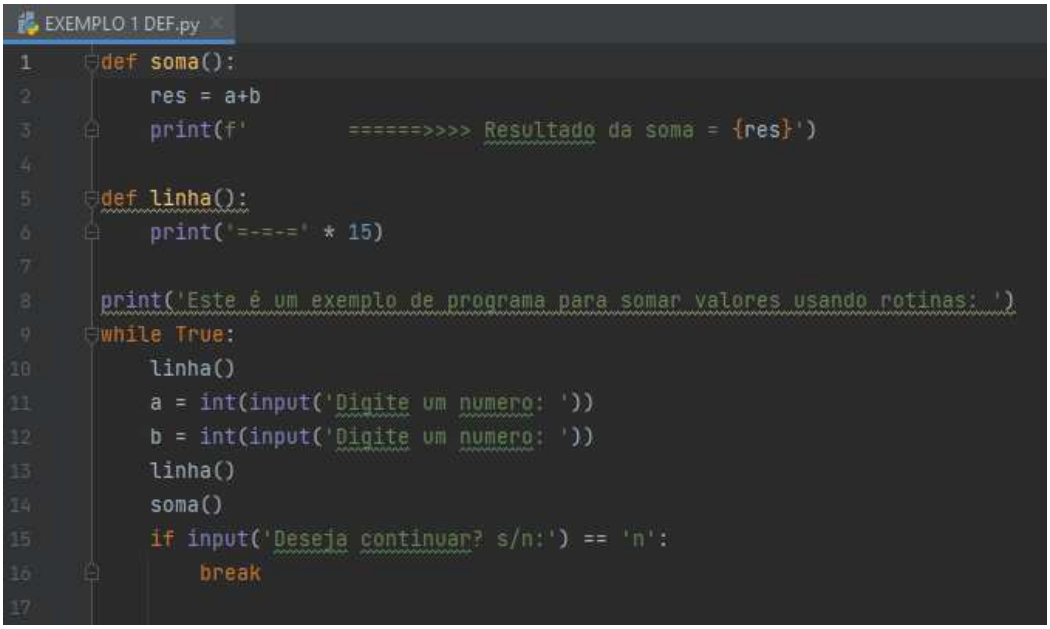
Exemplo 2:

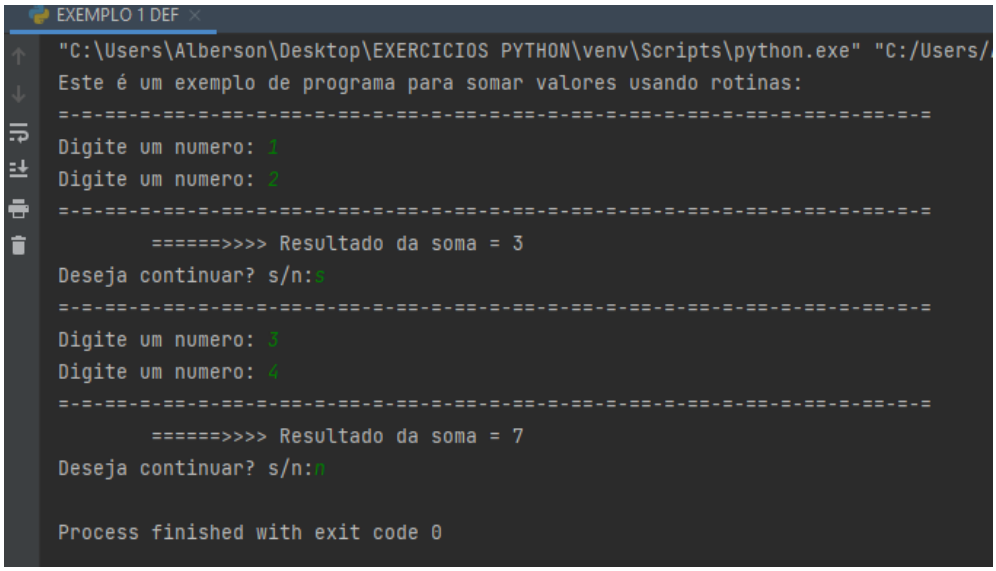


ATENÇÃO COM ERRO ACIMA!!!

Neste caso tentei imprimir o resultado na área do programa principal, após a chamada da rotina. Repare que o comando print usa a variável ***res*** e esta é local da rotina soma.

Exemplo 3:





Neste exemplo:

a) Foi incluído uma estrutura de repetição para executar várias vezes o programa para **somar dois números.**
b) Repare que criei outra rotina para desenhar linhas de separação na tela. A ela dei o nome de ***linha()***.
c) Sempre que o usuário responde ***"s"*** que deseja continuar, o programa solicita a digitação de dois números e estes são somados.
d) O programa finaliza quando o usuário responde ***"n"***, ou seja, que não deseja mais continuar executando o programa.

29.2 – CRIAÇÃO E USO DE ROTINAS COM PASSAGEM DE PARÂMETROS SEM RETORNO DE VALORES – (PROCEDIMENTOS)

Para criação de ***def*** com passagem de parâmetro, precisamos somente indicar dentro dos parênteses os nomes das variáveis que receberão os dados enviados no momento da chamada.

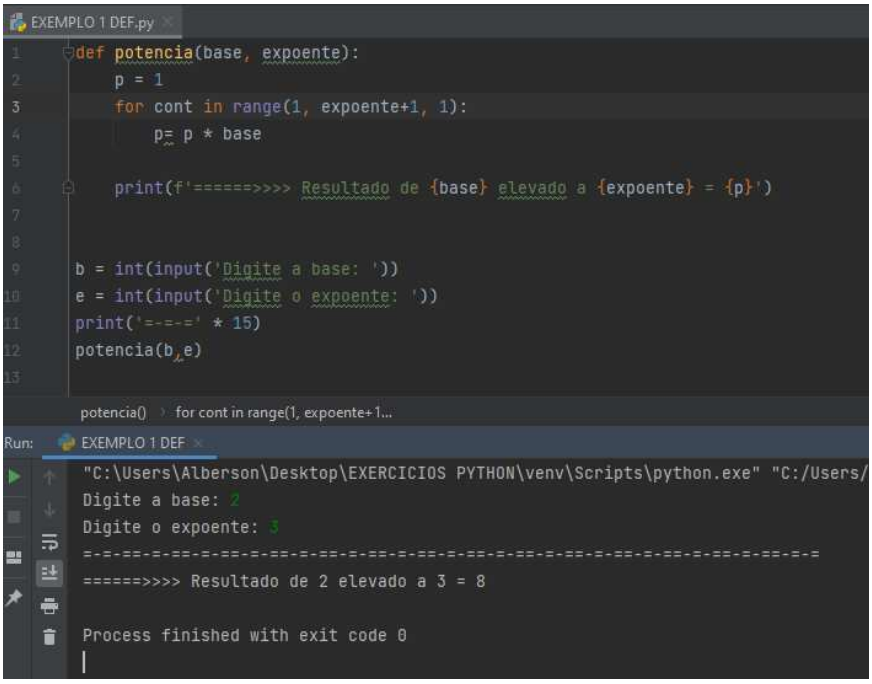
def <nome_rotina>(<var1>, <var2>,....,<varN>):
<comandos da rotina>

Vale lembrar que quando criamos o recurso de recebimento de parâmetros:

a) Será necessário passar a mesma quantidade de dados para a rotina, no momento da sua chamada, para que as variáveis recebam estes valores.
b) As variáveis que recebem os dados como parâmetros devem ser usadas dentro da rotina, senão não haveria motivo para criá-las.

Vamos aos exemplos:

<u>Exemplo 1</u>



Repare:

a) Foi criada a rotina chamada ***potencia(base, expoente)*** com variáveis base e expoente preparadas para receber dados de parâmetros;
b) Repare no módulo principal, que os dados informados pelo usuário são armazenados nas variáveis ***b*** e ***e***
c) Ao chamar a rotina os valores armazenados em ***b*** e ***e*** são passados automaticamente e respectivamente para ***base*** e ***expoente.***
d) Veja que as variáveis ***base*** e ***expoente*** foram usadas no interior da rotina ***potencia.***
e) Neste programa estamos calculando a potência, diante de um valor de base e expoente informados pelo usuário. Repare que não foi utilizada nenhuma função própria do python para este cálculo, visto que a estrutura de repetição está fazendo o cálculo.
f) Neste exemplo estamos calculando **2^3 = 2*2*2 = 8**

Exemplo 2:

```
EXEMPLO 1 DEF.py
def potencia(base, expoente):
    p = 1
    for cont in range(1, expoente+1, 1):
        p= p * base

    print(f'======>>>> Resultado de {base} elevado a {expoente} = {p}')


b = int(input('Digite a base: '))
e = int(input('Digite o expoente: '))
print('=-=-=' * 15)
#Na chamada abaixo indicamos os nomes das variáveis que vão receber
#os valores de e e b na área de parâmetros. ISSO NÃO É USUAL FAZER !!!
potencia(expoente=e, base=b)
```

```
Run: EXEMPLO 1 DEF
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberso
Digite a base: 2
Digite o expoente: 3
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
======>>>> Resultado de 2 elevado a 3 = 8

Process finished with exit code 0
```

IMPORTANTE NESTE EXEMPLO

Veja que no momento da chamada da rotina, indicamos quais variáveis receberão os dados que estão sendo passados, ou seja, *expoente* receberá *e*, *base* receberá *b*

Exemplo 3:

```
EXEMPLO 1 DEF.py
1  def potencia(base, expoente):
2      p = 1
3      for cont in range(1, expoente+1, 1):
4          p= p * base
5      print(f'======>>>> Resultado de {base} elevado a {expoente} = {p}')
6
7  b = int(input('Digite a base: '))
8  e = int(input('Digite o expoente: '))
9  print('=-=-=' * 15)
10 #repare no erro abaixo:
11 #Não podemos atribuir valor para expoente = e, sendo que b também está sendo
12 #passado como parâmetro para expoente, VEJA ERRO ABAIXO
13 potencia(expoente=e, b)
14
```

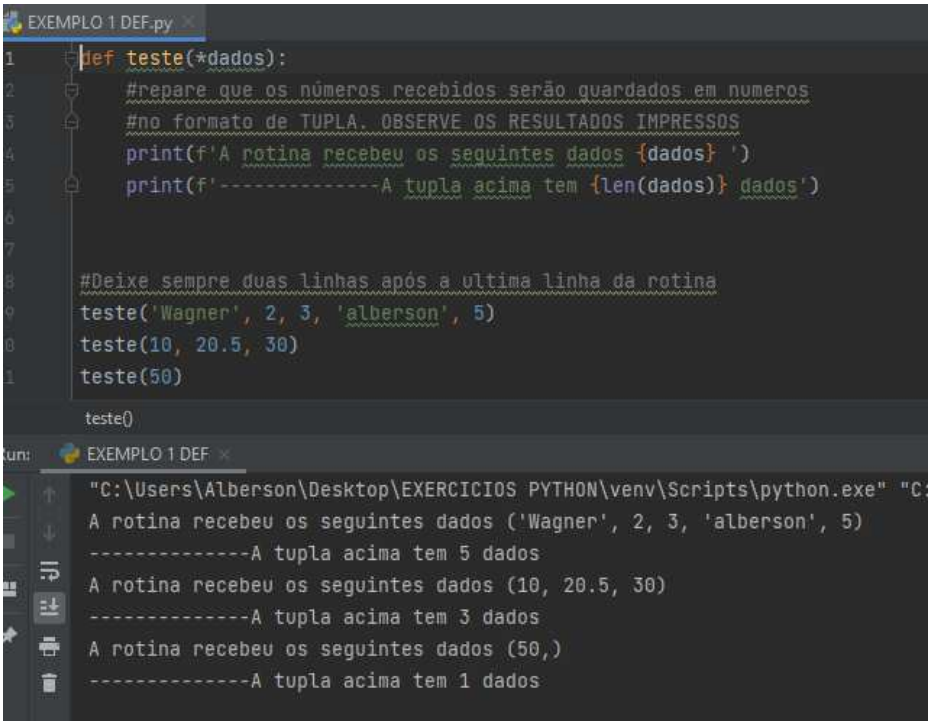
```
Run: EXEMPLO 1 DEF
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\EXEMPLO 1 DEF.py", line 13
    potencia(expoente=e, b)
                          ^
SyntaxError: positional argument follows keyword argument
```

MOTIVO DESTE ERRO

Neste exemplo repare o erro da linha 13, pois a variável ***b*** está no local de passagem de parâmetro para variável expoente, para a qual já atribuímos o valor da variável ***e***

29.2.1 – CRIAÇÃO E USO DE ROTINAS COM PASSAGEM DE PARÂMETROS DE DADOS EMPACOTADOS SEM RETORNO DE VALORES- (PROCEDIMENTOS)

Para criação de recebimento de vários parâmetros, quando o programador NÃO SABE QUANTOS SERÃO RECEBIDOS PELA ROTINA, usamos o recurso de EMPACOTAMENTO DE DADOS. Para isto usaremos um asterisco antes do nome de uma única variável colocada na área de parâmetros. Veja exemplo abaixo:



Observemos com atenção!!!

A VARIÁVEL dados, NA ÁREA DE PARÂMETROS, FOI CRIADA PRECEDIDA DE UM ASTERISCO, O QUAL RECEBE TODOS OS DADOS NO FORMATO DE TUPLA. QUANDO A ROTINA TESTE FOI CHAMADA, REPARE QUE OS PARÂMETROS QUE ESTÃO SENDO PASSADOS TEM QUANTIDADE DE DADOS DIFERENTES .

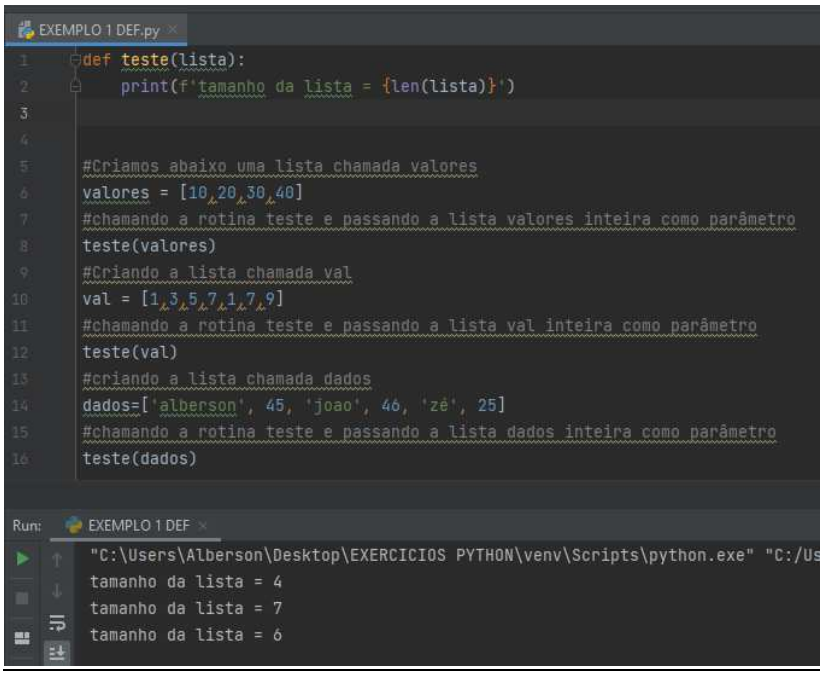
LEMBREM-SE QUE AS TUPLAS SÃO LIMITADAS NO SENTIDO DE SEREM IMUTÁVEIS

29.2.2 – CRIAÇÃO E USO DE ROTINAS COM PASSAGEM DE PARÂMETROS DE DADOS EM FORMAS DE LISTAS SEM RETORNO DE VALORES – (PROCEDIMENTOS)

Podemos criar uma lista e passá-la totalmente como parâmetros. Porém, para uso deste recurso **não se usa o *** precedendo a variável que receberá a lista como parâmetro. Aliás, para receber o parâmetro basta somente uma variável simples na área de parâmetro.

Vamos aos exemplos!!!!

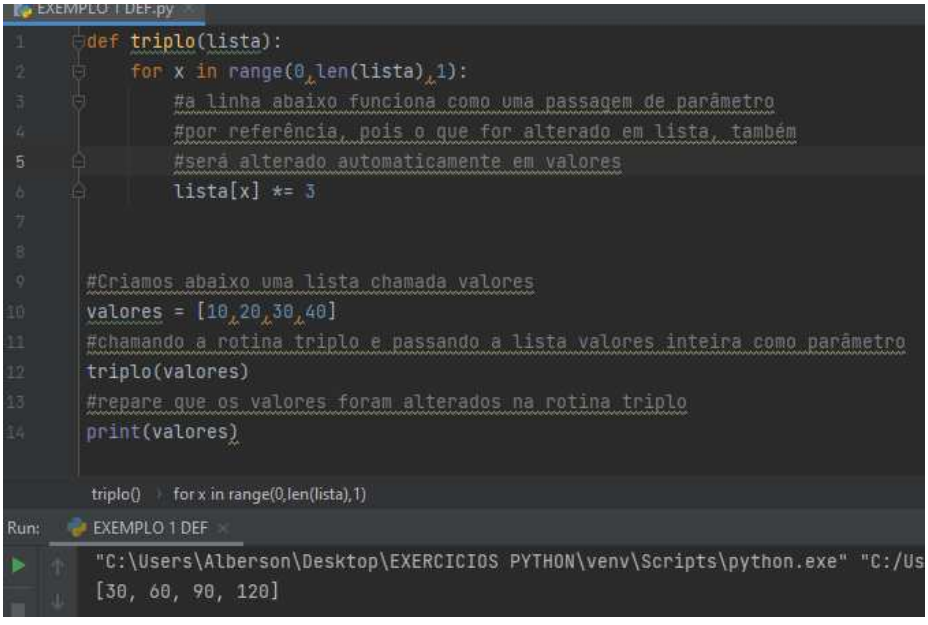
Exemplo 1:



Perceba no exemplo!!!

a) A rotina ***teste*** foi criada para receber um parâmetro, neste caso de exemplo, receberá uma lista.
b) A rotina contará e vai imprimir a quantidade de itens na lista recebida.
c) Foram criadas, no módulo principal do programa, várias listas de diferentes tamanhos.
d) Perceba que a cada chamada da rotina ***teste*** foi passada uma lista diferente e por este motivo, os resultados impressos demonstram o tamanho de cada uma delas.

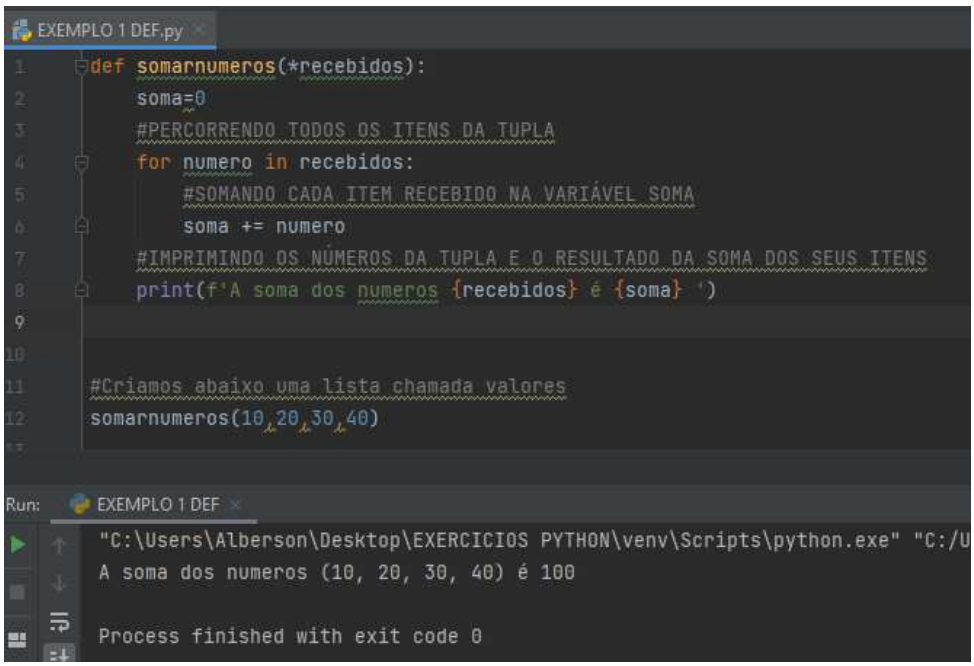
EXEMPLO 2:



IMPORTANTE !!!!

Em PYTHON TODA PASSAGEM DE PARÂMETRO É POR REFERÊNCIA

Exemplo 3:



Podemos também trabalhar com operações específicas ao desempacotar a TUPLA RECEBIDA dentro das rotinas, foi o caso acima, o qual somamos todos os ITENS recebidos COMO TUPLA.

29.3 – DESAFIOS DE ROTINAS – (PROCEDIMENTOS)

1) Fazer um programa em python para que o usuário digite um número decimal inteiro qualquer e armazene tal número em uma variável global qualquer. Em seguida, chame uma rotina **SEM PASSAGEM DE PARÂMETRO**, para convertê-lo este número em binário. A a conversão de decimal para binário deve usar estrutura de repetição que você quiser. Dentro da rotina imprima o resultado da conversão.

2) Refaça o programa 1, de tal forma que o usuário possa passar o número decimal digitado como parâmetro para rotina. Depois de passar este número como parâmetro e imprimir o resultado da conversão para binário, chame a rotina novamente no módulo principal do programa, porém desta vez, passando como parâmetro o número 59. Com isso, o programa dever exibir também o resultado desta conversão.

3) Fazer um programa com uma rotina que verifique se um número passado como parâmetro é ou não primo.

4) Fazer um programa em python que calcule a área total de uma casa. Para isso, o programa deve possuir uma rotina que receba como parâmetro USANDO RECURSO DE EMPACOTAMENTO os nomes de cada cômodo e as suas medidas dos comprimentos e larguras. Dentro da rotina, desempacote a tupla recebida e faça o cálculo mostrando o nome e a área de cada cômodo, bem como a soma de todas as áreas calculadas e exiba o resultado da área total do imóvel.

5) Fazer um programa em python que contenha uma rotina para verificar qual o maior e menor números contidos numa lista. O usuário deve digitar vários números e armazená-los na referida lista que será passada como parâmetro. Imprima qual número maior e o menor e qual a posição destes números dentro da referida lista. ATENÇÃO, NÃO PERMITA A DIGITAÇÃO DE NÚMEROS IGUAIS DENTRO DA LISTA.

30 – ROTINA COM DEFINIÇÃO DE PARÂMETROS OPCIONAIS

Quando chamamos uma rotina qualquer para execução, podemos definir valores padrões para as variáveis que receberão os dados na área de parâmetros de entrada.

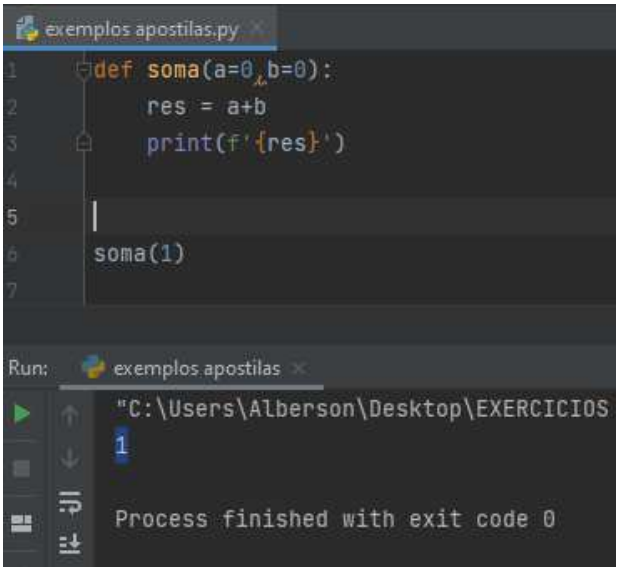
Este é um recurso muito útil, pois caso o programador escreva o comando de chamada da rotina deixando de passar algum parâmetro definido na área de parâmetro teremos um erro. Veja este erro:

Exemplo 1:

Neste caso, estou chamando a rotina *soma*, passando para ela somente o número *1*. Na verdade, a rotina esperava o recebimento de dois dados passados pelo programador e não um como foi passado. Daí o motivo do erro mostrado.

Como definir valores padrões para as variáveis na área de parâmetros da rotina soma. Vamos a solução:

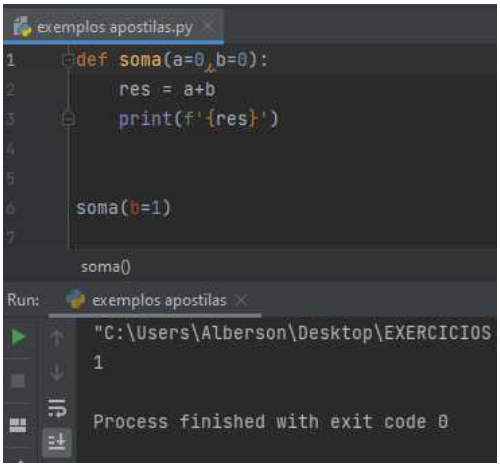
Exemplo 2:



Veja neste exemplo:

Na área de parâmetro da rotina *soma()* atribuímos o valor *0* (zero) para as variáveis *a* e *b*. Com isto, caso a chamada da rotina seja feita somente com um parâmetro, a variável que não recebeu dado assume *0*, não travando mais o programa. DESTA FORMA OS PARÂMETROS *a* E *b* SÃO OPCIONAIS NO MOMENTO DA CHAMADA DA ROTINA *soma*.

Exemplo 3:



Repare:

a) Neste caso na chamada da rotina ***soma*** passei para ***b*** o valor ***1.***
b) O valor de ***a*** foi assumido como sendo ***0***,

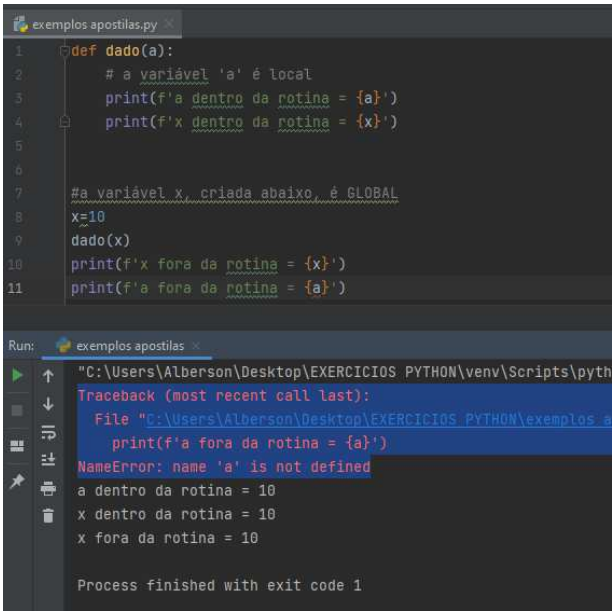
ATENÇÃO:

SE TENTARMOS PASSAR MAIS DE 2 PARÂMETROS PARA A ROTINA *soma* OCORRERÁ ERRO NA CHAMADA DESTA ROTINA, VISTO QUE SÓ TEMOS 2 PARÂMETROS DE ENTRADA DEFINIDOS. PARA UMA SITUAÇÃO COMO ESTA TEMOS QUE USAR O RECURSO DE EMPACOTAMENTO DE DADOS, VISTO ANTERIORMENTE

31 – VARIÁVEIS LOCAIS E GLOBAIS NO PYTHON – ESCOPO DE VARIÁVEIS

As variáveis criadas no módulo principal de um programa são GLOBAIS. As variáveis locais são as criadas na área de parâmetro, ou de uma rotina qualquer. Vejamos alguns detalhes importantes:

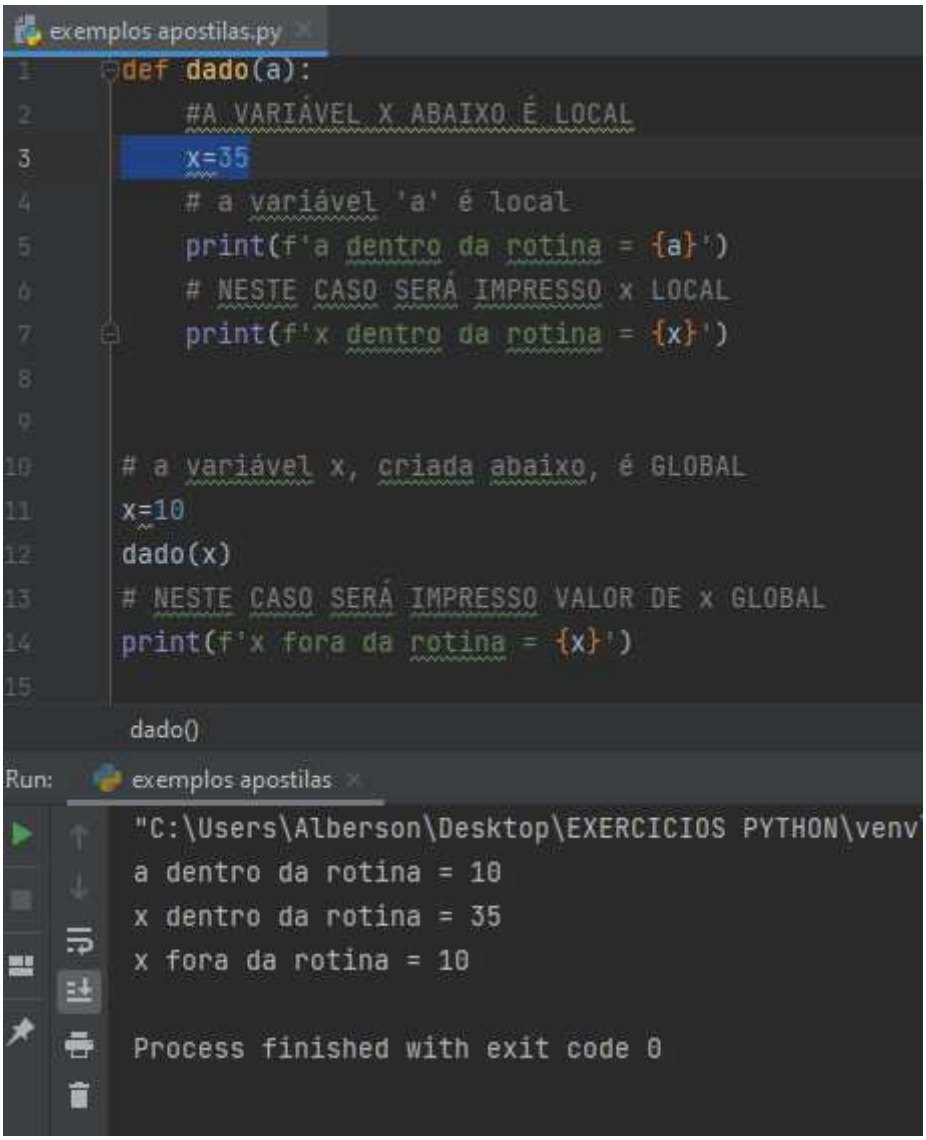
Exemplo 1:



Repare o erro ocorrido no exemplo:

Neste caso, a variável "***a***" não pode ser usada no módulo principal do programa, pois foi criada como parâmetro da rotina ***dado*** e portanto, só pode ser usada dentro daquela rotina. Observe que a variável ***x*** foi usada tanto no módulo principal, como também na rotina.

Exemplo 2:



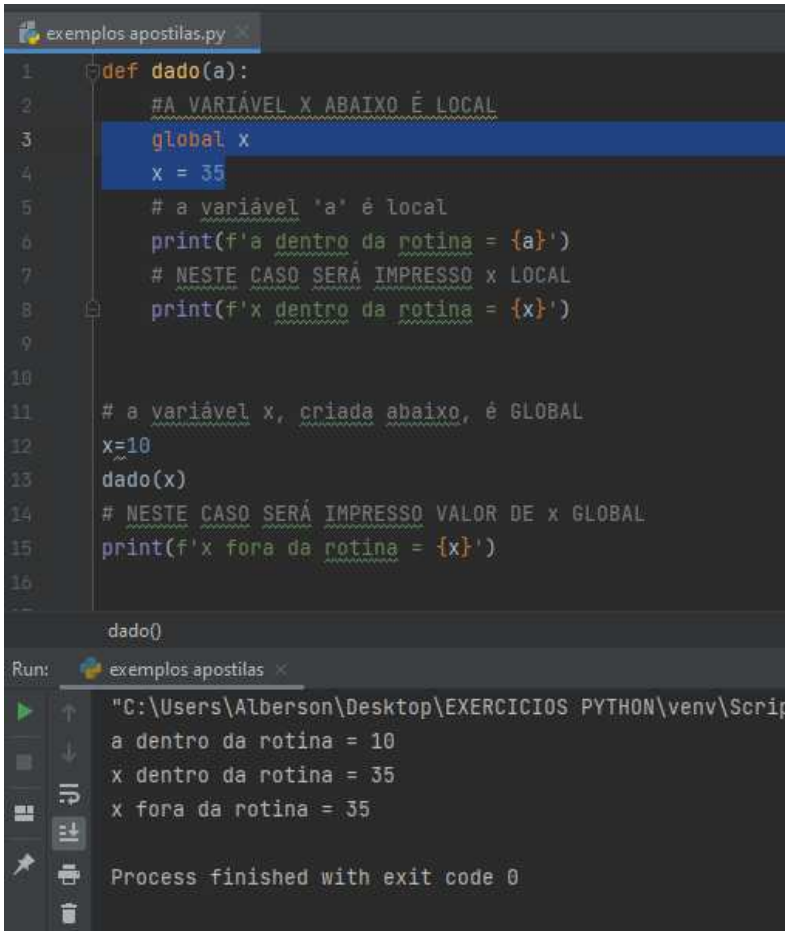
IMPORTANTE NESTE EXEMPLO:

Repare que a variável *x* criada no módulo principal que armazena *10*, não será reconhecida dentro da rotina soma, pois lá foi criada uma variável *x* que será local e armazenará *35*. Por este motivo tivemos dois resultados diferentes impressos para x na área de resultados.

31.1 – USANDO O COMANDO global DENTRO DE ROTINAS:

Quando usamos o comando ***global*** antes do nome de uma variável dentro de uma rotina, estamos informando ao python que qualquer valor atribuído a esta variável em questão, deverá ser armazenado na variável global já existente com este nome. Vejamos:

Exemplo 1:



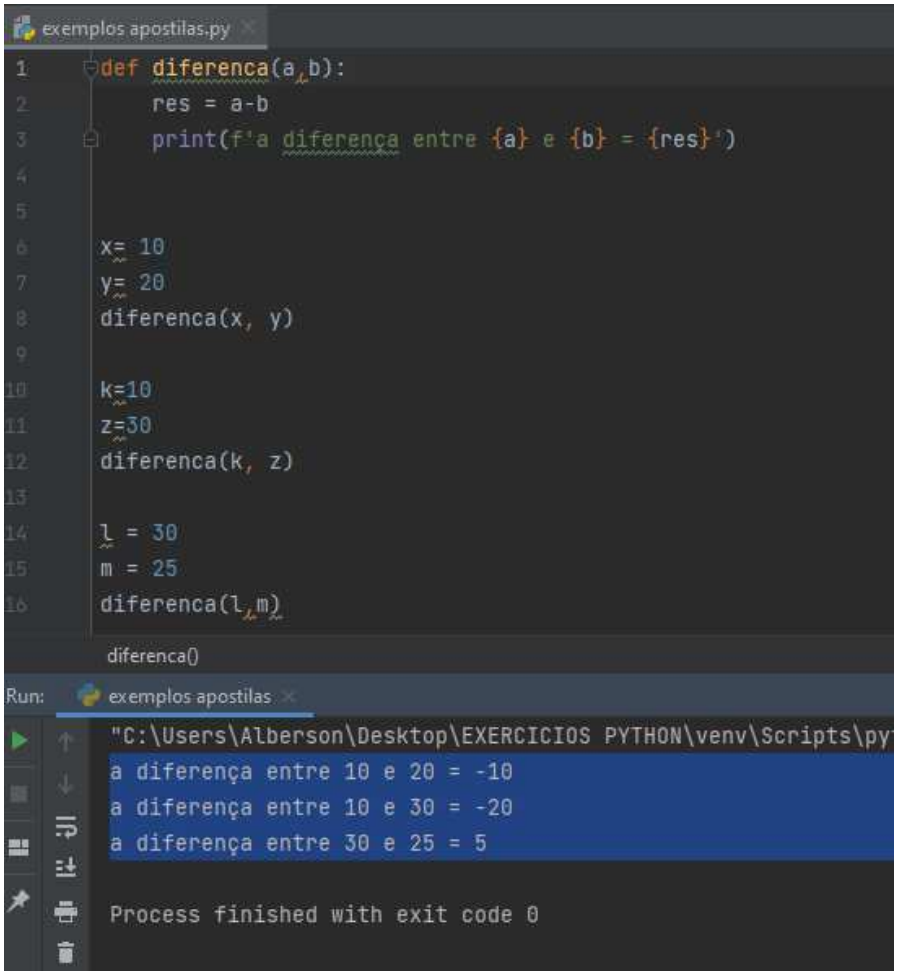
Observe IMPORTANTE!

NESTE CASO, ESTOU USANDO A VARIAVEL *X* GLOBAL dentro da rotina *soma*. Desta forma, a atribuição do número 35 ao *x* dentro da rotina, fará com que a variável *x* GLOBAL guarde este valor, ou seja, o python entenderá que NÃO FOI CRIADA uma variável *x* local.

32 – ROTINA COM RETORNO DE VALORES COMANDO return- FUNÇÕES

O uso de rotinas que retornam valores, conhecidas também como funções, serve para dinamizarmos mais alguns programas, veja um pequeno problema a seguir:

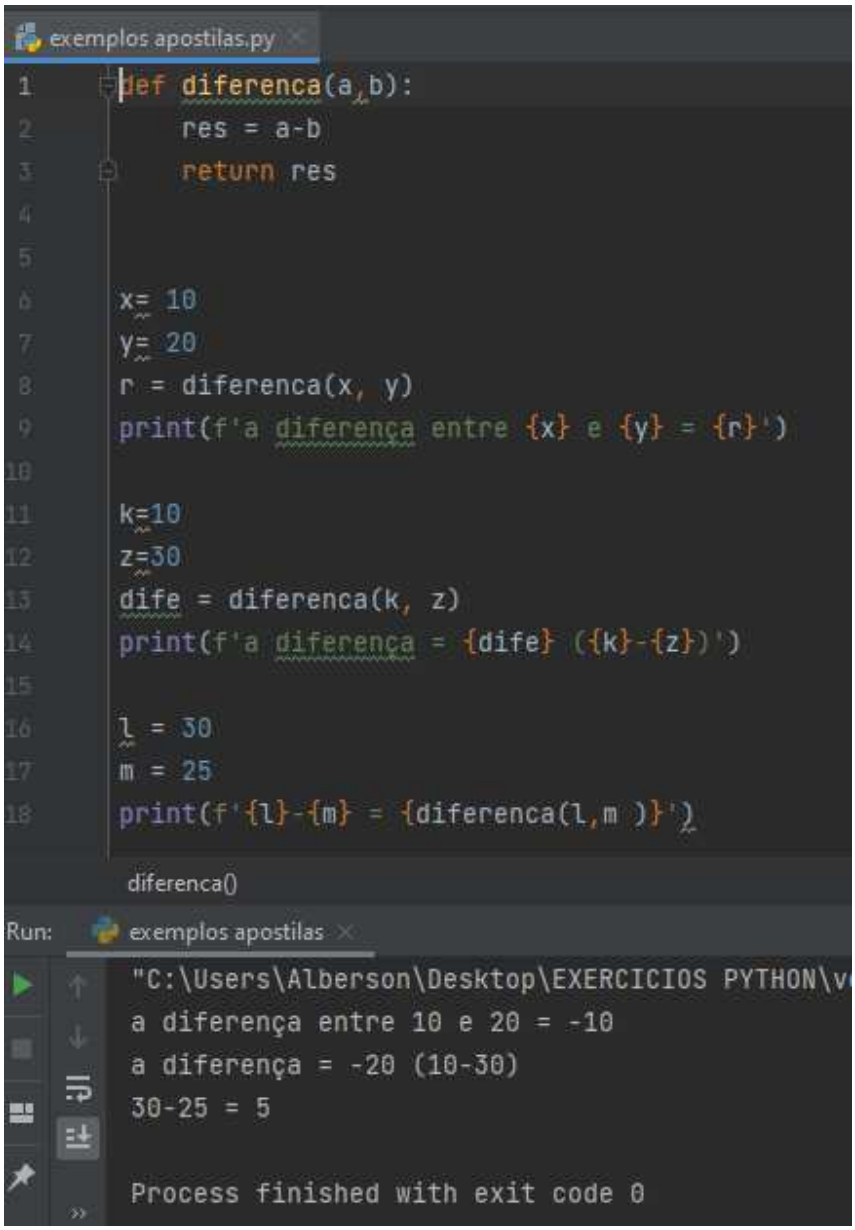
Exemplo 1:



Perceba que a impressão de cada resultado foi padronizada numa mesma mensagem, pois o comando print foi escrito dentro da rotina.

Posso mudar isso e criar uma mensagem distinta para cada resultado obtido. Contudo, devo usar o recurso de retorno do resultado com o comando return. Veja exemplo:

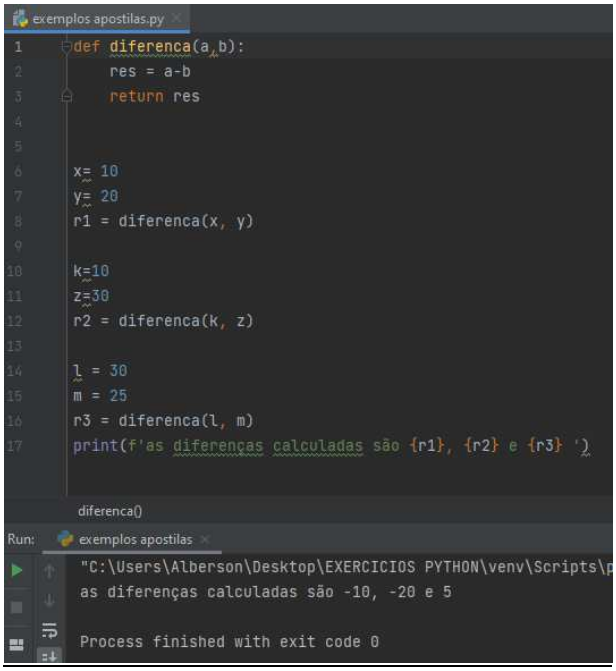
Exemplo 2:



OBSERVE ATENTAMENTE NO EXEMPLO:

a) CADA COMANDO DE CHAMADA NÃO FOI FEITO NUMA LINHA DE COMANDO ISOLADA, POIS SERÁ RETORNADO UM VALOR PELA FUNÇÃO E ESTE DEVE SER ARMAZENADO, OU ENTÃO USADO NUM COMANDO DE SAÍDA OU DE TESTE CONDICIONAL.
b) CADA IMPRESSÃO DE RESULTADO FOI FEITO USANDO UMA MENSAGEM DIFERENTE
c) DENTRO DA ROTINA FOI USADO O COMANDO ***return res***, PARA RETORNAR O RESULTADO DA DIFERENÇA ENTRE ***a*** E ***b***

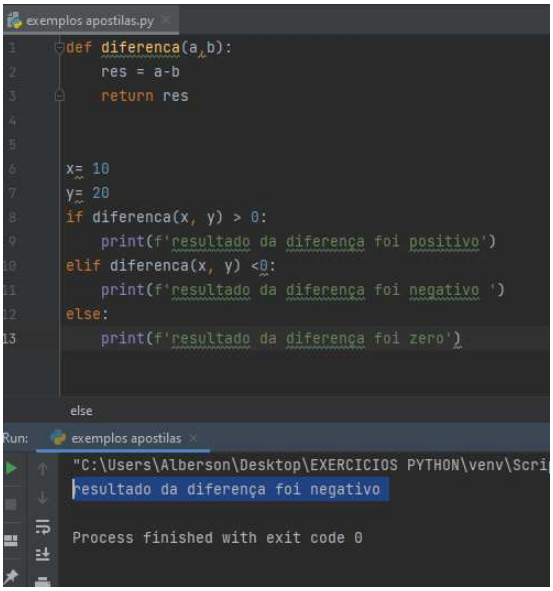
Exemplo 3:



Neste exemplo repare:

a) O resultado retornado para cada chamada da função ***diferença*** foi armazenado respectivamente em ***r1, r2*** e ***r3.***
b) Na última linha (17) usei apenas um comando ***print*** para exibir os 3 resultados das diferenças. Esta forma de impressão não seria possível sem o uso do comando ***return.***

Exemplo 4:



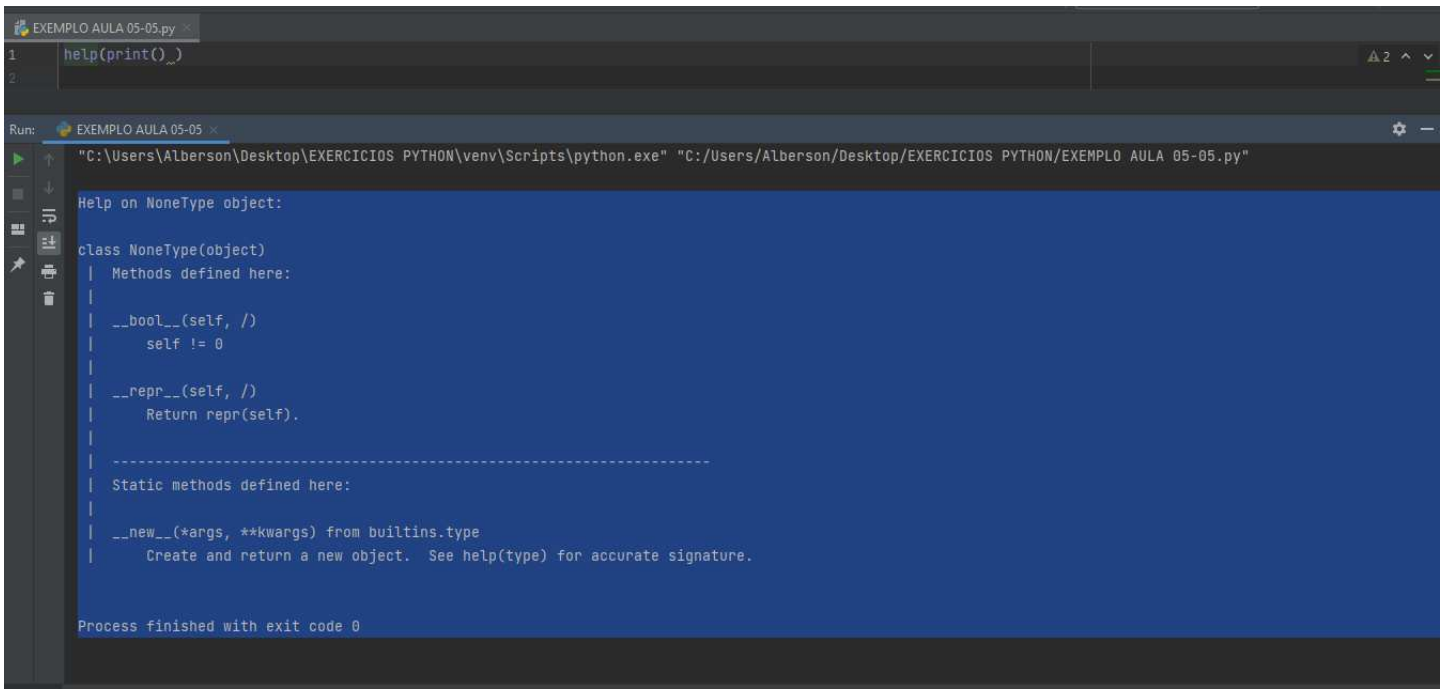
Observe mais um exemplo útil do uso do comando ***return***:

Neste caso a chamada da função foi realizada nos testes condicionais dos comandos ***if.*** O resultado retornado já será testado em relação ao número ***0***. Desta forma criamos um uma mensagem distinta para cada resultado.

33 – DOCSTRINGS – GERANDO DOCUMENTAÇÃO DE ROTINAS (procedimentos e funções) PARA USO NO COMANDO help()

A função ***help()*** é usada para consulta a informações de comandos e métodos do python. Com ela poderemos conhecer sintaxes de todos os comandos do python.

Exemplo 1:

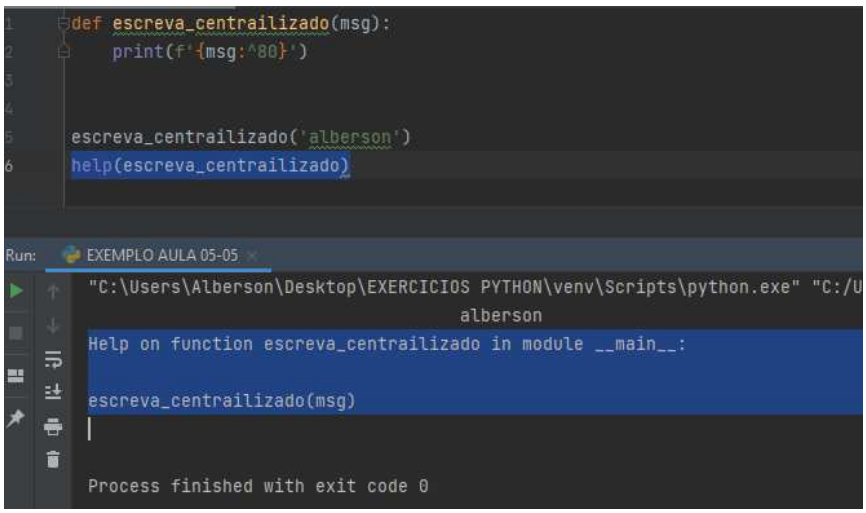


Repare!!!

a) Quando digitamos help(print()) o python mostra um as definições da função print.

O mesmo podemos fazer com nossas rotinas. Podemos documentá-las para facilitar o entendimento de outros programadores, que por ventura farão uso de uma rotina criada por nós. Vejamos um exemplo simples:

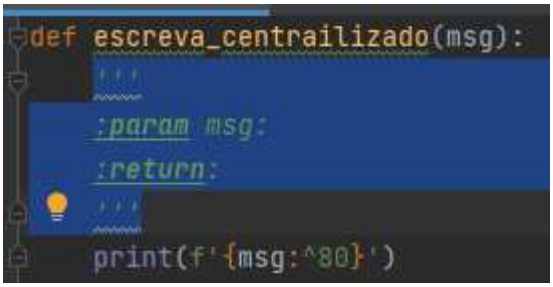
Exemplo 2:



Observem que neste caso, usei o *help()* para mostrar informações sobre a rotina "*escreva_centralizado()*" que criei no exemplo. Entretanto, foi mostrado somente o nome da rotina e o parâmetro esperado.

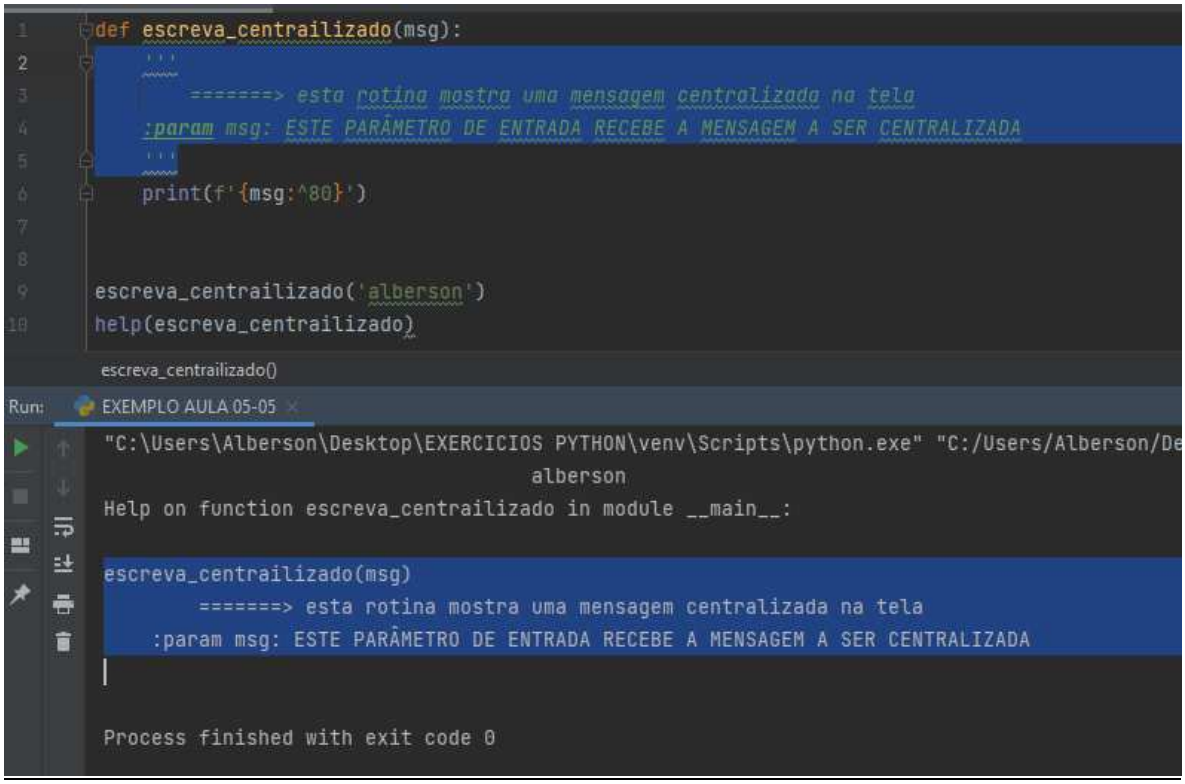
Para melhorarmos a exibição de informações sobre a rotina ***"escreva_centralizado()"*** vamos criar comentários no início da rotina. Estes serão mostrados pelo comando ***help()*** quando indicarmos o nome da referida rotina, veja:

Exemplo 3:



Repare que quando você criar a área de comentário, AUTOMATICAMENTE o Python já vai criar linhas para comentar sobre os parâmetros de entrada, e se for o caso, também irá preparar a linha do comentário do ***return***. Lembre-se que o ***return()*** só será tratado nos próximos capítulos, quando for abordado rotina com retorno de dados (funções).

POR EXEMPLO, PODEMOS PREENCHER ESTA ÁREA DA SEGUINTE FORMA:



Repare que quando o comando *help()* foi executado, os comentários criados no início da rotina foram mostrados para o programador.

34 – DESAFIOS SOBRE ROTINAS – (FUNÇÕES)

1) Desenvolver um programa python que tenha uma rotina para retornar uma MENSAGEM ao usuário informando se ALUNOS estão APROVADOS, REPROVADOS ou EM EXAMES.
   REGRAS:
   a) para ser APROVADO o aluno deve ter tirado média mínima 6;
   b) para estar EM EXAME o aluno deve ter tirado média entre 3.5 e menor que 6;
   c) para ser REPROVADO a média do aluno deve ser menor que 3.5; Neste programa peça ao usuário para digitar diretamente a média do aluno.

   NÃO É NECESSÁRIO REALIZAR O CÁLCULO DESTA MÉDIA.

2) Criar um programa python para solicitar ao usuário a digitação de vários números. A cada número digitado, chame uma função para realizar a soma dos seus antecessores até o número digitado. Entretanto, o usuário deve informar também para cada número digitado ele deseja ver o cálculo que está sendo feito. Assim sendo, a função recebe o número digitado e a resposta se deseja ou não ver o processo da soma. VALE INFORMAR O PADRÃO DA VISUALIZAÇÃO DO PROCESSO DA SOMA DEVERÁ SER **FALSO**. Veja os exemplos:
   a. Exemplo se usuário desejar visualizar o processo da soma sendo feito:
      Digite um número: 4
      Deseja visualizar o processo da soma? s: s
      Soma = 1+2+3+4 = 10

   b. Exemplo se usuário NÃO DESEJAR visualizar o processo da soma sendo feito:
      Digite um número: 5
      Deseja visualizar o processo da soma? s: (neste caso se o usuário não informar, por padrão será exibido resultado abaixo)
      Soma= 15

ATENÇÃO: CRIE O RECURSO PARA ENSINAR O PROGRAMADOR A USAR A FUNÇÃO DA SOMA CASO SEJA USADO O COMANDO help() DO PYTHON

3) Criar um programa python com uma função que receba parâmetros opcionais do nome e idade de uma pessoa. O programa deve informar mostrar o nome da pessoa e se ela é, ou não, maior de idade. ATENÇÃO: O NOME **E/OU** A IDADE DA PESSOA PODEM NÃO TEREM SIDO INFORMADOS E A FUNÇÃO DEVE CUIDAR DESTE CASO, VISTO QUE ESTAMOS USANDO UMA FUNÇÃO COM PARÂMETROS OPCIONAIS.

4) Criar um programa que solicite ao usuário a digitação de um número. O dado digitado deverá ser um número INTEIRO. Caso não tenha sido digitado um número inteiro, o programa deve solicitar novo número. ATENÇÃO: NO MÓDULO PRINCIPAL DO PROGRAMA DEVE TER SOMENTE AS DUAS LINHAS DE COMANDO ABAIXO:

```
n = digitenumero("Digite um numero:")
print(f"O número inteiro digitado foi {n}")
```

   observações:
   a. Repare que a função se chama "digitenumero"
   b. A função envia a mensagem como parâmetro

5) Fazer um programa em python que solicite ao usuário a digitação de vários faturamentos diários de um comércio. Passe estes N faturamentos para a função, a qual deverá retornar um DICIONÁRIO com os seguintes dados:
   - Total faturamento
   - Maior faturamento
   - Menor faturamento
   - Faturamento médio

35 – CRIANDO MODULOS DE PROGRAMAS

Agora veremos como criar módulos de programas para serem reaproveitados em outros projetos criados. Este assunto é muito importante, pois trata-se do reaproveitamento de códigos.

Imagine que você tenha rotinas criadas em um programa qualquer e queira utilizá-las em outros programas. Para isso, podemos criar um programa ".py" só com estas rotinas e importá-las sempre que necessário em programas desenvolvidos futuramente.

Vejamos o exemplo:

Vamos criar abaixo 2 programas python:

1) O primeiro, chamado "tratamentoidades.py": neste programa criaremos somente duas funções, a saber:
   - mn() - VERIFICARÁ SE IDADE RECEBIDA COMO PARÂMETRO CORRESPONDE A UMA PESSOA MAIOR OU MENOR DE IDADE, RETORNANDO UMA MENSAGEM AO PROGRAMA QUE A CHAMAR.
   - idadedaqui50anos() – SOMARÁ NA IDADE RECEBIDA COMO PARÂMETRO 50 ANOS E ROTORNARÁ ESTE RESULTADO AO PROGRAMA QUE A CHAMAR.

   Importante: Você perceberá que criei uma DocString para estas funcionalidades.

2) O segundo, chamado "testeidade_chamadaderotina.py": este programa importará o módulo "tratamentoidades.py" e chamará as duas funções nele criadas. Antes porém, você perceberá que também serão mostradas as funcionalidades do módulo, visto que usei o comando help(), que exibe as DOCSTRINGS criadas nas funções do módulo importado.

Vamos então criar o primeiro programa mencionado acima:

tratamentoidades.py

```
def mn(idade = 0):
    '''
    FUNÇÃO mn:
    ==========>>>>> ESTA FUNÇÃO DO MÓDULO maiormenor VERFICA SE UMA PESSOA É OU NÃO MAIOR DE IDADE
    :param idade: Dado do tipo inteiro que recebe uma idade no momento da chamada de mn do pacote maiormenor
    :return: retornará a mensagem 'maior' caso a idade recebida for maior ou igual a 18 anos;
             retornará a mensagem 'menor' caso a idade recebida form menor que 18 anos
    '''
    if (idade>=18):
        return 'maior'
    else:
        return 'menor'


def idadedaqui50anos(idade = 0 ):
    '''
    FUNÇÃO idadedaqui50anos:
    ====================> ESTA FUNÇÃO DO MÓDULO maiormenor CALCULA A IDADE DE UMA PESSOA DAQUI A 50 ANOS
    :param idade: Dado do tipo inteiro que recebe uma idade que será usada para somar 50 anos
    :return: Será retornado um dado do tipo inteiro que refere-se a idade recebida adicionada de 50 anos
    '''
    idadefutura = idade+50
    return idadefutura
```

INTERESSANTE DEMAIS!!!!! VALE COMENTÁRIO:

CASO ALTERARMOS O NOME DO PROGRAMA ACIMA, OS PROGRAMAS QUE O IMPORTARAM QUE ESTIVEREM NA MESMA PASTA QUE O PROGRAMA QUE O IMPORTOU, AUTOMATICAMENTE ADAPTARÃO AS LINHAS DE CÓDIGOS NO COMANDO IMPORT

Vamos criar agora o segundo programa:

testesidade_chamadaderotina.py

```
import tratamentoidades
#mostrando as funcionalidades do móudlo maiormenor
help (tratamentoidades)

#recebendo um idade informada pelo usário
id= int(input('digite sua idade'))
#chamando a funcionalidade mn() para verificar se idade corresponde a uma pessoa maior ou menor de idade
mensagem = tratamentoidades.mn(id)

#mostrando mensagem retornada pela função maiormenor
print(f'você é {mensagem}')

#Usando a funcionalidade idadedaqui50anos para mostrar a idade informada somada com 50 anos
novaid = tratamentoidades.idadedaqui50anos(id)
#mostrando a idade informada adicionada de 50 anos
print(f'Daqui a 50 anos você terá {novaid}')
```

Executando o programa acima teremos o seguinte resultado:

1) Primeiramente o resultado do comando help é mostrado, conforme destaque a seguir:

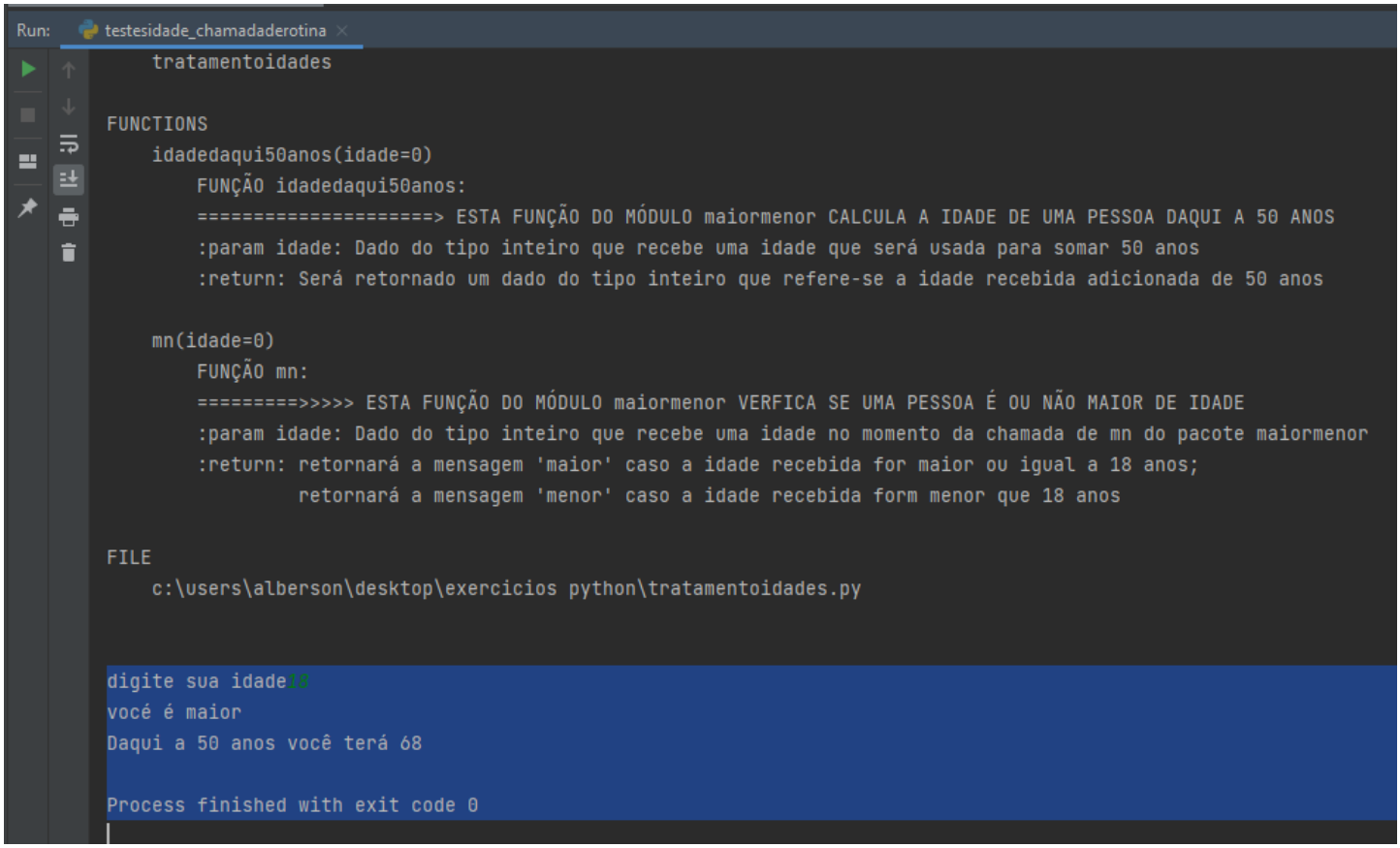
Run: testesidade_chamadaderotina

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/te
Help on module tratamentoidades:

NAME
    tratamentoidades

FUNCTIONS
    idadedaqui50anos(idade=0)
        FUNÇÃO idadedaqui50anos:
        ====================> ESTA FUNÇÃO DO MÓDULO maiormenor CALCULA A IDADE DE UMA PESSOA DAQUI A 50 ANOS
        :param idade: Dado do tipo inteiro que recebe uma idade que será usada para somar 50 anos
        :return: Será retornado um dado do tipo inteiro que refere-se a idade recebida adicionada de 50 anos

    mn(idade=0)
        FUNÇÃO mn:
        ==========>>>>> ESTA FUNÇÃO DO MÓDULO maiormenor VERFICA SE UMA PESSOA É OU NÃO MAIOR DE IDADE
        :param idade: Dado do tipo inteiro que recebe uma idade no momento da chamada de mn do pacote maiormenor
        :return: retornará a mensagem 'maior' caso a idade recebida for maior ou igual a 18 anos;
                 retornará a mensagem 'menor' caso a idade recebida form menor que 18 anos

FILE
    c:\users\alberson\desktop\exercicios python\tratamentoidades.py

digite sua idade
```

Veja que as DocString das funções existentes no módulo *"tratatamentoidades"* foram exibidas.

Em seguida, foi solicitado a digitação de uma idade.

Vamos digitar uma idade qualquer para testar a chamada das funções criadas no módulo importado, veja:



Em destaque acima, veja que digitamos 18. O programa chamou a rotina ***mn()*** e recebeu como retorno a mensagem ***"maior"***, que foi exibida concatenada numa frase. Depois foi chamada a função ***idadedaqui50anos()*** a qual retornou uma nova idade do usuário concatenada em outra mensagem, no caso ***68.***

35.1 – DESAFIOS DE MÓDULOS

1) Crie um módulo CHAMADO TESTE.PY que possua as seguintes funções:

a. Valida_CPF: Esta função deverá receber um parâmetro UM CPF EMPACOTADO. Em seguida, desempacote-o; verifique se é CPF recebido é ou não válido e retorne uma mensagem "CPF VÁLIDO" OU "CPF INVÁLIDO" para o programa que o chamou.
b. CRIE UM PROGRAMA QUALQUER QUE IMPORTE O MÓDULO TESTE.PY e use a função Valida_CPF, passando como parâmetro uma lista com os números de um cpf informado pelo usuário. Este programa irá receber e exibir a mensagem retornada pela função Valida_CPF.
c. Este programa deve terminar quando o usuário desejar.

36 – TRATAMENTOS DE ERROS E EXCEÇÕES - (try)

Quando desenvolvemos programas sabemos que erros podem ocorrer e com certeza aparecerão. Entretanto, sabemos que temos alguns tipos de erros tratáveis e outros não. Por exemplo:

- **ERROS DE SINTAXES:** Não devem ser tratados, pois cabe ao programador tomar o cuidado de usar o comando de forma correta.

- **ERROS DE SEMÂNTICOS:** Estes são exemplos de erros que podemos tratar em programas que desenvolveremos. Exemplos destes tipos de erros:
  - Tentar usar uma variável que não foi previamente criada
  - Tentar um abrir um arquivo que não existe
  - Não converter dados de uma variável para o tipo de dados necessário no momento de um cálculo
  - Solicitar a digitação de um número numa função que converte um dado digitado para inteiro ***int(input())*** e o usuário digita uma letra
  - Entre outros...

  O importante é saber que não se trata de erro de sintaxe.

36.1 – EXCEÇÕES

Normalmente criaremos trechos de programa para tratamentos de exceções que envolvam blocos de comandos os quais o programador imagina que pode ocorrer algum erro de semântica.

Vejamos alguns erros de exceções:

Exemplo 1: Uso de uma variável inexistente numa função print()

Observe!!!

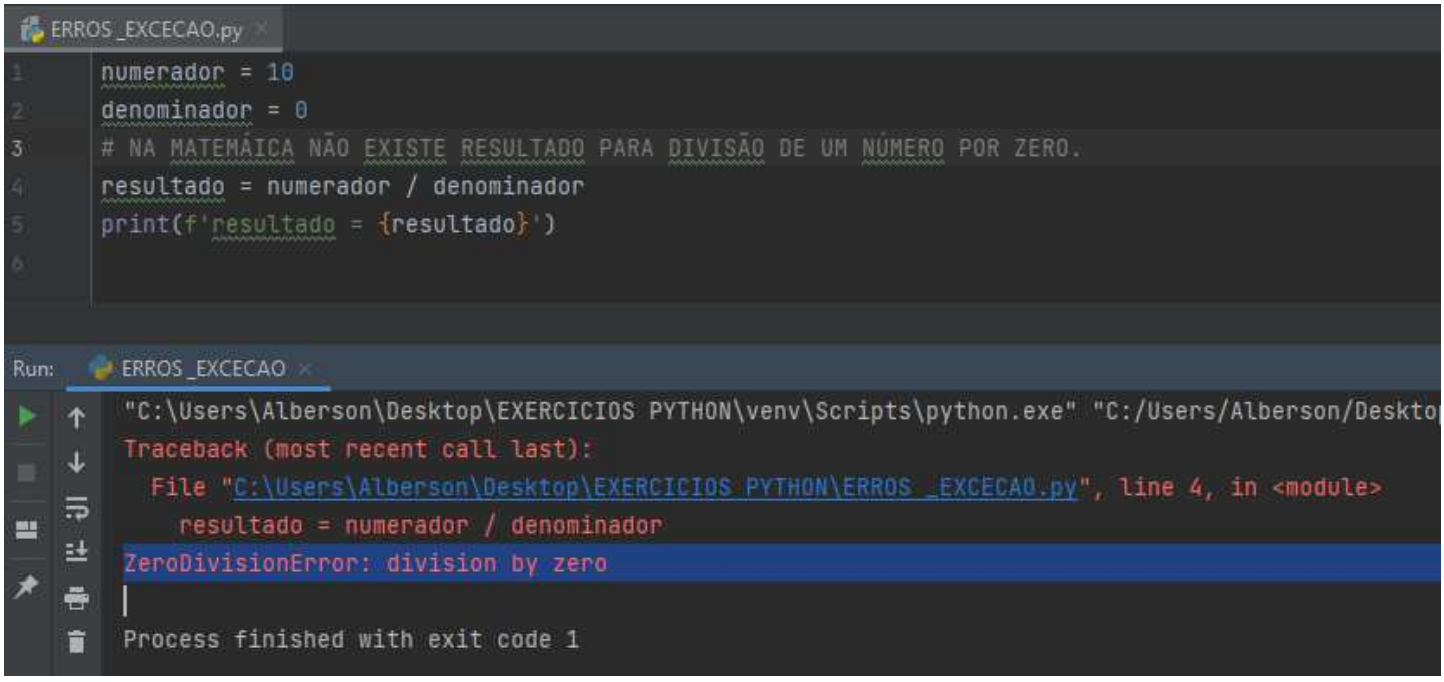
a) Neste comando **NÃO HÁ** erro de SINTAXE.
b) A variável ***nome*** não foi iniciada, isso provocou uma exceção chamada ***"NameError"***

Exemplo 2: **Um erro mais comum, tentando converter string para valores numéricos**

Observe!!!

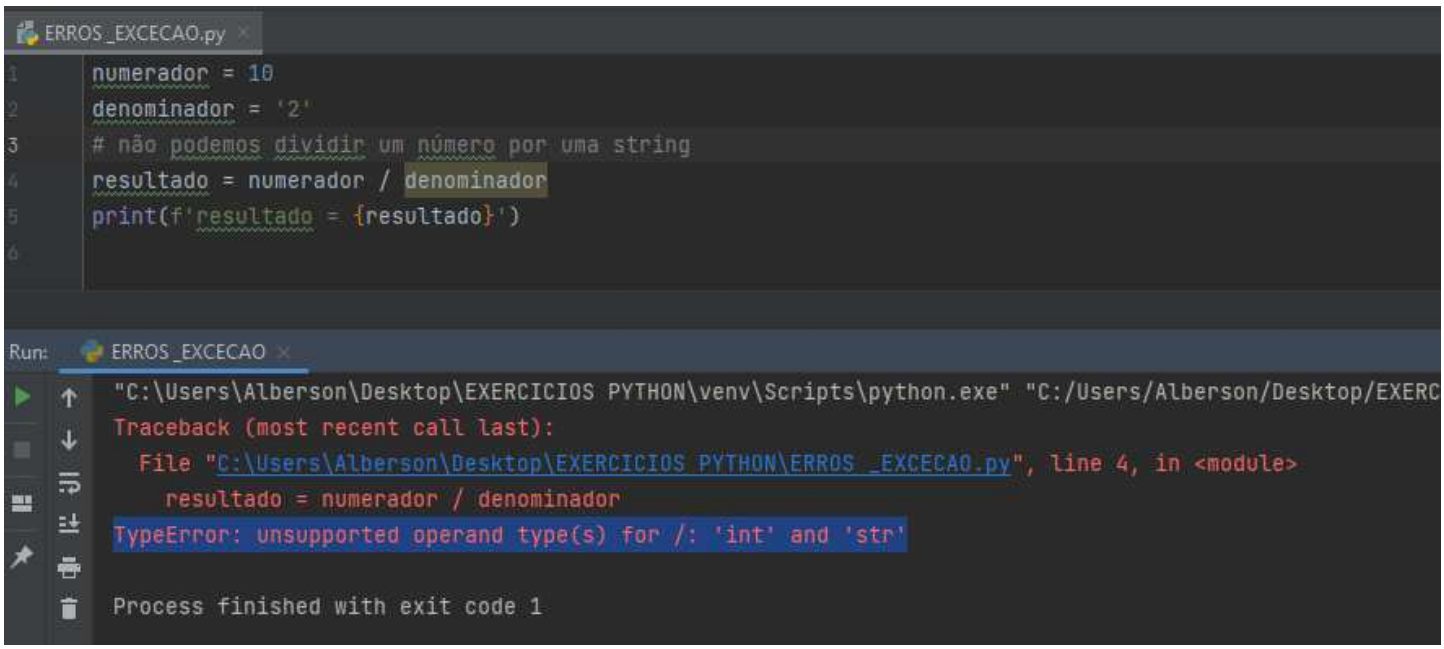
a) Observe acima que foi pedido para o usuário a digitação de um valor de produto. Ao invés de digitar um número foi digitado o meu nome.
b) Visto que esperava-se a digitação de uma string que pudesse ser convertida para float, neste caso ocorreu uma exceção chamada ***"ValueError".***

Exemplo 3: Erros de divisão por zero, é outro exemplo comum de exceção em programação python, veja:



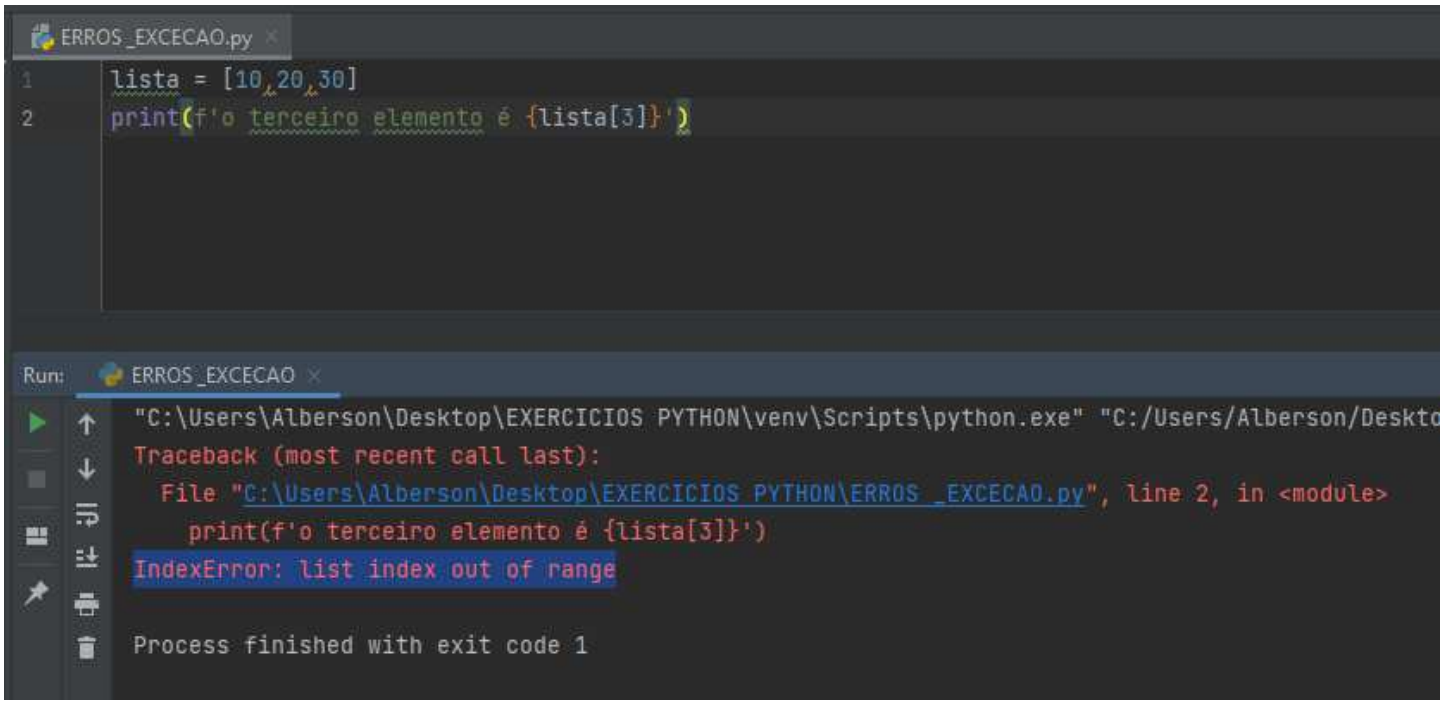
Neste caso a exceção gerada foi ***"ZeroDivisionError"***, pois tentamos dividir o numero 10 por 0.

Exemplo 4: Tentando dividir um numero por uma string. PYTHON NÃO ACEITA ISSO, VEJA:



Neste exemplo gerou-se uma exceção ***"TypeError"***, pois o denominador é de tipo string. Não podemos dividir um número por uma string

Exemplo 5: Quando trabalhamos com Tuplas, dicionários ou listas. Observe o exemplo abaixo com Lista:



Gerada exceção ***"IndexError"***, pois não existe a posição 3, ou seja, índice fora de range. Caso estivéssemos trabalhando com dicionários, a exceção seria ***"KeyError"*** porque dicionário não possui índice, mas sim chaves.

Exemplo 6: Importação de módulos inexistentes

```
ERROS_EXCECAO.py
1   import verificacpf
2

Run:  ERROS_EXCECAO
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Deskt
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\ERROS_EXCECAO.py", line 1, in <module>
    import verificacpf
ModuleNotFoundError: No module named 'verificacpf'

Process finished with exit code 1
```

Neste exemplo, repare que foi gerada uma exceção chamada ***"ModuleNotFound"***, pois não existe na pasta deste programa que estou iniciando um módulo ***verificacpf***

Assim sendo, temos uma quantidade grande de exceções que podem ser disparadas no seu programa python, observe uma lista de ALGUMAS:

ATENÇÃO!!!!
NÃO É NECESSÁRIO DECORAR OS NOMES DE EXCEÇÕES E QUANDO ESTAS VÃO APARECER.
QUANDO APARECEREM NOS PROGRAMAS QUE FOREM DESENVOLVIDOS BASTA TRATÁ-LAS.

36.2 – TRATANDO EXCEÇÕES COM COMANDO try:

Diante do que já conhecemos de exceções, vale ressaltar que uma exceção qualquer é "filha" de uma classe maior chamada ***Exception***.

Vamos usar o comando try: para tratar exceções, vejamos a 1ª sintaxe abaixo:

try:

<comandos que podem gerar exceções>

except:

<Comando(s) que deve(m) ser executado(s) no caso de EXCEÇÃO ocorrer>

else:

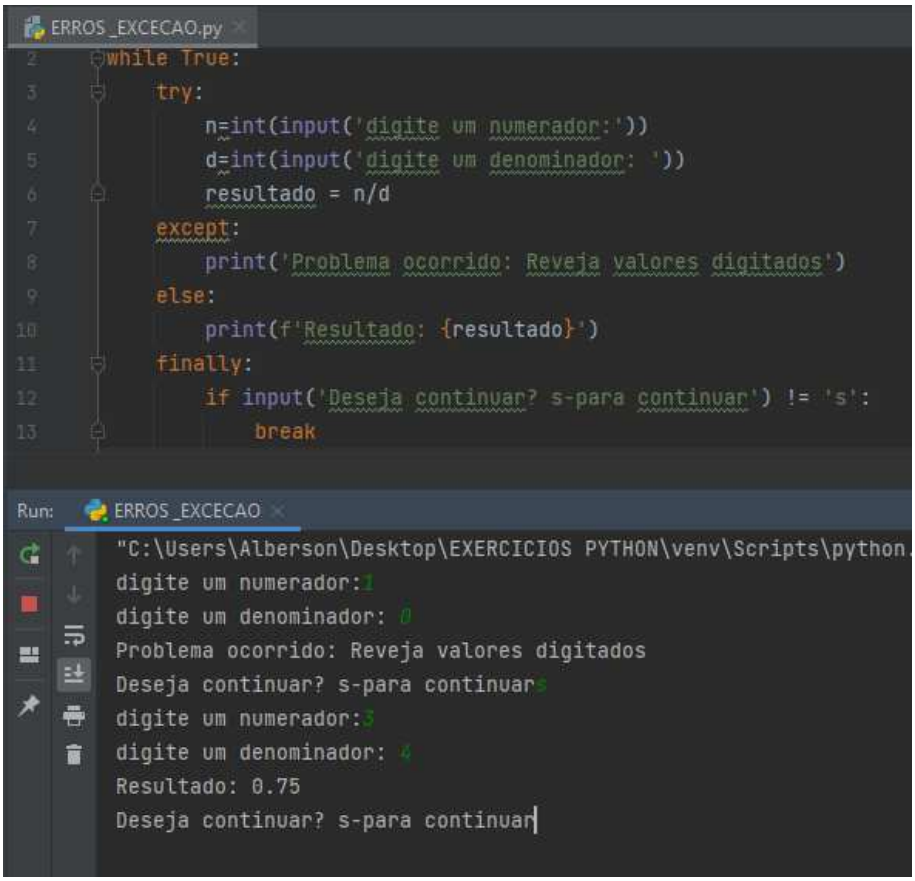
<comando(s) que deve(m) ser executado(s) quando NÃO OCORRER EXCEÇÃO>

finally:

<comando(s) que deve(m) ser executado(s) SE OCORRER OU NÃO EXCEÇÃO>

ATENÇÃO: *else:* e *finally:* são opcionais

Exemplo 1: REPARE QUE ESTE PROGRAMA NÃO TRAVARÁ EM HIPÓTESE ALGUMA



O que aconteceu quando entrou no laço de repetição pela primeira vez?

1) Foram digitados os números 1 para o denominador e 0 para denominador.
2) **Foi gerada exceção**, pois não podemos dividir o número 1 por 0. Assim sendo, a mensagem ***"Problema Ocorrido: Reveja valores digitados"*** foi mostrada, impedindo com isso que o programa travasse.
3) Em seguida, independente de ter ocorrido a exceção, o programa perguntou ***"Deseja continuar? s- para continuar"***. Veja que foi digitado "s" para o programa solicitar novos valores

O que aconteceu quando entrou no laço de repetição pela segunda vez?

a) Foram digitados os números 3 para o denominador e 4 para denominador.
b) **NÃO FOI GERADA EXCEÇÃO**, pois podemos dividir 3 por 4. Assim sendo, a mensagem ***"Resultado: 0.75"*** foi mostrada.
c) Em seguida, independente de ter ocorrido a exceção, o programa perguntou ***"Deseja continuar? s- para continuar"***.

Vejamos a 2ª sintaxe:

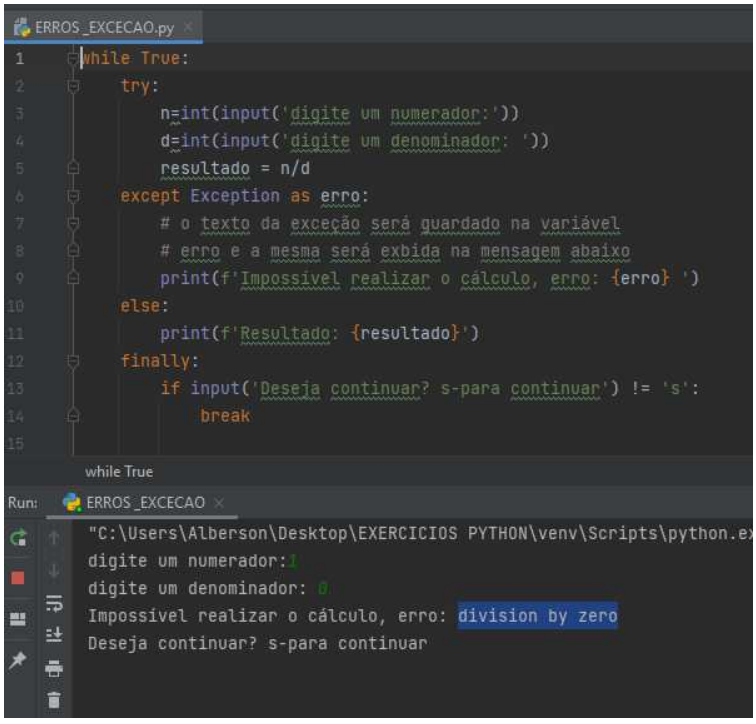
```
try:
    <comandos que podem gerar exceções>
except Exception as <nomevariávelerro>:
    <Comando(s) que deve(m) ser executado(s) no caso de EXCEÇÃO ocorrer>
```

ATENÇÃO:

else: **e** ***finally:*** **são opcionais, por isso não foram escritas acima, porém podem ser usadas como demonstradas na 1ª sintaxe**

Repare que na segunda sintaxe usaremos a classe Exception que irá atribuir a mensagem da exceção a uma variável, para que possamos reconhecer o erro e exibi-lo quando necessário, sem que trave o programa. Vejamos exemplo:

Exemplo1:

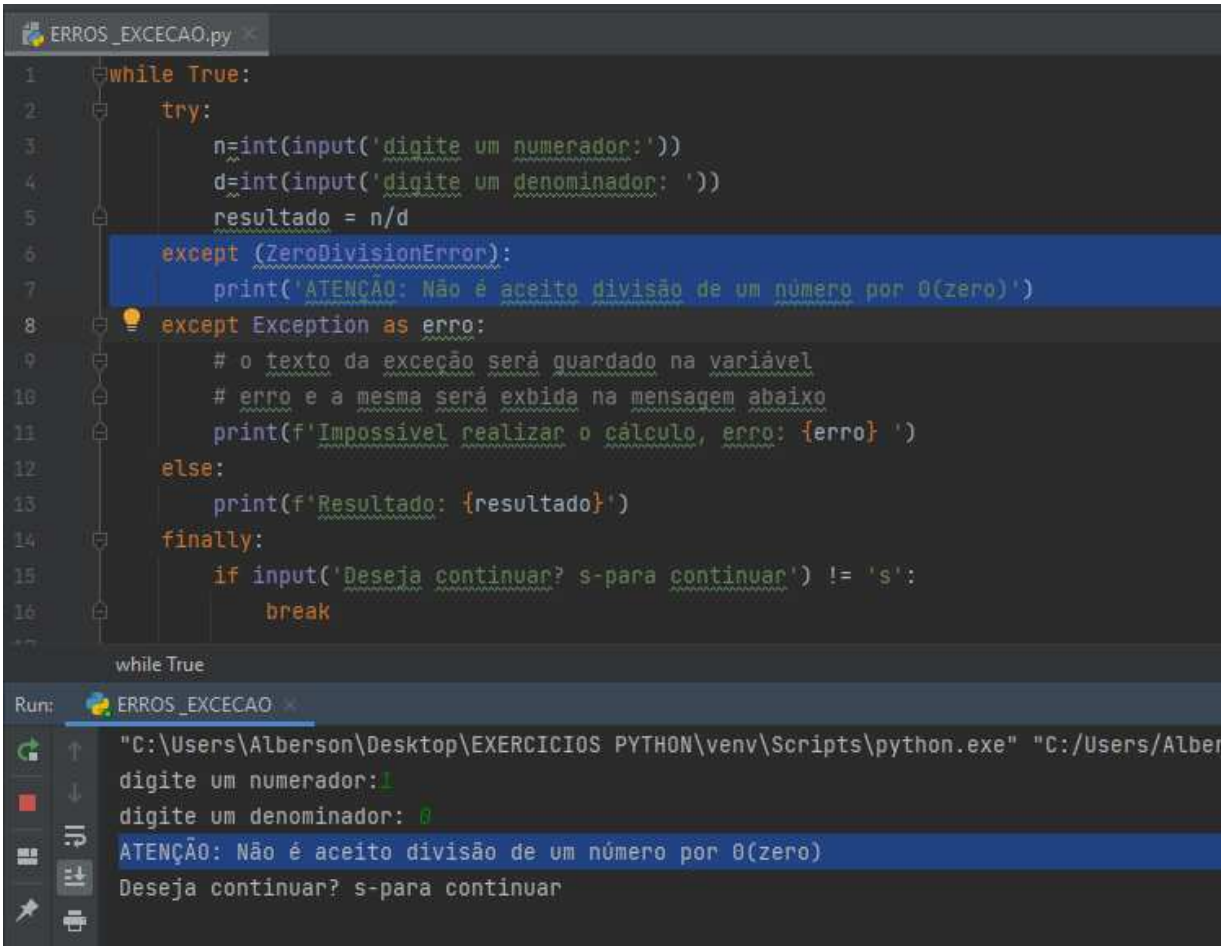


Observe!!!!

a) Na linha 6 o ***except*** utiliza a classe ***Exception*** que retornará para a variável ***erro*** o texto da exceção, quando esta ocorrer.
b) Veja que o print está exibindo o erro gerado pela exceção, visto que a variável ***erro*** foi usada na linha 9
c) No teste realizado veja que a mensagem ***"Impossível realizar cálculo, erro: Division by zero"*** impediu o travamento do programa e também exibiu qual o texto da exceção (motivo do erro).

Podemos também prever o erro na própria cláusula except e já personalizarmos uma mensagem, de tal forma que o usuário entenda a exceção gerada, veja:

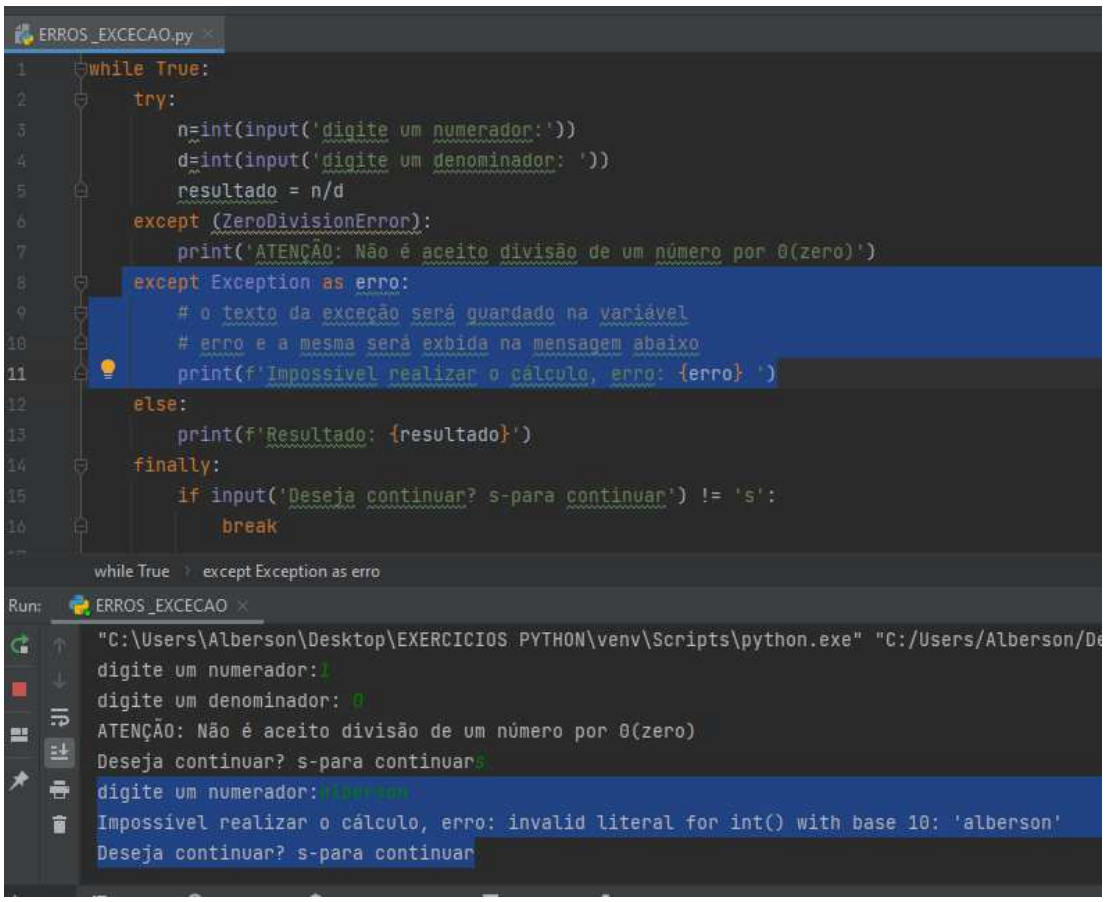
Exemplo 2:



Observem a vantagem!!!

a) Neste exemplo, eu programador, imagino que um dos erros que podem ocorrer é o ***"ZeroDivisionError"***, desta forma mencionarei este texto na frente o except.
b) Observe que propositalmente provoquei o erro e a mensagem exibida para o usuário ficou mais clara, de forma que ele entenda o motivo pelo qual a operação matemática não foi realizada.

Exemplo 3:



IMPORTATNE: AO CONTINUAR A EXECUÇÃO DO EXEMPLO PROVOQUEI UM OUTRO ERRO, PERCEBA QUE OUTRA MENSAGEM FOI EXIBIDA, VISTO QUE NÃO TINHA SIDO TRATADA ESTA POSSIBILIDADE DE TEXTO. ASSIM, FOI MOSTRADA A MENSAGEM GERADA PELO ***"except"*** ESCRITO NA LINHA 8.

36.3 – DESAFIOS DE TRATAMENTOS DE ERROS (try...)

a) Faça um programa que solicite a digitação de um número inteiro e outro real. Usando tratamentos de erros, quando o usuário digitar números de tipos diferentes dos que foram pedidos o programa o programa deve ficar pedindo até que ele digite o dado do tipo correto.

b) Desenvolver um programa que mostre para o usuário o seguinte menu:

1- QUADRADO DE UM NÚMERO
2- RAIZ CÚBICA DE UM NÚMERO
3- FATORIAL DE UM NÚMERO

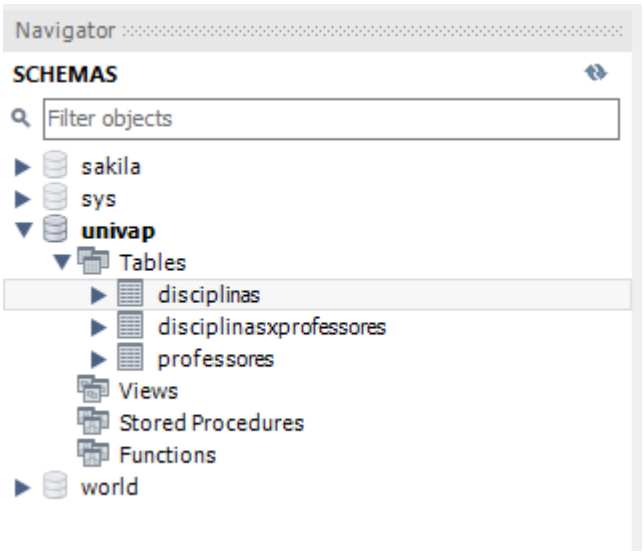
Após a exibição do menu acima, o programa deve solicitar que o usuário escolha uma destas opções.
Crie rotina de tratamento de erro (try) para que o usuário seja obrigado a digitar sempre um número de uma opção existente. Se por exemplo ele digitar "um", ou uma palavra qualquer ao invés do número da opção, o programa deve insistir na digitação correta e avisá-lo do motivo do erro. Outras rotinas try devem ser criadas para quando o usuário informar um número para cálculos escolhidos. Imagine se ele não digitar número algum e até mesmo digitar palavras. Isto não pode acontecer porque o python irá travar.

37 – BANCO DE DADOS MYSQL x PYTHON

Um importante assunto a ser abordado em Python é a manipulação de registros em banco de dados. Como exemplo, farei acesso a banco de dados com estrutura do MySql.

Os itens deste capítulo devem ser seguidos com muita atenção. Assim sendo, primeiramente vou criar um banco de dados no MySql (usei o WorkBench para tal) e nele criarei também as seguintes tabelas:

- disciplinas
- professores
- disciplinasxprofessores



IMPORTANTE:

PARA QUE TUDO FUNCIONE DE ACORDO COM OS EXEMPLOS QUE ABORDAREI, DÊ OS MESMOS NOMES PARA SEU BANCO DE DADOS E TABELAS, CONFORME A IMAGEM ACIMA.

Vejamos agora a estrutura de cada tabela que deve ser criada:

Tabela **disciplinas** - Crie a seguinte estrutura:

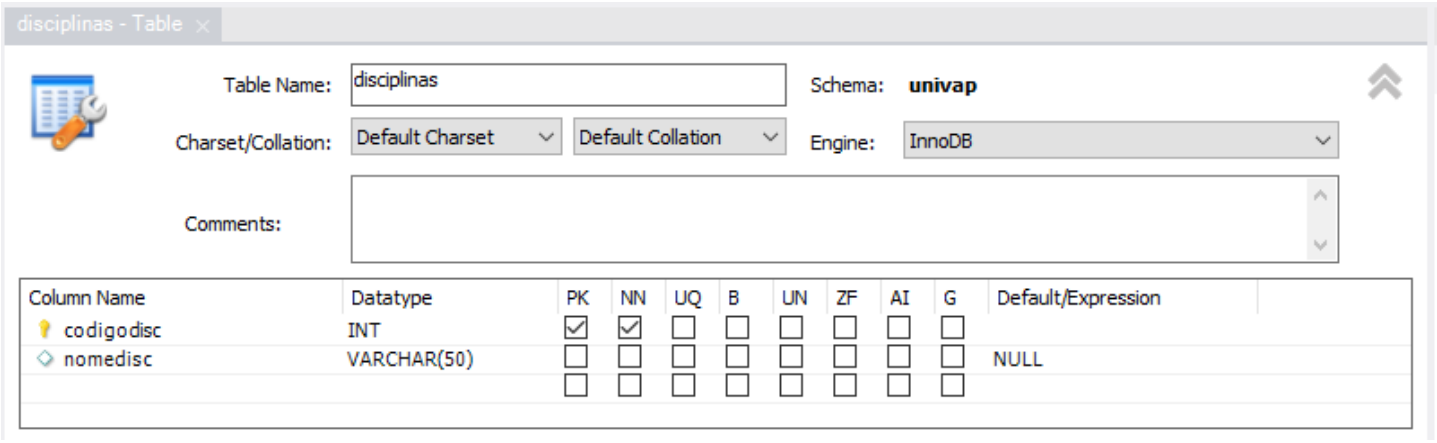
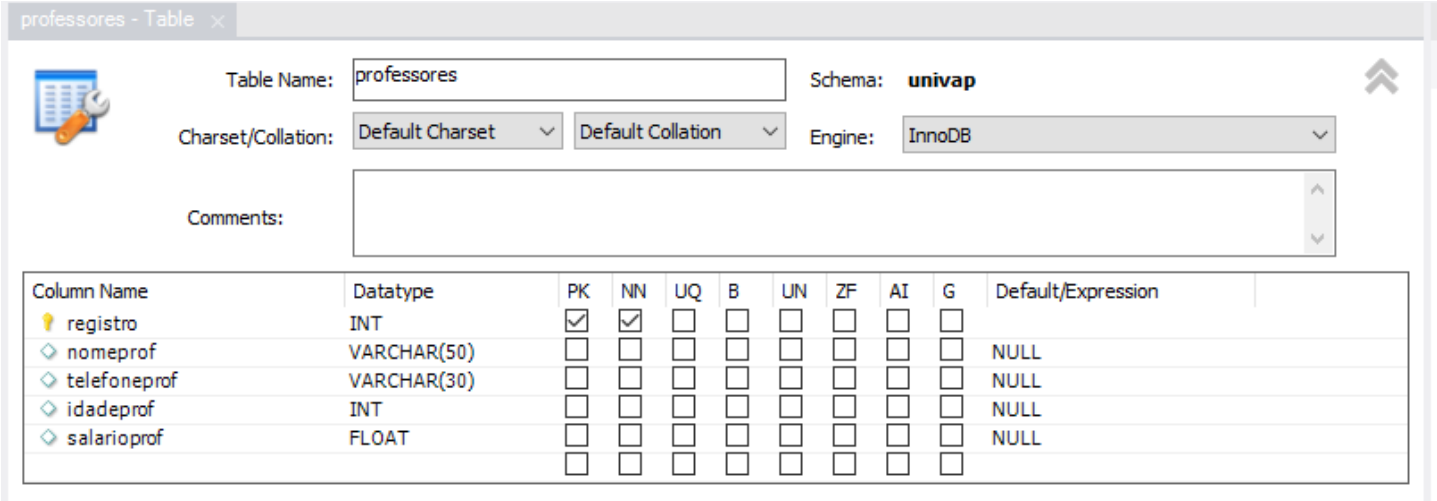
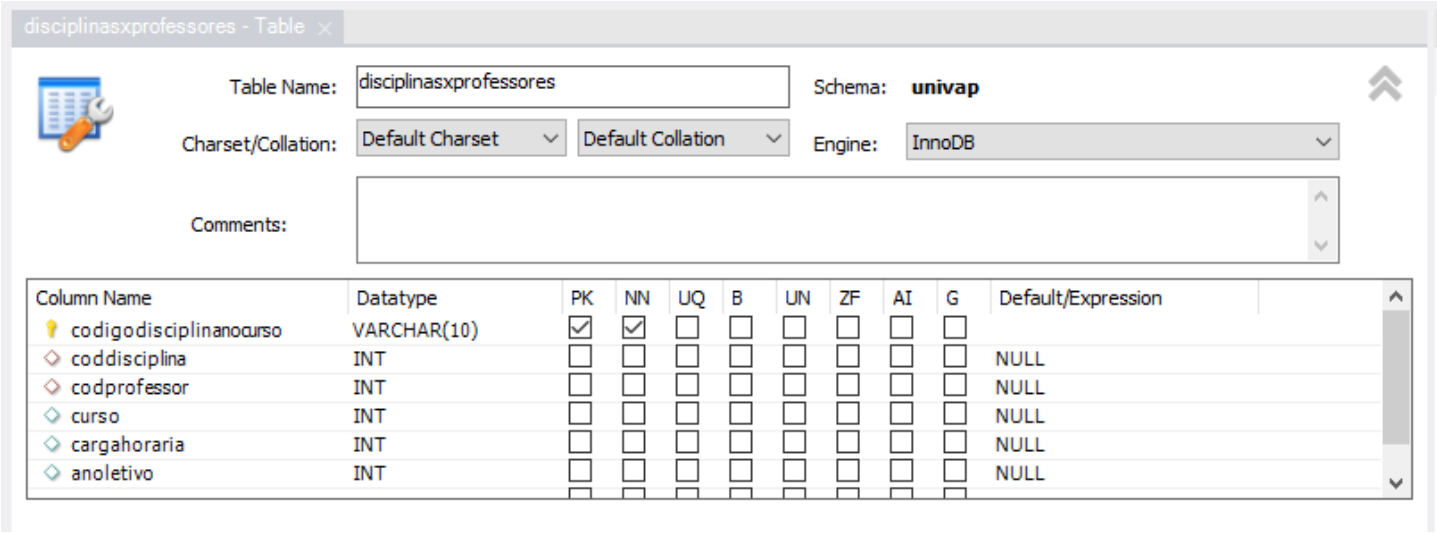


Tabela **professores** – Crie a seguinte estrutura:



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
registro	INT	☑	☑	☐	☐	☐	☐	☐	☐	
nomeprof	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
telefoneprof	VARCHAR(30)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
idadeprof	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
salarioprof	FLOAT	☐	☐	☐	☐	☐	☐	☐	☐	NULL

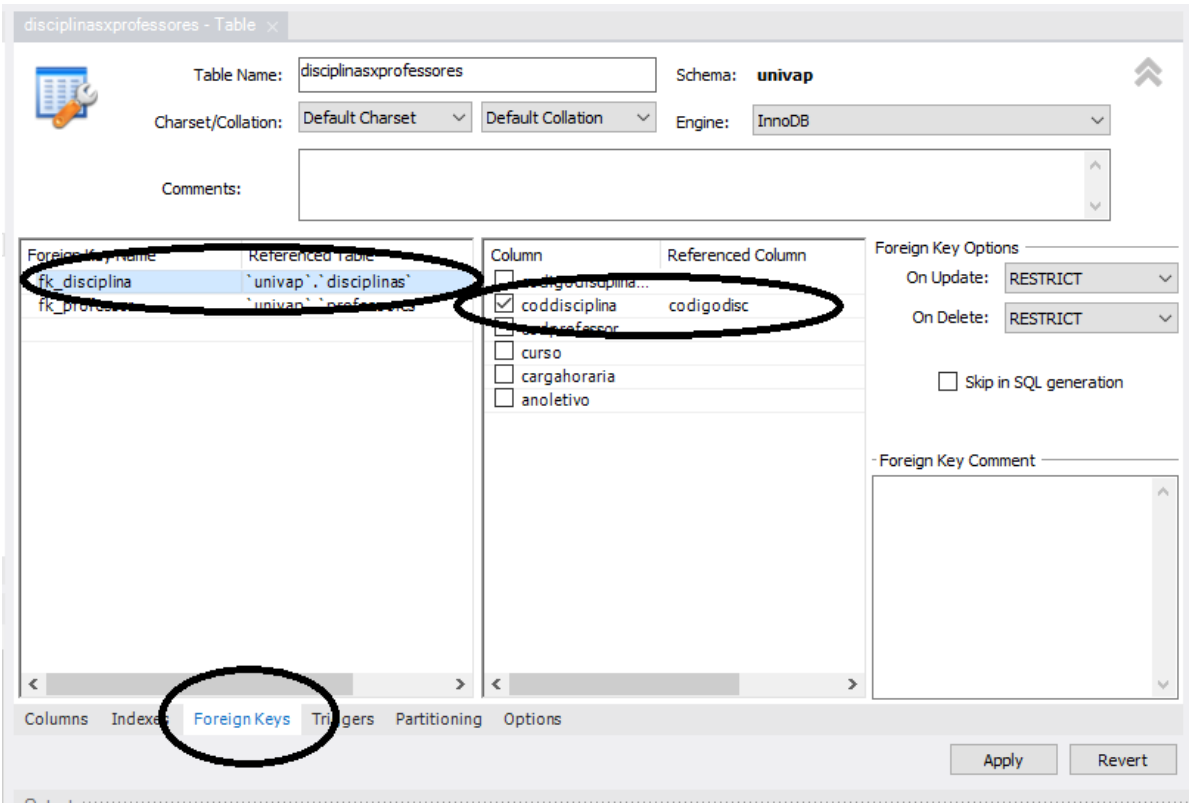
Tabela **disciplinaxprofessores:** Por fim, crie a seguinte estrutura observando as principalmente as definições de chaves estrangeiras:



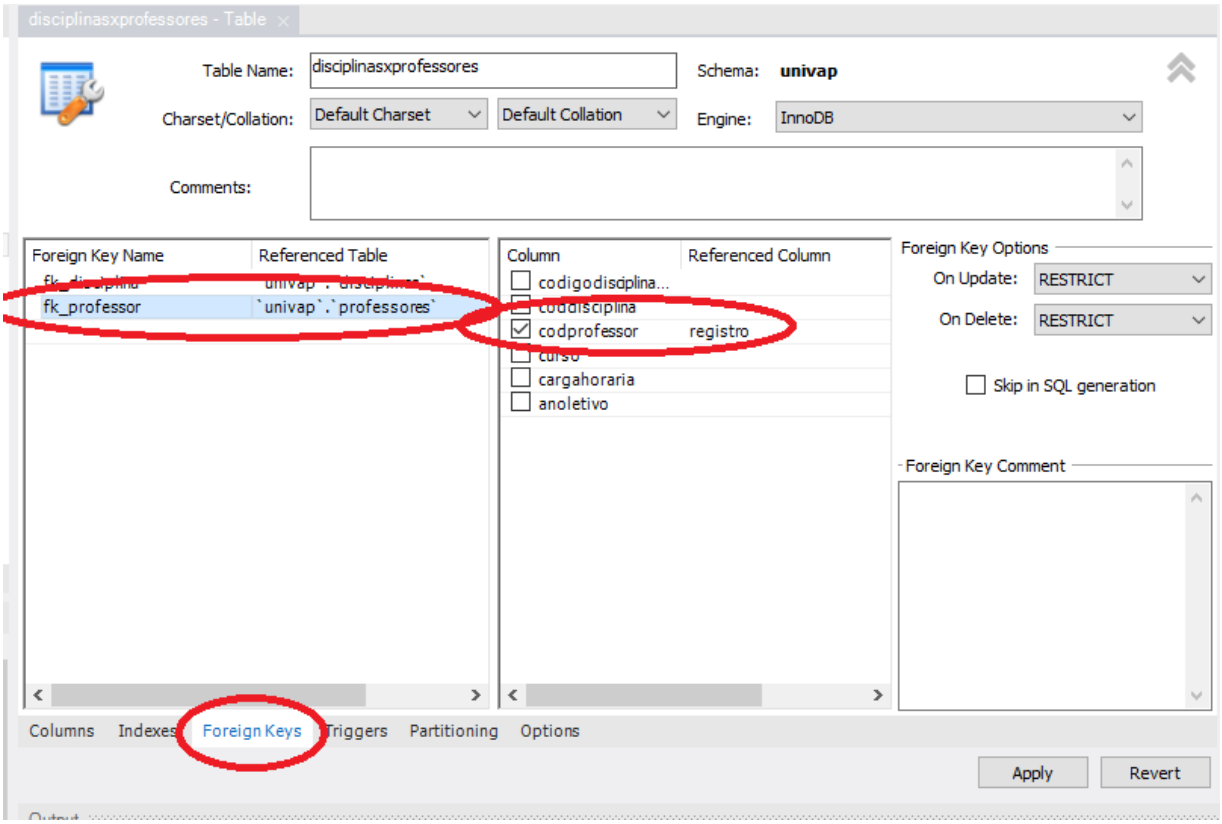
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
codigodisciplinanocurso	VARCHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
coddisciplina	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
codprofessor	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
curso	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
cargahoraria	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
anoletivo	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Repare que os campos coddicisplina e codprofessor são chaves estrangeiras desta tabela e, portanto, devem obrigatoriamente ser do mesmo tipo das respectivas chaves primárias nas tabelas “pai” do relacionamento que será criado.

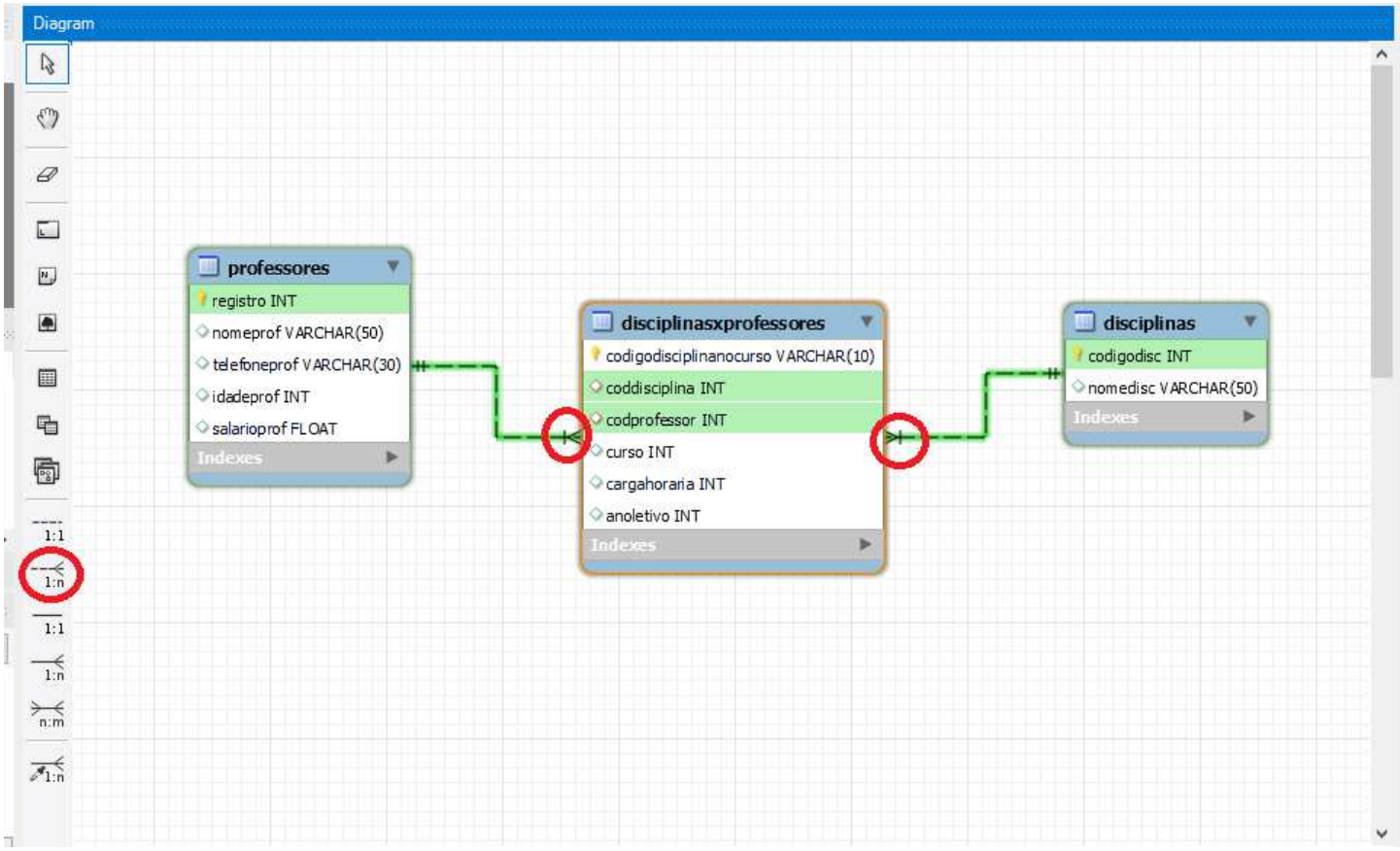
Veja tais definições de relacionamentos entre as tabelas a seguir:



Devemos criar também a seguinte relação:



Após criadas as tabelas e os relacionamentos deveremos ter o seguinte resultado, visualizado no diagrama do banco de dados a seguir:



Observe, que de acordo com as linhas de relacionamento foram criadas relações de 1:n (lê-se: 1 para n)

Diante das relações criadas, temos então a seguinte regra de negócio para esta escola:

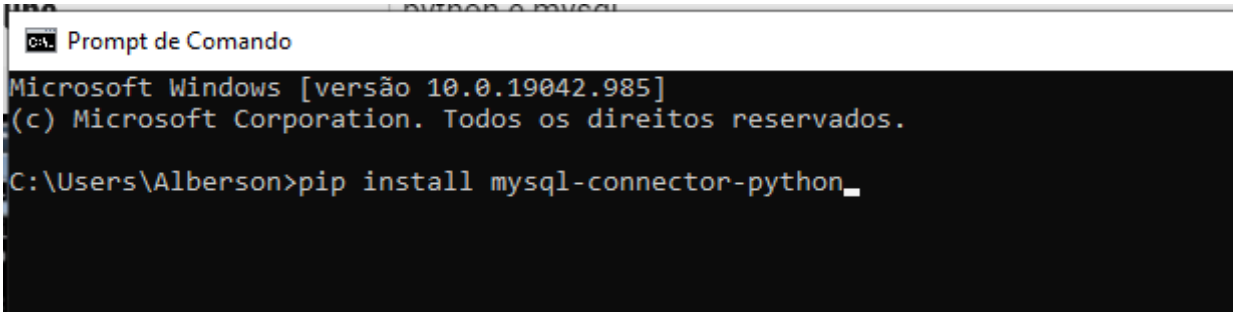
1 professor poderá dar aula de N disciplinas

1 disciplina poderá ser lecionada por N professores

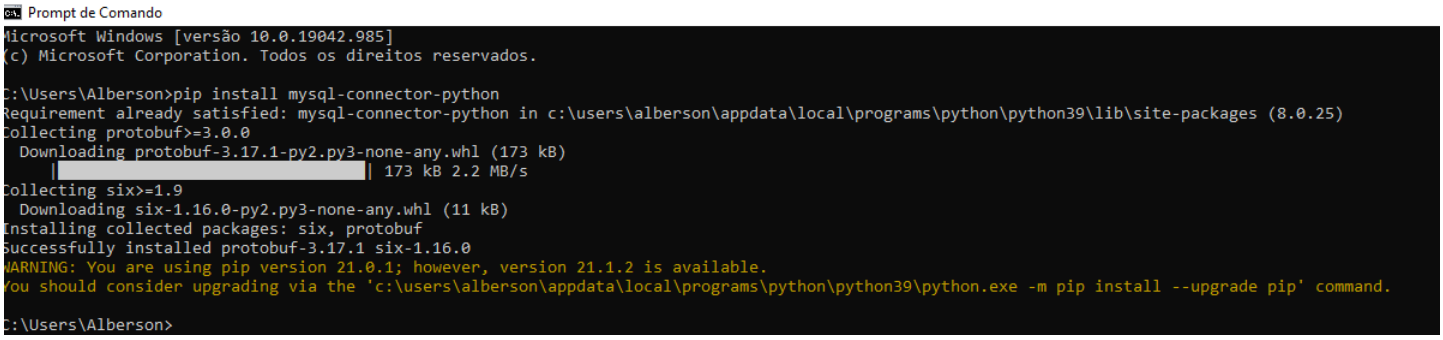
Nesta apostila não será abordado o assunto sobre uso do WorkBench, pois esse já foi abordado na disciplina de banco de dados do curso.

37.1 – Instalando a biblioteca mysql-connector-python com pip

Para acessar o banco de dados mysql que foi criado, precisaremos instalar o mysql-connector-python, usando o instalador pip do Python. Para isto digite a seguinte linha de comando, no seu **prompt de comandos do Windows**:

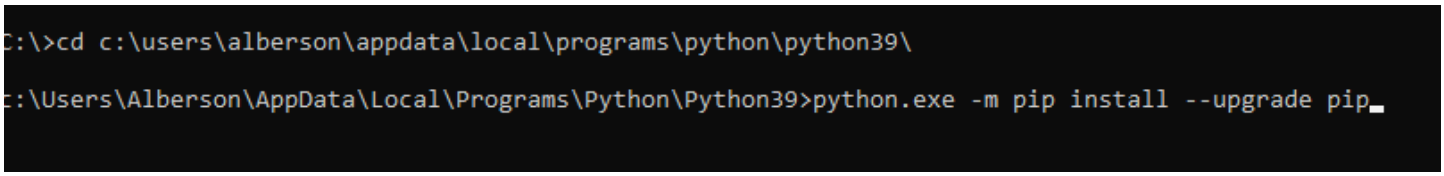


Ao pressionar <enter> será instalado o pacote de acesso a banco de dados do mysql, o qual iremos importar para dentro dos nossos programas Python de agora para frente, veja:

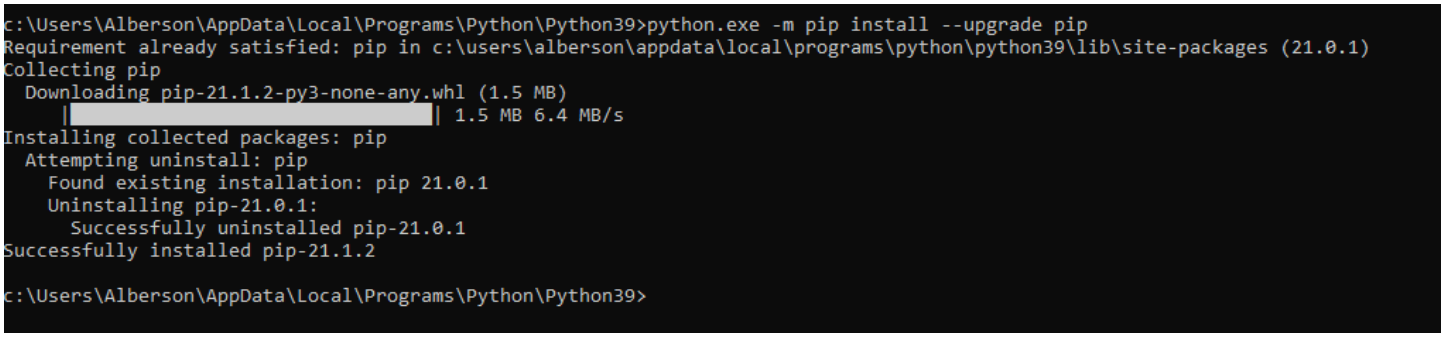


Agora podemos desenvolver nossos programas em Python, com acesso a banco de dados do mysql, pois já temos instalada a biblioteca mysql-connector-pyhon.

As linhas em amarelo, mostradas acima só estão indicando que o instalador pip está desatualizado, nada de grave para esta situação. Lembre-se que já tratamos sobre isso no início do nosso curso. Só para lembrar como atualizamos o instalador de bibliotecas pip, siga os passos a seguir:



Ao pressionar <enter> a atualização do pip será realizada e o resultado será:



Pronto!!!! Até que seja solicitado informado novamente, minha versão do pip install já está atualizada.

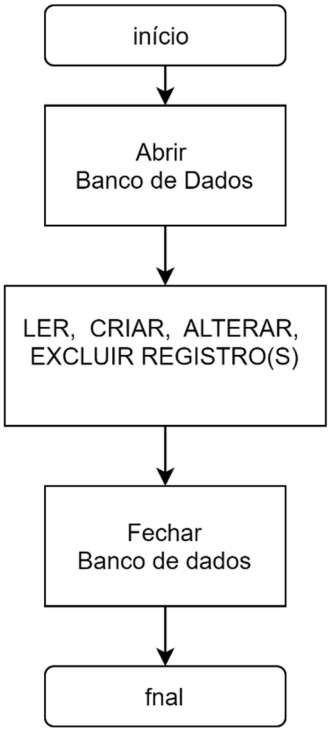
37.2 – Abrindo banco de dados num programa Python

Um CONCEITO muito usado por conhecedores de banco de dados é CRUD. CRUD é importante para o bom entendimento sobre uso de banco de dados. Você se lembra do que significa CRUD?

- **C (Create/insert) – Criar registros**
- **R (Read/select) – Ler registros**
- **U (Update) – Atualizar/Alterar Registros**
- **D (Delete) – Deletar/Excluir registros**

Diante do conhecimento sobre CRUD, podemos então dar início ao desenvolvimento de programas que acessam e **manipulam dados e estrutura de banco de dados**.

A lógica básica que utilizamos em todos os programas para manipulação de registros e estruturas de um banco de dados é representada no diagrama abaixo, independente da operação desejada, ou seja, cadastrar, alterar, consultar, excluir registros e também para criar estruturas gerais de tabelas num banco de dados. Veja:



O diagrama ilustra basicamente os processos normalmente realizados para usarmos trabalharmos com qualquer arquivo acessado por um software desenvolvido. **Por exemplo**: quando vamos usar um editor de texto para escrever um curriculum, abrimos o editor e um novo arquivo, criamos, alteramos, lemos e/ou excluímos partes indesejadas deste documento de texto e depois fechamos o arquivo e o editor. Os mesmos passos devemos seguir usando programação para OPERAÇÕES CRUD em tabelas de banco de dados, ou seja, abriremos o banco de dados para manipular o registros em suas tabelas e depois DEVEMOS FECHAR A CONEXÃO COM O BANCO DE DADOS.

Para abrir o banco de dados, use o código a seguir:

```python
import mysql.connector
try:
    #criando um objeto para conexão ao banco de dados
    #Devemos indicar em :
    # host = Servidor ou IP(caso esteja numa rede),
    # database = nome do banco de dados criado no workbenck
    # user = nome do usuário definido no momento de criação do banco de dados
    # password = senha de acesso do banco de dados
    conexao = mysql.connector.Connect(host='localhost',database='univap', user='root', password='mae240299')
    # testando se estamos conectado ao banco de dados
    if conexao.is_connected():
        informacaobanco = conexao.get_server_info()
        print(f'conectado ao servidor banco de dados - Versão {informacaobanco}')
        print('Conexão ok')
        # criando objeto cursor, responsável para trabalharmos com registros retornados pela tabela fisica
        comandosql = conexao.cursor()
        # Criando uma QUERY para mostrar as informações do banco de dados ao qual nos conectamos
        comandosql.execute('select database();')
        # usando método fetchone para buscar um dado do banco de dados e armazenálo na variável nomebanco
        nomebanco = comandosql.fetchone()
        print(f'Banco de dados acessado = {nomebanco}')
    else:
        print('Conexão não realizada com banco')
except Exception as erro:
    print(f'Erro : {erro}')

try
```

Run TODO Problems Terminal Python Packages Python Console

Este código, quando executado mostrará os seguintes dados para o usuário:

```
Run: Tela_disciplinas EXEMPLOS APOSTILA
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/Alberson/Desktop/BancoPython/EXEMPLOS APOSTILA.py"
conectado ao servidor banco de dados - Versão 8.0.25
Conexão ok
Banco de dados acessado = ('univap',)

Process finished with exit code 0
```

Run TODO Problems Terminal Python Packages Python Console

O programa mostrado anteriormente poderia ser resumido basicamente em:

```python
import mysql.connector

try:
    # criando um objeto para conexão ao banco de dados
    # Devemos indicar em :
    # host = Servidor ou IP(caso esteja numa rede),
    # database = nome do banco de dados criado no workbenck
    # user = nome do usuário definido no momento de criação do banco de dados
    # password = senha de acesso do banco de dados
    conexao = mysql.connector.Connect(host='localhost', database='univap', user='root', password='xxxxx')
    # testando se já estamos conectado ao banco de dados
    if conexao.is_connected():
        print('Banco de dados ABERTO: Conexão realizada com banco')
    else:
        # Se por algum motivo não estiver conectado mostrar mensagem abaixo
        print('Banco de dados FECHADO: NÃO OCORREU Conexão com banco')
except Exception as erro:
    # Caso tenha dado erro no momento da conexão, este será exibido abaixo:
    print(f'Erro : {erro}')
```

```
Run:  Tela_disciplinas  EXEMPLOS APOSTILA
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/Alberson/Desktop/BancoPython/EXEMPLOS APOSTILA.py"
Banco de dados ABERTO: Conexão realizada com banco

Process finished with exit code 0
```

IMPORTANTE DEMAIS NESTE TRECHO DE PROGRAMA !!!

Biblioteca ***"mysql.connector"*** = Permite que um programa se conecte a uma base de dados MySql pelo Python

Método ***"Connect()"*** do objeto "***connector***" ("**mysql.connector.Connect()")=** responsável por estabelecer a conexão com o banco de dados, ou seja, abrir o banco de dados.

Método ***"is.connected()"*** do objeto de conexão = Responsável por verificar se existe uma conexão com um banco de dados ou não. Retorna True se existir.

37.3 – Cadastrando registros numa tabela do Banco de Dados

Para exemplificar esta operação e esta funcionar, tenhamos em mente que o banco **deverá estar aberto**, pois caso contrário dará erro.

Exemplo: Vou cadastrar uma nova disciplina na tabela ***"disciplinas"***, no banco de dados ***"univap"***:

```
1  import mysql.connector
2  try:
3      conexao = mysql.connector.Connect(host='localhost',database='univap', user='root', password='mae240299')
4      if conexao.is_connected():
5          comandosql = conexao.cursor()
6          cd = int(input("CÓDIGO DA DISCIPLINA: "))
7          nd =(input("Nome da Disciplina: "))
8          # criando comando insert e concatenando os dados a serem gravados
9          comandosql.execute(f'insert into disciplinas(codigodisc, nomedisc) values({cd},"{nd}") ;')
10
11         # método commit é responsável por gravar de fato o novo registro de disciplina na tabela
12         conexao.commit()
13
14         print('Cadastro feito com sucesso ............!!!')
15
16         #fechando a conexão, ou seja, fechando o banco de dados aberto
17         comandosql.close()
18         conexao.close()
19 except Exception as erro:
20       print(f'Ocorreu erro: {erro}')
```

except Exception as erro

```
Run: CADASTRO
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Alberson/Desktop/BancoPython/CADASTRO.py
CÓDIGO DA DISCIPLINA: 11
Nome da Disciplina: matemática
Cadastro feito com sucesso ............!!!
```

IMPORTANTE DEMAIS NO PROGRAMA ACIMA !!!

Método ***cursor()***, usado na linha 5: Nesta linha estou criando um objeto cursor, para permitir navegar e manipular registros em tabelas no banco de dados

Método ***execute(),*** usado na linha 9: Este método executa a instrução SQL em questão, ou seja, será responsável por inserir na tabela um novo registro, quando o método ***commit()*** for executado***.***

Método ***commit(),*** usado na linha 12: Responsável **NESTE EXEMPLO**, por gravar fisicamente na tabela informada, um novo registro de disciplina (código da disciplina e nome da disciplina)

Método ***close()***, usados nas linhas 17 e 18 são importantes para limpar o objeto comandosql (***comandosql.close()***) e fechar o banco de dados (***conexao.close()***), respectivamente.

37.4 – Consultado por um registro numa tabela do Banco de Dados

O processo de consulta pode ser feito basicamente sobre qualquer dado gravado em campo(s) da tabela. É IMPORTANTE SABER QUE ALGUMAS CONSULTAS RETORNARM VÁRIOS REGISTROS E OUTRAS SOMENTE UM.

Para o exemplificar o processo de consulta, vamos realizar um teste, consultando um registro pela sua chave primária. Desta forma, não possibilidade de retorno de mais de um registro.

Exemplo: Vou consultar por um código de disciplina, cadastrada na tabela ***"disciplinas"*** no banco de dados ***"univap":***

```
1  import mysql.connector
2  try:
3      conexao = mysql.connector.Connect(host='localhost',database='univap', user='root', password='mae240299')
4      if conexao.is_connected():
5          comandosql = conexao.cursor()
6          cd = int(input("CÓDIGO DA DISCIPLINA: "))
7          comandosql.execute(f'select * from disciplinas where codigodisc = {cd};')
8
9          #O MÉTODO fetchall() retornará todos os registros filtrados (um ou mais registros) pelo comando select
10         tabela = comandosql.fetchall()
11
12         #O método rowcount conta quantos registros foram filtrados, caso tenha registro filtrado entra no if
13         if comandosql.rowcount > 0:
14             # se existir pelo menos uma disciplina na tabela temporária, mostre os dados da coluna 1
15             for registro in tabela:
16                 print(f'Nome da Disciplina: {registro[1]}')
17         comandosql.close()
18         conexao.close()
19 except Exception as erro:
20       print(f'Ocorreu erro: {erro}')
21
```

```
try
Run:  CONSULTA (1)
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Alberson/Desktop/BancoPython/CONSULTA.py
CÓDIGO DA DISCIPLINA: 1
Nome da Disciplina: Introdução a Programação de Computadores
```

IMPORTANTE DEMAIS NO PROGRAMA ACIMA !!!

Método ***"fetchall()"*** , usado na linha 10: cria uma tabela temporária com todos os registros filtrados pela **instrução SELECT, ARMAZENANDO-OS EM UMA TUPLA CHAMADA TABELA**

Método ***"rowcount"***, usado na linha 13: retorna quantos registros foram filtrados pela instrução ***"select",*** executada no método ***"execute()"***

Veja o comando ***"for"*** criado na linha 15: percorrerá pelos registros da ***"tabela"*** (TUPLA tabela). Estando posicionado em cada registro, mostrará o conteúdo da coluna 1 da TUPLA tabela.

IMPORTANTE MENCIONAR QUE A VARIÁVEL CHAMADA *"REGISTRO"* TEMOS AS COLUNAS *registro[0]* que armazena o código da disciplina e *registro[1]* que armazena o nome da disciplina. Assim sendo, basta neste exemplo mostrar o nome da disciplina, visto que o usuário já tinha informado por digitação o código da disciplina.

Método ***"close()"*** usados nas linhas 19 e 20 são importantes, estamos limpando o comando usql usado no objeto ***comandosql*** (***comandosql.close()***)e fechando o a conexão com o banco de dados (***conexão.close()***). Ao final de cada operação é importante fechar o banco de dados, principalmente.

37.5 – Alterando dados de um registro numa tabela do Banco de Dados

Para alterar um registro é conveniente mostrá-lo para o usuário previamente na tela. Neste exemplo usarei os comandos ***SELECT*** e o ***UPDATE*** do MySql.

Tenha muito cuidado com o uso do comando *UPDATE*, pois a omissão da cláusula *WHERE*, ou uso incorreto deste critério de filtro de registros, pode trazer consequências graves para registros da tabela.

Exemplo: Neste exemplo vou solicitar ao usuário para digitar um código de disciplina, e em seguida, mostrar o nome da disciplina gravado na tabela ***"disciplinas"***. Logo depois, o programa pedirá ao usuário para informar o novo nome para a disciplina e regravará o registro com novo nome informado na tabela, quando o usuário pressionar <enter>.

É IMPORTANTE FRISAR QUE NÃO SE DEVE ALTERAR CONTEÚDO DE CHAVE PRIMÁRIA, OU SEJA, NESTE CASO NÃO SERÁ ALTERADO O CÓDIGO DA DISCIPLINA.

```
import mysql.connector
try:
    conexao = mysql.connector.Connect(host='localhost',database='univap', user='root', password='mae240299')
    if conexao.is_connected():
        comandosql = conexao.cursor()
        cd = int(input("CÓDIGO DA DISCIPLINA: "))
        comandosql.execute(f'select * from disciplinas where codigodisc = {cd};')
        tabela = comandosql.fetchall()
        if comandosql.rowcount > 0:
            for registro in tabela:
                print(f'Nome da Disciplina: {registro[1]}')
            #Após exibir o nome da disciplina, pede o novo nome da disciplina para alteração
            nn = input('Novo nome da disciplina: ')
            #o MÉTODO EXECUTE está executando a alteração no registro que foi chamado
            comandosql.execute(f'update disciplinas set nomedisc = "{nn}" where codigodisc = {cd};')
            conexao.commit()
            print('Alteração feita com sucesso .............')
except Exception as erro:
      print(f'Ocorreu erro: {erro}')
comandosql.close()
conexao.close()
```

Tivemos como resultado:

```
Run:  ALTERACAO
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Alberson/Desktop/BancoPython/ALTERACAO.py
CÓDIGO DA DISCIPLINA: 11
Nome da Disciplina: MATEMÁTICA
Novo nome da disciplina: HISTÓRIA
Alteração feita com sucesso .............

Process finished with exit code 0
```

IMPORTANTE DEMAIS NO PROGRAMA ACIMA !!!

REPARE QUE O CÓDIGO ACIMA USA BOA PARTE DO PROGRAMA DE CONSULTA, MOSTRADO ANTERIORMENTE. PORTANTO, NÃO FICAREI EXPLICANDO NOVAMENTE OS COMANDOS QUE FORAM USADOS PARA CONSULTA DO REGISTRO.

Entretanto, cabe explicar sobre as seguintes linhas de programação:

NA LINHA 13: Está sendo pedido o novo nome para a disciplina que foi exibida no comando ***for.***

NA LINHA 15: Veja que recriei uma nova instrução sql no objeto ***"comandosql"***, ou seja, usei o ***"update"*** para alterarmos o nome da disciplina cujo código foi o informado pelo usuário.

As duas últimas linhas fecham as conexões abertas. Repare que neste exemplo fechei depois que todo o código foi executado.

37.6 – Excluindo um registro de uma tabela do Banco de Dados

Para exclusão um registro, ou vários, é conveniente mostrá-lo(s) para o usuário previamente na tela. Neste exemplo usarei os comandos ***SELECT*** e o ***DELETE*** do MySql.

Tenha muito cuidado com o uso do comando *DELETE*, pois a omissão da cláusula *WHERE*, ou uso incorreto de critérios de filtros, pode trazer consequências graves para registros gravados na tabela.

Exemplo: Neste exemplo vou solicitar ao usuário para digitar um código de disciplina, e em seguida, mostrar o nome da disciplina gravado na tabela ***"disciplinas"***. Logo depois, o programa pedirá confirmação de exclusão para o usuário. Caso o usuário confirme que realmente quer excluir o registro, a exclusão se concretizará.

```
import mysql.connector
try:
    conexao = mysql.connector.Connect(host='localhost',database='univap', user='root', password='mae240299')
    if conexao.is_connected():
        comandosql = conexao.cursor()
        cd = int(input("CÓDIGO DA DISCIPLINA: "))
        comandosql.execute(f'select * from disciplinas where codigodisc = {cd};')
        tabela = comandosql.fetchall()
        if comandosql.rowcount > 0:
            for registro in tabela:
                print(f'Nome da Disciplina: {registro[1]}')
            #Após exibir o nome da disciplina, PERGUNTA SE O USUÁRIO QUER EXCLUIR A DISCIPLINA EXIBIDA
            if input('Deseja Excluir a disciplina? S - Sim: ') == 'S':
                #o MÉTODO EXECUTE VAI EXCLUIR A DISCIPLINA DA TABELA NO BANCO DE DADOS
                comandosql.execute(f'delete from disciplinas where codigodisc = {cd};')
                conexao.commit()
                print('Exclusão feita com sucesso ..............')
except Exception as erro:
       print(f'Ocorreu erro: {erro}')
comandosql.close()
conexao.close()

```

Tivemos como resultado:

```
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Alberson/Desktop/BancoPython/EXCLUSAO.py
CÓDIGO DA DISCIPLINA: 11
Nome da Disciplina: HISTORIA
Deseja Excluir a disciplina? S - Sim: S
Exclusão feita com sucesso ..............

Process finished with exit code 0
```

IMPORTANTE DEMAIS NO PROGRAMA ACIMA !!!

REPARE QUE O CÓDIGO ACIMA USA BOA PARTE DO PROGRAMA DE CONSULTA, MOSTRADO ANTERIORMENTE. PORTANTO, NÃO NESTE EXEMPLO NÃO VOU EXPLICAR OS COMANDOS QUE FORAM USADOS PARA CONSULTA DO REGISTRO.

Entretanto, cabe explicar sobre as seguintes linhas de programação:

NA LINHA 13: Após mostrar o registro que o usuário quer excluir, o usuário será questionado se CONFIRMA A EXCLUSÃO DO REGISTRO da tabela, ou seja, se realmente quer excluir a disciplina. Se responder ***"S"*** o registro será excluído pela nova instrução Sql criada na linha 15.

OBSERVAÇÃO: QUANDO SOLICITAR A CONFIRMAÇÃO DE EXCLUSÃO DE UM REGISTRO PARA USUÁRIO, TOME CUIDADO DE OBRIGÁ-LO A USAR UMA TECLA DIFERENCIADA, TAL COMO O "S" MAIÚSCULO, POIS QUALQUER FACILIDADE QUE OFEREÇA A ELE, O REGISTRO PODERÁ SER EXCLUÍDO SEM QUERER.

NA LINHA 15: Veja que recriei uma nova instrução sql no objeto ***"comandosql"***, ou seja, usei o ***"DELETE"*** para excluir o registro da disciplina cujo código foi o informado pelo usuário.

As duas últimas linhas fecham as conexões abertas. Repare que neste exemplo fechei depois que todo o código foi executado.

37.7 – Consultando vários registros de uma tabela no Banco de Dados

Neste item da apostila vou criar uma grade na tela para exibir todos os dados de disciplinas gravadas para o usuário. Este exemplo será muito útil para nossos exercícios e trabalhos futuros.

Com este exemplo poderão consultar os dados, ou registros que desejarem de qualquer tabela existente num banco de dados. Basta para isso, definir na instrução ***SELECT*** o filtro necessário para consulta, podendo utilizar logicamente a cláusula ***where*** quando necessário. Para o exemplo criado, não usarei critério de filtro (critérios de seleção) de registros.

Exemplo:

```
import mysql.connector
from prettytable import PrettyTable
#criando duas colunas para o grid que exibirá todas as diciplinas cadastradas
grid = PrettyTable(['Códigos das Disciplinas', "Nomes de Disciplinas"])
try:
    conexao = mysql.connector.Connect(host='localhost',database='univap', user='root', password='mae240299')
    if conexao.is_connected():
        comandosql = conexao.cursor()
        #repare que NÃO USEI A CLÁUSULA WHERE, ou seja, todas as disciplinas gravadas serão consultadas
        comandosql.execute(f'select * from disciplinas;')
        #O MÉTODO fetchall() retornará todos os registros filtrados (um ou mais registros) pelo comando select
        tabela = comandosql.fetchall()
        #O método rowcount conta quantos registros foram filtrados, caso tenha registro filtrado entra no if
        if comandosql.rowcount > 0:
            # se existir pelo menos uma disciplina na tabela temporária, mostre-as no grid
            for registro in tabela:
                #criando as linhas do grid com os registros lidos da tabela temporária. Mostrando todas as disciplinas
                grid.add_row([registro[0],registro[1]])
            print(grid)
except Exception as erro:
      print(f'Ocorreu erro: {erro}')
comandosql.close()
conexao.close()
```

Tivemos como resultado:

```
Run:  CONSULTATODASDISCIPLINAS
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Alberson/Desktop/BancoPython/CONSULTATODASDISCIPLINAS.py
+-------------------------+-----------------------------------------+
| Códigos das Disciplinas |           Nomes de Disciplinas          |
+-------------------------+-----------------------------------------+
|            1            | Introdução a Programação de Computadores |
|            2            |          Linguagem Estruturada          |
|            3            |         Sistemas Computacionais         |
|            4            |         Desenvolvimento de Jogos        |
|            5            |          Redes de Computadores          |
|            8            |            COMPUTAÇÃO GRÁFICA           |
|            10           |            computação gráfica           |
+-------------------------+-----------------------------------------+

Process finished with exit code 0
```

IMPORTANTE DEMAIS NO PROGRAMA ACIMA !!!

REPARE QUE O CÓDIGO ACIMA USA BOA PARTE DO PROGRAMA DE CONSULTA, MOSTRADO ANTERIORMENTE. PORTANTO, NÃO NESTE EXEMPLO NÃO VOU EXPLICAR OS COMANDOS QUE FORAM USADOS PARA CONSULTA DO REGISTRO.

Entretanto, cabe explicar sobre as seguintes linhas de programação:

Na linha 2: Instalei a biblioteca ***PrettyTable*** e a importei para este programa de exemplo. Esta biblioteca foi a responsável para criar o grid de exibição das disciplinas cadastradas na tela para o usuário, conforme mostrado no resultado da execução.

Na linha 4: Iniciei a construção da grade de consulta, definindo os títulos de cada coluna que serão trazidas da tabela ***disciplinas***, no banco de dados ***univap***.

NA LINHA 18: Veja que estamos adicionando cada linha da tabela. Neste caso são os conteúdos das colunas 0 e 1 da tabela temporária REGISTRO. Quando o laço de repetição for finalizado, mando imprimir o grid, exibindo com isto todos os códigos e nomes de disciplinas cadastrados.

37.8 – PROGRAMA COMPLETO DE MANIPULAÇÃO DE DADOS DE DISCIPLINAS (CADASTRA, ALTERA, EXCLUI E CONSULTA REGISTROS)

Diante do estudo realizado das operações de manuseio de registros isoladamente, podemos então construir um único programa completo para possibilitar o usuário Cadastrar, Alterar, Excluir e Consultar registros de disciplinas cadastradas.

Exemplo: Neste exemplo uni todos os programas feitos isoladamente neste único exemplo a seguir

```
from prettytable import PrettyTable
import mysql.connector

def abrebanco():
    try:
        #criando um objeto para conexão ao banco de dados
        #Devemos indicar em :
        # host = Servidor ou IP(caso esteja numa rede),
        # database = nome do banco de dados criado no workbenck
        # user = nome do usuário definido no momento de criação do banco de dados
        # password = senha de acesso do banco de dados
        global conexao
        conexao = mysql.connector.Connect(host='localhost',database='univap',
user='root', password='mae240299')
        # testando se estamos conectado ao banco de dados
        if conexao.is_connected():
            informacaobanco = conexao.get_server_info()
            print(f'Conectado ao servidor banco de dados - Versão {informacaobanco}')
            print('Conexão ok')
            # criando objeto cursor, responsável para trabalharmos com registros
retornados pela tabela fisica
            global comandosql
            comandosql = conexao.cursor()
            # Criando uma QUERY para mostrar as informações do banco de dados ao qual nos
conectamos
            comandosql.execute('select database();')

            # usando método fetchone para buscar um dado do banco de dados e armazenálo
na variável nomebanco
            nomebanco = comandosql.fetchone()
            print(f'Banco de dados acessado = {nomebanco}')
            print('='*80)
            return 1
        else:
            print('Conexão não realizada com banco')
            return 0
    except Exception as erro:
        print(f'Erro : {erro}')
        return 0

def mostratodas():
    # criando duas colunas para o grid que exibirá todas as diciplinas cadastradas
    grid = PrettyTable(['Códigos das Disciplinas', "Nomes de Disciplinas"])
    try:
        comandosql = conexao.cursor()
        # repare que NÃO USEI A CLÁUSULA WHERE, ou seja, todas as disciplinas gravadas
serão consultadas
        comandosql.execute(f'select * from disciplinas;')
        # O MÉTODO fetchall() retornará todos os registros filtrados (um ou mais
registros) pelo comando select
        tabela = comandosql.fetchall()
        # O método rowcount conta quantos registros foram filtrados, caso tenha registro
filtrado entra no if
        if comandosql.rowcount > 0:
```

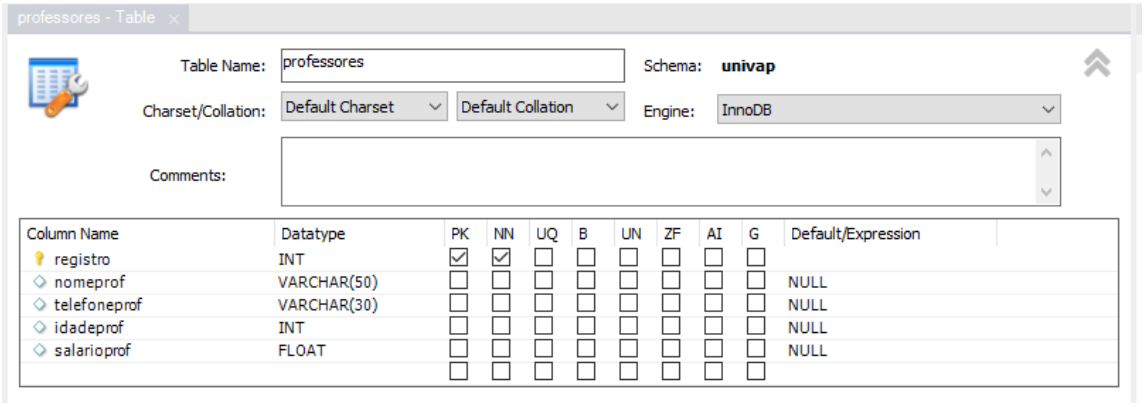
```
            # se existir pelo menos uma disciplina na tabela temporária, mostre-as no
grid
            for registro in tabela:
                # criando as linhas do grid com os registros lidos da tabela temporária.
Mostrando todas as disciplinas
                grid.add_row([registro[0], registro[1]])
            print(grid)
        else:
            print('Não existem disciplinas cadastradas!!!')
    except Exception as erro:
        print(f'Ocorreu erro: {erro}')

def consultardisciplina(cd=0):
    try:
        comandosql = conexao.cursor()
        comandosql.execute(f'select * from disciplinas where codigodisc = {cd};')
        tabela = comandosql.fetchall()
        # verivificando quanto registros de disciplinas de código igual ao digitado
        # filtrados pelo select foram guardados na tabela temporária
        if comandosql.rowcount > 0:
            # se existir pelo menos uma disciplina na tabela temporária, mostre os dados
da coluna 1
            for registro in tabela:
                print(f'Nome da Disciplina: {registro[1]}')
            return 'c'
        else:
            return 'nc'
    except Exception as error:
        return (f'Ocorreu erro ao tentar consultar esta disciplina: Erro===>>> {error}')


def cadastrardisciplina(cd=0,nd=''):
    try:
        comandosql = conexao.cursor()
        #criando comando insert e concatenando os dados a serem gravados, recebimdos em
cd e nd
        comandosql.execute(f'insert into disciplinas(codigodisc, nomedisc)
values({cd},"{nd}") ;')
        #método commit é responsável por gravar de fato o novo registro de disciplina na
tabela
        conexao.commit()
        return 'Cadastro da disciplina realizado com sucesso !!!! '
    except Exception as erro :
        print(f'Erro: {erro}')
        return 'Não foi possível cadastrar esta disciplina !!!'

def alterardisciplina(cd=0, nomedisciplina=''):
    try:
        comandosql = conexao.cursor()
        #criando comando update para atulizar o nome da disciplina em questão
        comandosql.execute(f'Update disciplinas SET nomedisc="{nomedisciplina}" where
codigodisc = {cd};')
        #método commit é responsável por REGRAVAR de fato o novo NOME DA DISCIPLINA
disciplina na tabela
        conexao.commit()
        return 'Disciplina alterada com sucesso !!! '
    except Exception as erro :
        print(f'Erro: {erro}')
        return 'Não foi possível alterada esta disciplina'

def excluirdisciplina(cd=0):
    try:
        comandosql = conexao.cursor()
```

```
        #criando comando delete e concatenando o código da disciplina para ser escluída
        comandosql.execute(f'delete from disciplinas where codigodisc = {cd};')
        #método commit é responsável por gravar de fato o novo registro de disciplina na
tabela
        conexao.commit()
        return 'Disciplina excluída com sucesso !!! '
    except Exception as erro :
        print(f'Erro: {erro}')
        return 'Não foi possível excluir esta disciplina'

'''
======================================== MÓDULO PRINCIPAL DO PROGRAMA
=============================================
'''
if abrebanco() == 1:
    resp = input('Deseja entrar no módulo de Disciplinas? (1-Sim, ou qualquer tecla para
sair) ==> ')
    while resp =='1':
        print('='*80)
        print('{:^80}'.format('SISTEMA UNIVAP - DISCIPLINAS'))
        print('='*80)

        while True:
            codigodisc = input('Código da Disciplina: (0- Mostra Todas) ')
            if codigodisc.isnumeric():
                codigodisc  = int(codigodisc)
                break
        if codigodisc == 0:
            mostratodas()
            #o comando continue volta a executar o laço de repetição do início do laço de
repetição
            continue

        #chamando função para consultardisciplina, se retornar 'nc' não está cadastrada
        if consultardisciplina(codigodisc) == 'nc':
            nomedisciplina = input('Nome da Disciplina: ')
            msg = cadastrardisciplina(codigodisc, nomedisciplina)
            print(msg)
        else:
            op = input("Escolha: [A]-Alterar  [E]-Excluir [C]-Cancelar Operações ==> ")
            while op!='A' and op!='E' and op!='C':
                op = input("ERRO !!! Escolha CORRETAMENTE : [A]-Alterar [E]-Excluir [C]-
Cancelar Operações ==> ")

            if op=='A':
                print('Atenção: Código da disciplina não pode ser alterado: ')
                nomedisciplina = input('Informe novo nome da disciplina: ')
                msg = alterardisciplina(codigodisc, nomedisciplina)
                print(msg)
            elif op == 'E':
                confirma = input('ATENÇÃO !!!! TEM CERTEZA, CONFIRMA EXCLUSÃO? S-SIM OU
N-NÃO:  ')
                while confirma != 'S' and confirma != 'N':
                    confirma = input('RESPOSTA INEXISTENTE !!!! TEM CERTEZA, CONFIRMA
EXCLUSÃO? S-SIM OU N-NÃO: ')
                msg = excluirdisciplina(codigodisc)
                print (msg)

        print('\n\n')
        print('='*80)
        if input('Deseja continuar usando o programa? 1- Sim OU qualquer tecla para sair:
') == '1':
            #o COMANDO CONTIUE ABAIXO, RETORNA PARA O WHILE QUE ESTÁ SENDO EXECUTADO
```

```
            continue
        else:
            break
            comandosql.close()
            conexao.close()
else:
    print('FIM DO PROGRAMA!!! Algum problema existente na conexão com banco de dados.')
```

37.9– DESAFIOS DE BANCO DE DADOS

1) Criar um programa completo em Python para cadastro, alteração, exclusão e consulta a registros da tabela professores. Lembrem-se de criar a tabela no mesmo banco de dados "univap", usado para os exemplos desta apostila:



O princípio de funcionamento desta tela deve ser o mesmo do programa completo apresentado para tela de Dados de Disciplinas, desenvolvida e apresentada em aulas.

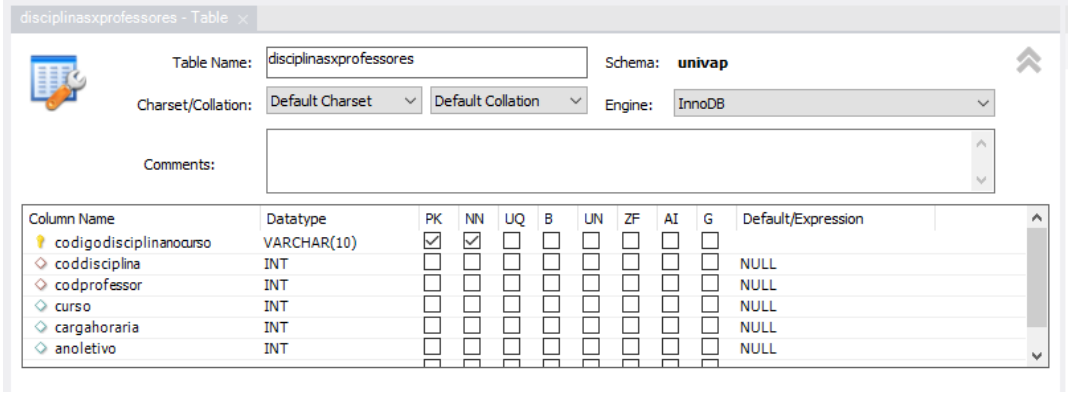
Este programa deve prever testes de consistências gerais, ou seja, exigir do usuário digitação correta do nome, telefone, idade e salário do professor.

Seu programa deve tratar os erros comuns, voltados principalmente a integridade dos dados, ou seja, só podemos alterar ou excluir algum professor caso não existam registros de disciplinas ligadas a ele na tabela ***"disciplinasxprofessores"***

Também cuide de tratar os erros comuns de conexão com banco de dados, exemplificados em aula.

Podem melhorar a aparência das telas de acordo que tenha boa aparência para o usuário final

2) Criar um programa Python que cadastre, altere, exclua e consulte registros na tabela ***"disciplinasxprofessores"***. Lembrem-se de criar a tabela para receber os registros dentro do banco de dados ***"univap"***. Esta tabela deve ter a seguinte estrutura e deve estar relacionada com as tabelas ***"disciplinas" e "professores",*** criadas até então.



Este programa deve prever os testes de possiblidades de erros comuns ao acessarmos o banco de dados.

Também deve ser previsto teste de integridade de dados, ou seja, ao tentarmos incluir um professor, ou disciplina que não tenham sido previamente cadastrados, o programa deve informar ao usuário que não foi possível realizar o processo.

Crie possiblidades de consultas para o usuário visualizar quais as disciplinas e professores foram previamente cadastrados e que podem ser usados na **relação de cadastro e alteração nesta tela**.

Logicamente preveja todos os tipos de erros comuns e trate-os nesta tela.