Conversão de Unidades

Gustavo J. V. Meira Filho

Table of contents

Objetivo

Introduzir a linguagem do python, funções, modularização e boas práticas.

- Conceitos de programação:
  - Definição de funções
  - Parâmetros
  - Retorno
  - Modularidade.
- Bibliotecas:
  - Nenhuma além do básico.
  - podemos usar o `pint`
    - Começando na Biblioteca
    - Guia do Usuário
    - Documentação
- Aplicações:
  - Criar funções para converter unidades comuns:
    - Pressão: atm Pa mmHg
    - Temperatura: K °C °F
    - Energia: J cal eV
  - Mostrar como organizar um mini "conversor" e aplicar em cálculos químicos simples (ex.: PV = nRT, convertendo pressão e temperatura em diferentes unidades).
  - Criar gráficos (ex.: massa atômica vs número atômico; cores por família).

Por Que o Python?

- **O que é Python?**
  - Linguagem de programação de alto nível, criada em 1991 por Guido van Rossum.
  - Fácil de aprender, legível, muito usada em ciência de dados, engenharia, inteligência artificial, automação etc.
- **Por que Python em engenharia química?**
  - Automação de cálculos repetitivos.
  - Análise e visualização de dados experimentais.
  - Modelagem e simulação de processos.
  - Integração com bibliotecas científicas (NumPy, SciPy, Pandas, Matplotlib).
  - Evita depender apenas de Excel/Matlab.
  - Permite realizar entregas complexas e contas que não podem ser feitas sem programação.
- **Como funciona em termos gerais:**
  - Código → interpretado (não precisa compilar).
  - Rodar em terminal, notebooks (Jupyter), ou IDEs (VS Code, Spyder, PyCharm).
  - Grande comunidade e suporte.

Introdução

```
print('Hello World!')
```

```
Hello World!
```

Operadores Matemáticos Básicos

Esses operadores são a base do cálculo em programação! Todos podem serimplementados na sua lógica para resolver problemas de engenharia ou de lógica. Aqui introduzimos dois tipos importantes de variável em python:

- **Variáveis Numéricas**: Usadas para armazenar números, sejam inteiros ou decimais. Exemplos:
  - `x = 10` (inteiro)
  - `y = 3.14` (decimal)
- **Variáveis Booleanas**: Representam valores de verdade, podendo ser `True` (verdadeiro) ou `False` (falso). São essenciais para controle de fluxo e tomadas de decisão em programas. Exemplos:
  - `verdade = True`
  - `falso = False`
- **Valor Ausente**: O `None` é usado para representar a ausência de valor ou um valor nulo. Ele é frequentemente utilizado para inicializar variáveis ou indicar que uma função não retorna nenhum valor significativo.

```
a = 2
b = 4
print(f'a = {a}, b = {b}')

print()
# Adição
print(f'{a} + {b} = {a + b}')
# Multiplicação
print(f'{a} * {b} = {a * b}')
# Subtração
print(f'{a} - {b} = {a - b}')

print()
# Divisão
print(f'{a} / {b} = {a / b}')
```

```python
# Divisão float
print(f'{b} / {a} = {b / a}')
# Divisão inteira
print(f'{b} // {a} = {b // a}')

print()
# Exponenciação
print(f'{a} ** {b} = {a ** b}')
# Raiz quadrada
print(f'{a} ** (1/{b}) = {a ** (1/b):.4f}')

print()
# Módulo
print(f'{a} % {b} = {a % b}')
print(f'{b} % {a} = {b % a}')

print()
import math
# Raiz quadrada usando a biblioteca math
print(f'math.sqrt({a}) = {math.sqrt(a):.4f}')
# Logaritmo natural
print(f'math.log({a}) = {math.log(a):.4f}')
# Logaritmo base 10
print(f'math.log10({a}) = {math.log10(a):.4f}')
# Fatorial
print(f'math.factorial({b}) = {math.factorial(b)}')
# Seno
print(f'math.sin({a}) = {math.sin(a):.4f}')
# Absoluto
print(f'math.fabs(-{a}) = {math.fabs(-a):.4f}')
```

```
a = 2, b = 4

2 + 4 = 6
2 * 4 = 8
2 - 4 = -2

2 / 4 = 0.5
4 / 2 = 2.0
4 // 2 = 2

2 ** 4 = 16
2 ** (1/4) = 1.1892

2 % 4 = 2
4 % 2 = 0

math.sqrt(2) = 1.4142
math.log(2) = 0.6931
math.log10(2) = 0.3010
math.factorial(4) = 24
```

```
math.sin(2) = 0.9093
math.fabs(-2) = 2.0000
```

```python
verdade = True
falso = False
vazio = None
print(f'verdade = {verdade}, falso = {falso}')
print(f'vazio = {vazio}')

# Operadores Lógicos
print()
print(f'{a} > {b} = {a > b}')   # Maior que
print(f'{a} < {b} = {a < b}')   # Menor que
print(f'{a} >= {b} = {a >= b}') # Maior ou igual a
print(f'{a} <= {b} = {a <= b}') # Menor ou igual a
print(f'{a} == {b} = {a == b}') # Igual a
print(f'{a} != {b} = {a != b}') # Diferente de
print()
print(f'not {verdade} = {not verdade}')               # Negação
print(f'{verdade} and {falso} = {verdade and falso}') # E lógico
print(f'{verdade} or {falso} = {verdade or falso}')   # Ou lógico
print(f'{verdade} ^ {falso} = {verdade ^ falso}')     # Ou exclusivo (XOR)
```

```
verdade = True, falso = False
vazio = None

2 > 4 = False
2 < 4 = True
2 >= 4 = False
2 <= 4 = True
2 == 4 = False
2 != 4 = True

not True = False
True and False = False
True or False = False
True ^ False = True
```

Condicionais

Condicionais permitem que o programa tome decisões com base em condições específicas. As principais estruturas condicionais em Python são:

- `if`: Executa um bloco de código se uma condição for verdadeira.
- `elif`: Permite verificar múltiplas condições, executando o bloco correspondente à primeira condição verdadeira.
- `else`: Executa um bloco de código se todas as condições anteriores forem falsas.

Podemos aninhar as condicionais!!

```
# Simulação simples de uma reação química
Ca0 = 2.0 # mol/L
Ca  = 0.5 # mol/L

X = (Ca0 - Ca) / Ca0
print(f'Conversão = {X:.1%}')

condicao_minima_de_projeto = 0.85
if X >= condicao_minima_de_projeto:
    print("Reação dentro da condição mínima de projeto.")
elif (X < condicao_minima_de_projeto) and (X >= 0.7):
    print("Reação abaixo da condição mínima de projeto, porém aceitável.")
else:
    print("Reação abaixo da condição mínima de projeto.")
```

```
Conversão = 75.0%
Reação abaixo da condição mínima de projeto, porém aceitável.
```

```
a = 10
b = 20
c = 30

if a > b:
    if a > c:
        maior = a
    else:
        maior = c
else:
    if b > c:
        maior = b
    else:
        maior = c

print(f'O maior valor entre {a}, {b} e {c} é {maior}.')
```

```
O maior valor entre 10, 20 e 30 é 30.
```

Strings

Strings são sequências de caracteres usadas para representar texto. Elas são imutáveis e suportam várias operações, como concatenação, fatiamento e formatação.

```
string = "AIChE"
print(f'String original: {string}')
```

```
String original: AIChE
```

Métodos específicos de strings!

```
print(f'String em maiúsculas: {string.upper()}')
print(f'String em minúsculas: {string.lower()}')
```

```
String em maiúsculas: AICHE
String em minúsculas: aiche
```

Fatiamento de strings: Podemos acessar partes específicas de uma string usando índices e fatias.

```
print(f'Primeira letra: {string[0]}')
print(f'Última letra: {string[-1]}')
print(f'Fatia da string (1 a 3): {string[1:4]}')
print(f'String invertida: {string[::-1]}')
```

```
Primeira letra: A
Última letra: E
Fatia da string (1 a 3): ICh
String invertida: EhCIA
```

Operações matemáticas em strings não são possíveis diretamente, mas são úteis para outros tipos de operações.

```
print(f'Concatenando com " - Python": {string + " - Python"}')
print()
print(f'String repetida 3 vezes: {(string+' ') * 3}')
print()
print(f'Comprimento da string: {len(string)}')
```

```
Concatenando com " - Python": AIChE - Python

String repetida 3 vezes: AIChE AIChE AIChE

Comprimento da string: 5
```

```
alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
print(f'Índice da letra C: {alfabeto.index("C")}')

print()
print(f'"abc" in "abcdef": {"abc" in "abcdef"}')
print(f'"xyz" in "abcdef": {"xyz" in "abcdef"}')
```

```
Índice da letra C: 2

"abc" in "abcdef": True
"xyz" in "abcdef": False
```

Listas

As listas são coleções ordenadas e mutáveis de itens. Elas permitem armazenar múltiplos valores em uma única variável, facilitando a manipulação e o acesso aos dados.

```
lista = [1, 2, 3, 4, 5]
print(f"Elementos da lista: {lista}")

lista_tipos_mistos = [1, "dois", 3.0, True, None]
print(f"Lista com tipos mistos: {lista_tipos_mistos}")
```

```
Elementos da lista: [1, 2, 3, 4, 5]
Lista com tipos mistos: [1, 'dois', 3.0, True, None]
```

Fatiamento de listas: Podemos acessar partes específicas de uma lista usando índices e fatias.

```
print(f'Primeiro elemento: {lista[0]}')
print(f'Último elemento: {lista[-1]}')
print(f'Fatia da lista (1 a 3): {lista[1:4]}')
print(f'Lista invertida: {lista[::-1]}')
print(f'Comprimento da lista: {len(lista)}')
print(f'Percorrendo a lista com passo 2: {lista[::2]}')
```

```
Primeiro elemento: 1
Último elemento: 5
Fatia da lista (1 a 3): [2, 3, 4]
Lista invertida: [5, 4, 3, 2, 1]
Comprimento da lista: 5
Percorrendo a lista com passo 2: [1, 3, 5]
```

Métodos de listas!

```
lista.append(6)
print(f'Lista após adicionar 6: {lista}')
lista.remove(3)
print(f'Lista após remover 3: {lista}')
popped = lista.pop()
print(f'Lista após remover o último elemento: {lista}')
print(f'Elemento removido: {popped}')
lista.sort(reverse=True)
print(f'Lista ordenada em ordem decrescente: {lista}')
lista.clear()
print(f'Lista após limpar todos os elementos: {lista}')
```

```
Lista após adicionar 6: [1, 2, 3, 4, 5, 6]
Lista após remover 3: [1, 2, 4, 5, 6]
Lista após remover o último elemento: [1, 2, 4, 5]
Elemento removido: 6
Lista ordenada em ordem decrescente: [5, 4, 2, 1]
Lista após limpar todos os elementos: []
```

Podemos criar listas aninhadas, formando matrizes ou tabelas.

```
import pprint
print()
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print(f'Matriz:')
pprint.pprint(matriz)
print(f'Elemento na posição [1][2]: {matriz[1][2]}')
print(f'Elemento na posição [0][1]: {matriz[0][1]}')
```

```
Matriz:
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Elemento na posição [1][2]: 6
Elemento na posição [0][1]: 2
```

O `range` é uma forma eficiente de gerar sequências de números inteiros, muito útil em loops e iterações.

```
print(f'range(5) = {list(range(5))}')
```

```
range(5) = [0, 1, 2, 3, 4]
```

List comprehension é uma maneira concisa e eficiente de criar listas em Python, permitindo gerar novas listas a partir de iteráveis existentes usando uma sintaxe compacta.

```
lista = [i for i in range(10)]
print(f'Lista criada com list comprehension: {lista}')

print()
# exponencial
exponencial = [2**i for i in range(10)]
print(f'Exponenciais de base 2: {exponencial}')

print()
# List comprehension com condição
pares = [i for i in range(10) if i % 2 == 0]
print(f'Números pares de 0 a 19: {pares}')
```

```
Lista criada com list comprehension: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Exponenciais de base 2: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

Números pares de 0 a 19: [0, 2, 4, 6, 8]
```

Loops

Loops são estruturas que permitem repetir um bloco de código várias vezes, facilitando a automação de tarefas repetitivas. Os principais tipos de loops em Python são:

- `for`: Usado para iterar sobre uma sequência (como listas, tuplas ou strings) ou qualquer objeto iterável;
- `while`: Executa um bloco de código enquanto uma condição for verdadeira.

Podemos usar `break` e `continue` para controlar o fluxo dentro dos loops. Além disso, podemos usar loops aninhados para iterar sobre estruturas de dados mais complexas.

```
lista = [1, 2, 3, 4, 5]
print(f"Elementos da lista: {lista}")

for elemento in lista:
    print(f'Elemento: {elemento}')
```

```
Elementos da lista: [1, 2, 3, 4, 5]
Elemento: 1
Elemento: 2
Elemento: 3
Elemento: 4
Elemento: 5
```

```
i = 0
while i < len(lista):
    print(f'Elemento na posição {i}: {lista[i]}')
    i += 1
```

```
Elemento na posição 0: 1
Elemento na posição 1: 2
Elemento na posição 2: 3
Elemento na posição 3: 4
Elemento na posição 4: 5
```

```
i = 0
while i < len(lista):
    print(f'Elemento na posição {i}: {lista[i]}')
    if lista[i] == 3:
        break
    i += 1
```

```
Elemento na posição 0: 1
Elemento na posição 1: 2
Elemento na posição 2: 3
```

```
i = 0
for elemento in lista:
    if elemento == 3:
        i += 1
        continue
    print(f'Elemento na posição {i}: {elemento}')
    i += 1
```

```
Elemento na posição 0: 1
Elemento na posição 1: 2
Elemento na posição 3: 4
Elemento na posição 4: 5
```

Tuplas

As tuplas são semelhantes às listas, mas são imutáveis. Elas são usadas para armazenar coleções de itens que não devem ser alterados após a criação.

```
tupla = (1, 2, 3, 4, 5)
print(f'Tupla: {tupla}')

print()
print(f'Primeiro elemento da tupla: {tupla[0]}')
print(f'Último elemento da tupla: {tupla[-1]}')
```

```
Tupla: (1, 2, 3, 4, 5)

Primeiro elemento da tupla: 1
Último elemento da tupla: 5
```

Tuplas são imutáveis, então não podemos adicionar ou remover elementos

- `tupla[0] = 10` Isso geraria um erro
- `tupla.append(6)` Isso também geraria um erro

Já no caso das listas, podemos modificar seus elementos livremente * `lista[0] = 10` Isso é válido

```
# Também é importante saber que listas apontam para o mesmo local na memória
lista1 = [1, 2, 3]
lista2 = lista1
lista2.append('x')
print(f'Lista 1: {lista1}')
print(f'Lista 2: {lista2}')

print()
# Para criar uma cópia independente, usamos o método copy()
lista1 = [1, 2, 3]
lista3 = lista1.copy()
```

```
lista3.append('x')
print(f'Lista 1 após modificar Lista 3: {lista1}')
print(f'Lista 3 (cópia independente): {lista3}')
```

```
Lista 1: [1, 2, 3, 'x']
Lista 2: [1, 2, 3, 'x']

Lista 1 após modificar Lista 3: [1, 2, 3]
Lista 3 (cópia independente): [1, 2, 3, 'x']
```

```
tupla1 = (1, 2, 3)
print(f'Tupla 1: {tupla1}')

# Tuplas também podem ser atribuídas a novas variáveis, mas continuam imutáveis
tupla2 = tupla1
print(f'Tupla 2 (cópia de Tupla 1): {tupla2}')
```

```
Tupla 1: (1, 2, 3)
Tupla 2 (cópia de Tupla 1): (1, 2, 3)
```

Dicionários

Dicionários são coleções não ordenadas de pares chave-valor. Eles permitem armazenar e acessar dados de forma eficiente usando chaves únicas.

```
dicionario = {'a': 1, 'b': 2, 'c': 3}
print(f'Dicionário original: {dicionario}')

print()
dicionario['d'] = 4
print(f'Dicionário após adicionar chave "d": {dicionario}')
```

```
Dicionário original: {'a': 1, 'b': 2, 'c': 3}

Dicionário após adicionar chave "d": {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

```
print(f'.keys(): {list(dicionario.keys())}')
print(f'.values(): {list(dicionario.values())}')

print()
for chave, valor in dicionario.items():
    print(f'Chave: {chave}, Valor: {valor}')
```

```
.keys(): ['a', 'b', 'c', 'd']
.values(): [1, 2, 3, 4]

Chave: a, Valor: 1
Chave: b, Valor: 2
Chave: c, Valor: 3
Chave: d, Valor: 4
```

Funções

Funções são blocos de código reutilizáveis que realizam tarefas específicas. Elas ajudam a organizar o código, melhorar a legibilidade e facilitar a manutenção.

- Definição de funções com `def`
- Parâmetros e argumentos

```
a, b = 5, 10
c = None

def soma(a, b):
    """Retorna a soma de dois números."""
    return a + b

resultado = soma(5, 10)
print(f'Resultado da soma: {resultado}')
print(f'a, b: {a, b}')
```

```
Resultado da soma: 15
a, b: (5, 10)
```

```
c = None

def sem_retorno(a, b):
    """Função que não retorna nada, apenas imprime a soma."""
    print(f'Soma dentro da função: {a + b}')

resultado = sem_retorno(5, 10)
print(f'Resultado da função sem_retorno: {resultado}')
print(f'c = {c}')
```

```
Soma dentro da função: 15
Resultado da função sem_retorno: None
c = None
```

```
def return_dict(x, y):
    """Retorna um dicionário com os valores e sua soma."""
    return {'x': x, 'y': y, 'soma': x + y}

result = return_dict(3, 7)
print(f'Resultado da função return_dict: {result}')
```

```
Resultado da função return_dict: {'x': 3, 'y': 7, 'soma': 10}
```

Conversor de Unidades

Podemos criar funções para converter unidades comuns:

$$\text{atm} \leftrightarrow \text{Pa} \leftrightarrow \text{mmHg}$$

$$\text{K} \leftrightarrow \text{°C} \leftrightarrow \text{°F}$$

$$\text{L} \leftrightarrow \text{m}^3 \leftrightarrow \text{mL}$$

$$\text{J} \leftrightarrow \text{cal} \leftrightarrow \text{eV}$$

Dicionários de Conversões

```python
def temperatura(T_celcius):
    """Converte temperatura de Celsius para Fahrenheit e Kelvin."""
    F = round((T_celcius * 9/5) + 32, 2)
    K = round(T_celcius + 273.15, 2)
    return {'C': T_celcius, 'F': F, 'K': K}

T = temperatura(25)
print(f'Temperaturas: {T}')
```

```
Temperaturas: {'C': 25, 'F': 77.0, 'K': 298.15}
```

```python
def pressao(P_atm):
    """Converte pressão de atm para Pa e bar."""
    Pa = P_atm * 101325
    bar = P_atm * 1.01325
    mmHg = P_atm * 760
    return {'atm': P_atm, 'Pa': Pa, 'bar': bar, 'mmHg': mmHg}

p = pressao(1)
print(f'Pressões: {p}')
```

```
Pressões: {'atm': 1, 'Pa': 101325, 'bar': 1.01325, 'mmHg': 760}
```

```python
def volume(V_litros):
    """Converte volume de litros para metros cúbicos e mililitros."""
    m3 = V_litros / 1000
    ml = V_litros * 1000
    return {'L': V_litros, 'm3': m3, 'mL': ml}

V = volume(1)
print(f'Volume: {V}')
```

```
Volume: {'L': 1, 'm3': 0.001, 'mL': 1000}
```

Constante dos Gases Ideais

Podemos construir um seletor de unidades de R (constante dos gases ideais) para facilitar nossos cálculos.

```python
def unidade_R(T, V, P):
    if T == 'K' and V == 'm3' and P == 'Pa':
        R = 8.314  # J/(mol·K)
    elif T == 'C' and V == 'L' and P == 'atm':
        R = 0.08206  # L·atm/(mol·K)
    else:
        R = None
        print("Unidades não reconhecidas para calcular R.")
    return R

R1 = unidade_R('K', 'm3', 'Pa')
print(f'Constante dos gases R (J/(mol·K)): {R1}')
```

```
Constante dos gases R (J/(mol·K)): 8.314
```

Usando para calcular o número de mols em diferentes unidades de pressão e temperatura.

$$\mathrm{pV} = \mathrm{nRT}$$

$$\therefore \mathrm{n} = \frac{\mathrm{pV}}{\mathrm{RT}}$$

```python
# p = 101325 Pa
# V = 0.0224 m3
# T = 273.15 K
n = (101325 * 0.0224) / (R1 * 273.15)
print(f'Número de mols n (usando R em J/(mol·K)): {n:.4} mols')
```

```
Número de mols n (usando R em J/(mol·K)): 0.9994 mols
```

```python
# Convertendo para unidades mais comuns
# p = 1 atm
# V = 22.4 L
# T = 0 °C
n = (
    pressao(1)['Pa'] * volume(22.4)['m3']
) / (
    unidade_R('K', 'm3', 'Pa') * temperatura(0)['K']
) # n = (101325 * 0.0224) / (R1 * 273.15)
print(f'Número de mols n (usando R em J/(mol·K)): {n:.4} mols')
```

```
Número de mols n (usando R em J/(mol·K)): 0.9994 mols
```

```
# Usando R em L·atm/(mol·K)
# p = 1 atm
# V = 22.4 L
# T = 0 °C -> note que tem que estar em K mesmo assim!
n = (1 * 22.4) / (unidade_R('C', 'L', 'atm') * temperatura(0)['K'])
print(f'Número de mols n (usando R em J/(mol·K)): {n:.4} mols')
```

```
Número de mols n (usando R em J/(mol·K)): 0.9993 mols
```

Operações em Mesma Base de Unidades

```
def conv_temp(T, unidade):
    """Converte temperatura para a unidade desejada."""
    if 'C' in unidade:
        return temperatura(T)
    elif 'F' in unidade:
        T_c = (T - 32) * 5/9
        return temperatura(T_c)
    elif 'K' in unidade:
        T_c = T - 273.15
        return temperatura(T_c)
    else:
        print("Unidade de temperatura não reconhecida.")
        return None

print(f'''10°C + 80°F - 300 K = {
    conv_temp(10, "C")["C"] +
    conv_temp(80, "F")["C"] -
    conv_temp(300, "K")["C"]:.2f} °C''')
print(f'''10°C + 80°F - 300 K = {
    conv_temp(10, "C")["F"] +
    conv_temp(80, "F")["F"] -
    conv_temp(300, "K")["F"]:.2f} °F''')
print(f'''10°C + 80°F - 300 K = {
    conv_temp(10, "C")["K"] +
    conv_temp(80, "F")["K"] -
    conv_temp(300, "K")["K"]:.2f} K''')

print()
print(f'Resposta = {temperatura(9.82)}')
```

```
10°C + 80°F - 300 K = 9.82 °C
10°C + 80°F - 300 K = 49.67 °F
10°C + 80°F - 300 K = 282.97 K

Resposta = {'C': 9.82, 'F': 49.68, 'K': 282.97}
```

Usando Bibliotecas

Podemos usar bibliotecas como `pint` para facilitar a conversão de unidades e garantir consistência nos cálculos.

```
from pint import UnitRegistry

ureg = UnitRegistry()
```

```
distancia = 100 * ureg.meter
print(f'Distância em metros: {distancia}')

print()
print(f'Magnitude = {distancia.magnitude}')
print(f'Distancia = {distancia.units}')
print(f'Dimensão = {distancia.dimensionality}')
```

```
Distância em metros: 100 meter

Magnitude = 100
Distancia = meter
Dimensão = [length]
```

```
print(f'magnitude = {distancia.magnitude}')
print(f'unidade = {distancia.units}')
```

```
magnitude = 100
unidade = meter
```

```
distancia_km = distancia.to(ureg.kilometer)
print(f'Distância (km): {distancia_km}')
```

```
Distância (km): 0.1 kilometer
```