Modulo1

Contents

1 Variaveis **3**
- 1.1 strings . . . 3
- 1.2 metodos usuais . . . 3

2 Coerçao **4**

3 Separador **4**

4 F-strings **5**

5 Formatacao strings usando format **5**

6 Operadores logicos **6**
- 6.1 exemplo and . . . 6
- 6.2 exemplo or . . . 6
- 6.3 operador not . . . 6
- 6.4 operador not in . . . 7

7 operadores aritmeticos **7**

8 interpolaçao de string com porcentagem em python **8**

9 Formatçao de strings com F-strings **8**

10 Fatiamento de strings **8**

11 Introduçao a Try e Except **9**

12 id - A identidade do valor que está na memoria **9**

13 Flags, is, is not, e None **9**

14 Tipos built-in, tipos imutaveis, metodos de strings, documentaçao **10**

15 while **10**

16 operadores de atribuiçao com operadores aritmeticos **10**

17 while + continue pulando repeticoes **11**

18 while + while(lacos internos) **11**

19 while-else **12**

20 for-in **12**

21 for + range **12**

22 for por baixo dos panos **13**

23 for e suas variações **13**

24 Tipo list **13**

25 introduçao aon desempacotamento e tuplas **15**
25.1 tuplas 15

26 enumerate **15**

27 imprecisao do numeros de pontos flutuante e decimal **15**

28 split e join **16**

29 Listas dentro de listas **16**

30 interpretador python **16**

31 Desempacotamento nas chamadas das funções **17**

32 Operacao ternario **18**

1 Variaveis

Ao contrário de linguagens como C ou Java, Python se destaca por sua tipagem dinâmica, um dos seus principais pontos fortes. Isso significa que nao é necessário definir o tipo de uma variável no momento de sua criaçao, o que facilita a escrita de código, mas também pode levar a erros em tempo de execuçao. Além disso, por ser uma linguagem multiparadigma, Python permite a criaçao de tipos de variáveis personalizados, oferecendo suporte à programaçao orientada a objetos.

Os tipos nativos do Python sao:

- **Inteiros (int)**: Suportam operações aritméticas comuns, como adiçao, subtraçao, multiplicaçao, entre outras.
- **Ponto flutuante (float)**: Representam números reais e também suportam operações aritméticas.
- **Strings (str)**: Sequências de caracteres, que permitem operações como concatenaçao, fatiamento, entre outras.
- **Listas (list)**: Coleções ordenadas e mutáveis, que permitem adiçao, remoçao e acesso por índice.
- **Tuplas (tuple)**: Semelhantes às listas, mas imutáveis.
- **Dicionários (dict)**: Coleções de pares chave-valor, que permitem acesso, adiçao e remoçao de elementos por chave.

1.1 strings

As strings em Python sao um dos tipos de variáveis mais utilizados, e a linguagem oferece amplo suporte para seu manuseio. Os dados textuais em Python sao tratados com objetos str, ou strings. Strings sao sequências imutáveis de pontos de código Unicode. As strings literais sao escritas de diversas maneiras:

Aspas simples: 'permitem aspas "duplas" internas'

Aspas duplas: "permitem aspas 'simples' internas"

Aspas triplas: "'Três aspas simples"', """Três aspas duplas"""

Strings de aspas triplas podem expandir por várias linhas – todos os espaços em branco associados serao incluídos em literal string.

Os literais strings que fazem parte de uma única expressao e têm apenas espaços em branco entre eles serao implicitamente convertidos em um único literal string. Isso é, ("spam " "eggs") == "spam eggs".

Veja Literais de string e bytes para mais informações sobre as várias formas de literal de string, incluindo o suporte a sequências de escape, e o prefixo r ("raw") que desabilita a maioria dos processos de sequência de escape.

As strings também podem ser criadas a partir de outros objetos usando o construtor str.

Uma vez que nao há nenhum tipo de "caractere" separador, indexar uma string produz strings de comprimento 1. Ou seja, para uma string nao vazia s, s[0] == s[0:1].

Também nao existe um tipo string mutável, mas o método str.join() ou a classe io.StringIO podem ser usados para construir strings de forma eficiente a partir de vários partes distintas.

1.2 metodos usuais

As constantes definidas neste módulo sao:

string.asciiletters A concatenaçao das constantes asciilowercase e asciiuppercase descritas abaixo. Este valor nao depende da localidade.

string.asciilowercase As letras minúsculas 'abcdefghijklmnopqrstuvwxyz'. Este valor nao depende da localidade e nao mudará.

string.asciiuppercase As letras maiúsculas 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'. Este valor nao depende da localidade e nao mudará.

string.digits A string '0123456789'.

string.hexdigits A string '0123456789abcdefABCDEF'.

string.octdigits A string '01234567'.

string.punctuation String de caracteres ASCII que sao considerados caracteres de pontuaçao na localidade C: !"#$

string.printable String de caracteres ASCII que sao considerados imprimíveis. Esta é uma combinaçao de digits, asciiletters, punctuation e whitespace.

string.whitespace Uma string contendo todos os caracteres ASCII que sao considerados espaços em branco. Isso inclui espaço de caracteres, tabulaçao, avanço de linha, retorno, avanço de formulário e tabulaçao vertical. site documetaçao.

2 Coerçao

A coerçao, ou conversao de tipos (*type conversion*, *typecasting*), é o processo de converter um tipo de dado em outro. Esse processo pode ocorrer entre tipos imutáveis e primitivos, como `str`, `int`, `float` e `bool`.

Exemplo de coerção em Python:

```
# Conversao de tipos (coercao)
# type conversion, typecasting, coercion
# O ato de converter um tipo em outro
# Tipos imutaveis e primitivos: str, int, float, bool

print(int('1'), type(int('1')))  # Converte a string '1' para
    inteiro
print(type(float('1') + 1))      # Converte a string '1' para float
     e soma 1
print(bool(' '))                 # Converte uma string nao vazia
    para booleano (True)
print(str(11) + 'b')             # Converte o numero 11 para string
     e concatena 'b'
```

3 Separador

No Python, a função `print()` permite personalizar a forma como os valores são impressos na tela. Entre os parâmetros dessa função, `sep` e `end` desempenham papéis importantes no controle do formato da saída. O parâmetro `sep` especifica o separador a ser utilizado entre os itens que são passados para a função `print()`, enquanto o parâmetro `end` define o que será impresso no final da linha. Essas opções permitem uma maior flexibilidade na exibição de dados, facilitando a formatação de resultados conforme a necessidade do usuário.

Por exemplo, podemos usar `sep` para separar números ou palavras com caracteres personalizados, como hífens ou espaços, e o `end` para controlar se haverá ou não uma quebra de linha, ou adicionar texto adicional no final da linha. A seguir, mostramos alguns exemplos de como esses parâmetros podem ser utilizados. No trecho de código a seguir, o parâmetro `sep` define o separador entre os números inteiros. Neste caso, os números serão separados por um hífen (-). Já o parâmetro `end` define o que será impresso no final de cada linha. No exemplo, após a impressão dos números, haverá uma quebra de linha seguida de `##`.

Exemplo de código:

```
# O parametro 'sep' separa os numeros por '-'
# O parametro 'end' determina o que sera impresso no final de cada
    linha

print(12, 34, 1011, sep="-", end='\n##')  # Numeros separados por
    '-' e quebra de linha com '##'
print(9, 10, sep="-", end='\n')          # Numeros separados por
    '-' e quebra de linha
```

```
print(56, 78, sep='-', end='\n')          # Numeros separados por
    '-' e quebra de linha
```

4 F-strings

As `f-strings` (ou `formatted string literals`) são uma maneira eficiente e legível de inserir variáveis diretamente dentro de strings no Python. Com elas, podemos incorporar valores de variáveis ou expressões dentro de strings, utilizando uma sintaxe simples e clara. Para usar uma `f-string`, basta colocar um `f` antes da string e, dentro das chaves (`{ }`), colocar o nome da variável ou a expressão que desejamos interpolar.

No exemplo a seguir, vemos o uso de `f-strings` para inserir variáveis dentro de uma string formatada. O código calcula o IMC de uma pessoa e o exibe com precisão específica:

```
# Definicao das variaveis
nome = 'Luiz Otavio'  # Nome da pessoa
altura = 1.80         # Altura da pessoa em metros
peso = 95             # Peso da pessoa em quilos
imc = peso / altura ** 2  # Calculo do Indice de Massa Corporal (
    IMC)

# Utilizacao das f-strings para formatacao de strings
linha1 = f'{nome} tem {altura:.2f} de altura,'  # A altura e
    formatada com 2 casas decimais
linha2 = f'pesa {peso} quilos e seu imc e'      # Exibe o peso
    diretamente
linha3 = f'{imc:.2f}'                           # O IMC e formatado
     com 2 casas decimais

# Impressao das linhas
print(linha1)
print(linha2)
print(linha3)

# Saida esperada:
# Luiz Otavio tem 1.80 de altura,
# pesa 95 quilos e seu IMC e
# 29.32
```

No código acima: - As variáveis `nome`, `altura`, `peso` e `imc` são inseridas nas strings através da sintaxe `f'variavel'`, com a possibilidade de aplicar formatação nas variáveis, como `:.2f` para exibir números flutuantes com 2 casas decimais. - O método `f'variavel:.2f'` é usado para formatar a variável `altura` e `imc`, garantindo que os valores sejam apresentados com duas casas decimais.

As `f-strings` são muito mais rápidas e legíveis em comparação com métodos mais antigos de formatação de strings, como o uso de `str.format()` ou o operador de percentagem `%`.

5 Formatacao strings usando format

setando a quantidade de casas apos a virgula

```
a = 'AAAAA'
b = 'BBBBBB'
c = 1.1
string = 'b={nome2} a={nome1} a={nome1} c={nome3:.2f}'
formato = string.format(nome1=a, nome2=b, nome3=c)
print(formato)
```

6 Operadores logicos

obs: print sempre avalia true por exemplo print (nome and idade) se nao houver nada nas duas variaveis a expressao retornar false, se houvera expressao retorna false and (e) or (ou) not (nao) and - Todas as condições precisam ser verdadeiras. Se qualquer valor for considerado falso, a expressao inteira será avaliada naquele valor Sao considerados falsy (que vc já viu) 0 0.0 ” False Também existe o tipo None que é usado para representar um nao valor

6.1 exemplo and

```
entrada = input('[E]ntrar [S]air: ')
senhadigitada = input('Senha: ')

senhapermitida = '123456'

if entrada == 'E' and senhadigitada == senhapermitida:
    print('Entrar')
else:
    print('Sair')
#Avaliacao de curto circuito
print(True and False and True)
print(True and 0 and True)
```

6.2 exemplo or

esse exemplo serve para verificar senha ou se nao há senha

```
# entrada = input('[E]ntrar [S]air: ')
# senhadigitada = input('Senha: ')

# senhapermitida = '123456'

# if (entrada == 'E' or entrada == 'e') and senhadigitada ==
   senhapermitida:
#     print('Entrar')
# else:
#     print('Sair')

# Avaliacao de curto circuito
senha = input('Senha: ') or 'Sem senha'
print(senha)
```

6.3 operador not

usado para inverter expressoes convem as vezes usar dentro de um print

```
# Operador logico "not"
# Usado para inverter expressoes
# not True = False
# not False = True
# senha = input('Senha: ')
print(not True)  # False
print(not False)  # True
```

6.4 operador not in

```
# Operadores in e not in
# Strings sao iteraveis
#  0 1 2 3 4 5
#  O t a v i o
# -6-5-4-3-2-1
# nome = 'Otavio'
# print(nome[2])
# print(nome[-4])
# print('vio' in nome)
# print('zero' in nome)
# print(10 * '-')
# print('vio' not in nome)
# print('zero' not in nome)

nome = input('Digite seu nome: ')
encontrar = input('Digite o que deseja encontrar: ')

if encontrar in nome:
    print(f'{encontrar} esta em {nome}')
else:
    print(f'{encontrar} nao esta em {nome}')
```

7 operadores aritmeticos

```
    adicao = 10 + 10
    print('Adicao', adicao)

    subtracao = 10 - 5
    print('Subtracao', subtracao)

    multiplicacao = 10 * 10
    print('Multiplicacao', multiplicacao)

    divisao = 10 / 3  # float
    print('Divisao', divisao)

    divisaointeira = 10 // 3
    print('Divisao inteira', divisaointeira)

    exponenciacao = 2 ** 10
    print('Exponenciacao', exponenciacao)

    modulo = 55 % 2  # resto da divisao
    print('Modulo', modulo)

    print(10 % 8 == 0)
    print(16 % 8 == 0)
    print(10 % 2 == 0)
    print(15 % 2 == 0)
    print(16 % 2 == 0)
```

8 interpolaçao de string com porcentagem em python

```
  """
s - string
d e i - int
f - float
x e X - Hexadecimal (ABCDEF0123456789)
"""
nome = 'Luiz'
preco = 1000.95897643
variavel = '%s, o preco e R$%.2f' % (nome, preco)
print(variavel)
#conversao de inteiro decimal para hexadecimal
print('O hexadecimal de %d e %08X' % (1500, 1500))
```

9 Formatçao de strings com F-strings

```
  """
  s - string
  d - int
  f - float
  .<numero de digitos>f
  x ou X - Hexadecimal
  (Caractere)(><^)(quantidade)
  > - Esquerda
  < - Direita
  ^ - Centro
  = - Forca o numero a aparecer antes dos zeros
  Sinal - + ou -
  Ex.: 0>-100,.1f
  Conversion flags - !r !s !a
  """
  variavel = 'ABC'
  print(f'{variavel}')
  print(f'{variavel: >10}')
  print(f'{variavel: <10}.')
  print(f'{variavel: ^10}.')
  print(f'{1000.4873648123746:0=+10,.1f}')
  print(f'O hexadecimal de 1500 e {1500:08X}')
  print(f'{variavel!r}')
```

10 Fatiamento de strings

```
  """
 012345678
 Ola mundo
-987654321
para que o fatiamento aconteca ate o final deve ser o indice final
   + 1
ou nao ter nada
Fatiamento [i:f:p] [::]
Obs.: a funcao len retorna a qtd
de caracteres da str
```

```
"""
variavel = 'Ola mundo'
print(variavel[::-1])
#contagem de tras pra frente -1 indica os passos , -1 indica de
   onde #comeca e -10 onde termina
#obs: o numero de passos pode ser maior que um
print(variavel[-1:-10:-1])
```

11 Introduçao a Try e Except

```
  """
try -> tentar executar o codigo
except -> ocorreu algum erro ao tentar executar
"""
numerostr = input(
    'Vou dobrar o numero que vc digitar: '
)

try:
    numerofloat = float(numerostr)
    print('FLOAT:', numerofloat)
    print(f'O dobro de {numerostr} e {numerofloat * 2:.2f}')
except:
    print('Isso nao e um numero')

# if numerostr.isdigit():
#     numerofloat = float(numerostr)
#     print(f'O dobro de {numerostr} e {numerofloat * 2:.2f}')
# else:
#     print('Isso nao e um numero')
```

12 id - A identidade do valor que está na memoria

entre duas variaveis na memoria com o mesmo valor , o python sempre mostra a primeira variavel endereçada na memória.

```
  #exemplo id
  v1 = 'a'
  v2 = 'a'
  #printa mesmo id de v1 devido o mesmo valor da memoria, se v1!=v2
      sera printado dois valores diferentes de id

  print(id(v1))
  print(id(v2))
```

13 Flags, is, is not, e None

```
  """
Flag (Bandeira) - Marcar um local
None = Nao valor
is e is not = e ou nao e (tipo, valor, identidade)
```

```
id = Identidade
"""
condicao = False
passounoif = None

if condicao:
    passounoif = True
    print('Faca algo')
else:
    print('Nao faca algo')


if passounoif is None:
    print('Nao passou no if')
else:
    print('Passou no if')
```

14 Tipos built-in, tipos imutaveis, metodos de strings, documentaçao

```
  """
https://docs.python.org/pt-br/3/library/stdtypes.html
Imutaveis que vimos: str, int, float, bool
"""
Tipos Built-in = tipos imbutidos
Tipos imutaveis =  nao podem acontecer mudancas no tipo
string = '1000'
# outravariavel = f'{string[:3]}ABC{string[4:]}'
# print(string)
# print(outravariavel)
print(string.zfill(10))
```

15 while

```
    """
  Repeticoes
  while (enquanto)
  Executa uma acao enquanto uma condicao for verdadeira
  Loop infinito -> Quando um codigo nao tem fim
  """
  condicao = True

  while condicao:
      nome = input('Qual o seu nome: ')
      print(f'Seu nome e {nome}')

      if nome == 'sair':
          break
  print('Acabou')
```

16 operadores de atribuiçao com operadores aritmeticos

```
"""
Operadores de atribuicao
= += -= *= /= //= **= %=
"""
contador = 10

###

contador /= 5
print(contador)
```

17 while + continue pulando repeticoes

```
"""
Repeticoes
while (enquanto)
Executa uma acao enquanto uma condicao for verdadeira
Loop infinito -> Quando um codigo nao tem fim
"""
contador = 0

while contador <= 100:
    contador += 1

    if contador == 6:
        print('Nao vou mostrar o 6.')
        continue

    if contador >= 10 and contador <= 27:
        print('Nao vou mostrar o', contador)
        continue

    print(contador)

    if contador == 40:
        break


print('Acabou')
```

18 while + while(lacos internos)

```
"""
Repeticoes
while (enquanto)
Executa uma acao enquanto uma condicao for verdadeira
Loop infinito -> Quando um codigo nao tem fim
"""
qtdlinhas = 5
qtdcolunas = 5

linha = 1
while linha <= qtdlinhas:
```

```
        coluna = 1
        while coluna <= qtdcolunas:
            print(f'{linha=} {coluna=}')
            coluna += 1
        linha += 1


    print('Acabou')
```

19 while-else

o else é sempre executado apos o while, porém se houver um break, o else nao é executado

```
  """ while/else """
string = 'Valor qualquer'

i = 0
while i < len(string):
    letra = string[i]

    if letra == ' ':
        break

    print(letra)
    i += 1
else:
    print('Nao encontrei um espaco na string.')
print('Fora do while.')
```

20 for-in

```
  # senhasalva = '123456'
# senhadigitada = ''
# repeticoes = 0

# while senhasalva != senhadigitada:
#     senhadigitada = input(f'Sua senha ({repeticoes}x): ')

#     repeticoes += 1

# print(repeticoes)
# print('Aquele laco acima pode ter repeticoes infinitas')
texto = 'Python'

novotexto = ''
for letra in texto:
    novotexto += f'*{letra}'
    print(letra)
print(novotexto + '*')
```

21 for + range

```
  """
For + Range
range -> range(start, stop, step)
"""
numeros = range(0, 100, 8)

for numero in numeros:
    print(numero)
```

22 for por baixo dos panos

```
  """
Iteravel -> str, range, etc (iter)
Iterador -> quem sabe entregar um valor por vez
next -> me entregue o proximo valor
iter -> me entregue seu iterador
"""
# for letra in texto
texto = 'Luiz'  # iteravel

# iteratador = iter(texto)  # iterator

# while True:
#     try:
#         letra = next(iteratador)
#         print(letra)
#     except StopIteration:
#         break

for letra in texto:
    print(letra)
```

23 for e suas variações

```
  for i in range(10):
  if i == 2:
      print('i e 2, pulando...')
      continue

  if i == 8:
      print('i e 8, seu else nao executara')
      break

  for j in range(1, 3):
      print(i, j)
else:
  print('For completo com sucesso!')
```

24 Tipo list

```
  """
Listas em Python
Tipo list - Mutavel
Suporta varios valores de qualquer tipo
Conhecimentos reutilizaveis - indices e fatiamento
Metodos uteis: append, insert, pop, del, clear, extend, +
"""
#         +01234
#         -54321
string = 'ABCDE'  # 5 caracteres (len)
# print(bool([]))  # falsy
# print(lista, type(lista))

#         0    1      2              3    4
#        -5   -4     -3             -2   -1
lista = [123, True, 'Luiz Otavio',  1.2, []]
lista[-3] = 'Maria'
print(lista)
print(lista[2], type(lista[2]))
```

métodos uteis

```
  #         0   1   2   3   4   5
  lista = [10, 20, 30, 40]
  # lista[2] = 300
  # del lista[2]
  # print(lista)
  # print(lista[2])
  lista.append(50)
  lista.pop()
  lista.append(60)
  lista.append(70)
  ultimovalor = lista.pop(3)
  print(lista, 'Removido,', ultimovalor)
#metodos pop, append, insert
  lista = [10, 20, 30, 40]
lista.append('Luiz')
nome = lista.pop()
lista.append(1233)
del lista[-1]
# lista.clear()
lista.insert(100, 5)
print(lista[4])
#metodos concatenar e extender Listas
listaa = [1, 2, 3]
listab = [4, 5, 6]
listac = listaa + listab
#o metodo extend extende a lista a para a lista b
listaa.extend(listab)
print(listaa)
```

cuidados com os tipos mutaveis

```
  """
Cuidados com dados mutaveis
= - copiado o valor (imutaveis)
= - aponta para o mesmo valor na memoria (mutavel)
"""
```

```
listaa = ['Luiz', 'Maria', 1, True, 1.2]
listab = listaa.copy()
listab = listaa #acontece q aqui listab aponta para listaa
listaa[0] = 'Qualquer coisa'
print(listaa)
print(listab)
```

25 introduçao aon desempacotamento e tuplas

```
, , nome, *resto = ['Maria', 'Helena', 'Luiz']
print(nome)
```

25.1 tuplas

```
  """
Tipo tupla - Uma lista imutavel
"""
nomes = ('Maria', 'Helena', 'Luiz')
# nomes = tuple(nomes)
# nomes = list(nomes)
print(nomes[-1])
print(nomes)
```

26 enumerate

```
  """
  enumerate - enumera iteraveis (indices)
  """
  # [(0, 'Maria'), (1, 'Helena'), (2, 'Luiz'), (3, 'Joao')]
  lista = ['Maria', 'Helena', 'Luiz']
  lista.append('Joao')

  for indice, nome in enumerate(lista):
      print(indice, nome, lista[indice])

  # for item in enumerate(lista):
  #     indice, nome = item
  #     print(indice, nome)


  # for tuplaenumerada in enumerate(lista):
  #     print('FOR da tupla:')
  #     for valor in tuplaenumerada:
  #         print(f'\t{valor}')
```

27 imprecisao do numeros de pontos flutuante e decimal

```
  """
Imprecisao de ponto flutuante
Double-precision floating-point format IEEE 754
https://en.wikipedia.org/wiki/Double-precisionfloating-pointformat
```

```
https://docs.python.org/pt-br/3/tutorial/floatingpoint.html
"""
import decimal

numero1 = decimal.Decimal('0.1')
numero2 = decimal.Decimal('0.7')
numero3 = numero1 + numero2
print(numero3)
print(f'{numero3:.2f}')
print(round(numero3, 2))
```

28 split e join

```
  """
split e join com list e str
split - divide uma string (list)
join - une uma string
strip - corta os espacos do comeco e o fim
rstrip - corta os espacos da Direita
lstrip - corta os espacos da Esquerda
"""
frase = '   Olha so que   , coisa interessante          '
# O ',' e o caracter de divisao.
listafrasescruas = frase.split(',')

listafrases = []
for i, frase in enumerate(listafrasescruas):
    listafrases.append(listafrasescruas[i].strip())

# print(listafrasescruas)
# print(listafrases)
frasesunidas = ', '.join(listafrases)
print(frasesunidas)
```

29 Listas dentro de listas

```
salas = [
    ['Maria', 'Helena'],
    ['Elena'],
    ["Luiz", 'Joao', 'Eduarda', (0, 10 , 10, 20 , 30)],

 ]
#print(salas[2][3][3])
for sala in salas:
    print(f'A sala e {sala}')
    for aluno in sala:
        print(aluno)
```

30 interpretador python

```
  """
Interpretador do Python

python mod.py (executa o mod)
python -u (unbuffered)
python -m mod (lib mod como script)
python -c 'cmd' (comando)
python -i mod.py (interativo com mod)

The Zen of Python, por Tim Peters

Bonito e melhor que feio.
Explicito e melhor que implicito.
Simples e melhor que complexo.
Complexo e melhor que complicado.
Plano e melhor que aglomerado.
Esparso e melhor que denso.
Legibilidade conta.
Casos especiais nao sao especiais o bastante para quebrar as regras
   .
Embora a praticidade venca a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambiguidade, recuse a tentacao de adivinhar.
Deve haver um -- e so um -- modo obvio para fazer algo.
Embora esse modo possa nao ser obvio a primeira vista a menos que
   voce seja holandes.
Agora e melhor que nunca.
Embora nunca frequentemente seja melhor que *exatamente* agora.
Se a implementacao e dificil de explicar, e uma ma ideia.
Se a implementacao e facil de explicar, pode ser uma boa ideia.
Namespaces sao uma grande ideia -- vamos fazer mais dessas!
"""
```

31 Desempacotamento nas chamadas das funções

```
  # Desempacotamento em chamadas
# de metodos e funcoes
string = 'ABCD'
lista = ['Maria', 'Helena', 1, 2, 3, 'Eduarda']
tupla = 'Python', 'e', 'legal'
salas = [
    # 0        1
    ['Maria', 'Helena', ],  # 0
    # 0
    ['Elaine', ],  # 1
    # 0       1       2
    ['Luiz', 'Joao', 'Eduarda', ],  # 2
]
# p, b, *, ap, u = lista
# print(p, u, ap)
# print('Maria', 'Helena', 1, 2, 3, 'Eduarda')
# print(*lista)
# print(*string)
# print(*tupla)
```

```
print(*salas, sep='\n')
```

32 Operacao ternario

```
  """
Operacao ternaria (condicional de uma linha)
<valor> if <condicao> else <outro valor>
"""
# condicao = 10 == 11
# variavel = 'Valor' if condicao else 'Outro valor'
# print(variavel)
# digito = 9  # > 9 = 0
# novodigito = digito if digito <= 9 else 0
# novodigito = 0 if digito > 9 else digito
# print(novodigito)
print('Valor' if False else 'Outro valor' if False else 'Fim')
```