

3º BIMESTRE – PROGRAMAÇÃO ORIENTADA A OBJETOS - PYTHON

Sumário	
29 – CRIAÇÃO DE ROTINAS DE PROGRAMAS – def ____()	114
29.1 – CRIAÇÃO E USO DE ROTINAS SEM PASSAGEM DE PARÂMETROS E SEM RETORNO DE VALORES – (PROCEDIMENTOS)	114
29.2 – CRIAÇÃO E USO DE ROTINAS COM PASSAGEM DE PARÂMETROS SEM RETORNO DE VALORES – (PROCEDIMENTOS)	116
29.2.1 – CRIAÇÃO E USO DE ROTINAS COM PASSAGEM DE PARÂMETROS DE DADOS EMPACOTADOS SEM RETORNO DE VALORES- (PROCEDIMENTOS)	119
29.2.2 – CRIAÇÃO E USO DE ROTINAS COM PASSAGEM DE PARÂMETROS DE DADOS EM FORMAS DE LISTAS SEM RETORNO DE VALORES – (PROCEDIMENTOS)	119
29.3 – DESAFIOS DE ROTINAS – (PROCEDIMENTOS)	121
30 – ROTINA COM DEFINIÇÃO DE PARÂMETROS OPCIONAIS	122
31 – VARIÁVEIS LOCAIS E GLOBAIS NO PYTHON – ESCOPO DE VARIÁVEIS	123
31.1 – USANDO O COMANDO global DENTRO DE ROTINAS:	125
32 – ROTINA COM RETORNO DE VALORES COMANDO return- FUNÇÕES	126
33 – DOCSTRINGS – GERANDO DOCUMENTAÇÃO DE ROTINAS (procedimentos e funções) PARA USO NO COMANDO help()	129
34 – DESAFIOS SOBRE ROTINAS – (FUNÇÕES)	131
35 – CRIANDO MODULOS DE PROGRAMAS	132
35.1 – DESAFIOS DE MÓDULOS	134
36 – TRATAMENTOS DE ERROS E EXCEÇÕES - (try)	135
36.1 – EXCEÇÕES	135
36.2 – TRATANDO EXCEÇÕES COM COMANDO try:	139
36.3 – DESAFIOS DE TRATAMENTOS DE ERROS (try...)	142
37 – BANCO DE DADOS MYSQL x PYTHON	143
37.1 – Instalando a biblioteca mysql-connector-python com pip	147
37.2 – Abrindo banco de dados num programa Python	148
37.3 – Cadastrando registros numa tabela do Banco de Dados	151
37.4 – Consultado por um registro numa tabela do Banco de Dados	152
37.5 – Alterando dados de um registro numa tabela do Banco de Dados	153
37.6 – Excluindo um registro de uma tabela do Banco de Dados	155
37.7 – Consultando vários registros de uma tabela no Banco de Dados	157
37.8 – PROGRAMA COMPLETO DE MANIPULAÇÃO DE DADOS DE DISCIPLINAS (CADASTRA, ALTERA, EXCLUI E CONSULTA REGISTROS)	159
37.9 – DESAFIOS DE BANCO DE DADOS	162

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.

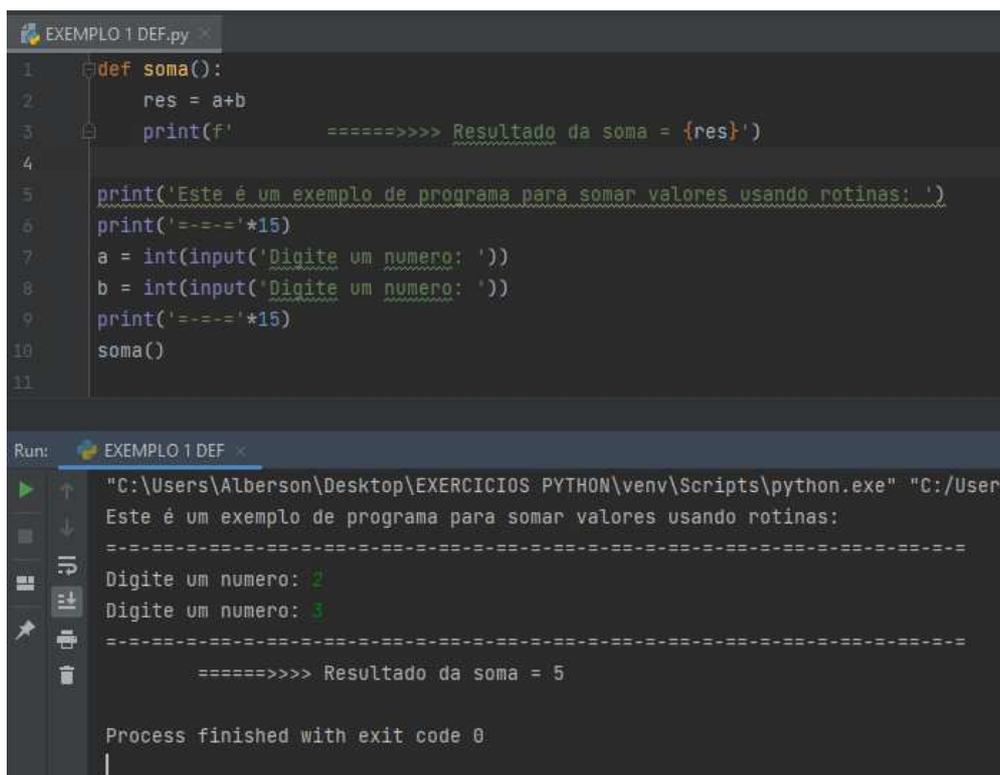
29.1 – CRIAÇÃO E USO DE ROTINAS SEM PASSAGEM DE PARÂMETROS E SEM RETORNO DE VALORES – (PROCEDIMENTOS)

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

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

Vamos aos exemplos:

Exemplo 1:



```
EXEMPLO 1 DEF.py
1 def soma():
2     res = a+b
3     print(f'      =====>>> Resultado da soma = {res}')
4
5     print('Este é um exemplo de programa para somar valores usando rotinas: ')
6     print('-----'*15)
7     a = int(input('Digite um numero: '))
8     b = int(input('Digite um numero: '))
9     print('-----'*15)
10    soma()
11

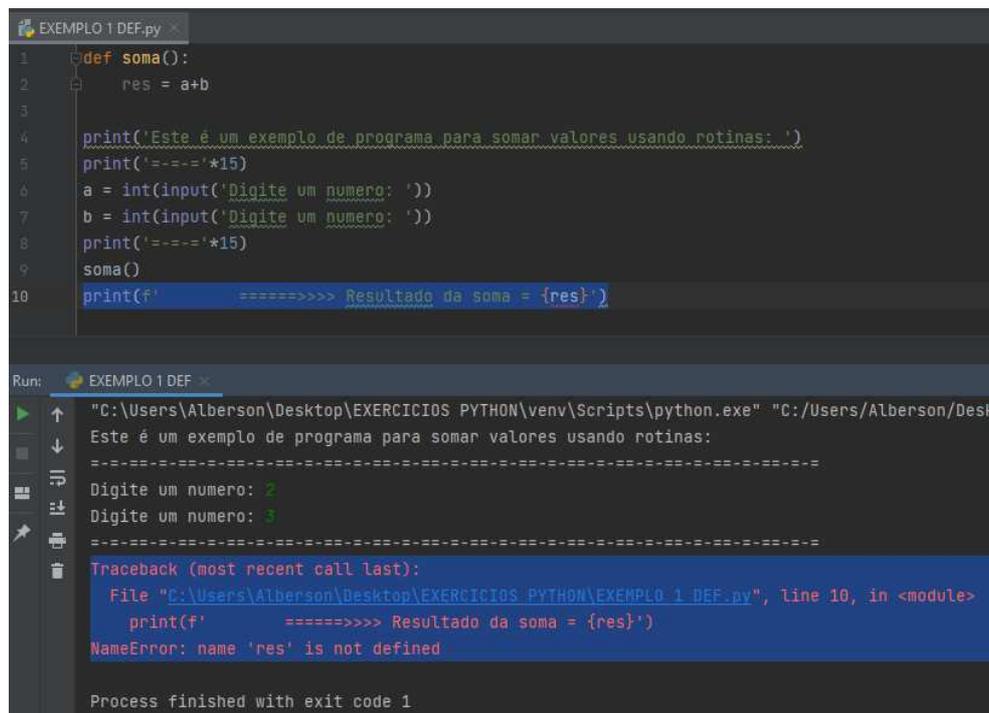
Run: EXEMPLO 1 DEF
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users
Este é um exemplo de programa para somar valores usando rotinas:
-----
Digite um numero: 2
Digite um numero: 3
-----
      =====>>> Resultado da soma = 5

Process finished with exit code 0
```

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:



```
EXEMPLO 1 DEF.py
1 def soma():
2     res = a+b
3
4     print('Este é um exemplo de programa para somar valores usando rotinas: ')
5     print('===='*15)
6     a = int(input('Digite um numero: '))
7     b = int(input('Digite um numero: '))
8     print('===='*15)
9     soma()
10    print(f'====>>> Resultado da soma = {res}')
```

Run: EXEMPLO 1 DEF

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desk
Este é um exemplo de programa para somar valores usando rotinas:
====
Digite um numero: 2
Digite um numero: 3
====
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\EXEMPLO 1 DEF.py", line 10, in <module>
    print(f'====>>> Resultado da soma = {res}')
```

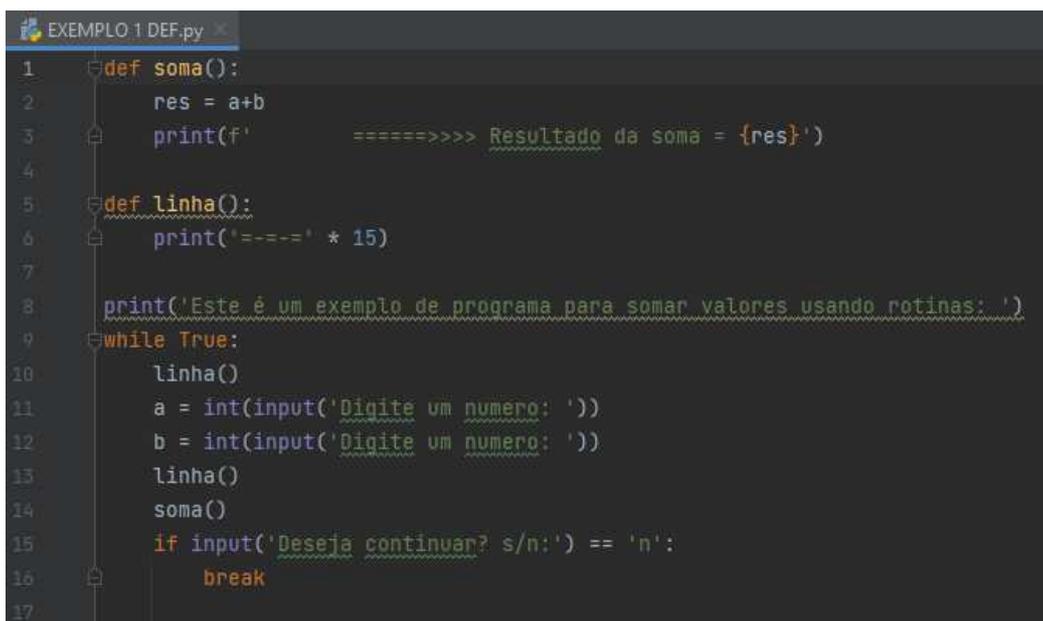
NameError: name 'res' is not defined

Process finished with exit code 1

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:



```
EXEMPLO 1 DEF.py
1 def soma():
2     res = a+b
3     print(f'====>>> Resultado da soma = {res}')
```

```
4
5 def linha():
6     print('====' * 15)
7
8 print('Este é um exemplo de programa para somar valores usando rotinas: ')
9 while True:
10    linha()
11    a = int(input('Digite um numero: '))
12    b = int(input('Digite um numero: '))
13    linha()
14    soma()
15    if input('Deseja continuar? s/n:') == 'n':
16        break
17
```

```
EXEMPLO 1 DEF x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/
Este é um exemplo de programa para somar valores usando rotinas:
-----
Digite um numero: 3
Digite um numero: 2
-----
=====>>> Resultado da soma = 3
Deseja continuar? s/n:s
-----
Digite um numero: 3
Digite um numero: 4
-----
=====>>> Resultado da soma = 7
Deseja continuar? s/n:n
Process finished with exit code 0
```

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.

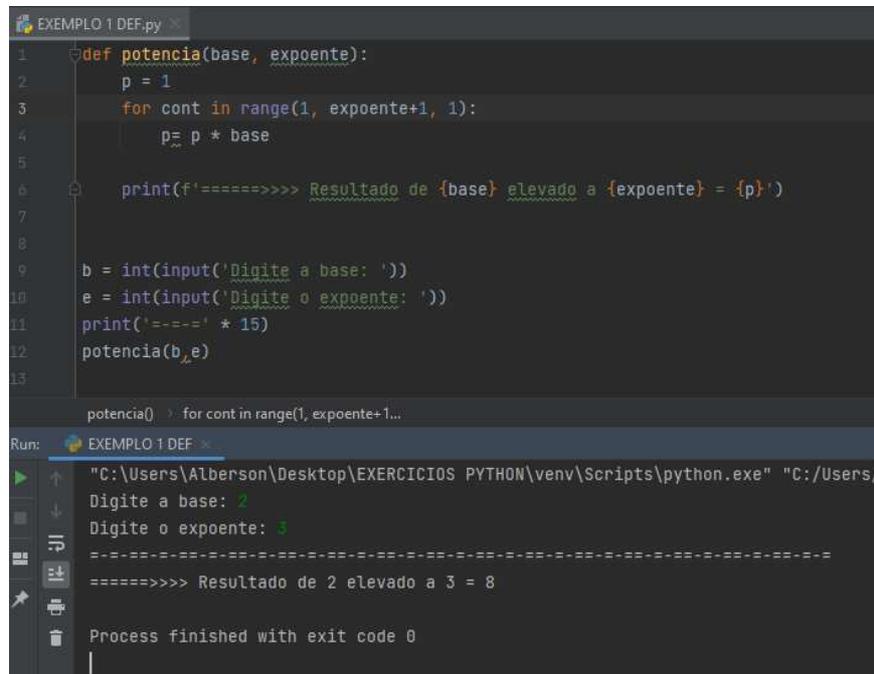
def <nome_rotina>(<var1>, <var2>,.....,<varN>):
<comandos da rotina>

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:

Exemplo 1



```
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
6     print(f'====>>> Resultado de {base} elevado a {expoente} = {p}')
7
8
9     b = int(input('Digite a base: '))
10    e = int(input('Digite o expoente: '))
11    print('---' * 15)
12    potencia(b,e)
13

potencia()  for cont in range(1, expoente+1...

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

Process finished with exit code 0
```

Repare:

- Foi criada a rotina chamada **potencia(base, expoente)** com variáveis **base** e **expoente** preparadas para receber dados de parâmetros;
- Repare no módulo principal, que os dados informados pelo usuário são armazenados nas variáveis **b** e **e**
- Ao chamar a rotina os valores armazenados em **b** e **e** são passados automaticamente e respectivamente para **base** e **expoente**.
- Veja que as variáveis **base** e **expoente** foram usadas no interior da rotina **potencia**.
- 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.
- Neste exemplo estamos calculando $2^3 = 2*2*2 = 8$

Exemplo 2:

```
EXEMPLO 1 DEF.py x
1 def potencia(base, expoente):
2     p = 1
3     for cont in range(1, expoente+1, 1):
4         p = p * base
5
6     print(f'====>>> Resultado de {base} elevado a {expoente} = {p}')
7
8
9     b = int(input('Digite a base: '))
10    e = int(input('Digite o expoente: '))
11    print('==== * 15')
12    #Na chamada abaixo indicamos os nomes das variáveis que vão receber
13    #os valores de e e b na área de parâmetros. ISSO NÃO É USUAL FAZER !!!
14    potencia(expoente=e, base=b)
15

Run: EXEMPLO 1 DEF x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberse
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 x
1 def potencia(base, expoente):
2     p = 1
3     for cont in range(1, expoente+1, 1):
4         p = p * base
5
6     print(f'====>>> Resultado de {base} elevado a {expoente} = {p}')
7
8
9     b = int(input('Digite a base: '))
10    e = int(input('Digite o expoente: '))
11    print('==== * 15')
12    #repare no erro abaixo:
13    #Não podemos atribuir valor para expoente = e, sendo que b também está sendo
14    #passado como parâmetro para expoente, VEJA ERRO ABAIXO
15    potencia(expoente=e, b)

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

MOTIVO DESTES 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:

```
EXEMPLO 1 DEF.py
1 def teste(*dados):
2     #prepare que os números recebidos serão guardados em numeros
3     #no formato de TUPLA. OBSERVE OS RESULTADOS IMPRESSOS
4     print(f'A rotina recebeu os seguintes dados {dados} ')
5     print(f'-----A tupla acima tem {len(dados)} dados')
6
7
8     #Deixe sempre duas linhas após a ultima linha da rotina
9     teste('Wagner', 2, 3, 'alberson', 5)
10    teste(10, 20.5, 30)
11    teste(50)
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2
```

Vamos aos exemplos!!!!

Exemplo 1:

```
EXEMPLO 1 DEF.py x
1 def teste(lista):
2     print(f'tamanho da lista = {len(lista)}')
3
4
5     #Criamos abaixo uma lista chamada valores
6     valores = [10,20,30,40]
7     #chamando a rotina teste e passando a lista valores inteira como parâmetro
8     teste(valores)
9     #Criando a lista chamada val
10    val = [1,3,5,7,1,7,9]
11    #chamando a rotina teste e passando a lista val inteira como parâmetro
12    teste(val)
13    #criando a lista chamada dados
14    dados=['albersson', 45, 'joao', 40, 'zé', 25]
15    #chamando a rotina teste e passando a lista dados inteira como parâmetro
16    teste(dados)

Run: EXEMPLO 1 DEF x
"C:\Users\Albersson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Us
tamanho da lista = 4
tamanho da lista = 7
tamanho da lista = 6
```

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:

```
EXEMPLO 1 DEF.py x
1 def triplo(lista):
2     for x in range(0, len(lista), 1):
3         #a linha abaixo funciona como uma passagem de parâmetro
4         #por referência, pois o que for alterado em lista, também
5         #será alterado automaticamente em valores
6         lista[x] *= 3
7
8
9     #Criamos abaixo uma lista chamada valores
10    valores = [10,20,30,40]
11    #chamando a rotina triplo e passando a lista valores inteira como parâmetro
12    triplo(valores)
13    #repare que os valores foram alterados na rotina triplo
14    print(valores)

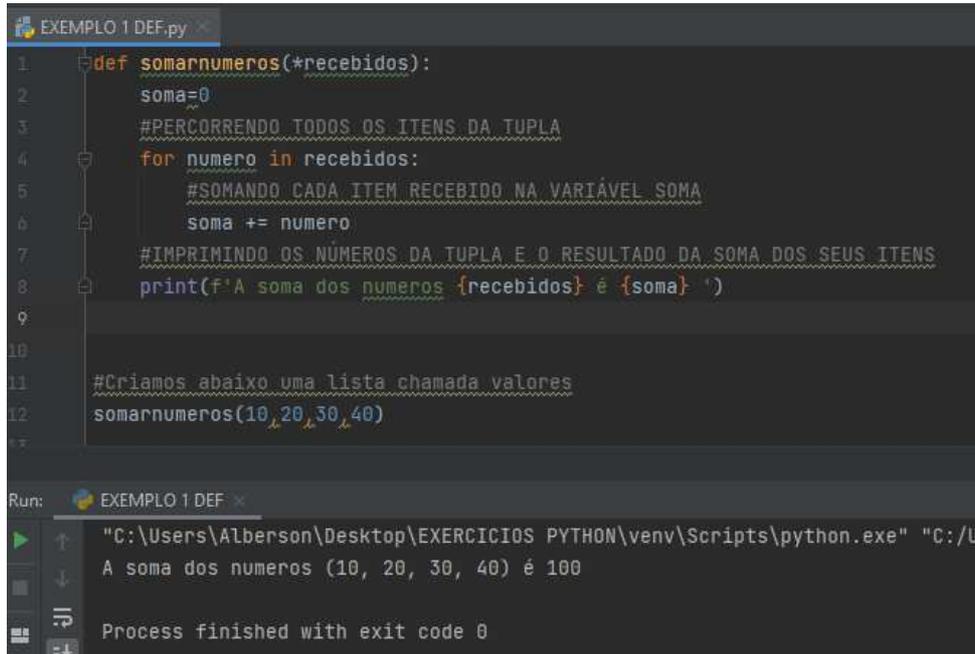
triplo()  for x in range(0, len(lista), 1)

Run: EXEMPLO 1 DEF x
"C:\Users\Albersson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Us
[30, 60, 90, 120]
```

IMPORTANTE !!!!

Em PYTHON TODA PASSAGEM DE PARÂMETRO É POR REFERÊNCIA

Exemplo 3:



```
EXEMPLO 1 DEF.py
1 def somarNumeros(*recebidos):
2     soma=0
3     #PERCORRENDO TODOS OS ITENS DA TUPLA
4     for numero in recebidos:
5         #SOMANDO CADA ITEM RECEBIDO NA VARIÁVEL SOMA
6         soma += numero
7     #IMPRIMINDO OS NÚMEROS DA TUPLA E O RESULTADO DA SOMA DOS SEUS ITENS
8     print(f'A soma dos numeros {recebidos} é {soma} ')
9
10
11 #Criamos abaixo uma lista chamada valores
12 somarNumeros(10, 20, 30, 40)

Run: EXEMPLO 1 DEF
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/U
A soma dos numeros (10, 20, 30, 40) é 100
Process finished with exit code 0
```

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 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 deve 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:



```
exemplos apostilas.py
def soma(a,b):
    res = a + b

soma(1)

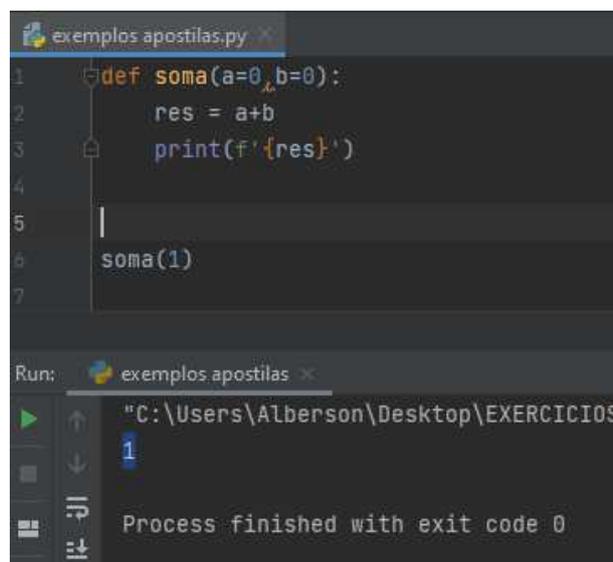
Run: exemplos apostilas
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/exemplos apostilas.py"
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\exemplos apostilas.py", line 4, in <module>
    soma(1)
TypeError: soma() missing 1 required positional argument: 'b'

Process finished with exit code 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:



```
exemplos apostilas.py
1 def soma(a=0,b=0):
2     res = a+b
3     print(f'{res}')
4
5 soma(1)
6
7

Run: exemplos apostilas
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/exemplos apostilas.py"
1
Process finished with exit code 0
```

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:

```
exemplos apostilas.py
1 def soma(a=0, b=0):
2     res = a+b
3     print(f'{res}')
4
5 soma(b=1)
6
7 soma()
```

Run: exemplos apostilas x

```
"C:\Users\Alberson\Desktop\EXERCICIOS
1
Process finished with exit code 0
```

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:

```
exemplos apostilas.py
1 def dado(a):
2     # a variável 'a' é local
3     print(f'a dentro da rotina = {a}')
4     print(f'x dentro da rotina = {x}')
5
6
7 #a variável x, criada abaixo, é GLOBAL
8 x=10
9 dado(x)
10 print(f'x fora da rotina = {x}')
11 print(f'a fora da rotina = {a}')
```

Run: exemplos apostilas x

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\pyth
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\exemplos a
    print(f'a fora da rotina = {a}')
NameError: name 'a' is not defined
a dentro da rotina = 10
x dentro da rotina = 10
x fora da rotina = 10
Process finished with exit code 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 **dados** 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:

```
exemplos apostilas.py x
1 def dados(a):
2     #A VARIÁVEL X ABAIXO É LOCAL
3     x=35
4     # a variável 'a' é local
5     print(f'a dentro da rotina = {a}')
6     # NESTE CASO SERÁ IMPRESSO x LOCAL
7     print(f'x dentro da rotina = {x}')
8
9
10 # a variável x, criada abaixo, é GLOBAL
11 x=10
12 dados(x)
13 # NESTE CASO SERÁ IMPRESSO VALOR DE x GLOBAL
14 print(f'x fora da rotina = {x}')
15
dados()
```

Run: exemplos apostilas x

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv
a dentro da rotina = 10
x dentro da rotina = 35
x fora da rotina = 10

Process finished with exit code 0
```

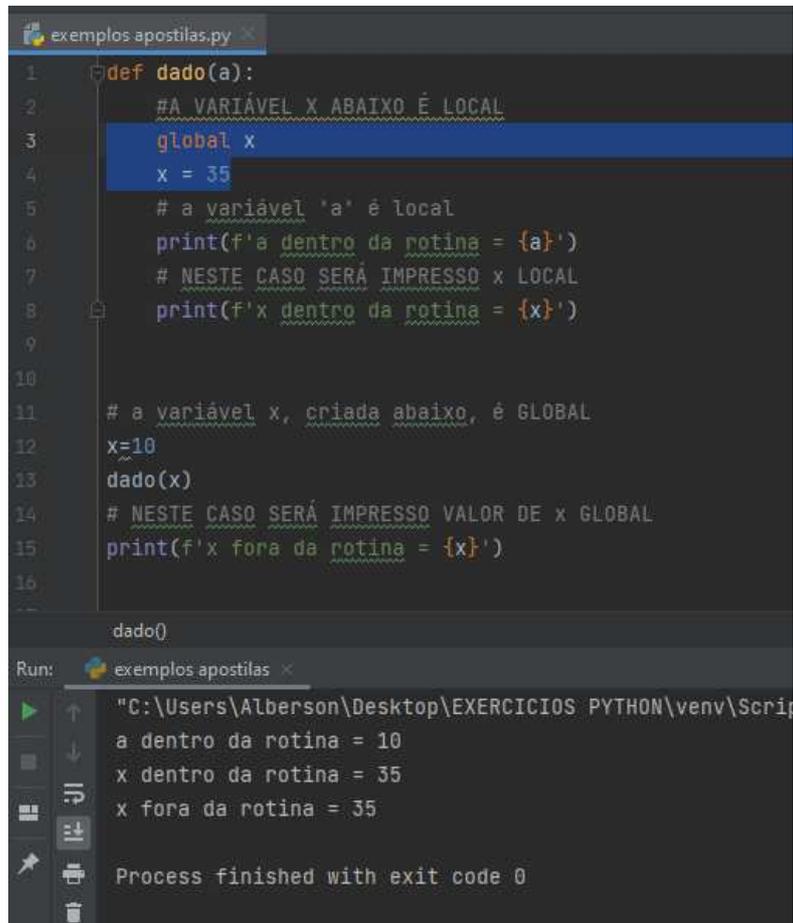
IMPORTANTE NESTE EXEMPLO:

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:



```
exemplos apostilas.py
1 def dado(a):
2     #A VARIÁVEL X ABAIXO É LOCAL
3     global x
4     x = 35
5     # a variável 'a' é local
6     print(f'a dentro da rotina = {a}')
7     # NESTE CASO SERÁ IMPRESSO x LOCAL
8     print(f'x dentro da rotina = {x}')
9
10
11 # a variável x, criada abaixo, é GLOBAL
12 x=10
13 dado(x)
14 # NESTE CASO SERÁ IMPRESSO VALOR DE x GLOBAL
15 print(f'x fora da rotina = {x}')
16
17
18 dado()

Run: exemplos apostilas
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe"
a dentro da rotina = 10
x dentro da rotina = 35
x fora da rotina = 35
Process finished with exit code 0
```

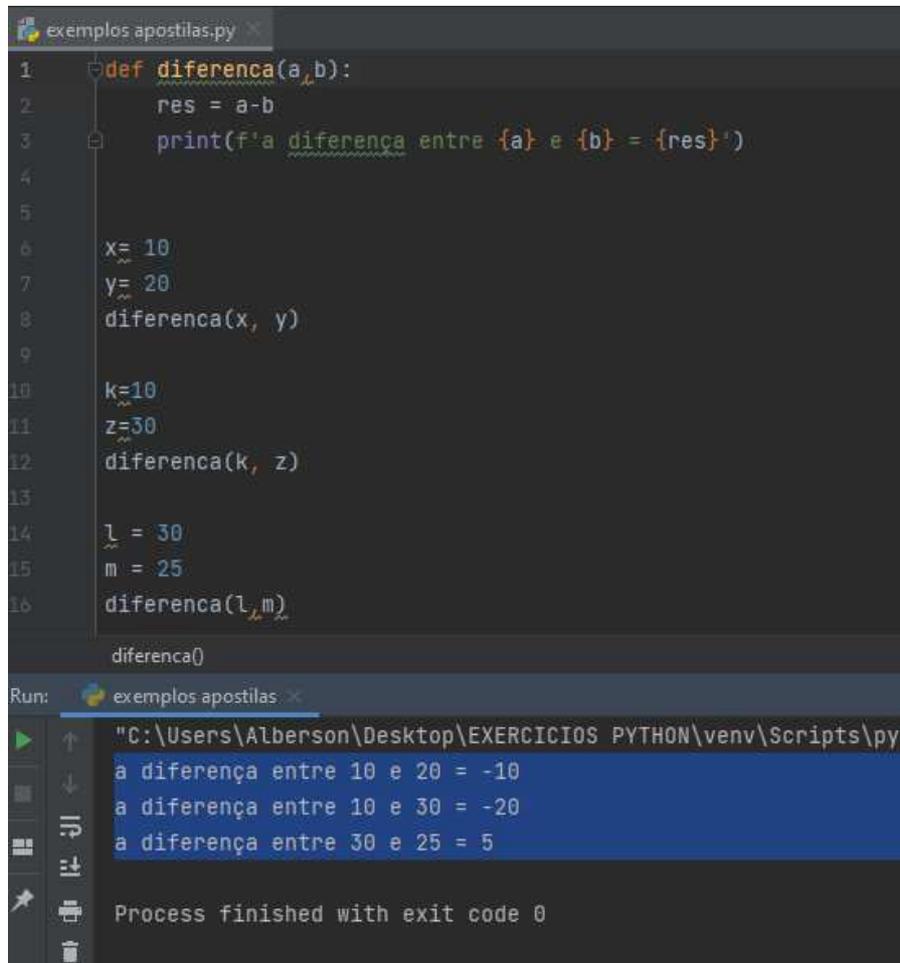
Observe IMPORTANTE!

NESTE CASO, ESTOU USANDO A VARIÁVEL 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:



```
exemplos apostilas.py
1 def diferenca(a,b):
2     res = a-b
3     print(f'a diferença entre {a} e {b} = {res}')
4
5
6     x= 10
7     y= 20
8     diferenca(x, y)
9
10    k=10
11    z=30
12    diferenca(k, z)
13
14    l = 30
15    m = 25
16    diferenca(l,m)
17
18    diferenca()
```

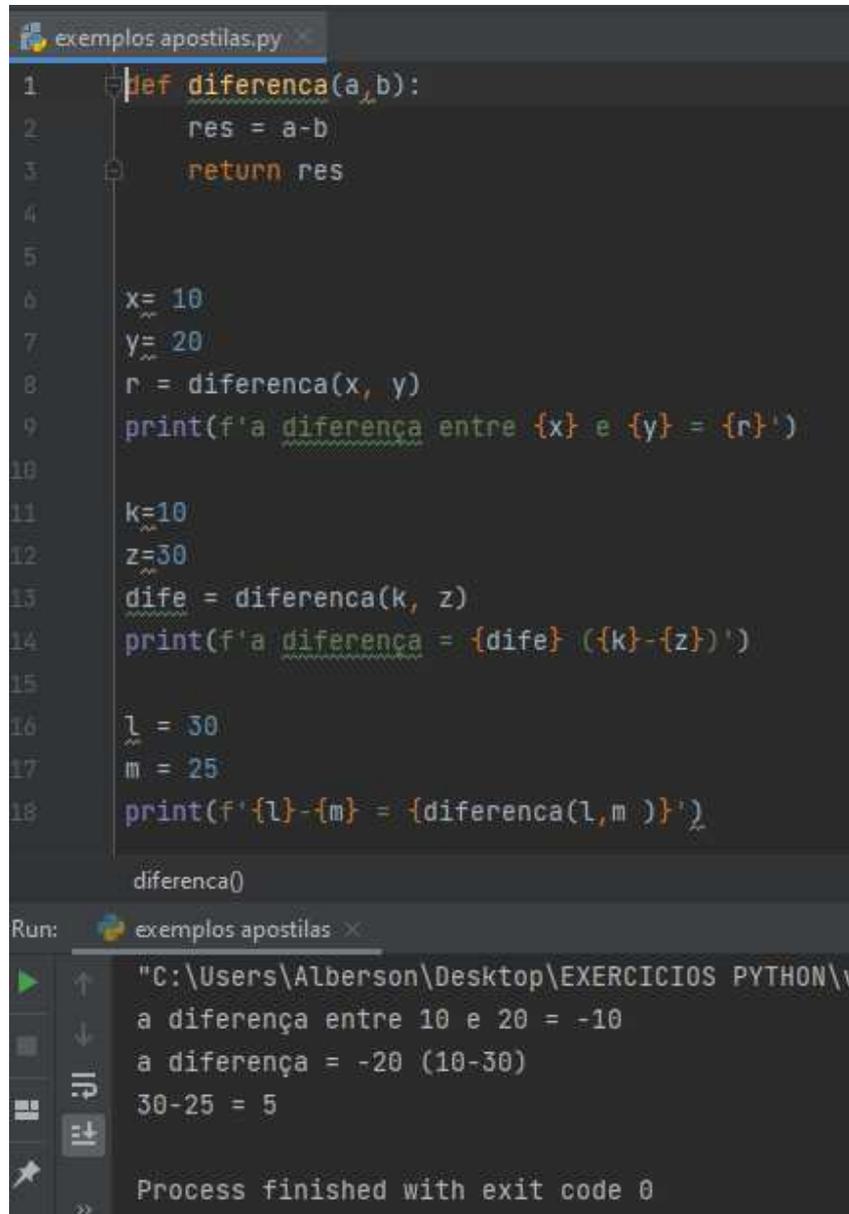
Run: exemplos apostilas ×

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\py
a diferença entre 10 e 20 = -10
a diferença entre 10 e 30 = -20
a diferença entre 30 e 25 = 5
Process finished with exit code 0
```

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:



```
exemplos apostilas.py
1 def diferenca(a,b):
2     res = a-b
3     return res
4
5
6 x= 10
7 y= 20
8 r = diferenca(x, y)
9 print(f'a diferenca entre {x} e {y} = {r}')
10
11 k=10
12 z=30
13 dife = diferenca(k, z)
14 print(f'a diferenca = {dife} ({k}-{z})')
15
16 l = 30
17 m = 25
18 print(f'{l}-{m} = {diferenca(l,m )}')

diferenca()

Run: exemplos apostilas
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\
a diferenca entre 10 e 20 = -10
a diferenca = -20 (10-30)
30-25 = 5
Process finished with exit code 0
```

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:

```
exemplos apostilas.py
1 def diferenca(a,b):
2     res = a-b
3     return res
4
5
6     x= 10
7     y= 20
8     r1 = diferenca(x, y)
9
10    k=10
11    z=30
12    r2 = diferenca(k, z)
13
14    l = 30
15    m = 25
16    r3 = diferenca(l, m)
17    print(f'as diferenças calculadas são {r1}, {r2} e {r3} ')
18
19    diferenca()
```

Run: exemplos apostilas

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\p
as diferenças calculadas são -10, -20 e 5
Process finished with exit code 0
```

Neste exemplo repare:

- O resultado retornado para cada chamada da função **diferença** foi armazenado respectivamente em **r1**, **r2** e **r3**.
- 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:

```
exemplos apostilas.py
1 def diferenca(a,b):
2     res = a-b
3     return res
4
5
6     x= 10
7     y= 20
8     if diferenca(x, y) > 0:
9         print(f'resultado da diferença foi positivo')
10    elif diferenca(x, y) < 0:
11        print(f'resultado da diferença foi negativo ')
12    else:
13        print(f'resultado da diferença foi zero')
14
15    else
```

Run: exemplos apostilas

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\scri
resultado da diferença foi negativo
Process finished with exit code 0
```

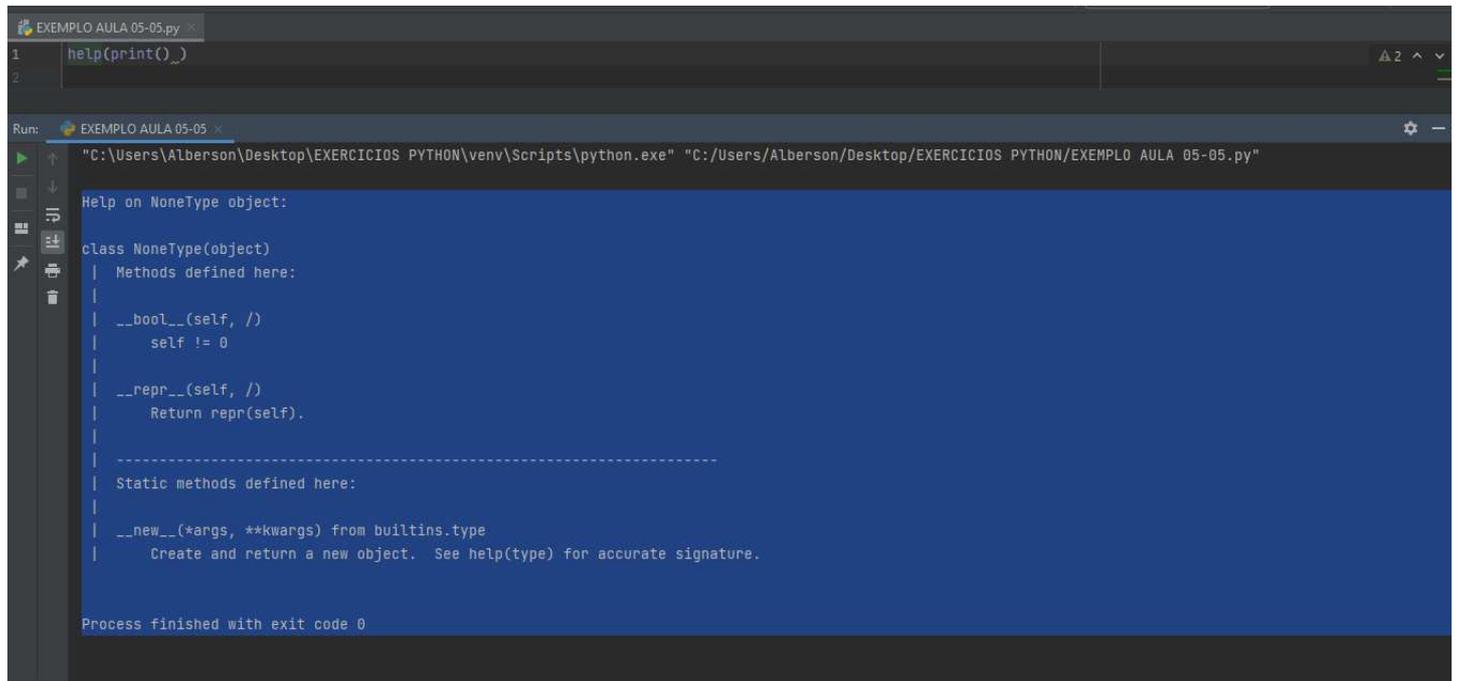
Observe mais um exemplo útil do uso do comando **return**:

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:



```
EXEMPLO AULA 05-05.py
1 help(print())
2

Run: EXEMPLO AULA 05-05
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/EXEMPLO AULA 05-05.py"

Help on NoneType object:

class NoneType(object)
| Methods defined here:
|
| __bool__(self, /)
|     self != 0
|
| __repr__(self, /)
|     Return repr(self).
|
|-----
| Static methods defined here:
|
| __new__(*args, **kwargs) from builtins.type
|     Create and return a new object.  See help(type) for accurate signature.

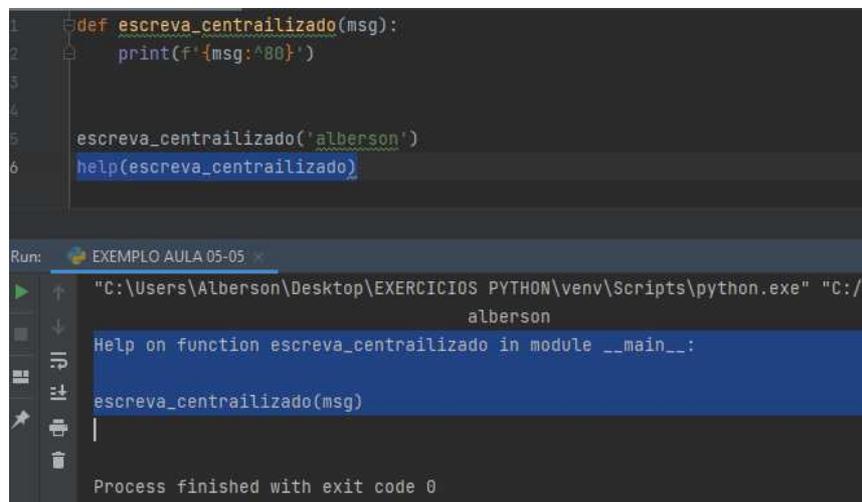
Process finished with exit code 0
```

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:



```
1 def escreva_centralizado(msg):
2     print(f'{msg:^80}')
3
4
5 escreva_centralizado('alberson')
6 help(escreva_centralizado)

Run: EXEMPLO AULA 05-05
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/U
alberson
Help on function escreva_centralizado in module __main__:
escreva_centralizado(msg)

Process finished with exit code 0
```

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:

```
def escreva_centralizado(msg):  
    """  
    :param msg:  
    :return:  
    """  
    print(f'{msg:^80}')
```

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:

```
1 def escreva_centralizado(msg):  
2     """  
3     =====> esta rotina mostra uma mensagem centralizada na tela  
4     :param msg: ESTE PARÂMETRO DE ENTRADA RECEBE A MENSAGEM A SER CENTRALIZADA  
5     """  
6     print(f'{msg:^80}')7  
8  
9     escreva_centralizado('alberson')  
10    help(escreva_centralizado)
```

Run: EXEMPLO AULA.05-05

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/De  
alberson  
Help on function escreva_centralizado in module __main__:  
escreva_centralizado(msg)  
    =====> esta rotina mostra uma mensagem centralizada na tela  
    :param msg: ESTE PARÂMETRO DE ENTRADA RECEBE A MENSAGEM A SER CENTRALIZADA
```

Process finished with exit code 0

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:

- para ser APROVADO o aluno deve ter tirado média mínima 6;
- para estar EM EXAME o aluno deve ter tirado média entre 3.5 e menor que 6;
- 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:

- 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

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

observações:

- Repare que a função se chama "digitenúmero"
- 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.
 - idadedaquia50anos() – 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 exhibe as DOCSTRINGS criadas nas funções do módulo importado.

Vamos então criar o primeiro programa mencionado acima:

```
tratamentoidades.py
1 def mn(idade = 0):
2     """
3     FUNÇÃO mn:
4     =====>>> ESTA FUNÇÃO DO MÓDULO maiormenor VERIFICA SE UMA PESSOA É OU NÃO MAIOR DE IDADE
5     :param idade: Dado do tipo inteiro que recebe uma idade no momento da chamada de mn do pacote maiormenor
6     :return: retornará a mensagem 'maior' caso a idade recebida for maior ou igual a 18 anos;
7             retornará a mensagem 'menor' caso a idade recebida form menor que 18 anos
8     """
9     if (idade>=18):
10        return 'maior'
11    else:
12        return 'menor'
13
14
15 def idadedaquia50anos(idade = 0 ):
16     """
17     FUNÇÃO idadedaquia50anos:
18     =====>>> ESTA FUNÇÃO DO MÓDULO maiormenor CALCULA A IDADE DE UMA PESSOA DAQUI A 50 ANOS
19     :param idade: Dado do tipo inteiro que recebe uma idade que será usada para somar 50 anos
20     :return: Será retornado um dado do tipo inteiro que refere-se a idade recebida adicionada de 50 anos
21     """
22    idadefutura = idade+50
23    return idadefutura
```

INTERESSANTE DE MAIS!!!! 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
1 import tratamentoidades
2 #mostrando as funcionalidades do módulo maiormenor
3 help_(tratamentoidades)
4
5 #recebendo um idade informada pelo usuário
6 id= int(input('digite sua idade'))
7 #chamando a funcionalidade mn() para verificar se idade corresponde a uma pessoa maior ou menor de idade
8 mensagem = tratamentoidades.mn(id)
9
10 #mostrando mensagem retornada pela função maiormenor
11 print(f'você é {mensagem}')
12
13 #Usando a funcionalidade idadedequi50anos para mostrar a idade informada somada com 50 anos
14 novalid = tratamentoidades.idadedequi50anos(id)
15 #mostrando a idade informada adicionada de 50 anos
16 print(f'Daqui a 50 anos você terá {novalid}')
17
```

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
  idadedequi50anos(idade=0)
    FUNÇÃO idadedequi50anos:
    =====>>> 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 VERIFICA 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 “tratamentoidades” 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:

```
Run: testesidade_chamadaderotina x
tratamentoidades

FUNCTIONS
  idadedaquia50anos(idade=0)
  FUNÇÃO idadedaquia50anos:
  =====> 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 VERIFICA 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 idade18
você é maior
Daqui a 50 anos você terá 68

Process finished with exit code 0
```

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:

- 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.
- 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.
- 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:

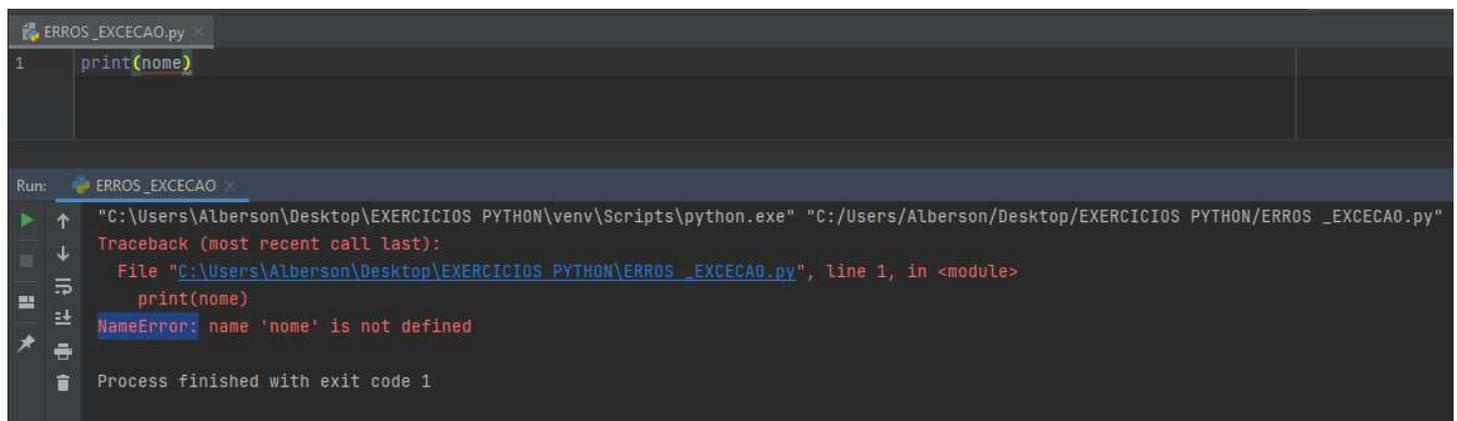
- **ERROS DE SINTAXES:** Não devem ser tratados, pois cabe ao programador tomar o cuidado de usar o comando de forma correta.
- **ERROS DE SEMÂNTICOS:** Estes são exemplos de erros que podemos tratar em programas que desenvolvermos. Exemplos destes tipos de erros:
 - Tentar usar uma variável que não 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()



```
ERROS_EXCECAO.py x
1 print(nome)

Run: ERROS_EXCECAO x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\env\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/ERROS_EXCECAO.py"
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\ERROS_EXCECAO.py", line 1, in <module>
    print(nome)
NameError: name 'nome' is not defined

Process finished with exit code 1
```

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



```
ERROS_EXCECAO.py x
1 valorproduto = float(input('digite o valor do produto R$: '))
2 print(f'O valor do produto é {valorproduto}')
5

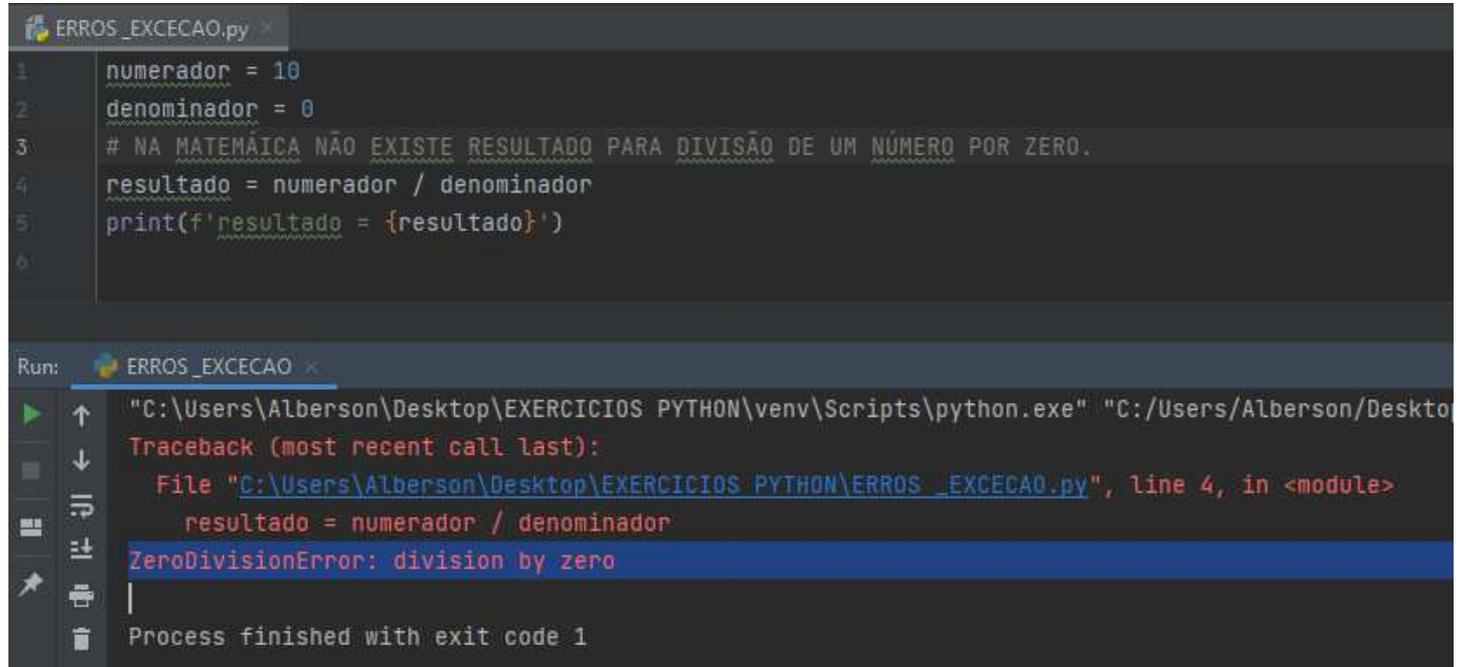
Run: ERROS_EXCECAO x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/ERROS_EXCECAO.py"
digite o valor do produto R$: albertson
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\ERROS_EXCECAO.py", line 1, in <module>
    valorproduto = float(input('digite o valor do produto R$: '))
ValueError: could not convert string to float: 'albertson'

Process finished with exit code 1
```

Observe!!!

- 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.
- 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:



```
ERROS_EXCECAO.py x
1 numerador = 10
2 denominador = 0
3 # NA MATEMÁTICA NÃO EXISTE RESULTADO PARA DIVISÃO DE UM NÚMERO POR ZERO.
4 resultado = numerador / denominador
5 print(f'resultado = {resultado}')
6

Run: ERROS_EXCECAO x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON/ERROS_EXCECAO.py"
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\ERROS_EXCECAO.py", line 4, in <module>
    resultado = numerador / denominador
ZeroDivisionError: division by zero

Process finished with exit code 1
```

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:

```
ERROS_EXCECAO.py x
1  numerador = 10
2  denominador = '2'
3  # não podemos dividir um número por uma string
4  resultado = numerador / denominador
5  print(f'resultado = {resultado}')
6

Run: ERROS_EXCECAO x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON\ERROS_EXCECAO.py"
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\ERROS_EXCECAO.py", line 4, in <module>
    resultado = numerador / denominador
TypeError: unsupported operand type(s) for /: 'int' and 'str'
Process finished with exit code 1
```

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:

```
ERROS_EXCECAO.py x
1  lista = [10, 20, 30]
2  print(f'o terceiro elemento é {lista[3]}')

Run: ERROS_EXCECAO x
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/Desktop/EXERCICIOS PYTHON\ERROS_EXCECAO.py"
Traceback (most recent call last):
  File "C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\ERROS_EXCECAO.py", line 2, in <module>
    print(f'o terceiro elemento é {lista[3]}')
IndexError: list index out of range
Process finished with exit code 1
```

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 x
1 import verificacpf
2

Run: ERROS_EXCECAO x
"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:

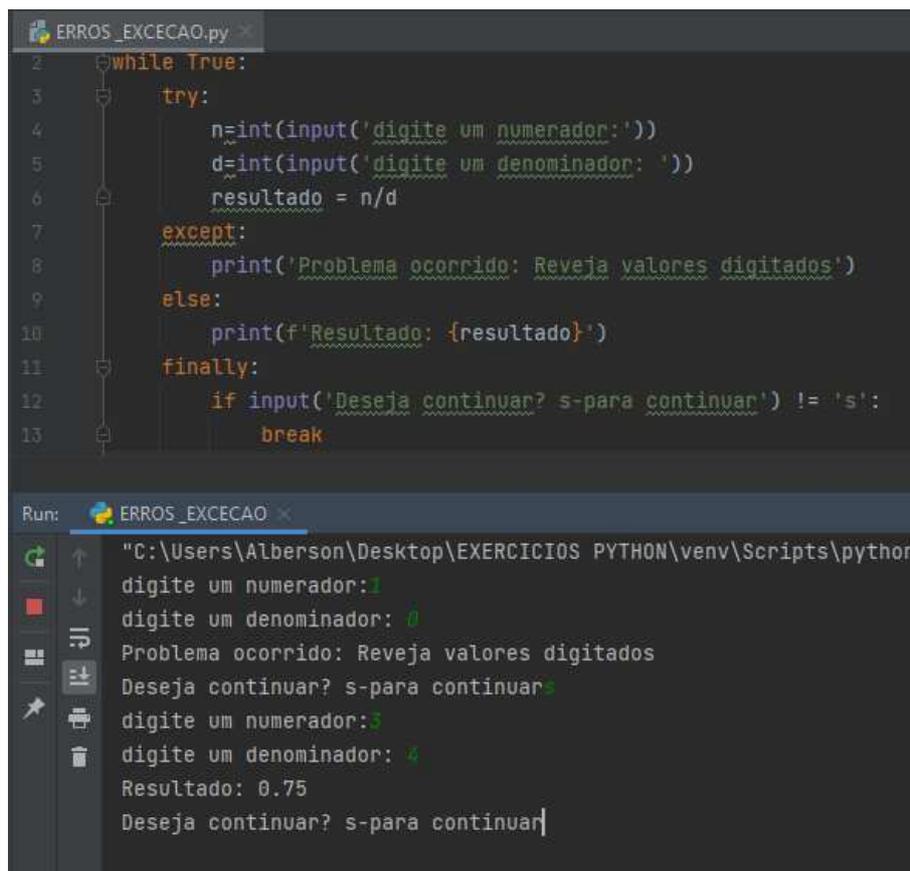
<comando(s) que deve(m) ser executado(s) quando NÃO OCORRER EXCEÇÃO>

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



```
ERROS_EXCECAO.py
2 while True:
3     try:
4         n=int(input('digite um numerador:'))
5         d=int(input('digite um denominador: '))
6         resultado = n/d
7     except:
8         print('Problema ocorrido: Reveja valores digitados')
9     else:
10        print(f'Resultado: {resultado}')
11    finally:
12        if input('Deseja continuar? s-para continuar') != 's':
13            break

Run: ERROS_EXCECAO
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python
digite um numerador:1
digite um denominador: 0
Problema ocorrido: Reveja valores digitados
Deseja continuar? s-para continuar
digite um numerador:3
digite um denominador: 4
Resultado: 0.75
Deseja continuar? s-para continuar
```

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

- 1) Foram digitados os números 1 para o numerador 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>:

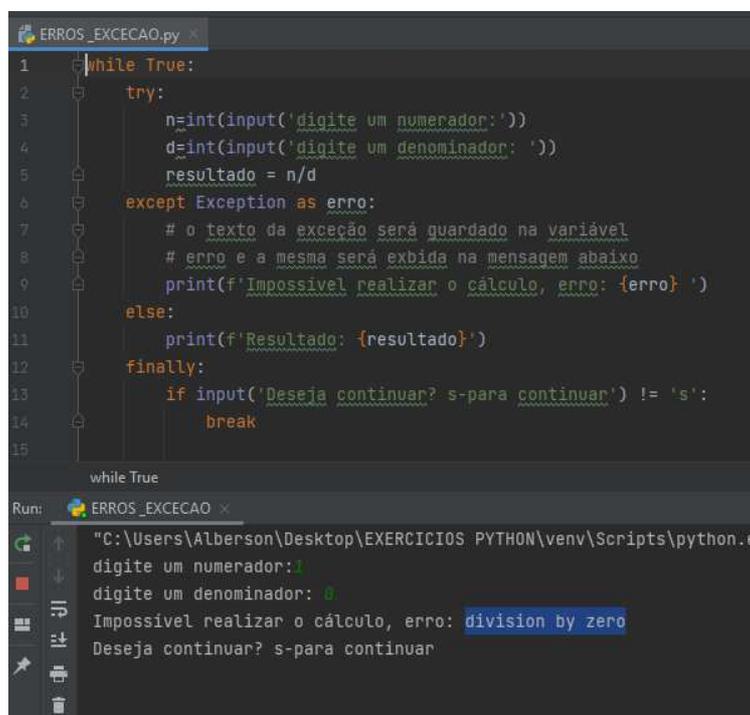
<Comando(s) que deve(m) ser executado(s) no caso de EXCEÇÃO ocorrer>

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:



```
ERROS_EXCECAO.py
1 while True:
2     try:
3         n=int(input('digite um numerador:'))
4         d=int(input('digite um denominador: '))
5         resultado = n/d
6     except Exception as erro:
7         # o texto da exceção será guardado na variável
8         # erro e a mesma será exibida na mensagem abaixo
9         print(f'Impossível realizar o cálculo, erro: {erro} ')
10    else:
11        print(f'Resultado: {resultado}')
12    finally:
13        if input('Deseja continuar? s-para continuar') != 's':
14            break
15
```

while True

Run: ERROS_EXCECAO ×

```
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe
digite um numerador:1
digite um denominador: 0
Impossível realizar o cálculo, erro: division by zero
Deseja continuar? s-para continuar
```

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:

```
ERROS_EXCECAO.py
1 while True:
2     try:
3         n=int(input('digite um numerador:'))
4         d=int(input('digite um denominador: '))
5         resultado = n/d
6     except (ZeroDivisionError):
7         print('ATENÇÃO: Não é aceito divisão de um número por 0(zero)')
8     except Exception as erro:
9         # o texto da exceção será guardado na variável
10        # erro e a mesma será exibida na mensagem abaixo
11        print(f'Impossível realizar o cálculo, erro: {erro} ')
12    else:
13        print(f'Resultado: {resultado}')
14    finally:
15        if input('Deseja continuar? s-para continuar') != 's':
16            break

while True
Run: ERROS_EXCECAO
"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alber
digite um numerador:1
digite um denominador: 0
ATENÇÃO: Não é aceito divisão de um número por 0(zero)
Deseja continuar? s-para continuar
```

Observem a vantagem!!!

- Neste exemplo, eu programador, imagino que um dos erros que podem ocorrer é o **"ZeroDivisionError"**, desta forma mencionarei este texto na frente o except.
- 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:

```

ERROS_EXCECAO.py
1 while True:
2     try:
3         n=int(input('digite um numerador:'))
4         d=int(input('digite um denominador: '))
5         resultado = n/d
6     except (ZeroDivisionError):
7         print('ATENÇÃO: Não é aceito divisão de um número por 0(zero)')
8     except Exception as erro:
9         # o texto da exceção será guardado na variável
10        # erro e a mesma será exibida na mensagem abaixo
11        print(f'Impossível realizar o cálculo, erro: {erro} ')
12
13    else:
14        print(f'Resultado: {resultado}')
15    finally:
16        if input('Deseja continuar? s-para continuar') != 's':
17            break
    
```

Run: ERROS_EXCECAO

```

"C:\Users\Alberson\Desktop\EXERCICIOS PYTHON\venv\Scripts\python.exe" "C:/Users/Alberson/D
digite um numerador:1
digite um denominador: 0
ATENÇÃO: Não é aceito divisão de um número por 0(zero)
Deseja continuar? s-para continuar s
digite um numerador:alberson
Impossível realizar o cálculo, erro: invalid literal for int() with base 10: 'alberson'
Deseja continuar? s-para continuar
    
```

IMPORTANTE: 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...)

- 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.
- Desenvolver um programa que mostre para o usuário o seguinte menu:
 - QUADRADO DE UM NÚMERO
 - RAIZ CÚBICA DE UM NÚMERO
 - 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.

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

professores - Table

Table Name: Schema: **univap**

Charset/Collation: Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
registro	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						
nomeprof	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
telefoneprof	VARCHAR(30)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
idadeprof	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
salarioprof	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

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

disciplinaxprofessores - Table

Table Name: Schema: **univap**

Charset/Collation: Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
codigodisciplinacurso	VARCHAR(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						
coddisciplina	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
codprofessor	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
curso	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
cargahoraria	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
anoletivo	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Repare que os campos coddisciplina 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:

Table Name: `disciplinasxprofessores` Schema: `univap` Engine: `InnoDB`

Foreign Key Name	Referenced Table	Column	Referenced Column
<code>fk_disciplina</code>	<code>`univap`.`disciplinas`</code>	<input checked="" type="checkbox"/> <code>coddisciplina</code>	<code>codigodisc</code>
<code>fk_professor</code>	<code>`univap`.`professores`</code>	<input type="checkbox"/> <code>codprofessor</code>	

Foreign Key Options: On Update: `RESTRICT`, On Delete: `RESTRICT`

Foreign Key Comment:

Columns | Indexes | **Foreign Keys** | Triggers | Partitioning | Options

Devemos criar também a seguinte relação:

Table Name: `disciplinasxprofessores` Schema: `univap` Engine: `InnoDB`

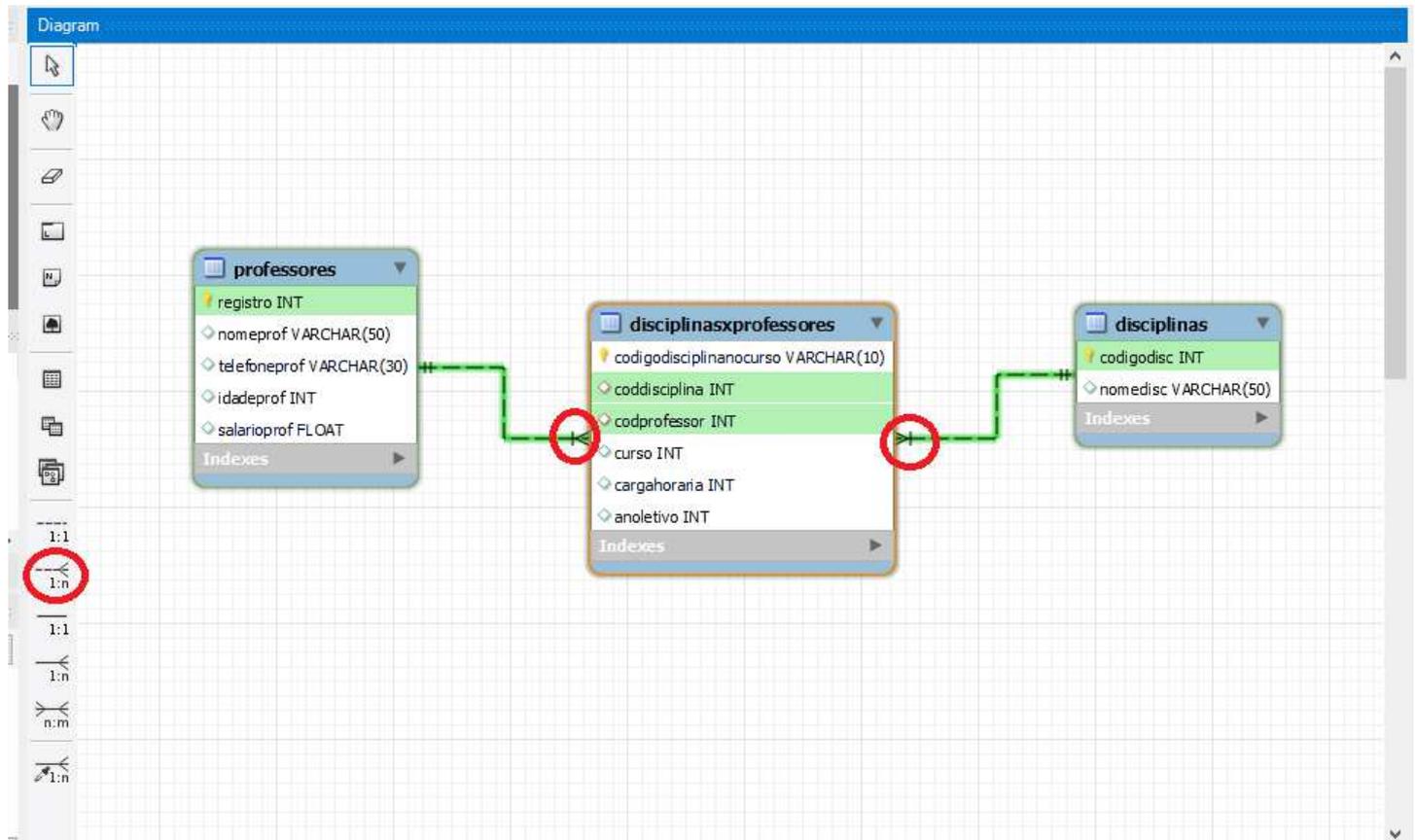
Foreign Key Name	Referenced Table	Column	Referenced Column
<code>fk_disciplina</code>	<code>`univap`.`disciplinas`</code>	<input type="checkbox"/> <code>codigodisciplina</code>	
<code>fk_professor</code>	<code>`univap`.`professores`</code>	<input checked="" type="checkbox"/> <code>codprofessor</code>	<code>registro</code>

Foreign Key Options: On Update: `RESTRICT`, On Delete: `RESTRICT`

Foreign Key Comment:

Columns | Indexes | **Foreign Keys** | Triggers | Partitioning | Options

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**:

```
C:\> Prompt de Comando
Microsoft Windows [versão 10.0.19042.985]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Alberson>pip install mysql-connector-python_
```

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:

```
C:\> Prompt de Comando
Microsoft Windows [versão 10.0.19042.985]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Alberson>pip install mysql-connector-python
Requirement already satisfied: mysql-connector-python in c:\users\alberson\appdata\local\programs\python\python39\lib\site-packages (8.0.25)
Collecting protobuf>=3.0.0
  Downloading protobuf-3.17.1-py2.py3-none-any.whl (173 kB)
    |#####| 173 kB 2.2 MB/s
Collecting six>=1.9
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, protobuf
Successfully installed protobuf-3.17.1 six-1.16.0
WARNING: You are using pip version 21.0.1; however, version 21.1.2 is available.
You should consider upgrading via the 'c:\users\alberson\appdata\local\programs\python\python39\python.exe -m pip install --upgrade pip' command.

C:\Users\Alberson>
```

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

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:

```
C:\>cd c:\users\alberson\appdata\local\programs\python\python39\
C:\Users\Alberson\AppData\Local\Programs\Python\Python39>python.exe -m pip install --upgrade pip_
```

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

```
C:\Users\Alberson\AppData\Local\Programs\Python\Python39>python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\alberson\appdata\local\programs\python\python39\lib\site-packages (21.0.1)
Collecting pip
  Downloading pip-21.1.2-py3-none-any.whl (1.5 MB)
    |#####| 1.5 MB 6.4 MB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.0.1
    Uninstalling pip-21.0.1:
      Successfully uninstalled pip-21.0.1
  Successfully installed pip-21.1.2

C:\Users\Alberson\AppData\Local\Programs\Python\Python39>
```

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

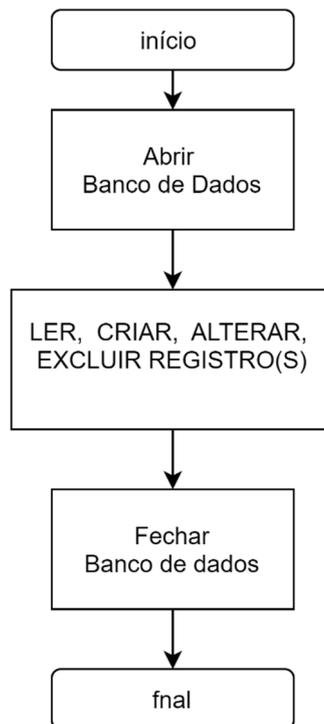
37.2 – Abrindo banco de dados num programa Python

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 excluimos 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:

```
1 import mysql.connector
2 try:
3     #criando um objeto para conexão ao banco de dados
4     #Devemos indicar em :
5     # host = Servidor ou IP(caso esteja numa rede),
6     # database = nome do banco de dados criado no workbenck
7     # user = nome do usuário definido no momento de criação do banco de dados
8     # password = senha de acesso do banco de dados
9     conexao = mysql.connector.Connect(host='localhost', database='univap', user='root', password='mae240299')
10    # testando se estamos conectado ao banco de dados
11    if conexao.is_connected():
12        informacaobanco = conexao.get_server_info()
13        print(f'conectado ao servidor banco de dados - Versão {informacaobanco}')
14        print('Conexão ok')
15        # criando objeto cursor, responsável para trabalharmos com registros retornados pela tabela fisica
16        comandosql = conexao.cursor()
17        # Criando uma QUERY para mostrar as informações do banco de dados ao qual nos conectamos
18        comandosql.execute('select database();')
19        # usando método fetchone para buscar um dado do banco de dados e armazená-lo na variável nomebanco
20        nomebanco = comandosql.fetchone()
21        print(f'Banco de dados acessado = {nomebanco}')
22    else:
23        print('Conexão não realizada com banco')
24    except Exception as erro:
25        print(f'Erro : {erro}')
26
```

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

```
Run: Tela_disciplinas x EXEMPLOS APOSTILA x
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
```

O programa mostrado anteriormente poderia ser resumido basicamente em:

```
1 import mysql.connector
2
3 try:
4     # criando um objeto para conexão ao banco de dados
5     # Devemos indicar em :
6     # host = Servidor ou IP(caso esteja numa rede),
7     # database = nome do banco de dados criado no workbenck
8     # user = nome do usuário definido no momento de criação do banco de dados
9     # password = senha de acesso do banco de dados
10    conexao = mysql.connector.Connect(host='localhost', database='univap', user='root', password='xxxxx')
11    # testando se já estamos conectados ao banco de dados
12    if conexao.is_connected():
13        print('Banco de dados ABERTO: Conexão realizada com banco')
14    else:
15        # Se por algum motivo não estiver conectado mostrar mensagem abaixo
16        print('Banco de dados FECHADO: NÃO OCORREU Conexão com banco')
17 except Exception as erro:
18    # Caso tenha dado erro no momento da conexão, este será exibido abaixo:
19    print(f'Erro : {erro}')
20
```

Run: Tela_disciplinas x EXEMPLOS APOSTILA x
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}')
21
22 except Exception as erro
```

Run: 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 DE MAIS 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 RETORNARÃO VÁRIOS REGISTROS E OUTRAS SOMENTE UM.

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

Run: CONSULTA (1) x

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 DE MAIS 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 a conexão com o banco de dados (*conexao.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.

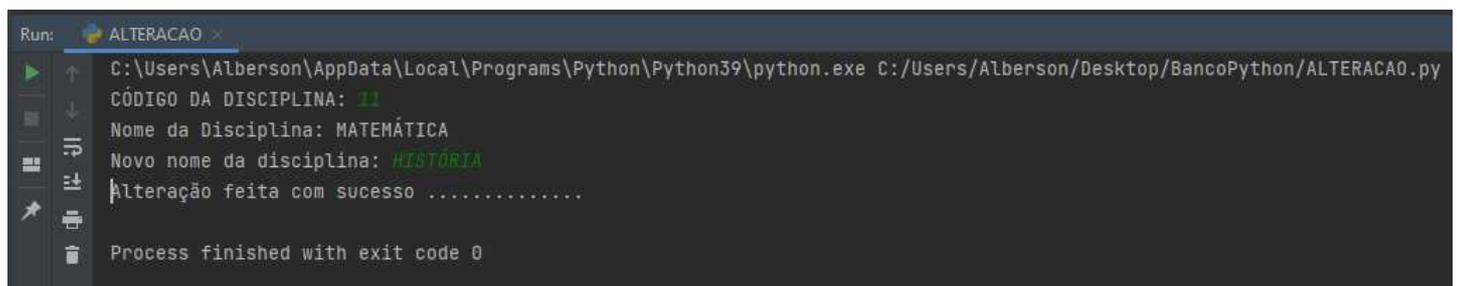
Tenha muito cuidado com o uso do comando *UPDATE*, pois a omissão da cláusula *WHERE*, ou uso incorreto deste critério de filtro de registros, pode trazer consequências graves para registros da tabela.

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.

```
1 import mysql.connector
2
3 try:
4     conexao = mysql.connector.Connect(host='localhost', database='univap', user='root', password='mae240299')
5     if conexao.is_connected():
6         comandosql = conexao.cursor()
7         cd = int(input("CÓDIGO DA DISCIPLINA: "))
8         comandosql.execute(f'select * from disciplinas where codigodisc = {cd};')
9         tabela = comandosql.fetchall()
10        if comandosql.rowcount > 0:
11            for registro in tabela:
12                print(f'Nome da Disciplina: {registro[1]}')
13                #Após exibir o nome da disciplina, pede o novo nome da disciplina para alteração
14                nn = input('Novo nome da disciplina: ')
15                #o MÉTODO EXECUTE está executando a alteração no registro que foi chamado
16                comandosql.execute(f'update disciplinas set nomedisc = "{nn}" where codigodisc = {cd};')
17                conexao.commit()
18                print('Alteração feita com sucesso .....')
19        except Exception as erro:
20            print(f'Ocorreu erro: {erro}')
21        comandosql.close()
22        conexao.close()
```

Tivemos como resultado:



IMPORTANTE DE MAIS NO PROGRAMA ACIMA !!!

REPERE 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.

```
1 import mysql.connector
2
3 try:
4     conexao = mysql.connector.Connect(host='localhost', database='univap', user='root', password='mae240299')
5     if conexao.is_connected():
6         comandosql = conexao.cursor()
7         cd = int(input("CÓDIGO DA DISCIPLINA: "))
8         comandosql.execute(f'select * from disciplinas where codigodisc = {cd};')
9         tabela = comandosql.fetchall()
10        if comandosql.rowcount > 0:
11            for registro in tabela:
12                print(f'Nome da Disciplina: {registro[1]}')
13                #Após exibir o nome da disciplina, PERGUNTA SE O USUÁRIO QUER EXCLUIR A DISCIPLINA EXIBIDA
14                if input('Deseja Excluir a disciplina? S - Sim: ') == 'S':
15                    #o MÉTODO EXECUTE VAI EXCLUIR A DISCIPLINA DA TABELA NO BANCO DE DADOS
16                    comandosql.execute(f'delete from disciplinas where codigodisc = {cd};')
17                    conexao.commit()
18                    print('Exclusão feita com sucesso .....')
19        except Exception as erro:
20            print(f'Ocorreu erro: {erro}')
21        comandosql.close()
22        conexao.close()
```

Tivemos como resultado:

```
C:\Users\Alberson\AppData\Local\Programs\Python\Python39\python.exe C:/Users/Alberson/Desktop/BancoPython/EXCLUSA0.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 !!!

REPRE 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.

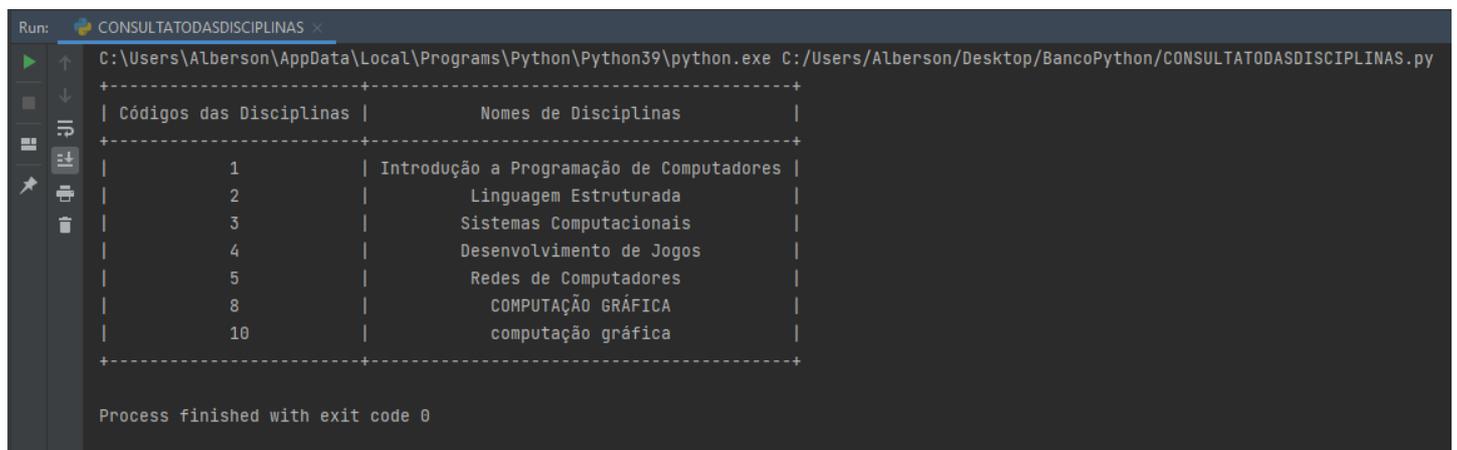
Com este exemplo poderão consultar os dados, ou registros que desejarem de qualquer tabela existente num banco de dados. Basta para isso, definir na instrução **SELECT** o filtro necessário para consulta, podendo utilizar logicamente a cláusula **where** quando necessário. Para o exemplo criado, não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

Exemplo:

```

1  import mysql.connector
2  from prettytable import PrettyTable
3  #criando duas colunas para o grid que exibirá todas as disciplinas cadastradas
4  grid = PrettyTable(['Códigos das Disciplinas', 'Nomes de Disciplinas'])
5  try:
6      conexao = mysql.connector.Connect(host='localhost', database='univap', user='root', password='mae240299')
7      if conexao.is_connected():
8          comandosql = conexao.cursor()
9          #repare que NÃO USEI A CLÁUSULA WHERE, ou seja, todas as disciplinas gravadas serão consultadas
10         comandosql.execute(f'select * from disciplinas;')
11         #O MÉTODO fetchall() retornará todos os registros filtrados (um ou mais registros) pelo comando select
12         tabela = comandosql.fetchall()
13         #O método rowcount conta quantos registros foram filtrados, caso tenha registro filtrado entra no if
14         if comandosql.rowcount > 0:
15             # se existir pelo menos uma disciplina na tabela temporária, mostre-as no grid
16             for registro in tabela:
17                 #criando as linhas do grid com os registros lidos da tabela temporária. Mostrando todas as disciplinas
18                 grid.add_row([registro[0], registro[1]])
19             print(grid)
20     except Exception as erro:
21         print(f'Ocorreu erro: {erro}')
22     comandosql.close()
23     conexao.close()
24
    
```

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 DE MAIS NO PROGRAMA ACIMA !!!

REPREME 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, EXCLUÍ 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 conectados 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 física
            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 armazená-lo
            # 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 disciplinas 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()
        # verificando 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 atualizar 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 excluída
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 CONTINUE ABAIXO, RETORNA PARA O WHILE QUE ESTÁ SENDO EXECUTADO
```


Este programa deve prever os testes de possibilidades 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 possibilidades 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.