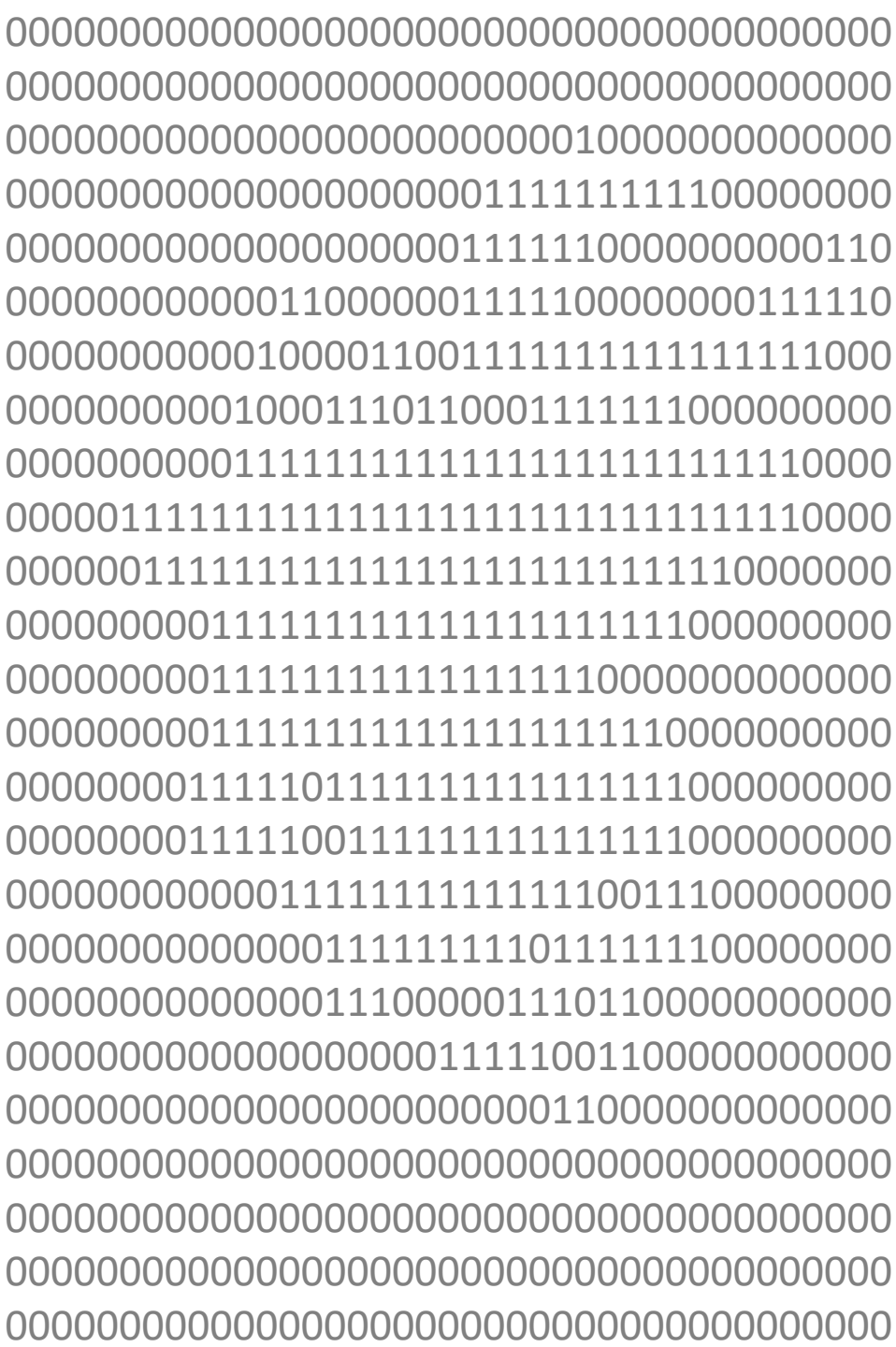


azevedolab.net

Introdução à Física Computacional

Aula 04

Implemente em Python as seguintes equações da cinemática. O programa mostrará o resultado (variável dependente à esquerda) a partir de dados de entrada (variáveis independentes à esquerda). Para testar o seu código, use valores para os quais você calculou a resposta.

Equação
$v = v_0 + at$
$x = x_0 + v_0 t + \frac{1}{2} a t^2$
$v^2 = v_0^2 + 2a(x - x_0)$
$x = x_0 + \frac{1}{2}(v + v_0)t$
$x = x_0 + vt - \frac{1}{2} a t^2$

Muitas vezes, para desenvolvermos códigos de interesse da Física Computacional, temos que fazer uso de métodos matemáticos disponíveis no módulo *math* da linguagem Python. Para o uso desses métodos matemáticos, precisamos chamar o módulo *math*, com o comando *import* indicado abaixo.

```
import math
```

A linha acima tem que aparecer antes de qualquer uso dos métodos matemáticos do módulo *math*. A descrição completa das funções disponíveis módulo *math* podem ser acessadas em: < https://docs.python.org/3/library/math.html >.

A seguir temos a lista das principais funções matemáticas do módulo *math*.

Funções matemáticas	Funções do módulo math	Exemplos de implementação
Arco-seno, arco-cosseno, arco-tangente	*math.asin(x), math.acos(x), math.atan(x)*	*function1 = math.asin(x)* *function2 = math.acos(x)* *function3 = math.atan(x)*
Funções trigonométricas hiperbólicas	*math.sinh(x), math.cosh(x), math.tanh(x)*	*function1 = math.sinh(x)* *function2 = math.cosh(x)* *function3 = math.tanh(x)*
Inversas das funções trigonométricas hiperbólicas	*math.asinh(x), math.acosh(x), math.atanh(x)*	*function1 = math.asinh(x)* *function2 = math.acosh(x)* *function3 = math.atanh(x)*
Número de Euler	*math.e*	*function1 = math.e*
Potência na base e	*math.exp(x)*	*function1 = math.exp(x)*
Logaritmo na base y	*math.log(x,y)*	*function1 = math.log(x,y)*

A linguagem de programação Python tem um arsenal de métodos para manipulação de strings. O conceito de string é simples. **Uma string é um conjunto de caracteres, que pode conter números, letras e símbolos especiais.** Toda leitura de informação via teclado (função *input()*) considera que o que está sendo digitado é uma string. Como vimos, para que o programa considere a informação digitada, como um número inteiro, temos que usar a função *int(),* antes da função *input()*, que converte a string para inteiro, como indicado abaixo.

```
my_var = int(input("Type an integer =>"))
```

Vimos, também, que podemos usar uma entrada de dados de ponto flutuante, a partir da função *float()*, que converte de string, ou de inteiro, para *float*, como indicado abaixo.

```
my_var = float(input("Type a float =>"))
```

Quando **não** indicamos as funções *float()* ou *int()*, antes da função *input()*, o interpretador Python assume que a informação sendo digitada é uma string, como indicado no trecho de código a seguir.

```
my_var = input("Type a string =>")
```

Variáveis em Python fazem referencia às posições na memória, onde estão armazenados dados, como no exemplo das strings, indicadas abaixo.

```
my_var1 = "Here we have a string"
my_var2 = "Here we have a new string with 1 e 2"
my_var3 = "112358132134455"
```

Usamos aspas duplas para indicar a string em Python, podemos usar, também, aspas simples, **a única exigência é que, uma vez iniciado com um tipo de aspa, esta deve estar no final**, como no exemplo abaixo.

```
my_var = input('Type a string =>')
```

As variáveis podem ser mostradas na tela com a função *print()*, como no exemplo a seguir.

```
my_var = "A new string"
print(my_var)
```

Podemos usar a função *print()* para mostrar resultados compostos na tela, onde strings, inteiros, *floats* e sequências de escape aparecem numa única linha, o código a seguir, *show_strings1.py,* ilustra a situação.

```
my_var1 = "My string"
my_var2 = 112358
my_var3 = 3.14159
print("String: ",my_var1,", Integer: ",my_var2,", Float: ",my_var3,"\n\n")
```

Ao executarmos o código *show_strings1.py*, temos o seguinte resultado.

```
String:  My string , Integer:  112358 , Float:  3.14159
```

Vamos olhar alguns detalhes importantes do código *show_strings1.py*. A primeira linha traz a atribuição do conteúdo da memória *"My string"* à variável *my_var1*. Usamos aspas duplas, poderíamos ter usado aspas simples, o resultado seria o mesmo. A segunda linha traz a variável *my_var2*, veja que não usamos aspas, o interpretador Python considera a variável *my_var2* como inteira, sem necessidade que explicitemos com a função *int()*. Para a variável *my_var3*, temos situação similar, só que agora a presença do ponto decimal indica que o dado é do tipo *float.* A função *print()* traz conteúdo entre aspas, as variáveis e a sequência de escape *\n*. Veja que separamos por vírgulas os conteúdos a serem mostrados na tela. Se omitirmos a vírgula, que separa, por exemplo, o primeiro conteúdo entre aspas e a variável *my_var1*, teremos um **erro de execução** do programa, também chamado de **erro de sintaxe**.

```
my_var1 = "My string"
my_var2 = 112358
my_var3 = 3.14159
print("String: ",my_var1,", Integer: ",my_var2,", Float: ",my_var3,"\n\n")
```

Veja, ainda, que as vírgulas inseridas entre aspas são caracteres de uma string, por isso são mostradas na tela.

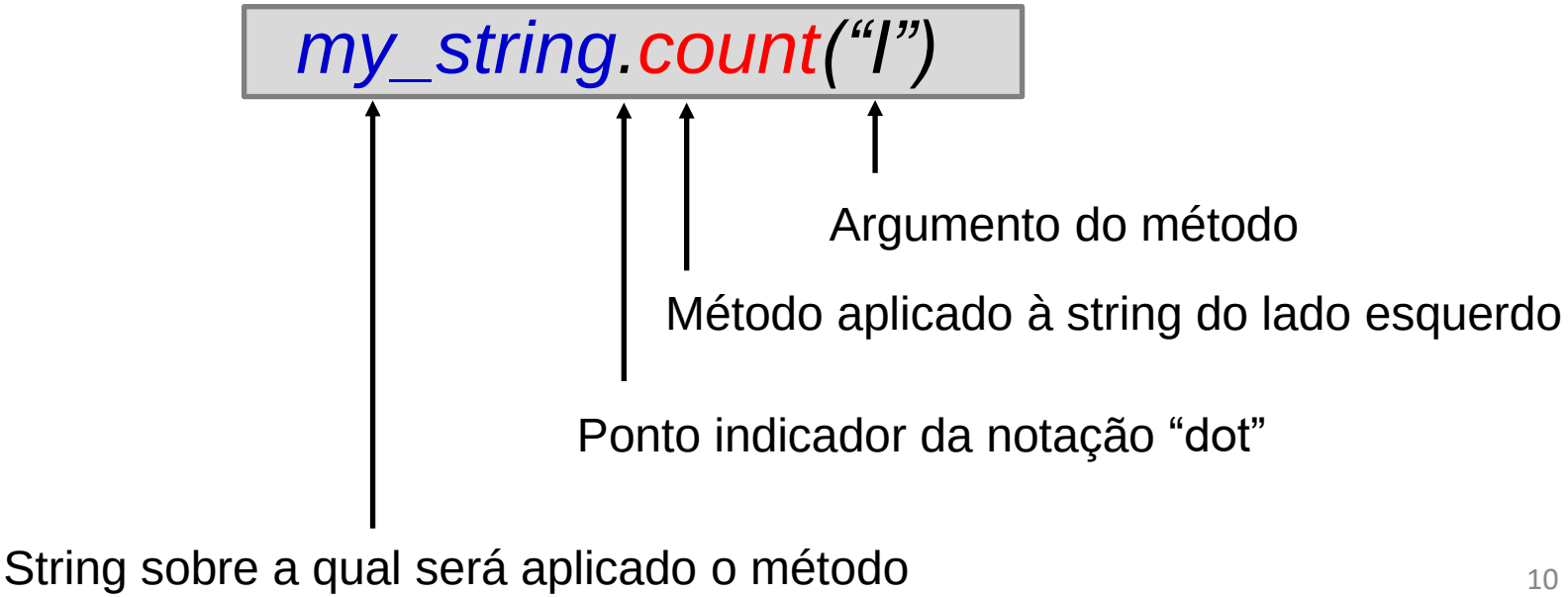
Vimos o uso de funções internas em Python (*Python's built-in functions*) para manipulação de strings, como as funções *print(), input(), float()* e *int().* Temos, também, duas funções adicionais para strings, são elas as funções *len()* e *str()*. A função *len()* retorna o tamanho de uma string que é dada como argumento, ou seja, o número de caracteres da string. A função *str()* converte o argumento para string, por exemplo, de inteiro para string. Vejamos um exemplo simples onde as duas funções aparecem. O código *show_strings2.py* converte um inteiro para string e, depois, mostra o número de caracteres da string.

```
my_int = int(input("Type an integer => "))        # Reads an integer and assigns it to my_int
my_string = str(my_int)                           # Converts to string
count_char = len(my_string)                       # Counts characters in my_string
print("Number of characters of ",my_string," is ",count_char) # Shows results
```

Ao executarmos o código acima, temos o resultado mostrado abaixo.

```
Type an integer => 12345
Number of characters of  12345  is  5
```

Como vimos, há diversos tipos de valores que podem ser usados como argumentos das funções internas da linguagem Python. Contudo, a maioria das funções, aplica-se a um tipo específico de dados, por exemplo, às strings. Tais **recursos da linguagem Python específicos para um tipo de dado são chamados de métodos**. A chamada de um método em Python, ocorre de forma similar às funções, exceto que o primeiro argumento aparece antes do nome do método, seguido por um ponto (.). Essa forma de chamar um método é denominada de **notação *dot***. Vejamos, como exemplo, o método *.count()*, que retorna o número de vezes que uma dada string, fornecida como argumento, aparece na string, *my_string*.



O programa *show_strings3.py* conta o número de vezes que o caractere “I” aparece na string atribuída à variável *my_string*.

```
my_string = input("Type a string => ") # Reads a string and assigns it to variable my_string
# .count("I") method to count the number of times its argument appears in the string
count_I = my_string.count("I")
print("I appears ",count_I, "time(s)") # Shows results
```

Ao executarmos o código, temos o resultado abaixo.

```
Type a string => PHYSICS
I appears  1 time(s)
```

Uma lista tem a vantagem de apresentar a informação de forma indexada, cada trecho da informação é chamado elemento da lista. Por exemplo, podemos criar uma lista com os 10 primeiros elementos da sequência de Fibonacci, como indicado abaixo.

```
my_list = [1,1,2,3,5,8,13,21,34,55]
```

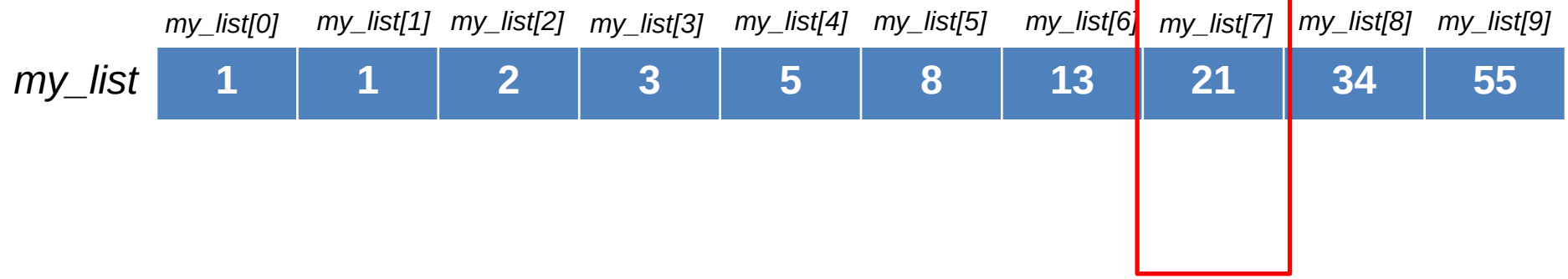
Para nos referirmos diretamente a um elemento da lista, chamamos a ordem do elemento, tendo em mente que em Python usamos o índice “0” para o primeiro elemento. Assim, a lista *my_list* tem a seguinte distribuição.

	my_list[0]	*my_list[1]*	*my_list[2]*	*my_list[3]*	*my_list[4]*	*my_list[5]*	*my_list[6]*	*my_list[7]*	*my_list[8]*	*my_list[9]*
my_list	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**

O trecho de código abaixo cria a lista *my_list* e imprime o elemento 7 da lista, ou seja, o oitavo elemento da lista.

```
my_list = [1,1,2,3,5,8,13,21,34,55]
print(my_list[7])
```

O resultado das linhas de código acima, cria a lista, e imprime o elemento “7” da lista, indicado no diagrama abaixo.



A função para abertura de arquivos em Python é *open()*. Vamos destacar abaixo as principais características da função *open()* para leitura de arquivos pré-existentes. Usamos a linha de comando abaixo para leitura do arquivo *my_file*, como segue:

```
my_info = open(my_file,'r')
```

my_file é uma variável para a qual foi atribuída o nome do arquivo, a informação *'r'*, é chamada de **modo de acesso ao arquivo**, no caso indica que será realizada uma operação de leitura do conteúdo do arquivo *my_file*. A informação contida no arquivo de entrada será referenciada à variável *my_info*. Resumindo, usaremos a variável *my_info* para manipularmos o conteúdo lido do arquivo *my_file*. Por exemplo, consideremos que a informação do nosso arquivo de entrada está contida numa linha de texto. Para lermos essa informação, usamos o comando,

```
my_line = my_info.readline()
```

.readline() é um método que lerá uma linha da variável *my_info* e atribuirá à variável *my_line.* O método *.readline()* é aplicado à variável *my_info.*

Leitura de Arquivos

Antes de irmos para um programa completo, vejamos um programa simples, que só lê o conteúdo de um arquivo e mostra o resultado na tela, o programa *read_file_and_show1.py*. A primeira linha em vermelho indica a leitura do nome do arquivo de entrada. Nesta versão, não estamos nos precavendo da possibilidade do usuário digitar um nome de arquivo que não existe. Veremos, mais adiante no curso, como prevenir tais situações. A segunda linha vermelha, traz a abertura do arquivo, bem como a atribuição do conteúdo à variável *my_info*, como mostrado abaixo. A opção *'r'* indica que realizaremos a leitura do arquivo.

```
# Program to demonstrate open() function in Python
my_file = input("Enter file name = >")

# Opens input file
my_info = open(my_file,'r')

# Reads one line of the input file
my_line = my_info.readline()

# Closes file
my_info.close()

# Shows file content on screen
print(my_line)
```

A terceira linha em vermelho traz a atribuição do conteúdo à variável *my_line*. Tal passagem pode parecer desnecessária, mas veja que à variável *my_info* foi referenciado o conteúdo do arquivo, mas para manipularmos o conteúdo, temos que atribuí-lo a uma nova variável, no caso a *my_line*. O método usado é o *.readline()*, que é aplicado à variável *my_info*. Uma vez que o conteúdo foi atribuído a uma variável, podemos fechar o arquivo, com o método *.close()*, aplicado ao *my_info*. Por último, mostramos o conteúdo atribuído à variável *my_line* na tela, com a função *print().* Algumas linguagens, como Perl, chamam variáveis como *my_info* de *filehandle*.

```
# Program to demonstrate open() function in Python
my_file = input("Enter file name = >")

# Opens input file
my_info = open(my_file,'r')

# Reads one line of the input file
my_line = my_info.readline()

# Closes file
my_info.close()

# Shows file content on screen
print(my_line)
```

Leitura de Arquivos

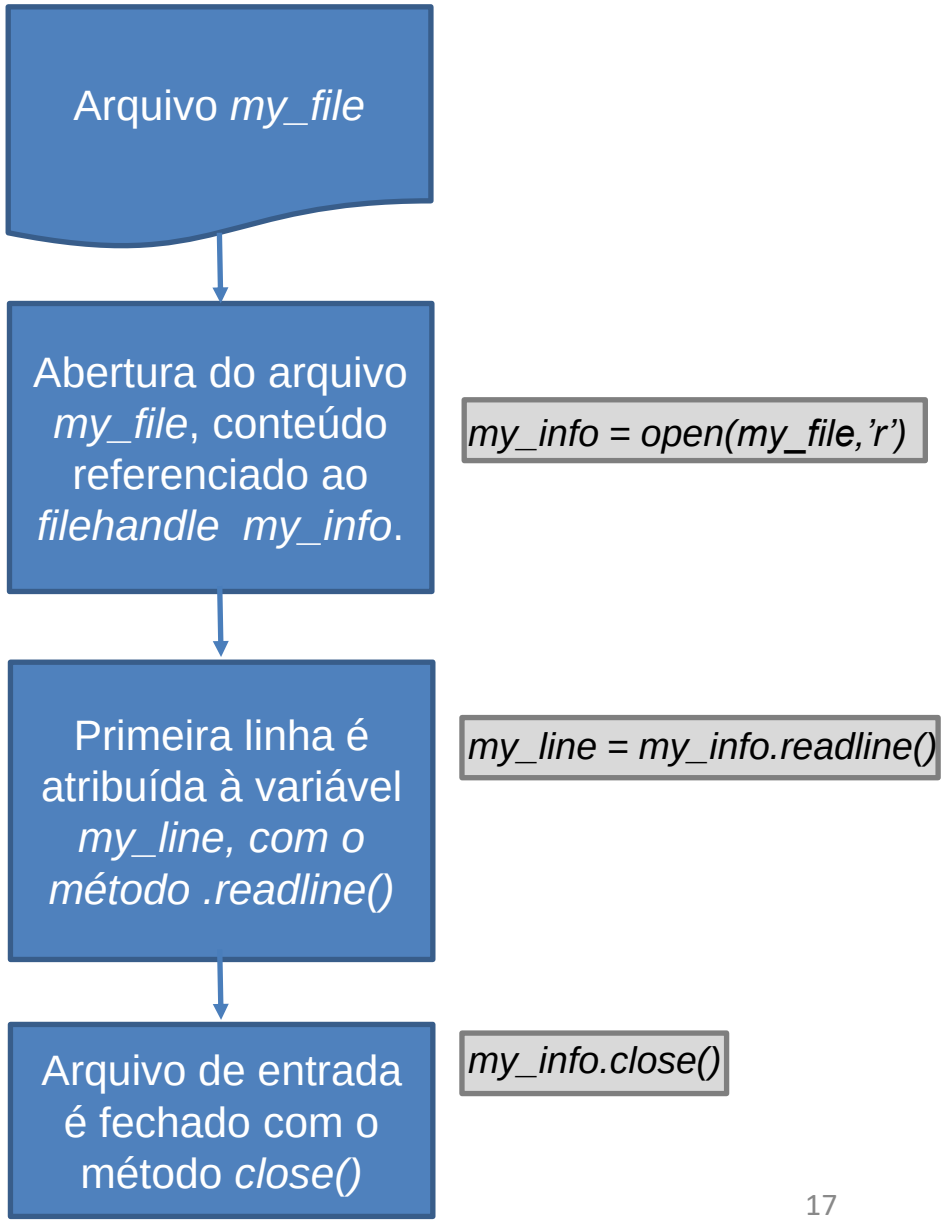
A partir do *filehandle my_info,* podemos manipular o arquivo de entrada, contudo, para acessar seu conteúdo de forma específica, temos que atribuí-lo a uma variável, como a *my_line* do programa. O fluxograma ao lado ilustra os principais conceitos na abertura e leitura de um arquivo com Python. Inicialmente temos o arquivo *my_file*, que será aberto com a função *open()* e seu conteúdo referenciado ao *filehandle my_info*. A linha de comando para a tarefa, está indicada abaixo:

my_info = open(my_file, 'r')

A seguir, a primeira linha do arquivo, via *filehandle* é atribuída à variável *my_line,* com a aplicação do método *.readline(),*

my_line = my_info.readline()

Podemos fechar o arquivo como *my_info.close().*



Rodaremos o programa *read_file_and_show1.py* e usaremos o arquivo *file1.txt* como entrada. O resultado está mostrado abaixo.

```
Enter file name => file1.txt
Let the light of science end the darkness of denialism.
```

Método Associados à Manipulação de Strings

Há diversos métodos em Python para manipulação de strings, o código *methods4strings1.py* traz alguns deles aplicados à string “GATTACA”. Os próximos slides destacam em vermelho o trecho de código discutido.

```
# Program to handle strings, using string methods available in Python 3 (part 1)
seqIn = "GATTACA"           # Initial sequence
print("\nInitial sequence:\t\t",seqIn)
# s.lower() method to generate lowercase characters for string s
seqOut = seqIn.lower()
print("\nSequence in lowercase:\t\t",seqOut)
# s.upper() method to generate uppercase characters for string s
seqOut = seqOut.upper()
print("\nSequence in uppercase:\t\t",seqOut)
# len(s) method to show the length of the string s
countBases = len(seqOut)    # len(seqOut) is assigned to countBases
print("\nNumber of bases:\t\t",countBases)
# s.count("X") method to count substring "X" in string s, used to calculate percentage of CG
countC = seqOut.count("C")  # Number of substrings "C" is assigned to countC
countG = seqOut.count("G")  # Number of substrings "G" is assigned to countG
perCG = float(100*(countC+countG)/len(seqOut))  # Calculates percentage of C+G in the sequence
print("\nPorcentage of CG:\t\t",perCG)
# s.replace("X","Y") method to replace substring "X" for "Y" in the string s
seqOut = seqOut.replace("T","U")
print("\nSequence after replacing (t->U):",seqOut)
# s.find("X") method to return position of substring "X" in the string s
posA = seqOut.find("A")
print("\nPosition of substring A:\t",posA)
posUU = seqOut.find("UU")
print("\nPosition of substring UU:\t",posUU)
posU = seqOut.find("U")          # Returns the position of first "U" found in the string
print("\nPosition of first substring U:\t",posU)
```

Método Associados à Manipulação de Strings

O método s.*lower()* transforma a string *s* para letras minúsculas e o *s.upper()* para letras maiúsculas. Ambos resultados são mostrados na tela com a função *print()*.

```
# Program to handle strings, using string methods available in Python 3 (part 1)
seqIn = "GATTACA"           # Initial sequence
print("\nInitial sequence:\t\t",seqIn)
# s.lower() method to generate lowercase characters for string s
seqOut = seqIn.lower()
print("\nSequence in lowercase:\t\t",seqOut)
# s.upper() method to generate uppercase characters for string s
seqOut = seqOut.upper()
print("\nSequence in uppercase:\t\t",seqOut)
# len(s) method to show the length of the string s
countBases = len(seqOut)    # len(seqOut) is assigned to countBases
print("\nNumber of bases:\t\t",countBases)
# s.count("X") method to count substring "X" in string s, used to calculate percentage of CG
countC = seqOut.count("C")  # Number of substrings "C" is assigned to countC
countG = seqOut.count("G")  # Number of substrings "G" is assigned to countG
perCG = float(100*(countC+countG)/len(seqOut))  # Calculates percentage of C+G in the sequence
print("\nPorcentage of CG:\t\t",perCG)
# s.replace("X","Y") method to replace substring "X" for "Y" in the string s
seqOut = seqOut.replace("T","U")
print("\nSequence after replacing (t->U):",seqOut)
# s.find("X") method to return position of substring "X" in the string s
posA = seqOut.find("A")
print("\nPosition of substring A:\t",posA)
posUU = seqOut.find("UU")
print("\nPosition of substring UU:\t",posUU)
posU = seqOut.find("U")          # Returns the position of first "U" found in the string
print("\nPosition of first substring U:\t",posU)
```

Método Associados à Manipulação de Strings

O método *len(s)* mostra o número de caracteres na string *s* e o método *s.count("X")* conta o número de vezes que a substring "X" aparece na string *s*.

```
# Program to handle strings, using string methods available in Python 3 (part 1)
seqIn = "GATTACA"           # Initial sequence
print("\nInitial sequence:\t\t",seqIn)
# s.lower() method to generate lowercase characters for string s
seqOut = seqIn.lower()
print("\nSequence in lowercase:\t\t",seqOut)
# s.upper() method to generate uppercase characters for string s
seqOut = seqOut.upper()
print("\nSequence in uppercase:\t\t",seqOut)
# len(s) method to show the length of the string s
countBases = len(seqOut)    # len(seqOut) is assigned to countBases
print("\nNumber of bases:\t\t",countBases)
# s.count("X") method to count substring "X" in string s, used to calculate percentage of CG
countC = seqOut.count("C")  # Number of substrings "C" is assigned to countC
countG = seqOut.count("G")  # Number of substrings "G" is assigned to countG
perCG = float(100*(countC+countG)/len(seqOut))  # Calculates percentage of C+G in the sequence
print("\nPorcentage of CG:\t\t",perCG)
# s.replace("X","Y") method to replace substring "X" for "Y" in the string s
seqOut = seqOut.replace("T","U")
print("\nSequence after replacing (t->U):",seqOut)
# s.find("X") method to return position of substring "X" in the string s
posA = seqOut.find("A")
print("\nPosition of substring A:\t",posA)
posUU = seqOut.find("UU")
print("\nPosition of substring UU:\t",posUU)
posU = seqOut.find("U")          # Returns the position of first "U" found in the string
print("\nPosition of first substring U:\t",posU)
```

Método Associados à Manipulação de Strings

A partir do conteúdo atribuído às variáveis *countC* e *countG*, é calculada a porcentagem de C + G na sequência.

```
# Program to handle strings, using string methods available in Python 3 (part 1)
seqIn = "GATTACA"           # Initial sequence
print("\nInitial sequence:\t\t",seqIn)
# s.lower() method to generate lowercase characters for string s
seqOut = seqIn.lower()
print("\nSequence in lowercase:\t\t",seqOut)
# s.upper() method to generate uppercase characters for string s
seqOut = seqOut.upper()
print("\nSequence in uppercase:\t\t",seqOut)
# len(s) method to show the length of the string s
countBases = len(seqOut)    # len(seqOut) is assigned to countBases
print("\nNumber of bases:\t\t",countBases)
# s.count("X") method to count substring "X" in string s, used to calculate percentage of CG
countC = seqOut.count("C")  # Number of substrings "C" is assigned to countC
countG = seqOut.count("G")  # Number of substrings "G" is assigned to countG
perCG = float(100*(countC+countG)/len(seqOut))  # Calculates percentage of C+G in the sequence
print("\nPorcentage of CG:\t\t",perCG)
# s.replace("X","Y") method to replace substring "X" for "Y" in the string s
seqOut = seqOut.replace("T","U")
print("\nSequence after replacing (t->U):",seqOut)
# s.find("X") method to return position of substring "X" in the string s
posA = seqOut.find("A")
print("\nPosition of substring A:\t",posA)
posUU = seqOut.find("UU")
print("\nPosition of substring UU:\t",posUU)
posU = seqOut.find("U")          # Returns the position of first "U" found in the string
print("\nPosition of first substring U:\t",posU)
```

Método Associados à Manipulação de Strings

O método *replace("X","Y")* troca a substring "X" pela substring "Y".

```
# Program to handle strings, using string methods available in Python 3 (part 1)
seqIn = "GATTACA"           # Initial sequence
print("\nInitial sequence:\t\t",seqIn)
# s.lower() method to generate lowercase characters for string s
seqOut = seqIn.lower()
print("\nSequence in lowercase:\t\t",seqOut)
# s.upper() method to generate uppercase characters for string s
seqOut = seqOut.upper()
print("\nSequence in uppercase:\t\t",seqOut)
# len(s) method to show the length of the string s
countBases = len(seqOut)    # len(seqOut) is assigned to countBases
print("\nNumber of bases:\t\t",countBases)
# s.count("X") method to count substring "X" in string s, used to calculate percentage of CG
countC = seqOut.count("C")  # Number of substrings "C" is assigned to countC
countG = seqOut.count("G")  # Number of substrings "G" is assigned to countG
perCG = float(100*(countC+countG)/len(seqOut))  # Calculates percentage of C+G in the sequence
print("\nPorcentage of CG:\t\t",perCG)
# s.replace("X","Y") method to replace substring "X" for "Y" in the string s
seqOut = seqOut.replace("T","U")
print("\nSequence after replacing (t->U):",seqOut)
# s.find("X") method to return position of substring "X" in the string s
posA = seqOut.find("A")
print("\nPosition of substring A:\t",posA)
posUU = seqOut.find("UU")
print("\nPosition of substring UU:\t",posUU)
posU = seqOut.find("U")          # Returns the position of first "U" found in the string
print("\nPosition of first substring U:\t",posU)
```

Método Associados à Manipulação de Strings

O método *s.find("X")* retorna a posição da substring "X". O primeiro caractere da string tem posição zero "0", o segundo caractere tem posição "1", e assim sucessivamente.

```
# Program to handle strings, using string methods available in Python 3 (part 1)
seqIn = "GATTACA"           # Initial sequence
print("\nInitial sequence:\t\t",seqIn)
# s.lower() method to generate lowercase characters for string s
seqOut = seqIn.lower()
print("\nSequence in lowercase:\t\t",seqOut)
# s.upper() method to generate uppercase characters for string s
seqOut = seqOut.upper()
print("\nSequence in uppercase:\t\t",seqOut)
# len(s) method to show the length of the string s
countBases = len(seqOut)    # len(seqOut) is assigned to countBases
print("\nNumber of bases:\t\t",countBases)
# s.count("X") method to count substring "X" in string s, used to calculate percentage of CG
countC = seqOut.count("C")  # Number of substrings "C" is assigned to countC
countG = seqOut.count("G")  # Number of substrings "G" is assigned to countG
perCG = float(100*(countC+countG)/len(seqOut))  # Calculates percentage of C+G in the sequence
print("\nPorcentage of CG:\t\t",perCG)
# s.replace("X","Y") method to replace substring "X" for "Y" in the string s
seqOut = seqOut.replace("T","U")
print("\nSequence after replacing (t->U):",seqOut)
# s.find("X") method to return position of substring "X" in the string s
posA = seqOut.find("A")
print("\nPosition of substring A:\t",posA)
posUU = seqOut.find("UU")
print("\nPosition of substring UU:\t",posUU)
posU = seqOut.find("U")          # Returns the position of first "U" found in the string
print("\nPosition of first substring U:\t",posU)
```

Método Associados à Manipulação de Strings

Ao executarmos o código *methods4strings1.py*, temos os resultados mostrados abaixo.

```
Initial sequence: GATTACA

Sequence in lowercase: gattaca

Sequence in uppercase: GATTACA

Number of bases: 7

Porcentage of CG: 28.571428571428573

Sequence after replacing (t->U): GAUUACA

Position of substring A: 1

Position of substring UU: 2

Position of first substring U: 2
```

Método Associados à Manipulação de Strings

O código *methods4strings2.py* traz outros métodos aplicados à string “GaTTaCa”, como segue.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

O método *s.swapcase()* troca de maiúscula para minúscula e vice-versa.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

O método *s.strip()* elimina espaços em branco, tabs e *newlines* no início e final da string.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

O método *list(s)* transforma uma string numa lista, com vírgulas separando cada caractere da string, inclusive espaços, tab e *newlines* existentes.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

O método 'X'.*join(ls)* transforma uma lista *ls* numa string, inserindo o argumento *X* entre os elementos da lista na composição da string.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

O método *s.split()* divide uma string *s* em palavras, formando uma lista.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

No final do programa temos a inserção de espaço ‘ ‘ e traço ‘-’ entre os elementos da lista *c3po*, com os métodos *‘ ‘.join(c3po)* e *‘-’.join(c3po*), respectivamente.

```
# Program to handle strings, using string methods available in Python 3 (part 2).
seqIn = "GaTTaCa"
print("\nInitial sequence:\t\t",seqIn)
# s.swapcase() method to change from uppercase to lowercase and vice-versa in the string s
seqOut = seqIn.swapcase()
print("\nSwapped sequence:\t\t",seqOut)
# New initial sequence
seqIn = "  GATTACA "
print("\nNew initial sequence:\t\t",seqIn)
# s.strip() method to get rid of spaces, tabs, and newlines in the string s
seqOut = seqIn.strip()
print("\nSequence without spaces:\t",seqOut)
# list(s) method to separate the string s in elements of a list
seqList = list(seqIn)
print("\nSequence as a list: \t\t",seqList)
# ''.join() method to merge all elements of a list in one string
seqIn = ''.join(seqIn)
print("\nThe sequence is back:\t\t",seqIn)
# s.split() method to separate the string s in words, where each word is an element
darthVader = "You don't know the power of the dark side"
c3po = darthVader.split()
print("\nOriginal message:\t\t",darthVader)
print("\nMessage after .split():\t\t",c3po)
# Applying ''.join() method after .split method
darthVader = ' '.join(c3po)              # With ' '
print("\nMessage after ' '.join():\t\t",darthVader)
darthVader = '-'.join(c3po)              # With '-'
print("\nMessage after '-'.join():\t\t",darthVader)
```

Método Associados à Manipulação de Strings

Ao executarmos o código *methods4strings2.py*, temos os resultados mostrados abaixo.

```
Initial sequence: GaTTaCa

Swapped sequence: gAttAcA

New initial sequence:   GATTACA

Sequence without spaces: GATTACA

Sequence as a list:  [' ', ' ', 'G', 'A', 'T', 'T', 'A', 'C', 'A', ' ']

The sequence is back:   GATTACA

Original message: You don't know the power of the dark side

Message after .split(): ['You', "don't", 'know', 'the', 'power', 'of', 'the',
'dark', 'side']

Message after ' '.join(): You don't know the power of the dark side

Message after '-'.join(): You-don't-know-the-power-of-the-dark-side
Mensagem apos '-'.join: You-don't-know-the-power-of-the-dark-side
```

Método Associados à Manipulação de Strings

Uma lista completa dos métodos usados para manipulação de strings em Python pode ser encontrada em:

http://www.tutorialspoint.com/python/python_strings.htm

Vamos considerar o programa *show_characters.py*, que mostra cada caractere de uma string, um por vez. O loop *for* está destacado nas linhas vermelhas. O loop *for* atribui cada caractere contido na string *my_string* à variável *char*, no loop *for* temos a função *print(char)*, que será repetida enquanto houver caracteres na string *my_string*. Cada caractere é mostrado numa linha. Veja que a função *print()* está recuada para direita, este recurso é obrigatório quando usamos o loop *for*. Ele delimita o bloco que será executado no loop *for*. Veja, também, que finalizamos a linha onde está o loop *for* com ":", esta parte também é obrigatória para o uso do loop *for*.

```
# Program to show the characters in a string, one in each line
my_string = "Python"

# Looping through the string
for char in my_string:
    print(char)
```

Abaixo temos o resultado da execução do programa *show_characters.py*. Vemos que cada caractere da string “Python” é mostrado numa linha. O loop *for* caminha sobre a string *my_string*, um caractere por vez, este processo é chamada **iteração**, tem a ideia de repetição. Em inglês é usado o termo “iteration”. Outro ponto a ser destacado, o loop *for* usa o recurso do **recuo**, em inglês “indentation”, para destacar o bloco que será executado no *loop*. Especificamente, o bloco tem somente a função *print()*.

```
P
y
t
h
o
n
```

O loop *for* pode varrer listas, como vemos no código *show_elements.py*, que mostra os elementos na lista *my_list*. A lógica de programação é mesma do programa *show_characters.py*, só trocamos a string por lista. O resultado é que o loop *for* mostra agora um elemento da lista por vez, um em cada linha. O loop *for* está destacado em vermelho. Veja que cada elemento da lista *my_list* é uma string. Separamos os elementos da lista por vírgulas, como mostrado para a lista na definição da lista *my_list*, na segunda linha do programa abaixo.

```
# Program to show the elements in a list, one in each line
my_list = ["Python","is","cool"]

# Looping through the list
for element in my_list:
    print(element)
```

Ao executarmos o programa *show_elements.py*, temos o resultado abaixo

```
Python
is
cool
```

Os programas a seguir estão disponíveis no site: http://www.delmarlearning.com/companions/content/1435455002/downloads/index.asp?isbn=1435455002. Esses programas são discutidos no livro: DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.

Considere o programa *craps_roller.py*, mostrado abaixo. Vamos analisar cada linha de código nos próximos slides. O programa simula o lançamento de dois dados, como num jogo de cassino chamado “Craps Roller”, quem tiver interesse em saber quais são as chances de ganhar no “Craps Roller”, veja o site: http://www.math.uah.edu/stat/games/Craps.html.

```
import random
# Generates random numbers from 1 to 6
die1 = random.randint(1,6)
die2 = random.randrange(6) + 1

total = die1 + die2

print("\nYou rolled a ",die1,"and a ",die2," for a total of ",total)
```

A primeira linha importa o módulo *random*. Já vimos nas aulas anteriores, que o comando *import* é usado para carregar um arquivo com código previamente preparado. O código carregado passa a fazer parte do programa que o chama. Assim, podemos chamar uma função específica do módulo carregado. Normalmente, os módulos são preparados dentro de um tema, por exemplo, o módulo *random* traz funções relacionadas à geração de números aleatórios. Na verdade, o termo aleatório deveria ser substituído por "pseudoaleatório", visto que o interpretador Python usa uma equação para gerar os números ditos aleatórios, assim não podem ser considerados aleatórios no sentido restrito da palavra. Para termos números aleatórios, devemos usar fenômenos naturais, tais como decaimento de partículas alfa, para, desta forma, obtermos uma sequência de números aleatórios. Mais informações em: http://www.fourmilab.ch/hotbits/.

```
import random
# Generates random numbers from 1 to 6
die1 = random.randint(1,6)
die2 = random.randrange(6) + 1

total = die1 + die2

print("\nYou rolled a ",die1,"and a ",die2," for a total of ",total)
```

Para gerarmos os números com o módulo *random*, chamamos as funções necessárias. Por exemplo, a função *randint()* destacada em vermelho no código abaixo. Para chamarmos a função, usamos a notação *dot*, vista anteriormente. Especificamente, *random.randint(1,6)* retorna um número entre 1 e 6, incluindo os extremos. Esse número é atribuído à variável *die1*. Se chamássemos a função diretamente, com *randint(1,6)*, teríamos uma mensagem de erro, especificamente, um erro de sintaxe ou execução. **Assim, a regra para o uso das funções presentes nos módulos, é colocar o nome do módulo, ao qual pertence a função, seguido do ponto "." e o nome da função, como indicado no quadro abaixo.**

modulo.função(argumento(s))

```
import random
# Generates random numbers from 1 to 6
die1 = random.randint(1,6)
die2 = random.randrange(6) + 1

total = die1 + die2

print("\nYou rolled a ",die1,"and a ",die2," for a total of ",total)
```

A função *randrange(6)* gera um número entre 0 e 5, ou seja, o número indicado como argumento da função não faz parte do conjunto de números pseudoaleatórios a serem gerados pela função. Assim, se usarmos a função *randrange()* para gerar um número pseudoaleatório entre 1 e 6, temos que somar “1” ao resultado, como mostrado na linha de código em vermelho abaixo. Veja, como na função *randint()*, os números gerados são inteiros. A diferença é que na função *randrange()* não precisamos especificar o limite inferior, é assumido ser zero “0”. É usada a notação *dot* e o resultado atribuído à variável *die2*.

```
import random
# Generates random numbers from 1 to 6
die1 = random.randint(1,6)
die2 = random.randrange(6) + 1

total = die1 + die2

print("\nYou rolled a ",die1,"and a ",die2," for a total of ",total)
```

Como os números pseudoaleatórios foram atribuídos às variáveis *die1* e *die2*, podemos operar com os valores. A linha em destaque abaixo realiza a soma dos valores atribuídos às variáveis *die1* e *die2* e atribui o resultado à variável *total*. A linha seguinte mostra o resultado na tela.

```
import random
# Generates random numbers from 1 to 6
die1 = random.randint(1,6)
die2 = random.randrange(6) + 1

total = die1 + die2

print("\nYou rolled a ",die1,"and a ",die2," for a total of ",total)
```

Vimos anteriormente o comando *if* e seu uso na ramificação da execução de um código em Python. Se não tivéssemos a possibilidade de ramificação da execução do código, os programas seguiriam seu caminho definido, do primeiro ao último comando, sem a possibilidade de ramificações. O comando *if* tem esta qualidade, ramificar a execução, vinculado a um teste que tomará um caminho ou outro. Vimos até o momento o *if* isolado, bem como o i*f* com o *else*. Além dos citados, temos o *elif*, o quadro no próximo slide ilustra o uso do *if*, *else* e *elif*.

Comando	Descrição
If condição : *bloco...*	O comando *if* testa a condição, e, caso seja verdadeira, executa o bloco de comandos. Caso seja falsa, pula o bloco de comandos.
If condição: *bloco 1* *else:* *bloco 2*	O comando *if* testa a condição, e, caso seja verdadeira, executa o bloco 1 de comandos. Caso seja falsa, executa o bloco 2 de comandos, vinculado ao *else*.
If condição 1: *bloco 1* *elif condição 2:* *bloco 2* *elif condição 3:* *bloco 3* *....* *elif condição N:* *bloco N* *else:* *bloco N + 1*	O comando *if* testa a condição 1, e, caso seja verdadeira, executa o bloco 1 de comandos e ignora os *elif* e *else*, mesmo que estes tenham condições verdadeiras. Caso seja falsa, testa a condição 2, se verdadeira, executa o bloco 2 e ignora os outros *elif* e *else*, se presentes. Essa abordagem é seguida pelos outros *elif* e *else*, se presentes. Veja, uma vez que uma das condições é satisfeita, o bloco referente a esta condição é executado e o programa ignora os outros *elif* e *else*, se presentes.

Veremos o programa *mood_computer.py*, disponível no site: http://www.delmarlearning.com/companions/content/1435455002/downloads/index.asp?isbn=1435455002. Como o código tem mais de 50 linhas, mostraremos por trechos. O programa é uma simulação de um teste de humor. Não se preocupe, não teremos eletrodos colocados na sua cabeça para testarmos o seu humor. O humor mostrado será resultado de um número pseudoaleatório. Só para ilustrarmos o uso do comando *if*. Nas décadas de 1970 e 1980 era comum as pessoas usarem um anel de humor que, conforme a sua coloração, poderíamos inferir o humor. Na verdade o suposto "anel do humor" era simplesmente uma bijuteria, onde a pedra do anel tinha um líquido que mudava de cor conforme a temperatura, nada relacionado ao humor da pessoa.

No primeiro trecho, temos o comando *import*, onde o módulo *random* é chamado. Depois temos duas funções *print()*, que mostram mensagens referentes à simulação do humor. Em seguida, temos a chamada da função *randint(1,3)*, que gera um número pseudoaleatório inteiro entre 1 e 3. Temos, então, o teste da primeira condição com o comando *if*, que caso seja “1”, mostra uma face feliz.

```
import random

print("I sense your energy.  Your true emotions are coming across my screen.")
print("You are...")

mood = random.randint(1, 3)

if mood == 1:
    # happy
    print( \
    """
       -----------
       |         |
       | O     O |
       |    <    |
       |         |
       | .     . |
       |  `...`  |
       -----------
                  """)
```

No segundo trecho, temos o *elif*, onde realizamos o teste da segunda condição, que caso seja “2”, mostra uma face neutra.

```
elif mood == 2:
    # neutral
    print( \
    """
       -----------
       |         |
       | O     O |
       |    <    |
       |         |
       | ------  |
       |         |
       -----------
                  """)
```

No terceiro trecho, temos o último *elif*, onde realizamos o teste da terceira condição, que caso seja “3”, mostra uma face triste.

```
elif mood == 3:
    # sad
    print( \
    """
       -----------
       |         |
       | O     O |
       |    <    |
       |         |
       |  .'.    |
       | '   '   |
       -----------
                  """)
```

Em seguida temos o *else,* que neste caso é desnecessário, visto que todas as possibilidades já foram testadas, mas é uma boa política deixarmos como uma cláusula de segurança, onde se números que não estivessem entre 1 e 3 surgissem, a situação seria tratada. Em um código relativamente simples como este, pode parecer excesso de zelo, mas imagine um programa com 10 mil linhas de código, com centenas de ramificações. Tal zelo pode evitar resultados catastróficos. Depois do bloco do *else*, temos uma função *print().*

```
else:
    print("Illegal mood value!  (You must be in a really bad mood).")

print("...today.")
```

Vamos ao resultado do programa *mood_computer.py*.

```
I sense your energy.  Your true emotions are coming across my screen.
You are...

        -----------
        |         |
        | O   O   |
        |   <     |
        |         |
        | .     . |
        |  `...`  |
        -----------

...today.
```

Outra forma de mudarmos a sequência de execução de cima para baixo de um programa é por meio de loops, já vimos o loop *for*. Outro tipo de loop é o *while,* que executa um bloco de código, enquanto uma determinada condição for satisfeita. Uma vez que a condição não seja mais satisfeita, o programa sai do loop *while.* Abaixo temos a estrutura geral do loop *while*.

```
while (condição):
        bloco vinculado à condição
```

O loop *while* assemelha-se ao comando *if*, o bloco de código é executado se a condição for satisfeita. A diferença reside que o loop *while* executa o bloco de forma repetida, até que a condição não seja mais satisfeita. Isto pode levar a situação onde a condição nunca seja satisfeita, o que chamamos de **loop infinito**. A condição testada no início é avaliada depois da execução do bloco de código, assim, espera-se que tenhamos uma variável que durante uma dada execução do bloco irá mudar para que a condição não seja mais satisfeita e a execução do bloco vinculada à condição do loop *while* seja encerrada.

Os loops infinitos podem ser considerados erros de lógica, mas podemos ter um loop potencialmente infinito que tenha uma condição de saída, conseguida com o comando *break*. Podemos, ainda, ter a necessidade de omitirmos a execução de parte de um bloco de um loop, voltando para o topo do bloco do loop. Isto é conseguido com o comando *continue*. Veremos o programa *skip_7.py* que mostra na tela uma contagem de 1 até 10, mas omite o número 7. O código chama-se *skip_7.py*, indicado abaixo.

```
count = 0
while True :
    count += 1
    # end loop if count greater than 10
    if count > 10:
        break
    # skip 7
    if count == 7:
        continue
    print(count)
```

A primeira linha do código é a atribuição de zero à variável *count*, que será usada como contador de iterações do loop. Em seguida temos o início do loop *while*, veja que a condição é sempre verdadeira, *while True*: gera um loop infinito, que será interrompido com o teste de uma condição de saída. Antes desta condição, somamos 1 ao valor atribuído à variável *count* e atribuímos o resultado ao contador *count*, com *count += 1*. Cada passagem do loop teremos "1" somado e atribuído à variável *count*.

```
count = 0
while True :
    count += 1
    # end loop if count greater than 10
    if count > 10:
        break
    # skip 7
    if count == 7:
        continue
    print(count)
```

Depois testamos se o contador é maior que 10, com o comando *if*, caso seja maior que 10, o bloco do *if* é executado. Vemos no bloco abaixo que temos o comando *break*, que faz exatamente isto, quebra a sequência de execução do bloco associado ao *while* e sai do loop. Em seguida testamos se o contador é 7, caso seja, o bloco do *if* é executado, onde temos o comando *continue,* que pula para o início do loop e testa a condição do loop, e como é sempre verdadeira, continua a execução. O resultado líquido é que o número 7 não é mostrado na tela.

```
count = 0
while True :
    count += 1
    # end loop if count greater than 10
    if count > 10:
        break
    # skip 7
    if count == 7:
        continue
    print(count)
```

Vamos ao resultado do programa *skip_7.py*.

```
1
2
3
4
5
6
8
9
10
```

Veremos a aplicação dos conceitos vistos hoje num programa de jogo, onde você tenta adivinhar um número pseudoaleatório entre 1 e 100 gerado pelo computador, o programa chama-se *guess_my_number.py*. Veremos a execução do programa antes de vermos seu código.

```
Welcome to 'Guess My Number'!

I'm thinking of a number between 1 and 100.
Try to guess it in as few attempts as possible.

Take a guess: 50
Lower...
Take a guess: 25
Higher...
Take a guess: 38
Higher...
Take a guess: 44
You guessed it!  The number was 44
And it only took you 4 tries!
```

O programa *guess_my_number.py* está mostrado abaixo. A primeira linha importa o módulo *random*, que tem as funções para gerarmos números pseudoaleatórios.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

As três linhas seguintes são funções *print()* que mostram uma mensagem de boas-vindas e informações sobre o jogo.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

Depois usamos a função *randint(1,100)* do módulo *random*, para gerar um número pseudoaleatório entre 1 e 100. O número gerado será atribuído à variável *the_number*.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

O programa agora pergunta ao usuário pelo o número que ele acha que foi escolhido pelo computador, e atribui este número à variável *guess.* O número 1 é atribuído à variável *tries,* que indica o número de tentativas que o jogador fez.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

A condição do loop *while* é que o valor atribuído à variável *guess* seja diferente do atribuído à variável *the_number*, ou seja, o bloco do loop *while* será executado, enquanto os números forem diferentes.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

No bloco do loop temos o comando *if,* que testa se o valor atribuído à variável *guess* é maior que ao atribuído à variável *the_number,* caso seja, o programa mostra a mensagem *"Lower..."*, indicando que o jogador deve digitar um número menor.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

Caso o valor atribuído à variável *guess* não seja maior que o atribuído à variável *the_number* é mostrada a mensagem *"Higher ..."*, para que o jogador digite um número mais alto.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

Depois é lido um novo valor para o número e atribuído à variável *guess*. É somado 1 ao valor atribuído à variável *tries.* O bloco de comandos é executado, até que a condição do *while* não seja mais satisfeita, ou seja, o jogador achou o número.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

Como o programa só sairá do bloco do loop *while*, quando o jogador acertar o número, colocamos, logo após os comandos do loop *while*, duas funções *print()* com as informações sobre o número encontrado e o número de tentativas necessárias.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

Execute o programa *guess_my_number.py*. Apesar de termos um número aleatório, podemos imaginar uma abordagem de jogo para minimizar o número de tentativas. Jogue algumas vezes e pense como minimizar o número de tentativas.

```
import random
print("\tWelcome to 'Guess My Number'!")
print("\nI'm thinking of a number between 1 and 100.")
print("Try to guess it in as few attempts as possible.\n")
the_number = random.randint(1, 100)
guess = int(input("Take a guess: "))
tries = 1
while guess != the_number:    # guessing loop
    if guess > the_number:
        print("Lower...")
    else:
        print("Higher...")
    guess = int(input("Take a guess: "))
    tries += 1
print("You guessed it!  The number was", the_number)
print("And it only took you", tries, "tries!\n")
```

Exercício de programação 1. Modifique o código fonte do programa *show_strings1.py*, use aspas simples para a variável *my_var1*. Salve o novo programa com o nome *show_strings1a.py*. Rode o programa. Houve diferenças nos resultados? Explique.

Exercício de programação 2. Modifique o código fonte do programa *show_strings1.py,* crie um nova variável, a variável *my_var4,* que recebe o resultado da divisão da variável *my_var2* pela *my_var3* (`my_var4 = my_var2/my_var3`). Insira a variável *my_var4* na função *print()*, mas antes coloque uma informação que o resultado mostrado é a divisão (sem acento, por exemplo, *Division*). Salve o novo programa com o nome *show_strings1b.py*. Rode o programa. **Nota: Não use acentos nas strings. Abaixo temos o resultado esperado**

```
String:  My string , Integer:  112358 , Float:  3.14159 Division:  35764.69240098167
```

Exercício de programação 3. Modifique o código fonte do programa *show_strings2.py, e* coloque aspas simples nos conteúdos atribuídos às variáveis *my_var2* e *my_var3*. Mantenha a variável *my_var4* inalterada. Salve o novo programa com o nome *show_strings2a.py*. Rode o programa. O que aconteceu? Explique.

Exercício de programação 4. Modifique o código fonte do programa *methods4strings1.py*, de forma que a nova versão leia o conteúdo de um arquivo de entrada. Salve o novo programa com o nome *methods4strings1a.py*. Rode o programa.

Exercício de programação 5. Escreva um programa que simula um biscoito da sorte chinês. O programa deve mostrar uma entre cinco previsões, de forma aleatória, cada vez que é executado. Nome do programa: *fortune_cookie.py*.

Exercício de programação 6. Escreva um programa que simula o lançamento de uma moeda 100 vezes. Depois o programa mostra o número de vezes que deu cara e que deu coroa. Nome do programa: *flip_a_coin.py* .

Exercício de programação 7. Modifique o programa *guess_my_number.py*, de forma que o jogador tenha um número limitado de tentativas. Se o jogador não consegue acertar o número gerado pelo computador, num número definido de tentativas, serão mostradas na tela o número certo e uma mensagem para o jogador. Nome do programa: *limited_guess_my_number.py* .

-BRESSERT, Eli. **SciPy and NumPy**. Sebastopol: O'Reilly Media, Inc., 2013. 56 p.
-DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.
-HAL, Tim, STACEY, J-P. **Python 3 for Absolute Beginners.** Springer-Verlag New York, 2009. 295 p.
-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** Nova York: Springer Science+Business Media LLC, 2010. 316 p.
-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p.
-KIUSALAAS, Jaan. **Numerical Methods in Engineering with Python.** 2ed. Nova York: Cambridge University Press, 2010. 422 p.
-LANDAU, Rubin H. **A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran90**. Princeton: Princeton University Press, 2005. 481p.
-LANDAU, Rubin H., PÁEZ, Manuel José, BORDEIANU, Cristian C. **A Survey of Computational Physics. Introductory Computational Physics.** Princeton: Princeton University Press, 2008. 658 p.
-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p.
-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p.