Programação em Python visando a modelagem de processos agrícolas
Aula 01

PREFÁCIO – LEIA-ME!

1. Este texto é parte das disciplinas de graduação 1100222 (Modelagem do Crescimento de Culturas Agrícolas) e de pós-graduação LEB5048 (Modelagem de Culturas Agrícolas I), onde um dos objetivos é introduzir ao aluno à modelagem de processos envolvendo culturas agrícolas. A programação computacional é parte essencial disso.
O texto tem como premissa introduzir os conceitos fundamentais necessários para a realização das tarefas propostas nas semanas seguintes, ficando sob responsabilidade do aluno maior aprofundamento, caso tenha interesse.

2. Antes de iniciar, o aluno já deve ter instalado o Python 3 (Anaconda) e a interface Spyder no Windows. Veja instruções em separado.

3. Para se referir a códigos neste texto, será utilizado a seguinte estrutura:

```
aqui é o código digitado pelo aluno
aqui é a resposta fornecida pelo Python
```

O campo em cinza com fonte `Consolas` faz referência ao código escrito pelo aluno. A linha seguinte, sem preenchimento cinza e em letra com fonte `Consolas`, é a resposta do Python ao código executado. Toda vez que aparecer no corpo do texto a fonte `Consolas`, está sendo feita referência ao a um código ou parte de um.

4. Vários livros e páginas de internet foram consultados para a elaboração do texto, porém o mais consultado foi o livro *Learning Python 5th ed.* de Mark Lutz.

Conteúdo

1 Python e Spyder 3

1.1 Por que Python para este curso? 3
1.2 Spyder como interface 3
1.3 Criar e salvar um novo arquivo 4

2 Iniciar a programação com a interface Spyder 5

2.1 Introdução 5
2.2 A função input( ) 7
2.3 Operadores numéricos 8
2.4 Comentários 9
2.5 Lists (listas) – variáveis indexadas 9
2.6 Trabalhar com arquivos (Files) 10

3 Loops e condições: for, if else, while. 12

3.1 Loop: for 12
3.2 A função `range( )`. 13
3.3 Condição: if else 13
3.4 Loop: While 14
3.5 Exemplo de *arquivo* para *list*. 14

4 Funções especiais e o comando import. 18

1 Python e Spyder

1.1 Por que Python para este curso?

Python é uma linguagem que possui uma sintaxe simples, permitindo que o código seja lido, escrito e alterado com maior facilidade, se comparado com outras linguagens de "propósito geral" (do inglês *general purpose*). É otimizada para um desenvolvimento rápido e permite códigos menores do que linguagens como Fortran ou C.
Como programação científica e numérica, Python se destaca como uma das linguagens mais utilizadas atualmente.
A linguagem Python compila o código sem a criação de um arquivo executável para um determinado sistema operacional. Esta característica permite que o código seja compilado e executado em partes (similar a linguagem R, por *scripts*).
É gratuito.

1.2 Spyder como interface

Para o curso foi escolhida a interface Spyder. É similar à interface do Matlab e R.
Possui alguns recursos interessantes que facilitarão no desenvolvimento e execução dos programas em Python durante o curso.
O Spyder, por padrão, se encontra na pasta *Anaconda3* no menu iniciar.

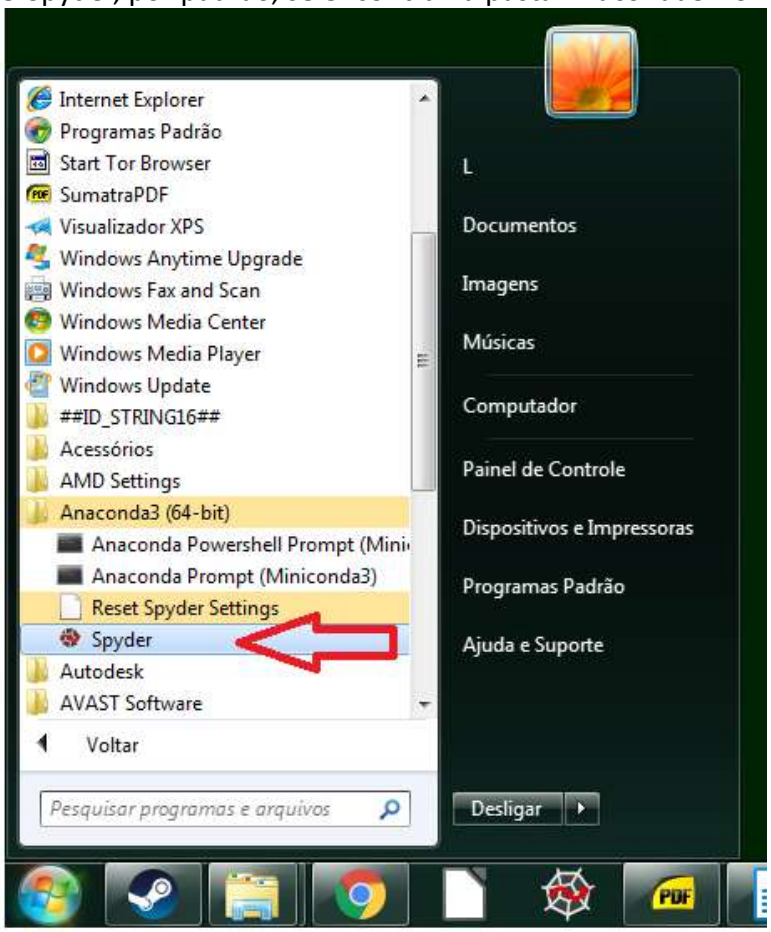

Figura 1

Depois de abrir o Spyder aparece a seguinte tela, onde podemos reconhecer três áreas:

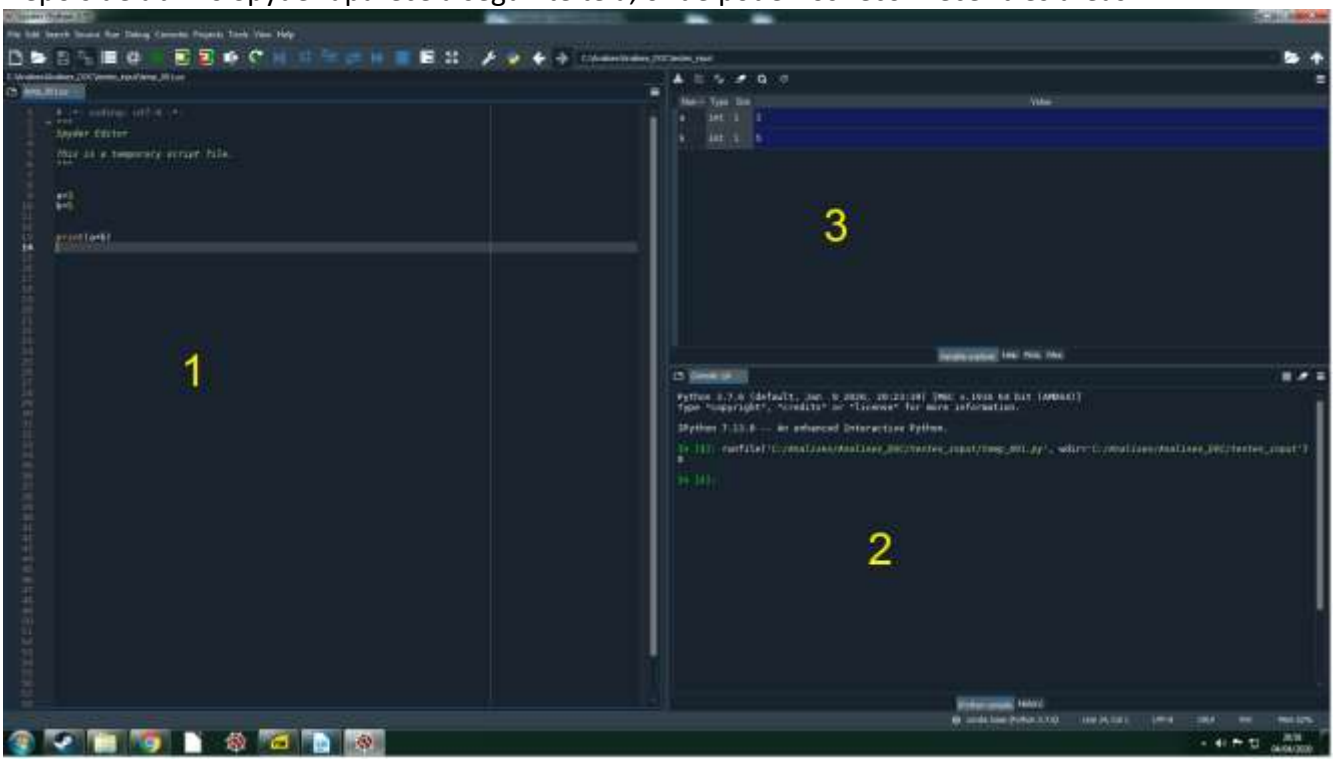

Figura 2

A janela 1, onde serão digitados os comandos, funciona como um editor de texto simples, se chama *Editor*. A janela 2, onde visualizaremos a saída do programa e mensagens sobre o funcionamento do programa, se chama *IPhyton Console*. A janela 3 pode ser configurada para mostrar algumas opções, dentre elas, *ajuda*, e*xploração de variáveis*. Para isso, basta clicar na aba desejada (na parte inferior desta mesma janela). Além dessas, o Spyder possui outras opções de janelas que não estão na figura, para mostrá-las, é necessário configurar. A maior parte destes recursos podem ser gerenciados no menu *view*, que se encontra no menu superior da interface. Caso alguma destas janelas, principalmente a 1 e 2, sejam fechadas sem querer, é possível abri-las no menu superior *view* > *panes*.
A interface Spyder apresenta muitos recursos e ferramentas, não sendo possível abordá-las todas no decorrer do curso.

1.3 Criar e salvar um novo arquivo

Para iniciarmos, é interessante começar um novo arquivo e salvá-lo na sequência em um local desejado. Este processo é semelhante a diversos programas, então não será apresentado aqui o procedimento. A extensão do arquivo será *.*py*.

Observa-se que ao criar um novo arquivo, o Spyder já implementa algumas linhas escritas por padrão, como é observado na Figura 2. Não é obrigatório mantê-las, fazem pouca diferença no programa.
O comando na linha 1 permite a utilização de caracteres UTF-8 em algumas variáveis que serão explicadas mais adiante. Se o aluno tiver curiosidade, pesquise no Google sobre caracteres UTF-8.
O comando em verde, da linha 2 até a linha 6 é um *comentário*. Por característica, os comentários não interferem na execução do programa, sua utilidade é ser explicativo e é dedicada ao usuário.

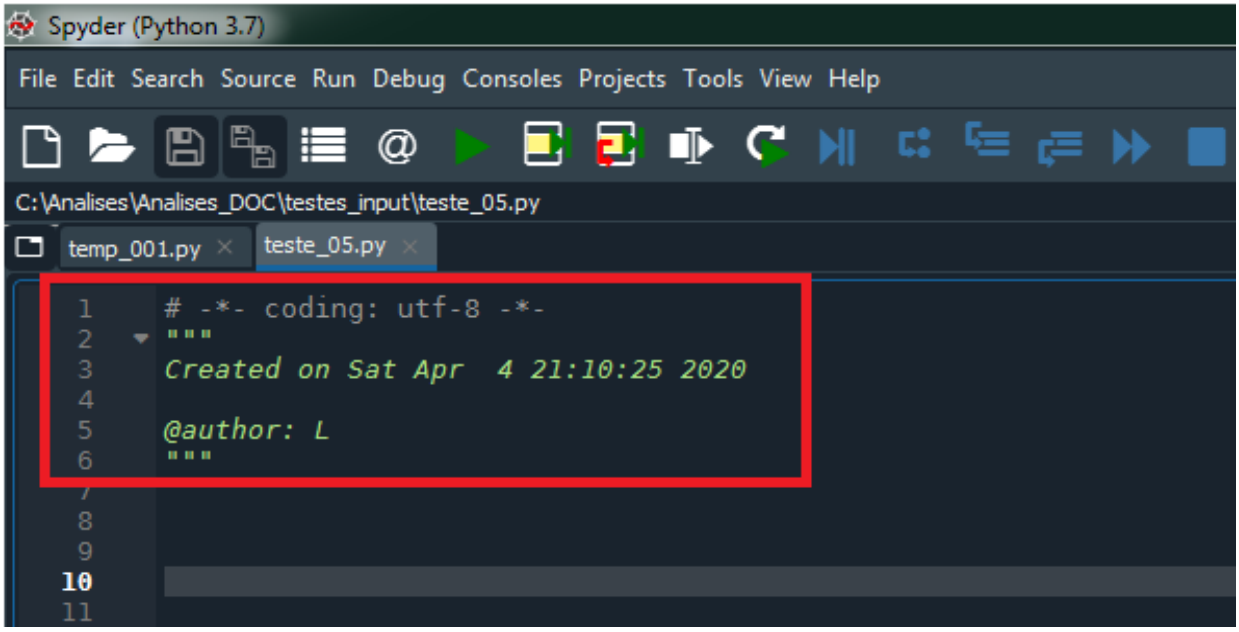

Figura 3

2 Iniciar a programação com a interface Spyder

2.1 Introdução

Neste exemplo será utilizado um código-exemplo. Para executar qualquer código, clicar no botão 
O nosso primeiro código irá somar dois valores, a e b, e imprimir o resultado na tela.
A sequência de comandos é

```
a = 3
b = 5
c = a + b
print(c)
```

Alternativamente, pode-se realizar a operação dentro da função `print()`, evitando a definição da variável c, como no exemplo:

```
a = 3
b = 5
print(a + b)
```

O resultado será o mesmo, pois o Python irá realizar a soma e depois mostrar o resultado na tela.

Os nomes das variáveis devem ser exatamente iguais na definição e no seu uso, isso inclui letras maiúsculas e minúsculas. Por exemplo, a variável `"a"`, é diferente da variável `"A"`.
Por exemplo, se inserir o código:

```
a = 5
b = 3
print(a + B)                    ***NÃO FUNCIONA!***
```

A resposta do Python será uma mensagem de erro: `NameError: name 'B' is not defined`.

Na Figura 4, o quadrado vermelho indica o local do botão para "rodar" o programa. E a seta vermelha indica o resultado da soma.
Neste exemplo, a função `print( )` mostra o resultado da soma. Sempre que esta função for utilizada, será mostrado o resultado, ou o que estiver escrito no código, nesta mesma janela, chamada de console.

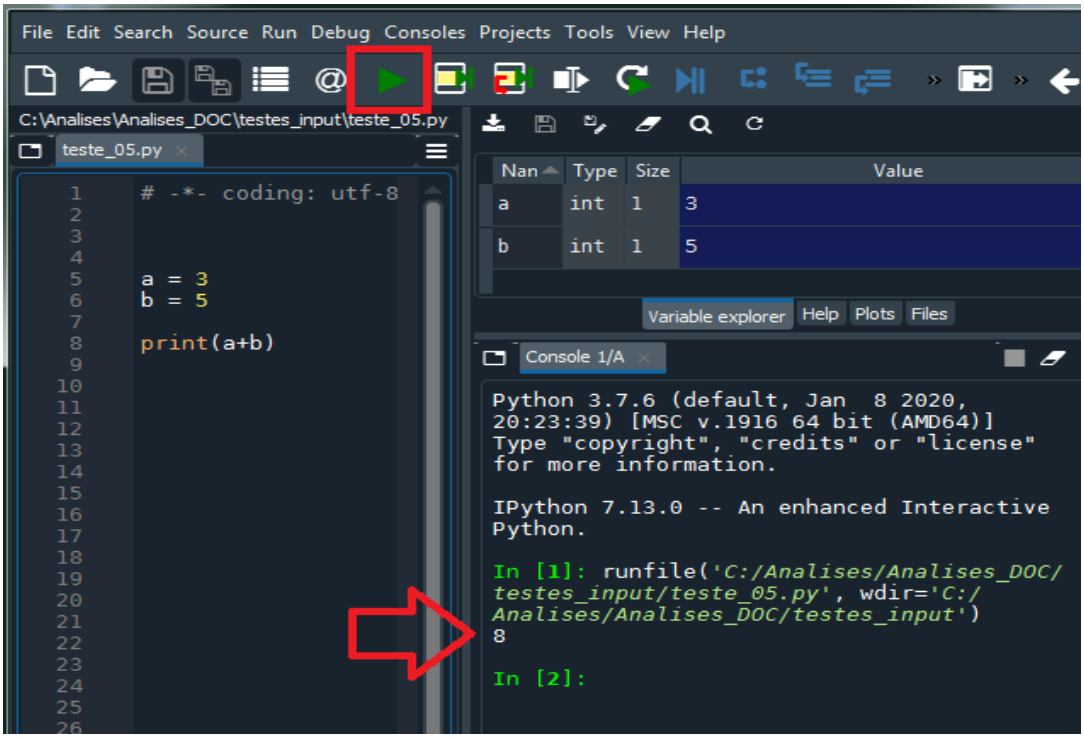

Figura 4

Exercício 1

Para se ambientar com a linguagem, fazer a soma de 3 valores:

a) Somar os valores 5, 56 e 7 diretamente dentro da função `print()`.

b) Somar os valores 5, 56 e 7 atribuindo-os a variáveis.

2.2 A função input()

Às vezes é interessante possibilitar que o usuário do programa insira alguma variável enquanto o programa é executado. Para isso existe a função `input()`. Vamos programar e executar o exemplo. Observe que variáveis alfanuméricas (em inglês: *String*) devem ser colocadas entre aspas:

```
print('Insira seu nome: ')
nome = input()
print('Olá, seja bem-vindo ' + nome)
```

Strings podem também ser multiplicadas. Execute o programa a seguir para ver o efeito:

```
print('Insira seu nome: ')
nome = input()
print('Olá, seja bem-vindo ' + 3*nome)
```

Nos exemplos acima o programa irá parar e esperar o usuário digitar o nome e apertar a tecla Enter. O texto digitado será armazenado em uma string denominada `nome`.
Se for de interesse fazer operações numéricas com a variável inserida com uso da função `input()`, é necessário converter a variável String para um número inteiro (função `int()`, ou decimal (função `float()`), veja abaixo.

Para entender a diferença entre uma String e número, execute os dois códigos abaixo:

```
print('Insira um número: ')
num = input()
print(3*num)
```

Entrando com '12', o retorno do programa será
`121212`
A entrada '12' foi tratada como uma string, e não como um número.

Transformando a String para um número através da função `int()` [ou `float()`], obtemos

```
print('Insira um número: ')
num = input()
num = int(num)
print(3*num)
```

Entrando com '12', o retorno do programa será
`36 (com int) ou 36.0 (com float)`
e agora foi tratado como número.

Exercício 2

Fazer um programa que forneça a soma e o produto de dois valores inseridos pela função input, e então mostre o resultado na tela.

Possível resposta:

```
print('Insira um número: ')
a = float(input())      # Observe que o input() foi
                        # colocado dentro do float()
print('Insira mais um número: ')
b = float(input())
print('A soma é: ', a + b)
print('A multiplicação é: ', a * b)
```

2.3 Operadores numéricos

Além de algumas outras opções menos comuns, os operadores matemáticos básicos são (+) soma, (-) subtração, (*) multiplicação, (/) divisão, (**) exponenciação. Alguns exemplos.

```
print(125 + 450)
```
575

```
print(1.2 * 3)
```
3.6

```
print(2 ** 10)
```
1024

Os operadores % e // fornecem respectivamente o resto de uma divisão e a "divisão inteira" de dois valores.

```
a = 7
b = 3
print(a % b)
```
1
(Na divisão de 7 por 3, o resto é 1)

```
a = 7
b = 2
print(a // b)
```
3
(A divisão inteira de 7 por 2 resulta em 3)
**Importante observar que o separador decimal no Python é o ponto (e nunca a vírgula).

2.4 Comentários

Comentários são textos que não pertencem ao código, que não serão compilados e executados no programa. São úteis principalmente para leitura posterior do código, ou quando mais de uma pessoa trabalha no mesmo código. É um texto no corpo do código, que tem significado para o aluno, e é descartado pelo compilador.
Para utilizá-lo, basta colocar o caractere # em qualquer linha, da esquerda para a direita. Depois do caractere #, o Python não considerará a linha como parte do código do programa.

```
print('aqui jaz uma string')   # teste com comentário
aqui jaz uma string
```

Outra forma de marcar um bloco de texto como comentário é colocá-lo entre três aspas duplas, como por exemplo

```
""" (três aspas duplas no início do bloco de comentário)
Toda essa parte é comentário
Não será interpretada pelo compilador
@author: XXX
(... e três aspas duplas no fim) """
```

2.5 Lists (listas) – variáveis indexadas

É muito comum trabalhar com variáveis indexadas, variáveis que se repetem ao longo do tempo, por exemplo. Assim, podemos imaginar a temperatura média diária ao longo de um ano. Serão 365 valores. Esse tipo de informação é armazenado normalmente num objeto list (lista). É definida por colchetes e os valores são separados por vírgulas. Por exemplo, as temperaturas diárias da primeira semana de março poderiam ser guardadas como

```
Temp = [26.3, 28.2, 25.8, 25.1, 24.7, 25.0, 26.1]
```

Observe que os sete valores dessa lista são numerados pelo Python de 0 a 6. Assim, se eu quiser saber a temperatura do 1º dia (primeiro valor da lista), acrescento

```
print(Temp[0])
26.3
```

ou do 4º dia:

```
print(Temp[3])
25.1
```

Para o último valor podemos usar o índice -1:

```
print(Temp[-1])
26.1
```

ou para o penúltimo dia, -2:

```
print(Temp[-2])
25.0
```

O exemplo acima é uma lista em uma dimensão (um índice), mas podemos criar listas de duas ou mais dimensões. Por exemplo os dados diários de temperatura durante 10 anos compões uma lista de duas dimensões, os dez anos (0 a 9) e os 365 dias (0 a 364).
Como exemplo um exemplo de uma lista em duas dimensões "3 por 3" de alguns valores:

```
lista2D = [[1, 8, 2], [3, 4, 5], [7, 7, 2]]
```

Por questão de clareza, uma maneira usual de expressarmos a mesma lista "3 por 3" que o Python nos permite, é separando as linhas da lista, escrevendo-a como:

```
lista2D = [[1, 8, 2],
           [3, 4, 5],
           [7, 7, 2]]
```

Nesse caso, observe a resposta quando segue o comando:

```
print(lista2D[1])
[3,4,5]
```

E seu eu quiser imprimir somente o primeiro valor da segunda linha ('3')?

```
print(lista2D[1][0])
3
```

2.6 Trabalhar com arquivos (Files)

Os objetos *files* são a principal interface com arquivos externos ao Python. Podem ser utilizados para ler e escrever dados ou resultados de uma simulação, por exemplo. Para criar ou alterar um arquivo, será utilizado a função `open()`. A estrutura da função é:

`open(`*`nome do arquivo , modo`*`)`

onde o *nome do arquivo* é o nome do arquivo que será criado ou aberto (na mesma pasta que o arquivo .py), e o *modo* é o modo que o arquivo será trabalhado, ou seja, se o arquivo será lido, escrito ou alterado.
Para utilização do arquivo em local/pasta diferente do arquivo .py, deve-se colocar o caminho completo do arquivo. Segue um exemplo para gravar um arquivo a partir do Python:

```
f = open('data.txt', 'w')          # abra/cria arquivo nomeado data.txt
f.write('Esta é a primeira linha\n')   # insere string no arquivo
                                    # seguida de uma quebra de linha (\n)
f.write('Esta é a segunda linha do arquivo. ')
f.write('Ainda é a segunda linha')   # pois não tinha \n na anterior.
f.close()                         # é necessário para fechar o arquivo
```

Nesse exemplo, será criado ou sobrescrito um arquivo com nome data.txt. Na função `open()`, o modo `'w'`, (do inglês *write*) no significa que o Python irá criar o arquivo, na mesma pasta que o arquivo *.py* está localizado.

Toda vez que se abre um arquivo, é necessário fecha-lo com a função `close()`.
A função `write()` insere uma string no arquivo `f`. Para mudar para uma próxima linha, deve se terminar o string com \n.

Para ler um arquivo existente sem alterá-lo, utiliza-se o modo `'r'` (do inglês *read*), na função `open()`.
A função `read()` coloca o arquivo todo em uma string. No exemplo a seguir, será aberto o arquivo criado no exemplo acima.

```
f = open('data.txt', 'r')        # o 'r' é para ler o arquivo apenas
texto = f.read()               # coloca o arquivo inteiro em uma string
f.close()

print(texto)
```
```
Esta é a primeira linha
Esta é a segunda linha do arquivo. Ainda é a segunda linha
```

A última linha do código digitado acima, com a função `print(texto)`, mostra o conteúdo do arquivo lido.
Observe que ao abrir um arquivo com a opção `'w'`, um arquivo eventualmente existente com o mesmo nome será apagado sem confirmação. Alternativamente, a opção `'a'` (do inglês *append*) na função `open('file.txt', 'a')`, permite abrir e acrescentar dados ao final do arquivo existente.
Há opções além das apresentadas aqui para se trabalhar com arquivos, mas não pertencem ao escopo do curso. Exemplo importante são os arquivos de bytes binários e arquivos de texto *non-ASCII Unicode*.

Exercício 3

O arquivo de texto Dados.prn (fornecido) possui dados diários contendo temperatura (em ^{o}C) e precipitação (em mm), na primeira e na segunda coluna respectivamente.

Copiar o arquivo para dentro da pasta C:\Python_Curso
Fazer um programa para abrir e mostrar no console o conteúdo do arquivo Dados.prn.

Possível resposta:

```
f = open('C:\Python_Curso\Dados.prn' , 'r')
a = f.read()
f.close()
print(a)
```

3 Loops e condições: for, if else, while.

3.1 Loop: for

O *loop `for`* é o iterável padrão no Python.

Como exemplo

```
for i in range(3):
    print(i)            # statement 1
print('terminou o Loop')   # statement 2
0
1
2
terminou o Loop
```

O loop `for` executa o `statement 1` um número determinado de vezes. No exemplo acima, é repetido três vezes por causa da função `range(3)`.
A função `range()` vai varrer o valor inteiro dentro do parêntese, iniciando-se do zero. Neste exemplo será executado o comando três vezes. A variável `i` começa do zero, este é um padrão em Python.
Para o exemplo funcionar é necessário *indentar* os statement. Indentar é afastar o texto da margem, esta ferramenta é a maneira que o Python entende que os comandos não terminaram. Ou seja, se eu colocar algum código sem indentação, o Python vai considerá-lo fora do loop.

```
for i in range(2):
    print('iteração nº:', i)
    print('OLÁ')
print('terminou')
iteração nº: 0
OLÁ
iteração nº: 1
OLÁ
terminou
```

Os objetos, como os produzidos pela função `range()`, são objetos denominados iteráveis.
É importante entender como o Python interpreta isso.
Por exemplo, strings são iteráveis:

```
for i in 'Quirijn':        # para aprender a soletrar o nome do professor
    print(i, end = '-')    # statement
Q-u-i-r-i-j-n-
```

Listas são iteráveis também:

```
lis = [5, 23, 'coisa', 1, 'fim']
for i in lis:
    print(i)
```

```
5
23
coisa
1
fim
```

3.2 A função `range()`.

A função `range()` pode ser utilizada de maneira diferente da descrita. As vezes se deseja iniciar um loop em um valor inteiro sem ser o zero, para isso basta colocar o valor como no exemplo:

```
for i in range(1, 10):
    print(i)
```

O código acima irá mostrar no console valores de 1 até 9, o passo é 1 por padrão (uma unidade de incremento a cada iteração). Ao chegar no valor 10, o loop é interrompido sem executar o comando.

```
for i in range(-2, 10):
    print(i, end = ' ')
```

```
-2 -1 0 1 2 3 4 5 6 7 8 9
```

No código acima, será iniciado o loop com valor -2 indo até 9. O loop irá executar o comando 12 vezes (do -2 até 9).

Outro exemplo, agora com *passo* 2.

```
for i in range(0, 10, 2):
    print(i, end = ' ')
```

```
0 2 4 6 8
```

```
for i in range(1, 10, 2):
    print(i, end = ' ')
```

```
1 3 5 7 9
```

Como esperado foram obtidos valores do início até o valor 10 (sem incluí-lo).

3.3 Condição: if else

O comando “if” permite a execução condicional de parte de um algoritmo. Veja como ele funciona em função de alguns exemplos:

```
if 3 < 8.5:
    print('é verdadeiro')
```

```
é verdadeiro
```

Outro exemplo, incluindo “else”:

```
if 5 == 6:        # test1
    print('é verdadeiro')  # statement1
else:
    print('o test1 é falso')
o test1 é falso
```

No exemplo acima, foi verificado que o valor 5 é diferente de 6, no `test1`, então não executou o `statement1`, acionando o `statement` presente em `else`.

3.4 Loop: While

O *loop `while`* executa um determinado `statement` enquanto um teste seja avaliado como verdadeiro. Segue um exemplo:

```
i = 0
while 3 > i:
    print('o valor de i é:', i)
    i = i+1
print('Terminei, o valor de i é:', i) #comando não-indentado, fora do loop!
o valor de i é: 0
o valor de i é: 1
o valor de i é: 2
Terminei, o valor de i é: 3
```

Inicialmente o valor de `i` é zero, então a cada loop é somado 1 ao valor de `i,` até que o valor de `i` se torna igual a 3, saindo do loop.

3.5 Exemplo de *arquivo* para *list*.

Durante o curso serão utilizados arquivos de texto como dados de entrada (por exemplo, um arquivo com dados meteorológicos, outro da cultura, do solo). Ao utilizar a função `read()`, mencionada no tópico 7, o conteúdo do arquivo será lido como uma única string, onde as quebras de linhas, são representadas pelo código \n. Para separá-lo em linhas, pode-se utilizar a função `splitlines()`, como em: *`string`*`.splitlines()`.
Se necessário separar cada linha em uma nova lista, pode-se utilizar a função `split()`.

Para exemplificar, segue o exemplo onde o arquivo de texto contendo:

```
30.0   2.2
35.0   50
20.5   33.3
```

Será transformado em uma lista com valores numéricos *float*.
Para ler e transformar o arquivo em uma lista de strings:

```
f = open(nome do arquivo , 'r')
file = f.read()               # importa o arquivo inteiro numa string file
f.close()
list_str = file.splitlines()  # separa o string file por linhas
```

Nesta etapa a lista list_str será composta por strings, cada uma representando uma linha do arquivo original, ou seja:

```
['30.0   2.2', '35.0   50', '20.5   33.3']
```

Na próxima etapa, cada string da lista é transformado em uma nova lista interna contendo valores numéricos, utilizando a função `split()`. A função `split(sep)` separa uma string em uma lista, onde a string *sep* é o separador considerado, por exemplo:

```
a = list_str[0].split('   ')   # valores separados por três espaços
print(a)
['30.0', '2.2']
```

No exemplo acima, a primeira linha foi separada por 3 espaços, que é o formato da lista fornecida. Agora é necessário fazer um loop para repetir o processo para todas as linhas do arquivo original.

```
list_str2 = []
for i in range(len(list_str)):
    a = list_str[i].split('   ')
    b = [float(a[0]), float(a[1])]
    list_str2.append(b)

print(list_str2)
[[30.0, 2.2], [35.0, 50.0], [20.5, 33.3]]
```

Exercício 4

Seguindo o exercício **3**, após importar os dados do arquivo Dados.prn, transformar os dados em formato de lista contendo números do tipo *float*.

Possível resposta:

```
f = open('C:\Python_Curso\Dados.prn', 'r')
file = f.read()                    # lê o conteúdo inteiro do arquivo
f.close()

list_str = file.splitlines()           # separa por linha

list_val=[]   # lista vazia para acrescentar os valores
for i in range(len(list_str)):                                    # (1)
    list_spl = list_str[i].split('   ')                           # (2)
    list_flo = [float(list_spl[0]), float(list_spl[1])]           # (3)
    list_val.append(list_flo)                                     # (4)
print(list_val)
```

No exercício **4**, a linha com o marcador (1), é onde foi definido o loop `for`. O código indentado após está linha será executado um determinado número de vezes, neste caso uma vez para cada item na lista `list_str`, o valor de `i` começa em zero e vai aumentado em uma unidade a cada ciclo.
Na linha com o marcador (2) é acessada a string na lista `list_str[i]`, e esta string é separada em outras strings onde houver `'   '`. É criado uma nova lista, `list_spl`, com as estas strings separadas.
Na linha com o marcador (3), é transformada as strings em número do tipo float. E então os valores são adicionados a uma nova lista `list_flo`, neste caso, com dois valores.
E na linha com marcador (4), é adicionado ao final da lista inicialmente vazia, a lista `list_val`. Ou seja, será obtido uma lista com listas de pares de números.

Exercício 5

Utilize o arquivo Dados .prn para:

a) Fornecer a temperatura média e a soma das precipitações (lembrando que o arquivo possui dados diários de temperatura [^{o}C] e precipitação [mm]).

b) Fornecer a temperatura média dos dias em que chove pelo menos 1,0 mm.

c) Criar outro arquivo com as temperaturas em Fahrenheit e precipitações diárias em centímetro.

Possível resposta do exercício 5:

a)

```
temp = 0   # necessário definir a variável antes de usá-la
prec = 0
for i in range(len(list_val)):
    temp = temp + list_val[i][0]
    prec = prec + list_val[i][1]
temp_med1 = temp / len(list_val)
print(round(temp_med1, 5), round(prec, 5))
```

Na resposta acima, foi utilizado a lista `list_val` do exercício **4**. A função `round(`*`valor, decimais`*`)` arredonda o *`valor`* para um determinado número de casas *`decimais`*.
O uso desta função é interessante pois os objetos do tipo *float* possuem um número limitado (em torno de 16 casas decimais) de precisão.
A linha com o código:
`for i in range(len(list_val)):`
inicia o loop que será executado uma vez para cada objeto presente na lista.

b)

```
temp2 = 0
cont = 0
for i in range(len(list(list_val))):
    if list_val[i][1] >= 1:   # verifica se a precipitação é maior que 1
        temp2 = temp2 + list_val[i][0]
        cont = cont + 1
temp_med2 = round(temp2 / cont, 5)
print(cont, temp_med2)   # imprime o número de cias e a temperature média
```

c)

```
lista_mod = []    # lista vazia
for i in range(len(list_val)):
    tf = list_val[i][0] * (9/5) + 32  # formula para conversão C=>F
    p_cm = list_val[i][1] / 10        # conversão mm em cm
    lista_mod.append([round(tf, 5), round(p_cm, 5)])
f2 = open('C:\Python_Curso\SAIDA.txt', 'w')
for i in range(len(lista_mod)):
    linha_str = str(lista_mod[i][0]) + '   ' + str(lista_mod[i][1]) + "\n"
    f2.write(linha_str)
f2.close()
```

Exercício 6 – Desafio (OPCIONAL)

Organizar os dados diários do arquivo Dados.prn em ordem crescente de temperatura, e gerar arquivo de saída com os dados em ordem.

4 Funções especiais e o comando import.

As classes e funções incorporadas no Python muitas vezes não são suficientes para a criação de um programa. Nesses casos utiliza-se o comando `import` para importar funções, bibliotecas e mesmo outros arquivos.
Uma biblioteca útil que vem incorporada no Python, mas sendo necessário carregá-la para utilizá-la, é a biblioteca *`math`*, que fornece algumas funções matemáticas úteis, como:

Função	Código
Seno	`math.sin(x)`
Cosseno	`math.cos(x)`
Tangente	`math.tan(x)`
Exponencial	`math.exp(x)`
Logaritmo	`math.log(x, base)`

Além de algumas constantes como: π (`math.pi`) e e (`math.e`).
É necessário primeiro importar a biblioteca `math`, normalmente no início do programa.

```
import math
```

Depois as funções e constantes ficam disponíveis.
Exemplos:

```
print(math.e)             # retorna o valor de e (base log natural)
print(math.pi)            # retorna o valor de pi
2.718281828459045
3.141592653589793

print(math.exp(3))             # retorna o valor de e elevado a 3 [exp(3)]
20.085536923187668

z = math.log(2, 10)    # retorna o log à base 10 de 2
print(z)
0.30102999566398114

z = math.log(2)    # retorna o log à base e (o ln) de 2
print(z)
0.6931471805599453

r = 2.5
área = math.pi * r**2    # calcula a área do círculo
print(área)
print(round(área,3))     # imprime o mesmo valor arredondado em 3 casas
19.634954084936208
19.635
```

Exercício 7

Escreva um programa que calcula

a) A área e o perímetro de um círculo de raio R ($2\pi R$ e πR^2)
b) A área e o volume de uma esfera de raio R ($4\pi R^2$ e $[4\pi R^3]/3$)

Possível resposta do exercício 7:

a)
```
import math

print('Insira o raio: ')
raio = float(input())
area = round(math.pi*raio**2., 3)
perim = round(2 * math.pi * raio, 3)
print(area, perim)
```

b)
```
import math

print('Insira o raio: ')
raio = float(input())
area_esf = round(4 * math.pi * raio**2, 3)
vol_esf = round(4/3 * math.pi * raio**3, 3)
print(area_esf, vol_esf)
```

Exercício 8

Uma função muito simples que pode ser usada para descrever o crescimento de uma cultura é a *função logística*, dada por

$$Y = \frac{Y_{\max}}{1 + e^{-k\left(t - \frac{t_c}{2}\right)}}$$

Nessa equação, Y (kg/ha) é a biomassa em função do tempo t (d), Y_{max} (kg/ha) é o rendimento no fim do ciclo e t_c (d) é a duração do ciclo. A constante k (d^{-1}) determina a inclinação da curva de crescimento.

Escreva um programa que solicita ao usuário entrar com os valores de Y_{max}, k e t_c e calcula os valores diários de Y ao longo do ciclo, gravando o resultado num arquivo de saída e mostrando também na tela.

Possível resposta do exercício 8:

```
import math     # para disponibilizar as funções matemáticas

ymax = float(input('Insira o valor de Ymax [kg/ha]: '))  # (1)
k = float(input('Insira o valor de k (0.05 < k < 0.1): '))
tc = float(input('Insira o valor de tc [dias]: '))

f3 = open('C:\Python_Curso\SAIDA2.txt', 'w')
f3.write('t[dias]   Y[kg/ha]\n')
for t in range(int(tc + 1)):     # (2)
    y = ymax / (1 + math.exp(-k*(t - (tc/2))))     # (3)
    print('Para t = ' + str(t) + ', Y =', round(y, 1))     #(4)
    f3.write(str(t) + '    ' + str(round(y, 1)) + '\n')    # (5)
f3.close()
```

Observações:

No início do programa inserem-se as variáveis. Dentro dos parênteses da função `input(`*`string`*`)`, pode-se colocar um texto que é exibida ao executar o programa, como é visto na linha com o marcador `(1)`. Isso é útil para evitar o uso da função `print()`, deixando o código menor.

Na linha com o marcador `(2)` foi feito um loop `for` que será executado até atingir `tc` (utiliza-se tc+1 pois o loop termina 1 antes do fim). O uso da função `int()` é importante pois a função `range()` admite apenas números inteiros.

Na linha com o (3), é inserida a função fornecida no enunciado do problema. Note que foi utilizado a variável `t` como contador.

Na linha com o (4), é mostrado no monitor (console) a string na forma:

`Para t =` *`valor`*`, Y =` *`valor`*

onde o valor de y é arredondado com duas casas decimais.

Na linha (5), os valores são exportados para o arquivo SAIDA2.txt (identificado como f3)