Aula 2 – Fundamentos da Linguagem Python

EMC410235 - Programação Científica para Engenharia e Ciência Térmicas

Prof. Rafael F. L. de Cerqueira

2025.2

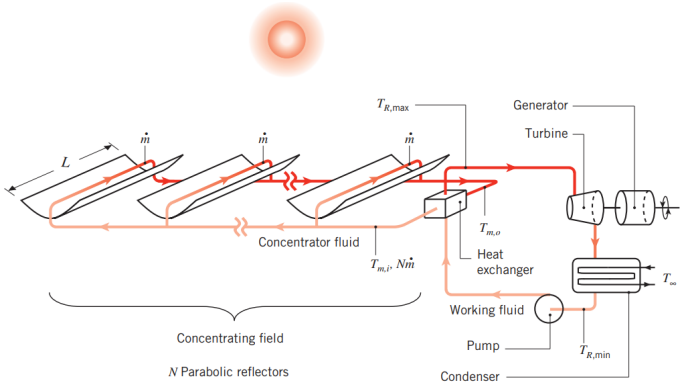
Objetivos da Aula

- Usar Python de forma interativa
- Compreender variáveis e operadores básicos
- Trabalhar com listas e tuplas
- Entender laços de repetição
- Aplicar estruturas condicionais
- Criar e utilizar funções
- Realizar entrada e saída de dados
- Explorar funções mais avançadas

Exemplo de Aplicação - Sistema de Concentração Solar

Um método para geração de energia elétrica por radiação solar envolve:

- Concentração da luz solar em tubos absorvedores.
- Tubos localizados nos focos de refletores parabólicos.
- Um fluido concentrador é aquecido ao escoar pelos tubos.
- Após o aquecimento, o fluido troca calor com o fluido de trabalho de um ciclo de Rankine.
- O fluido resfriado retorna ao concentrador solar.



[Incropera 8a edição - Exemplo 8.5]

Usando Python Interativamente

- Shell interativo: IPython, Jupyter, terminal, ***debug* interativo**
- Execução linha a linha

```
>>> 2 + 2
4
>>> x = 10
>>> print(x)
10
```

Declarando Variáveis

- Atribuição com =
- Tipagem dinâmica

```
x = 42
nome = "Gremio"
altura = 1.65
```

Operadores Básicos

- Aritméticos, comparação, lógicos

```
a = 7
b = 3
print(a + b, a / b, a % b)
print(a > b, a == b)
```

Comentários e Importações

Exemplos de comentários e importações:

```python
# Comentario de uma linha
import math  # importa o modulo completo
print(math.pi)

from math import sin # importa apenas a funcao sin
print(sin(math.pi / 2))

from math import * # importa tudo (desaconselhado)
print(cos(0))

import numpy as np  # importa com alias (pratica comum)
valores = np.array([1, 2, 3])
```

Observação: **evite usar** `from module import *`, pois pode causar conflitos de nomes e dificultar a leitura do código.

Precedência e Módulo

Python segue regras de precedência similares às da matemática. Parênteses podem ser usados para garantir a ordem de execução desejada. Além disso, operadores como divisão inteira (‘//‘) e módulo (‘%‘) são úteis em muitas situações de engenharia e ciência.

Exemplos:

```
print(3 + 4 * 2)    # Multiplicação ocorre antes
print((3 + 4) * 2)  # Parênteses alteram a ordem

print(10 // 3)      # Divisão inteira: resultado = 3
print(10 % 3)       # Módulo (resto): resultado = 1
```

Use parênteses sempre que quiser deixar clara a ordem das operações.

Listas e Tuplas

Listas e tuplas são estruturas de dados fundamentais em Python. Listas são mutáveis (podem ser alteradas), enquanto tuplas são imutáveis. Ambas armazenam coleções ordenadas de elementos.

Exemplos:

```
valores = [10, 20, 30] # Lista
valores.append(40)     # Adiciona um elemento
print(valores)         # Saída: [10, 20, 30, 40]

coordenadas = (2, 4)   # Tupla
print(coordenadas[0])  # Acesso ao primeiro valor
```

Use listas quando precisar modificar os dados; use tuplas para dados fixos.

Indexação e Fatiamento

Elementos de listas e tuplas podem ser acessados individualmente ou por fatias (slices). A indexação começa do zero e também aceita índices negativos para contar de trás para frente.

Exemplos:

```
letras = ['a', 'b', 'c', 'd']
print(letras[-1])   # Ultimo elemento: 'd'
print(letras[1:3])  # Elementos do índice 1 até 2: ['b', 'c']
```

Lembre-se que o intervalo '[i:j]' inclui o 'i', mas exclui o 'j'.

Copiando Listas

Atribuir uma lista a outra variável não cria uma nova cópia, apenas uma nova referência. Para copiar de verdade (shallow copy), use o método ‘.copy()‘.

Exemplo:

```
lista1 = [1, 2, 3]
lista2 = lista1.copy()  # Cria uma cópia independente
lista2.append(4)

print(lista1)           # Saída: [1, 2, 3]
print(lista2)           # Saída: [1, 2, 3, 4]
```

Evite usar ‘lista2 = lista1‘ quando desejar cópia real dos dados.

Laços de Repetição

Laços (‘for‘ e ‘while‘) permitem executar blocos de código repetidamente. O ‘for‘ é usado para percorrer sequências, enquanto o ‘while‘ repete até que uma condição deixe de ser verdadeira.

Exemplos:

```
# Laço for
for i in range(5):
  print(i)

# Laço while
i = 0
while i < 5:
  print(i)
  i += 1
```

Use ‘for‘ quando souber o número de repetições; ‘while‘ quando depender de uma condição.

Estruturas Condicionais

Condicionais (‘if‘, ‘elif‘, ‘else‘) permitem executar blocos diferentes de código com base em testes lógicos. São essenciais para controle de fluxo.

Exemplo:

```
temp = 35

if temp > 40:
  print("Muito quente!")
elif temp > 30:
  print("Quente")
else:
  print("Agradável")
```

Use expressões lógicas com operadores como ‘>‘, ‘<‘, ‘==‘, ‘!=‘ etc.

Funções

Funções permitem organizar o código em blocos reutilizáveis. São úteis para evitar repetição e melhorar a legibilidade. Usam a palavra-chave `def`.

Exemplo:

```
def saudacao(nome):
  """
  Exibe uma saudação personalizada na tela.

  Parâmetros:
  nome (str): Nome da pessoa a ser saudada.
  """

  print(f"Olá, {nome}!")

saudacao("Ana")
```

Use docstrings (entre aspas triplas) para explicar o que a função faz, seus parâmetros e o

Funções com Múltiplos Retornos

Em Python, funções podem retornar múltiplos valores separadamente.

```
def estatisticas(lista):
  """
  Calcula a média e o valor máximo de uma lista numérica.

  Parâmetros:
  lista (list of float): Conjunto de valores numéricos.

  Retorno:
  tuple: Média e valor máximo da lista.
  """

  media = sum(lista) / len(lista)
  maximo = max(lista)
  return (media, maximo)

m, M = estatisticas([10, 20, 30])
print(m, M)
```

Use tuplas para retornar múltiplos resultados e desempacote-os diretamente na chamada.

Escopo de Variáveis

O escopo de uma variável determina onde ela pode ser acessada. Variáveis definidas dentro de funções são locais e não afetam o valor de variáveis com o mesmo nome fora da função.

Exemplo:

```
x = 5  # variável global

def exemplo():
  x = 10 # variável local, diferente da global
  print("dentro:", x)

exemplo()
print("fora:", x)
```

Evite depender de variáveis globais dentro de funções. Prefira passar valores como argumentos.

Funções Lambda e Argumentos Nomeados

Funções 'lambda' são funções anônimas e compactas. Já os argumentos nomeados permitem definir valores padrão, facilitando chamadas mais flexíveis.

```
# Função lambda: usada para operações simples
dobro = lambda x: x * 2
print(dobro(5)) # Saída: 10

# Função com argumento nomeado (valor padrão)
def potencia(base, expoente=2):
  """
  Retorna a base elevada ao expoente.
  Se nenhum expoente for fornecido, usa 2.
  """
  return base ** expoente

print(potencia(3))    # Usa expoente padrão: 9
print(potencia(3, 3)) # Fornece expoente: 27
```

Use 'lambda' para funções curtas e argumentos nomeados para tornar funções mais versáteis.

Funções como Argumentos

Em Python, funções são objetos de primeira classe: podem ser passadas como argumentos para outras funções. Isso permite alto grau de flexibilidade e programação funcional.

```
def aplicar(f, x):
  """
  Aplica a função 'f' ao valor 'x'.

  Parâmetros:
  f (function): Função a ser aplicada.
  x (any): Valor de entrada.

  Retorno:
  Resultado de f(x).
  """
  return f(x)

print(aplicar(lambda x: x + 10, 5)) # Saída: 15
```

Essa abordagem é útil para abstrair operações genéricas, como filtros, transformações e integrações.

I/O - Entrada e Saída

Python usa a função `input()` para receber dados do usuário como texto. Para converter entradas para números, usamos funções como `int()` ou `float()`.

```
nome = input("Digite seu nome: ")
idade = int(input("Idade: "))
print(f"{nome} tem {idade} anos")
```

A função `input()` sempre retorna uma string; é necessário converter para o tipo desejado.

Escrita em Arquivo

Para gravar dados em um arquivo, usamos a função `open()` com o modo `'w'` (escrita). Sempre feche o arquivo com `close()` ou use `with`.

```
with open("saida.txt", "w") as f:
  f.write("Nome: Ana\n")
  f.write("Idade: 25\n")
```

O modo 'w' sobrescreve o conteúdo. Use 'a' para adicionar sem apagar.

Escrita com open() e close()

Embora o uso de `with` seja mais seguro, é possível abrir e fechar arquivos manualmente usando `open()` e `close()`.

```
f = open("dados.txt", "w")
  f.write("Temperatura: 37.5\n")
  f.write("Pressão: 101.3\n")
f.close()
```

Lembre-se de sempre chamar `close()` para garantir que o conteúdo seja gravado.

Leitura com read()

O método read() retorna todo o conteúdo do arquivo como uma única string.

```
with open("dados.txt", "r") as f:
  conteudo = f.read()

print(conteudo)
```

Use quando quiser ler todo o conteúdo de uma vez.

Leitura com readlines()

O método `readlines()` retorna uma lista de strings, cada uma representando uma linha do arquivo.

```
with open("dados.txt", "r") as f:
  linhas = f.readlines()

for linha in linhas:
  print(linha.strip())
```

Útil para processar arquivos linha a linha.

Iteração Direta sobre Arquivo

Arquivos podem ser usados diretamente em laços `for`, lendo linha por linha com menor uso de memória.

```
with open("dados.txt", "r") as f:
  for linha in f:
    print(linha.strip())
```

Mais eficiente para arquivos grandes.

Argumentos via Linha de Comando: sys.argv

O módulo sys permite acessar os argumentos passados ao script. O primeiro elemento de sys.argv é sempre o nome do arquivo Python.

```
import sys

nome = sys.argv[1]       # Primeiro argumento após o nome do script
idade = int(sys.argv[2]) # Segundo argumento convertido para inteiro

print(f"{nome} tem {idade} anos")
```

Execute no terminal como:
`python script.py Paulo 30`

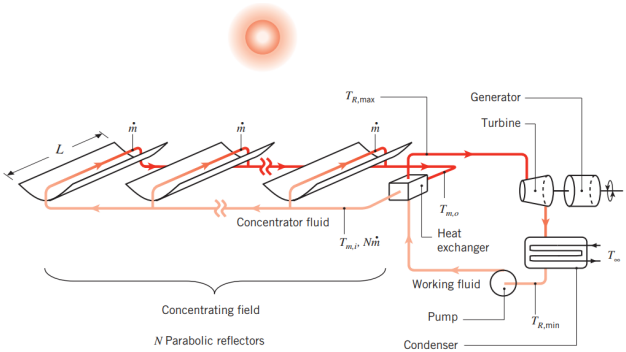
Exemplo de Aplicação

Problema aplicado de transferência de calor em sistemas de concentração solar

Problema de Transferência de Calor Solar

Um método para geração de energia elétrica por radiação solar envolve:

- Concentração da luz solar em tubos absorvedores.
- Tubos localizados nos focos de refletores parabólicos.
- Um fluido concentrador é aquecido ao escoar pelos tubos.
- Após o aquecimento, o fluido troca calor com o fluido de trabalho de um ciclo de Rankine.
- O fluido resfriado retorna ao concentrador solar.



[Incropera 8a edição - Exemplo 8.5]

Problema de Transferência de Calor Solar

Sabendo que o fluido concentrador:

- entra no tubo com $T_{m,i} = 400°C$
- e sai com $T_{m,o} = 450°C$

Propriedades:

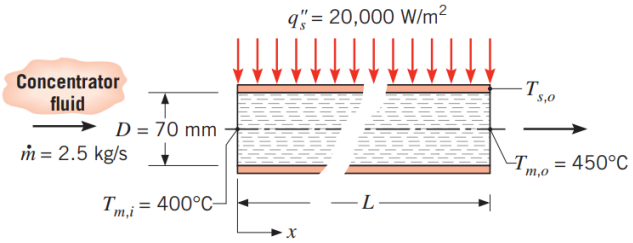
- $\rho = 700 kg/m^3$
- $k = 0.078 W/(m \cdot K)$
- $c_p = 2590 J/(kg \cdot K)$
- $\mu = 0.15 \times 10^{-3}\ N \cdot s/m^2$

Dado os seguintes parâmetros:

- $D = 70mm$
- $\dot{m} = 2.5 kg/s$
- $q''_s = 20000 W/m^2$

Responda:

1. Qual o comprimento necessário do concentrador, L?
2. A que taxa q o calor é transferido para o fluido em um único tubo?
3. Qual é a temperatura da superfície externa do tubo na saída do concentrador, $T_s(L)$?



Exercício para Casa

Objetivo: Estender o código do exemplo para permitir análises paramétricas com diferentes geometrias e condições.

Tarefas:

1. Alterar o código para incluir a opção de seção **quadrada**.
2. Analisar os seguintes cenários para **tubo circular** e **seção quadrada**:
   - 3 valores diferentes de fluxo térmico q''
   - 3 valores diferentes de vazão mássica $\dot{m}$
   - 3 valores diferentes de diâmetro (ou lado da seção)
3. Requisitos:
   - As **propriedades físicas** devem ser lidas de um `arquivo texto`.
   - As **saídas dos cenários** devem ser gravadas em um `arquivo texto`.

Bônus: permitir execução via linha de comando:

```
python analise_concentrador.py props.txt param1 param2 param3, etc...
```