Modulo1

Contents

1 Variaveis

Python = Linguagem programação
tipo de tipagem = Dinamica/Forte
str − > string − > texto
Strings são textos que estão dentro das aspas

2 Coerção

conversão de tipos , coerção type convertion, typecasting, coercion é o ato de converter um tipo em outro tipos imutaveis e primitivos str, int , float, bool

```
  # conversao de tipos, coercao
# type convertion, typecasting, coercion
# e o ato de converter um tipo em outro
#  tipos imutaveis e primitivos:
# str, int, float, bool
print(int('1'), type(int('1')))
print(type(float('1') + 1))
print(bool(' '))
print(str(11) + 'b')
```

3 Separador

nesse trecho de codigo sep indica que os numeros inteiros seram separados por - e ao final e o end barra n indica que havera uma quebra de linha

```
print(12, 34, 1011, sep= "-", end='\n##')
print(9, 10, sep= "-", end= '\n')
print(56, 78, sep= '-', end='\n')
```

4 F-strings

consigo inserir dentro de uma string as variaveis dentro do codigo

```
nome = 'Luiz Otavio'
altura = 1.80
peso = 95
imc = peso / altura ** 2

"f-strings"
linha_1 = f'{nome} tem {altura:.2f} de altura,'
linha_2 = f'pesa {peso} quilos e seu imc e'
linha_3 = f'{imc:.2f}'

print(linha_1)
print(linha_2)
print(linha_3)

# Luiz Otavio tem 1.80 de altura,
# pesa 95 quilos e seu IMC e
# 29.320987654320987
```

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 não houver nada nas duas variaveis a expressão retornar false, se houvera expressão retorna false and (e) or (ou) not (não) and - Todas as condições precisam ser verdadeiras. Se qualquer valor for considerado falso, a expressão inteira será avaliada naquele valor São considerados falsy (que vc já viu) 0 0.0 " False Também existe o tipo None que é usado para representar um não valor

6.1 exemplo and

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

senha_permitida = '123456'

if entrada == 'E' and senha_digitada == senha_permitida:
    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 não há senha

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

# senha_permitida = '123456'

# if (entrada == 'E' or entrada == 'e') and senha_digitada ==
   senha_permitida:
#     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)

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

    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 interpolação 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 Formatção 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 Introdução a Try e Except

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

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

# if numero_str.isdigit():
#     numero_float = float(numero_str)
#     print(f'O dobro de {numero_str} e {numero_float * 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
passou_no_if = None

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


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

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

```
"""
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'
# outra_variavel = f'{string[:3]}ABC{string[4:]}'
# print(string)
# print(outra_variavel)
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 atribuição 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
    """
    qtd_linhas = 5
    qtd_colunas = 5

    linha = 1
    while linha <= qtd_linhas:
        coluna = 1
        while coluna <= qtd_colunas:
            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 não 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

```
  # senha_salva = '123456'
# senha_digitada = ''
# repeticoes = 0

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

#     repeticoes += 1

# print(repeticoes)
```

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

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

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)
    ultimo_valor = lista.pop(3)
    print(lista, 'Removido,', ultimo_valor)
#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
lista_a = [1, 2, 3]
lista_b = [4, 5, 6]
```

```
lista_c = lista_a + lista_b
#o metodo extend extende a lista a para a lista b
lista_a.extend(lista_b)
print(lista_a)
```

cuidados com os tipos mutaveis

```
  """
Cuidados com dados mutaveis
= - copiado o valor (imutaveis)
= - aponta para o mesmo valor na memoria (mutavel)
"""
lista_a = ['Luiz', 'Maria', 1, True, 1.2]
lista_b = lista_a.copy()
lista_b = lista_a #acontece q aqui lista_b aponta para lista_a
lista_a[0] = 'Qualquer coisa'
print(lista_a)
print(lista_b)
```

25 introdução 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 tupla_enumerada in enumerate(lista):
```

```
  #     print('FOR da tupla:')
  #     for valor in tupla_enumerada:
  #         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-precision_floating-
   point_format
https://docs.python.org/pt-br/3/tutorial/floatingpoint.html
"""
import decimal

numero_1 = decimal.Decimal('0.1')
numero_2 = decimal.Decimal('0.7')
numero_3 = numero_1 + numero_2
print(numero_3)
print(f'{numero_3:.2f}')
print(round(numero_3, 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.
lista_frases_cruas = frase.split(',')

lista_frases = []
for i, frase in enumerate(lista_frases_cruas):
    lista_frases.append(lista_frases_cruas[i].strip())

# print(lista_frases_cruas)
# print(lista_frases)
frases_unidas = ', '.join(lista_frases)
print(frases_unidas)
```

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
# novo_digito = digito if digito <= 9 else 0
# novo_digito = 0 if digito > 9 else digito
# print(novo_digito)
print('Valor' if False else 'Outro valor' if False else 'Fim')
```