Treinamento Python Foundation – Conteudo Teorico/Prático

Sumário

Python - Visão geral

Python é uma linguagem de script de alto nível, interpretada, interativa e orientada a objetos. O Python foi projetado para ser altamente legível. Ele usa palavras-chave em inglês frequentemente, enquanto outros idiomas usam pontuação e tem menos construções sintáticas do que outros idiomas.

Python é Interpretado - Python é processado em tempo de execução pelo intérprete. Você não precisa compilar seu programa antes de executá-lo. Isso é semelhante ao PERL e PHP.

Python é interativo - Você pode sentar-se em um prompt do Python e interagir diretamente com o intérprete para escrever seus programas.

Python é orientado a objetos - O Python suporta o estilo ou a técnica de programação orientada a objetos que encapsula o código dentro dos objetos.

História do Python

Python foi desenvolvido por Guido van Rossum no final dos anos 80 e início dos 90, no Instituto Nacional de Pesquisa de Matemática e Ciência da Computação, na Holanda.

O Python é derivado de muitas outras linguagens, incluindo ABC, Modula-3, C, C ++, Algol-68, SmallTalk e shell do Unix e outras linguagens de script.

O Python é protegido por direitos autorais. Como o Perl, o código fonte do Python agora está disponível sob a GNU General Public License (GPL).

Python agora é mantido por uma equipe de desenvolvimento do instituto, embora Guido van Rossum ainda tenha um papel vital na direção de seu progresso.

Os recursos do Python incluem:

Fácil de aprender - o Python possui poucas palavras-chave, estrutura simples e uma sintaxe claramente definida. Isso permite que o aluno compreenda o idioma rapidamente.

Fácil de ler - o código Python é mais claramente definido e visível aos olhos.

Fácil de manter - o código fonte do Python é bastante fácil de manter.

Uma ampla biblioteca padrão - a maior parte da biblioteca do Python é muito portátil e compatível com várias plataformas no UNIX, Windows e Macintosh.

Modo Interativo - O Python suporta um modo interativo que permite testes interativos e depuração de trechos de código.

Portável - Python pode rodar em uma ampla variedade de plataformas de hardware e tem a mesma interface em todas as plataformas.

Extensível - Você pode adicionar módulos de baixo nível ao interpretador Python. Esses módulos permitem que os programadores adicionem ou personalizem suas ferramentas para serem mais eficientes.

Bancos de dados - Python fornece interfaces para todos os principais bancos de dados comerciais.

Programação GUI - O Python suporta aplicativos GUI que podem ser criados e portados para muitas chamadas de sistema, bibliotecas e sistemas Windows, como Windows MFC, Macintosh e o sistema X Window do Unix.

Escalonável - Python fornece uma estrutura e suporte melhores para programas grandes do que scripts de shell.

Além dos recursos mencionados acima, o Python possui uma grande lista de bons recursos, poucos estão listados abaixo -

Ele suporta métodos de programação funcionais e estruturados, bem como OOP.

Ele pode ser usado como uma linguagem de script ou pode ser compilado em código de bytes para criar aplicativos grandes.

Ele fornece tipos de dados dinâmicos de alto nível e suporta a verificação dinâmica de tipos.

Ele suporta a coleta automática de lixo.

Pode ser facilmente integrado com C, C ++, COM, ActiveX, CORBA e Java.

Operadores Básicos Python

Operadores são as construções que podem manipular o valor dos operandos. Considere a expressão 4 + 5 = 9. Aqui, 4 e 5 são chamados de operandos e + é chamado de operador.

Tipos de Operador

A linguagem Python suporta os seguintes tipos de operadores -

- Operadores aritméticos
- Operadores de comparação (relacional)
- Operadores de atribuição
- Operadores lógicos
- Operadores bit a bit
- Operadores de associação
- Operadores de identidade

Vamos dar uma olhada em todos os operadores, um por um.

Operadores aritméticos em Python

Assuma que a variável **a** mantém o valor 10 e a variável **b** mantém o valor 21:

Operador	Descrição	Exemplo

+ Adição	Adiciona valores em ambos os lados do operador.	a + b = 31
- Subtração	Subtrai o operando do lado direito do operando do lado esquerdo.	a - b = -11
* Multiplicação	Multiplica valores em ambos os lados do operador	a * b = 210
/ Divisão	Divide o operando da mão esquerda pelo operando da mão direita	b / a = 2,1
% De módulo	Divide o operando da mão esquerda pelo operando da mão direita e retorna o restante	b% a = 1
** Expoente	Executa cálculo exponencial (potência) em operadores	a ** b = 10 à potência 20
//	Divisão de piso (Floor Division) - a divisão dos operandos em que o resultado é o quociente no qual os dígitos após o ponto decimal são removidos. Mas se um dos operandos for negativo, o resultado será calculado, ou seja, arredondado para longe de zero (em direção ao infinito negativo):	9 // 2 = 4 e 9,0 // 2,0 = 4,0, -11 // 3 = -4, -11,0 // 3 = -4,0

Operadores de comparação Python

Esses operadores comparam os valores em ambos os lados e decidem a relação entre eles. Eles também são chamados de operadores relacionais.

Assuma que a variável **a** mantém o valor 10 e a variável **b** mantém o valor 20, então:

Operador	Descrição	Exemplo
==	Se os valores de dois operandos forem iguais, a condição se tornará verdadeira.	(a == b) não é verdadeiro.
! =	Se os valores de dois operandos não forem iguais, a condição se tornará verdadeira.	(a! = b) é verdadeiro.
>	Se o valor do operando esquerdo for maior que o valor do operando direito, a condição se tornará verdadeira.	(a> b) não é verdadeiro.
<	Se o valor do operando esquerdo for menor que o valor do operando direito, a condição se tornará verdadeira.	(a <b) é verdadeiro.
> =	Se o valor do operando esquerdo for maior ou igual ao valor do operando direito, a condição se tornará verdadeira.	(a> = b) não é verdadeiro.
<=	Se o valor do operando esquerdo for menor ou igual ao valor do operando direito, a condição se tornará verdadeira.	(a <= b) é verdadeiro.

Operadores de atribuição do Python

Assuma que a variável **a** mantém o valor 10 e a variável **b** mantém o valor 20:

Operador	Descrição	Exemplo

=	Atribui valores de operandos do lado direito ao operando do lado esquerdo	c = a + b atribui valor de a + b em c
+ = Adicionar AND	Ele adiciona o operando direito ao operando esquerdo e atribui o resultado ao operando esquerdo	c + = a é equivalente a c = c + a
- = Subtrair AND	Subtrai o operando direito do operando esquerdo e atribui o resultado ao operando esquerdo	c - = a é equivalente a c = c - a
* = Multiplicar AND	Multiplica o operando direito pelo operando esquerdo e atribui o resultado ao operando esquerdo	c * = a é equivalente a c = c * a
/ = Dividir AND	Ele divide o operando esquerdo com o operando direito e atribui o resultado ao operando esquerdo	c / = a é equivalente a c = c / ac / = a é equivalente a c = c / a
% = Módulo AND	Ele pega o módulo usando dois operandos e atribui o resultado ao operando esquerdo	c% = a é equivalente a c = c% a
** = Expoente AND	Executa o cálculo exponencial (potência) em operadores e atribui valor ao operando esquerdo	c ** = a é equivalente a c = c ** a

// = Divisão de piso (Division floor)	Realiza divisão de piso nos operadores e atribui valor ao operando esquerdo	c // = a é equivalente a c = c // a

Operadores Python Bitwise

O operador bit a bit trabalha em bits e executa operação bit a bit. Suponha que a = 60; eb = 13; Agora, no formato binário, eles serão os seguintes -

a = 0011 1100

b = 0000 1101

---

a & b = 0000 1100

a | b = 0011 1101

a ^ b = 0011 0001

~ a = 1100 0011

A função interna bin do Python () pode ser usada para obter representação binária de um número inteiro.

Os seguintes operadores Bitwise são suportados pela linguagem Python:

Operador	Descrição	Exemplo
& Binário AND	O operador copia um bit, para o resultado, se existir nos dois operandos	(a & b) (significa 0000 1100)

| OR binário	Ele copia um bit, se existir em qualquer operando.	(a | b) = 61 (significa 0011 1101)
^ XOR binário	Ele copia o bit, se estiver definido em um operando, mas não em ambos.	(a ^ b) = 49 (significa 0011 0001)
~ Complemento binário	É unário e tem o efeito de 'inverter' os bits.	(~ a) = -61 (significa 1100 0011 no formulário do complemento 2 devido a um número binário assinado.
<< Deslocamento binário à esquerda	O valor do operando esquerdo é movido para a esquerda pelo número de bits especificado pelo operando direito.	a << 2 = 240 (significa 1111 0000)
>> Deslocamento binário à direita	O valor do operando esquerdo é movido para a direita pelo número de bits especificado pelo operando direito.	a >> 2 = 15 (significa 0000 1111)

Operadores lógicos Python

Os seguintes operadores lógicos são suportados pela linguagem Python. Assuma que a variável **a** contém True e a variável **b** contém False

Operador	Descrição	Exemplo
AND lógico	Se ambos os operandos forem verdadeiros, a condição se tornará verdadeira.	(a and b) é falso.

OR Lógico	Se qualquer um dos dois operandos for diferente de zero, a condição se tornará verdadeira.	(a or b) é verdadeiro.
NOT Logico	Usado para reverter o estado lógico de seu operando.	Not(a and b) é verdadeiro.

Operadores de associação do Python

Os operadores de associação do Python testam a associação em uma sequência, como cadeias, listas ou tuplas. Existem dois operadores de associação, conforme explicado abaixo:

Mostrar exemplo

Operador	Descrição	Exemplo
in	Avalia como true se encontrar uma variável na sequência especificada e false caso contrário.	x em y, aqui resulta em 1 se x é um membro da sequência y.
not in	Avalia como true se não encontrar uma variável na sequência especificada e false caso contrário.	x não em y, aqui não resulta em 1 se x não for um membro da sequência y.

Operadores de identidade (Identity) Python

Os operadores de identidade comparam os locais de memória de dois objetos. Existem dois operadores de identidade, conforme explicado abaixo -

Mostrar exemplo

Operador	Descrição	Exemplo
is	Avalia como true se as variáveis em ambos os lados do operador apontam para o mesmo objeto e false em caso contrário.	x é y, aqui **é o** resultado em 1 se id (x) for igual a id (y).
is not	Avalia como false se as variáveis de ambos os lados do operador apontam para o mesmo objeto e true, caso contrário.	x não é y, aqui **não** resulta em 1 se id (x) não for igual a id (y).

Precedência dos operadores Python

A tabela a seguir lista todos os operadores da precedência mais alta à mais baixa.

Mostrar exemplo

Operador e descrição
****** Exponenciação (aumento ao poder)
~ + - Complemento, unário mais e menos (os nomes dos métodos para os dois últimos são + @ e - @)
*** /% //** Divisão de multiplicação, divisão, módulo e piso

+ - Adição e subtração
>> << Deslocamento bit a direita e esquerda
& Bit a bit 'AND'
^ | `OR 'exclusivo bit a bit e` OR' regular
<= <>> = Operadores de comparação
<> ==! = Operadores de igualdade
=% = / = // = - = + = * = ** = Operadores de atribuição
is is not Operadores de identidade
in not in Operadores de associação

not or and

Operadores lógicos

Sintaxe Básica Python

Primeiro Programa Python

Vamos executar os programas em diferentes modos de programação.

Programação em modo interativo

Chamar o intérprete sem passar um arquivo de script como parâmetro exibe o seguinte prompt

```
print ("Ola Mundo Python!" )
```

```
Ola Mundo Python!
```

Se você estiver executando a versão mais antiga do Python (Python 2.x), o uso de parênteses como função inprint é opcional. Isso produz o seguinte resultado -

Tipos de variáveis

Variáveis nada mais são que locais de memória reservados para armazenar valores. Isso significa que, ao criar uma variável, você reserva algum espaço na memória.

Com base no tipo de dados de uma variável, o intérprete aloca memória e decide o que pode ser armazenado na memória reservada. Portanto, atribuindo diferentes tipos de dados às variáveis, você pode armazenar números inteiros, decimais ou caracteres nessas variáveis.

Atribuindo valores a variáveis

Variáveis Python não precisam de declaração explícita para reservar espaço na memória. A declaração acontece automaticamente quando você atribui um valor a uma variável. O sinal de igual (=) é usado para atribuir valores a variáveis.

O operando à esquerda do operador = é o nome da variável e o operando à direita do operador = é o valor armazenado na variável. Por exemplo

```
NumInt = 300          # Valor numero inteiro
NumFloat   = 3000.0       # Valor numero float/double
nome    = "Chilindrina"       # Valor String

print (NumInt)
print (NumFloat)
print (nome)
```

```
300
3000.0
Chilindrina
```

Aqui, 300, 3000.0 e "Chilindrina" são os valores atribuídos às variáveis NumInt, NumFloat e nome, respectivamente. Isso produz o resultado exibido acima

Atribuição múltipla O Python permite atribuir um único valor a várias variáveis simultaneamente.

Por exemplo: a = b = c = 1

Aqui, um objeto inteiro é criado com o valor 1 e todas as três variáveis são atribuídas ao mesmo local de memória. Você também pode atribuir vários objetos a várias variáveis. Por exemplo a, b, c = 1, 2, "Kiko"

Aqui, dois objetos inteiros com os valores 1 e 2 são atribuídos às variáveis a e b, respectivamente, e um objeto string com o valor "Kiko" é atribuído à variável c.

Tipos de dados padrão

Os dados armazenados na memória podem ser de vários tipos. Por exemplo, a idade de uma pessoa é armazenada como um valor numérico e seu endereço é armazenado como caracteres alfanuméricos. O Python possui vários

tipos de dados padrão que são usados para definir as operações possíveis neles e o método de armazenamento para cada um deles.

O Python possui cinco tipos de dados padrão:

- Números (Numbers)
- Strings
- Lists
- Tuple
- DictiOnary

Números Python

Os tipos de dados numéricos armazenam valores numéricos. Objetos numéricos são criados quando você atribui um valor a eles. Por exemplo

```
valorA = 1
valorB = 10
```

```
print(valorA, valorB)
```

```
1 10
```

O Python suporta três tipos numéricos diferentes:

- int (números inteiros assinados)
- float (valores reais de ponto flutuante)
- complex (números complexos)

Python Strings

Strings em Python são identificadas como um conjunto contíguo de caracteres representados entre aspas. O Python permite um par de aspas simples ou duplas. Subconjuntos de strings podem ser obtidos usando o operador de fatia ([] e [:]) com índices que começam em 0 no início da string e avançam de -1 até o final.

O sinal de mais (+) é o operador de concatenação de cadeias e o asterisco (*) é o operador de repetição. Por exemplo

```
str = 'Ola Mundo!'

print (str)              # Imprime o texto completo
print (str[0])           # Imprime o primeiro caractere do texto
print (str[2:5])         # Imprime caracteres no intervalo do 3º
ao 5º posições print (str[2:])          # Imprime sequência
iniciando no terceiro caractere print (str * 2)          #
Imprime a string duas vezes print (str + "TESTANDO") # Imprime
string concatenada
```

```
Ola Mundo!
O
a M
a Mundo!
Ola Mundo!Ola Mundo!
Ola Mundo!TESTANDO
```

Atualizando strings

Você pode "atualizar" uma string existente (re) atribuindo uma variável a outra string. O novo valor pode estar relacionado ao seu valor anterior ou a uma sequência completamente diferente. Por exemplo -

```
stringA = 'Hello World!'

print ("Atualizando minha string : ", stringA[:6] +
'código em Python')
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Atualizando minha string :  Hello código em Python
```

Scape Characters

A tabela a seguir mostra uma lista de caracteres de escape ou não imprimíveis que podem ser representados com a notação de barra invertida.

Um caractere de escape é interpretado; em uma única e entre aspas.

Notação de barra invertida	Caractere hexadecimal	Descrição

\a	0x07	Bell (sino) ou alerta
\ b	0x08	Backspace
\ cx		Control-x
\ Cx		Control-x
\ e	0x1b	Escapar
\ f	0x0c	Formfeed
\ M- \ Cx		Meta-Control-x
\ n	0x0a	Nova linha (new line)
\ nnn		Notação octal, em que n está na faixa de 0,7
\ r	0x0d	Retorno carriage (transporte)
\ s	0x20	Espaço
\ t	0x09	Tab
\ v	0x0b	Tab vertical
\ x		Caracter x

\ xnn		Notação hexadecimal, em que n está no intervalo 0,9, af ou AF

Operadores especiais de strings

Suponha que a variável de string **a** contenha 'Hello' e a variável **b** contenha 'Python';

Operador	Descrição	Exemplo
+	Concatenação - Adiciona valores em ambos os lados do operador	***a*** **+** ***b*** resulta em HelloPython
*	Repetição - Cria novas strings, concatenando várias cópias da mesma string	***a*** * 2 resulta em - HelloHello
[]	Slice (Fatia) - retorna o caractere do índice especificado	***a*** [1] resulta em e
[:]	Range slice (Faixa de intervalo) - fornece os caracteres do intervalo especificado	***a*** [1: 4] resulta em ell
in	Membership - Retorna true se um caractere existe na string especificada	H in ***a*** resulta em 1
not in	Membership - Retorna true se um caractere não existe na string especificada	M not in ***a*** resulta em 1

r / r	Raw string (Sequência bruta/crua) - Suprime o significado real dos caracteres de escape. A sintaxe para cadeias brutas é exatamente a mesma que para cadeias normais, com exceção do operador de cadeia bruta, a letra "r", que precede as aspas. O "r" pode ser minúsculo (r) ou maiúsculo (R) e deve ser colocado imediatamente antes da primeira citação.	imprima r '\ n' imprime \ ne imprima R '\ n'prints \ n
%	Formato - Executa a formatação de seqüência de caracteres	

Operador de formatação de string

Um dos recursos mais interessantes do Python é o operador de formato de string **%**. Esse operador é exclusivo para strings e compensa o pacote de funções da família printf () de C. A seguir, um exemplo simples:

```
print ("Meu nome é %s e peso %d kg!" % ('Zuleida', 61))
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Meu nome é Zuleida e peso 61 kg!
```

Aqui está a lista do conjunto completo de símbolos que podem ser usados junto com % :

Símbolo de formato	Conversão
% c	caracter
% s	conversão de string via str () antes da formatação

%i	número inteiro decimal signed (assinado)
% d	número inteiro decimal signed (assinado)
%u	número inteiro decimal unsigned (não assinado)
% o	número inteiro octal
% x	número inteiro hexadecimal (letras minúsculas)
% X	número inteiro hexadecimal (letras maiúsculas)
% e	notação exponencial (com 'e' minúsculo)
% E	notação exponencial (com UPPERcase 'E')
% f	número real de ponto flutuante
% g	o menor de% fe% e
% G	o menor de% fe% E

Outros símbolos e funcionalidades suportados estão listados na tabela a seguir -

Símbolo	Funcionalidade
*	argumento especifica largura ou precisão
-	justificação esquerda

+	exibir o sinal
<sp>	deixa um espaço em branco antes de um número positivo
#	adiciona o zero inicial octal ('0') ou o hexadecimal inicial '0x' ou '0X', dependendo se 'x' ou 'X' foram usados.
0	*Pad* da esquerda com zeros (em vez de espaços)
%	'%%' deixa um único literal '%'
(var)	variável de mapeamento (argumentos do dicionário)
mn	m é a largura total mínima en é o número de dígitos a serem exibidos após o ponto decimal (se aplicável)

Citações Triplas

As citações triplas do Python são úteis, permitindo que as seqüências de caracteres abranjam várias linhas, incluindo NEWLINEs, TABs e outros caracteres especiais.

A sintaxe para aspas triplas consiste em três aspas **simples ou duplas** consecutivas.

```
para_str = """esta é uma longa sequência composta de
várias linhas e caracteres não imprimíveis, como
TAB (\ t) e eles aparecerão dessa maneira quando exibidos.
NEWLINEs dentro da string, tem de ser explicitamente
declarada com [\n] entre colchetes ou apenas uma NEWLINE com a
atribuição de variável também será exibida..
```

```
    """

    print (para_str)
```

Quando o código acima é executado, produz o seguinte resultado. Observe como todos os caracteres especiais foram convertidos em seu formulário impresso, até a última NEWLINE no final da string entre "up". e fechando aspas triplas. Observe também que NEWLINEs ocorrem com um retorno de carro explícito no final de uma linha ou com seu código de escape (\ n) -

```
    esta é uma longa sequência composta de

    várias linhas e caracteres não imprimíveis, como

    TAB (\ t) e eles aparecerão dessa maneira quando exibidos.

    NEWLINEs dentro da string, tem de ser explicitamente declarada com [ ] entre colchetes ou apenas uma NEWLINE com a atribuição de v ariável também será exibida..
```

Seqüências brutas/cruas (raw) não tratam a barra invertida (backslash) como um caractere especial. Cada caractere que você coloca em uma sequência bruta permanece do jeito que você a escreveu -

```
    print ('C:\\nowhere')
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
    C:\nowhere
```

Agora vamos fazer uso da string bruta. Nós **colocaríamos** expressão em **r'expression 'da** seguinte maneira:

```
    print r'C:\\nowhere'
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
    C:\\nowhere
```

Unicode String

No Python 3, todas as strings são representadas no Unicode.No Python 2 são armazenadas internamente como ASCII de 8 bits, portanto, é necessário anexar 'u' para torná-lo Unicode. Não é mais necessário agora.

Métodos de seqüência de caracteres internos

O Python inclui os seguintes métodos internos para manipular strings

Métodos-padrão de strings (métodos internos)

O Python inclui os seguintes métodos internos para manipular strings:

Métodos com Descrição
capitalize() Coloca em maiúscula a primeira letra da string
center (largura, preenchimento) Retorna uma string preenchida com espaço com a string original centralizada em um total de colunas de largura.
count (str, beg = 0, end = len (string)) Conta quantas vezes str ocorre em uma string ou em uma subcadeia de strings se o índice inicial e o final do índice forem fornecidos.
decode (codificação = 'UTF-8', erros = 'estrito') Decodifica a string usando o codec registrado para codificação. padrões de codificação para a codificação de string padrão.
encode (codificação = 'UTF-8', erros = 'estrito') Retorna a versão da string codificada; em caso de erro, o padrão é gerar um ValueError, a menos que erros sejam fornecidos com 'ignorar' ou 'substituir'.

endswith (sufixo, beg = 0, end = len (string))

Determina se a string ou uma substring da string (se o índice inicial e final do índice forem fornecidos) termina com o sufixo; retorna true se sim e false caso contrário.

expandtabs (tabsize = 8)

Expande as guias na string para vários espaços; o padrão é 8 espaços por guia, se o tamanho da tabulação não for fornecido.

find (str, beg = 0 end = len (string))

Determina se str ocorre na string ou em uma subcadeia de strings se o índice inicial e final do índice receberem retornos index se encontrado e -1 caso contrário.

índex (str, beg = 0, end = len (string))

O mesmo que find (), mas gera uma exceção se str não for encontrado.

isalnum ()

Retorna true se a string tiver pelo menos 1 caractere e todos os caracteres forem alfanuméricos e false, caso contrário.

isalpha ()

Retorna true se a string tiver pelo menos 1 caractere e todos os caracteres forem alfabéticos e falsos.

isdigit ()

Retorna true se a string contém apenas dígitos e false, caso contrário.

islower()

Retornará true se a string tiver pelo menos 1 caractere e todos os caracteres estiverem em minúsculas e false, caso contrário.

isnumeric ()

Retorna true se uma string unicode contiver apenas caracteres numéricos e false caso contrário.
isspace () Retorna true se a string contiver apenas caracteres de espaço em branco e false caso contrário.
istitle () Retorna true se a string for "titlecased" corretamente e false caso contrário.
isupper () Retorna true se a string tiver pelo menos um caractere em caixa e todos os caracteres em caixa estiverem em maiúsculas e falso em contrário.
join (seq) Mescla (concatena) as representações de string dos elementos na sequência seq em uma string, com a string separadora.
len (string) Retorna o comprimento da string
ljust (width [, fillchar]) Retorna uma string preenchida com espaço com a string original justificada à esquerda para um total de colunas de largura.
lower() Converte todas as letras maiúsculas em sequência para minúsculas.
lstrip () Remove todo o espaço em branco à esquerda da string.

maketrans ()

Retorna uma tabela de conversão para ser usada na função de conversão.

max (str)

Retorna o máximo de caracteres alfabéticos da string str.

min (str)

Retorna o caractere alfabético mínimo da string str.

replace (old, noew [, max])

Substitui todas as ocorrências antigas na cadeia de caracteres por novas ou, no máximo, no máximo, se especificado.

rfind (str, beg = 0, end = len (string))

O mesmo que find (), mas pesquise para trás na string.

rindex (str, beg = 0, end = len (string))

O mesmo que index (), mas pesquise para trás na string.

rjust (width, [, fillchar])

Retorna uma string preenchida com espaço com a string original justificada à direita para um total de colunas de largura.

rstrip ()

Remove todo o espaço em branco à direita da sequência.

split (str = "", num = string.count (str))

Divide a string de acordo com o delimitador str (espaço se não for fornecido) e retorna a lista de substrings; dividir em no máximo substrings se for dado.

splitlines (num = string.count ('\ n'))

Divide a string em todas (ou num) NEWLINEs e retorna uma lista de cada linha com as NEWLINEs removidas.

startswith(str, beg = 0, end = len (string))

Determina se string ou uma substring de string (se o índice inicial e final do índice for fornecido) começa com substring str; retorna true se sim e false caso contrário.

strip ([chars])

Executa lstrip () e rstrip () na string.

swapcase ()

Inverte maiúsculas e minúsculas para todas as letras da string.

título()

Retorna a versão "titlecased" da string, ou seja, todas as palavras começam em maiúsculas e as demais em minúsculas.

translate(table, deletechars = "")

Traduz a string de acordo com a tabela de conversão str (256 caracteres), removendo as da string del.

upper()

Converte letras minúsculas em sequência para maiúsculas.

zfill (width)

Retorna a string original deixada com zeros para um total de caracteres de largura; destinado a números, zfill () mantém qualquer sinal fornecido (menos um zero).

isdecimal ()

Retorna true se uma string unicode contiver apenas caracteres decimais e false caso contrário.

Listas (Lists) de Python

As listas são os tipos de dados compostos mais versáteis do Python. Uma lista contém itens separados por vírgulas e entre colchetes ([]). Até certo ponto, as listas são semelhantes às matrizes em C. Uma das diferenças entre elas é que todos os itens pertencentes a uma lista podem ser de tipos de dados diferentes.

Os valores armazenados em uma lista podem ser acessados usando o operador de fatia ([] e [:]) com índices começando em 0 no início da lista e trabalhando até o final de -1. O sinal de mais (+) é o operador de concatenação da lista e o asterisco (*) é o operador de repetição. Por exemplo

```
UmaLista = [ 'xwyz', 123 , 5.15, 'Dexter',
92.5 ] listinha = [456, 'Corleone']

print (UmaLista)            # Imprime a lista
completa print (UmaLista[0])        # Imprime o
primeiro elemento da lista
print (UmaLista[1:3])       # Imprime elementos no intervalo do 2º
ao 3º posições print (UmaLista[2:])        # Imprime elementos a
partir do terceiro elemento print (listinha * 2)    # Imprime a
lista duas vezes print (UmaLista + listinha) # Imprime listas
concatenadas
```

```
['xwyz', 123, 5.15,
'Dexter', 92.5] xwyz
[123, 5.15]
[5.15, 'Dexter', 92.5]
[456, 'Corleone', 456, 'Corleone']
['xwyz', 123, 5.15, 'Dexter', 92.5, 456,
'Corleone']
```

Atualizando listas

Você pode atualizar elementos únicos ou múltiplos de listas, fornecendo a fatia no lado esquerdo do operador de atribuição e pode adicionar elementos a uma lista com o método append (). Por exemplo:

```
lista1 = ['matematica', 'portugues', 1992, 2005]
print ("Valor disponível no índice 2: ", lista1[2])
```

```
lista1[2] = 2001
print ("Novo valor disponível no índice 2: ", lista1[2])
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Valor disponível no índice 2 :  1997
Novo valor disponível no índice 2 :  2001
```

Excluir elementos da lista

Para remover um elemento da lista, você pode usar a instrução del se souber exatamente quais elementos estão excluindo ou o método remove () se não souber. Por exemplo:

```
lista1 = ['matematica', 'portugues', 1992, 2005]
print (lista1)

del lista1[2]
print ("Após excluir o valor no índice 2: ", lista1)
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
['matematica', 'portugues', 1992, 2005]
Após excluir o valor no índice 2:  ['matematica',
'portugues', 2005]
```

Operações básicas da lista

As listas respondem aos operadores + e * como as strings; eles significam concatenação e repetição aqui também, exceto que o resultado é uma nova lista, não uma string.

De fato, as listas respondem a todas as operações gerais de string que usamos nas seqüências de caracteres no capítulo anterior.

Expressão Python	Resultados	Descrição
len ([1, 2, 3])	3	comprimento
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenação

['Ola!'] * 4	['Ola!', 'Ola!', 'Ola!', 'Ola!']	Repetição
3 em [1, 2, 3]	true	Associação
for x in [1, 2, 3]: print x,	1 2 3	Iteração

Indexação, fatia(slice) e matrizes (arrays)

Como as listas são strings, a indexação e o fatiamento funcionam da mesma maneira para as listas e para as arrays.

Assumindo a seguinte entrada:

```
lingProg = ['Javascript'', 'Java', 'Python']
```

Expressão Python	Resultados	Descrição
LingProg[2]	'Python'	A contagem do índices começam em zero
LingProg[-2]	'Java'	Sentido contrário: conta a partir da direita
LingProg[1:]	['Java', 'Python']	Usando slice para trazer “pedaços”

Funções e métodos padrão de lists

O Python inclui as seguintes funções-padrão no elemento list

Função com Descrição
len (list) Dá o comprimento total da lista.
max (list) Retorna o item da lista com o valor máximo.

min (list) Retorna o item da lista com valor mínimo.
list (seq) Converte uma tupla em lista.

Python inclui os seguintes métodos de lista

Métodos com Descrição
list.append (obj) Anexa ao final da lista Anexa obj do objeto à lista
list.count (obj) Retorna a contagem de quantas vezes obj ocorre na lista
list.extend (seq) Anexa o conteúdo de seq à lista
list.index (obj) Retorna o índice mais baixo da lista que obj aparece
list.insert (índex, obj) Insere o objeto obj na lista no índice de deslocamento
list.pop (obj = list[-1]) Remove e retorna o último objeto ou obj da lista
list.remove (obj) Remove obj do objeto da lista
list.reverse () Inverte os objetos da lista no local
list.sort ([func]) Classifica objetos da lista, use compare func se for dado

Tuplas (Tuples) Python

Uma tupla é outro tipo de dados de sequência semelhante à lista. Uma tupla consiste em vários valores separados por vírgulas. Diferentemente das listas, no entanto, as tuplas são colocadas entre parênteses.

A principal diferença entre listas e tuplas é - As listas estão entre colchetes ([]) e seus elementos e tamanho podem ser alterados, enquanto as tuplas estão entre parênteses (()) e não podem ser atualizadas. As tuplas podem ser consideradas como listas somente leitura . Por exemplo

```
UmaTupla = ( 'xwyz', 123 , 5.15, 'Dexter',
92.5 ) tuplapeq = (456, 'Corleone')

print (UmaTupla)            # Imprime a tupla
completa print (UmaTupla[0])         # Imprime o
primeiro elemento da tupla
print (UmaTupla[1:3])       # Imprime elementos no intervalo do 2º
ao 3º posições print (UmaTupla[2:])        # Imprime elementos a
partir do terceiro elemento print (tuplapeq * 2)      # Imprime
a tupla duas vezes print (UmaTupla + tuplapeq) # Imprime tuplas
concatenadas
```

```
('xwyz', 123, 5.15,
'Dexter', 92.5) xwyz
(123, 5.15)
(5.15, 'Dexter', 92.5)
(456, 'Corleone', 456, 'Corleone')
('xwyz', 123, 5.15, 'Dexter', 92.5, 456,
'Corleone')
```

Atualizando Tuplas

As tuplas são imutáveis, o que significa que você não pode atualizar ou alterar os valores dos elementos das tuplas. Você pode usar partes das tuplas existentes para criar novas, como o exemplo a seguir demonstra:

```
tuplaNova = (23, 43.54)
tuplaNova2 = ('Vamos', 'nessa')

#A ação a seguir não é válida para tuplas
#tup1[0] = 100;

# Então, vamos criar uma nova tupla da seguinte maneira
tuplaResult = tuplaNova + tuplaNova2
print (tuplaResult)
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
(23, 43.54, 'Vamos', 'nessa')
```

Excluir elementos da tupla

A remoção de elementos individuais da tupla não é possível. Obviamente, não há nada errado em montar outra tupla com os elementos indesejados descartados.

Para remover explicitamente uma tupla inteira, basta usar a instrução **del** . Por exemplo:

```
tupla1 = ('filosofia', 'sociologia', 1993, 1996)

print (tupla1)
del tupla1;
print ("Depois de deltar a tupla1 : ")
print (tupla1)
```

Isso produz o seguinte resultado. Observe uma exceção levantada, porque a tupla **del tup** não existe mais:

```
('filosofia', 'sociologia', 1993, 1996)
After deleting tup :
Traceback (most recent call last):
   File "test.py", line 9, in <module>
      print tupla1;
NameError: name 'tupla1' is not defined
```

Operações básicas de tuplas

As tuplas respondem aos operadores + e * como as strings; eles significam concatenação e repetição aqui também, exceto que o resultado é uma nova tupla, não uma string.

De fato, as tuplas respondem a todas as operações gerais de sequência que usamos nas seqüências de caracteres no capítulo anterior:

Expressão Python	Resultados	Descrição
len ((1, 2, 3))	3	comprimento
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenação

('Hi!') *	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetição
3 in (1, 2, 3)	true	Associação
for x in (1, 2, 3): print x,	1 2 3	Iteração

Indexação, fatia (slice) e arrays

Como as tuplas são sequências, a indexação e o fatiamento funcionam da mesma maneira para as tuplas e para as arrays. Assumindo a seguinte entrada

```
elementos = ('spam', 'Spamzinho', 'MUITO SPAM!')
```

Expressão Python	Resultados	Descrição
elementos [2]	'MUITO SPAM!'	A contagem do índices começam em zero
elementos [-2]	'Spamzinho'	Sentido contrário: conta a partir da direita
elementos [1:]	['Spamzinho', 'MUITO SPAM!']	Usando slice para trazer "pedaços"

Funções e métodos padrão de TUPLES

O Python inclui as seguintes funções de tupla

Função com Descrição
len (tuple) Dá o comprimento total da tupla.

max (tuple) Retorna o item da tupla com o valor máximo.
min (tuple) Retorna o item da tupla com valor mínimo.
tuples (seq) Converte uma lista em tupla.

Dicionário Python

Dicionários são pares chave-valor (key-value).Cada chave é separada de seu valor por dois pontos (:), os itens são separados por vírgulas e colocados entre chaves. Um dicionário vazio sem itens é escrito com apenas duas chaves, assim: {}.

As chaves são únicas em um dicionário, enquanto os valores podem não ser. Os valores de um dicionário podem ser de qualquer tipo, mas as chaves devem ser de um tipo de dados imutáveis, como cadeias, números ou tuplas.

Acessando valores no dicionário

Para acessar os elementos do dicionário, você pode usar os colchetes familiares, juntamente com a chave, para obter seu valor. A seguir, um exemplo simples -

```
dici = {'Nome': 'Florinda', 'idade': 37, 'Curso': 'Pedagogia'}
print ("dici['Nome']: ", dici['Nome'])
print ("dici['idade']: ", dici['idade'])
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
dici['Nome']: Florinda

dici['idade']: 37
```

Se tentarmos acessar um item de dados com uma chave, que não faz parte do dicionário, obteremos um erro da seguinte maneira:

```
dici = {'Nome': 'Florinda', 'idade': 37, 'Curso': 'Pedagogia'}

print ("dici['Clotilde']: ", dici['Clotilde'])
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
dici['Clotilde']:

Traceback (most recent call last):

        print "dici['Clotilde']: ", dici['Clotilde'];

KeyError: 'Clotilde'
```

Acessando elementos do dicionário

Enquanto a indexação é usada com outros tipos de dados para acessar valores, um dicionário usa *keys*. As chaves podem ser usadas entre colchetes [] ou com o método get().

Se usarmos colchetes [ ], um KeyError é gerado caso uma chave não seja encontrada no dicionário. Por outro lado, o método get() retornará *None* se a chave não for encontrada.

```
# get vs [] para acessar elementos
dici = {'nome': 'Florinda', 'idade': 37}

print(dici['nome'])
# saida: Florinda

print(dici.get('idade'))
# saida: 37

print(dici.get('endereço'))

# saida: None
```

Ao executar o código acima, o resultado é o seguinte:

```
Florinda
37
None
```

Atualizando dicionário

Você pode atualizar/alterar um dicionário adicionando uma nova entrada ou um par de valores-chave, modificando uma entrada existente ou excluindo uma entrada existente, conforme mostrado abaixo no exemplo simples -

```
dici = {'Nome': 'Florinda', 'idade': 37, 'Curso':
'Pedagogia'}

dici['idade'] = 18; # atualiza/altera uma entrada existente

dici['Faculdade'] = "Colação de Grau" # Adiciona uma nova
entrada

     print ("dici['idade']: ", dici['idade'])

     print ("dici['Faculdade']: ", dici['Faculdade'])
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
dici['idade']:  18

dici['Faculdade']:  Colação de Grau
```

Excluir elementos do dicionário

Você pode remover elementos individuais do dicionário ou limpar todo o conteúdo de um dicionário. Você também pode excluir o dicionário inteiro em uma única operação.

Para remover explicitamente um dicionário inteiro, basta usar a instrução **del** . A seguir, um exemplo simples:

```
dici = {'Nome': 'Florinda', 'idade': 37, 'Curso':
'Pedagogia'}
```

```
del dici['Nome'] # remove o elemento com a chave 'Nome'
dici.clear()     # remove todos os elementos em dici
del dici         # deleta/exclui o dicionário inteiro

print ("dici['idade']: ", dici['idade'])
print ("dici['Faculdade']: ", dici['Faculdade'])
```

Isso produz o seguinte resultado: - observe que uma exceção é gerada porque o dicionário depois da instrução ***del dici*** não existe mais:

```
dici['Age']:

Traceback (most recent call last):

      print "dici['Age']: ", dici['Age'];

TypeError: 'type' object is unsubscriptable
```

Propriedades das chaves de dicionário

Os valores do dicionário não têm restrições. Eles podem ser qualquer objeto Python arbitrário, objetos padrão ou objetos definidos pelo usuário. No entanto, o mesmo não é verdadeiro para as chaves.

Há dois pontos importantes a serem lembrados sobre as chaves do dicionário:

1 - Não é permitido mais de uma entrada por chave. O que significa que nenhuma chave duplicada é permitida. Quando chaves duplicadas são encontradas durante a atribuição, a última atribuição vence. Por exemplo:

```
dici = {'Nome': 'Florinda', 'idade': 37, 'Nome': 'Kiko'}

print ("dici['Nome']: ", dici['Nome'])
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
dici['Nome']:  Kiko
```

Não podemos usar listas, porque são mutáveis

2 - as chaves devem ser imutáveis. O que significa que você pode usar strings, números ou tuplas como chaves de dicionário, mas algo como ['key'] não é permitido. A seguir, um exemplo simples:

```
dici = {['Nome']: 'Florinda', 'idade': 37}

print ("dici['Nome']: ", dici['Nome'])
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Traceback (most recent call last):

      dici = {['Nome']: 'Florinda', 'idade': 37}

TypeError: list objects are unhashable
```

Funções internas e métodos de um dicionário

Python inclui as seguintes funções de dicionário:

Função e Descrição	
Função	Descrição
all()	Retorna True se todas as *keys* do dicionário forem True (ou se o dicionário estiver vazio).
any()	Retorna True se qualquer *key* do dicionário for verdadeira. Se o dicionário estiver vazio, retorna False.
len()	Retorna o comprimento (o número de itens) no dicionário.
sorted()	Retorna uma nova lista classificada de keys no dicionário.

Aqui estão alguns exemplos que usam funções internas para trabalhar com um dicionário. Para observamos a execução, vamos criar um novo dicionário de pares chave-valor: a chave é um número inteiro e o valor respectivo de cada chave será o quadrado do número indicado na chave.

Vamos a construção:

```
quadrados = {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}


# saida: False
print(all(quadrados))


# saida: True
print(any(quadrados))


# saida: 6
print(len(quadrados))


# saida: [0, 1, 3, 5, 7, 9]
print(sorted(quadrados))
```

Não podemos ter chave 0

```
False
True
6
[0, 1, 3, 5, 7, 9]
```

Observe a saída acima

Python inclui os seguintes métodos de dicionário:

Métodos com Descrição
dict.clear () Remove todos os elementos do dicionário *dict*

dict.copy () Retorna uma cópia superficial do dicionário *dict*
dict.fromkeys () Cria um novo dicionário com chaves de seq e valores *definidos* como *valor* .
dict.get (key, default = None) Retorna o valor da chave . Se a chave não existir, retornará d (o padrão é None).
dict.has_key (key) Retorna *true* se a chave estiver no dicionário *dict* , *false* caso contrário
dict.items () Retorna um novo objeto dos itens do dicionário no formato (chave, valor).
dict.keys () Retorna a lista de chaves do dicionário
dict.setdefault (key, default = None) Semelhante a get (), mas definirá dict [key] = padrão se a *chave* ainda não estiver no dict
dict.update (dict2) Adiciona pares de valores-chave do dicionário *dict2* a *dict*
dict.values () Retorna a lista de dicionário *dict* valores 's

Exemplo de caso de uso desses métodos. Vamos criar um novo dicionário chamado *disciplinas* - como se segue abaixo:

```
    # Métodos do Dictionary
    disciplinas = {}.fromkeys(['Sociologia', 'História',
'Geografia'], 0)

    print(disciplinas)
    # Output: {'Sociologia': 0, 'História': 0, 'Geografia':
0}


    for i in disciplinas.items():
        print(i)

    print(list(sorted(disciplinas.keys())))
    # Output: ['Geografia', 'História', 'Sociologia']
```

Ao executar o código acima, o resultado é exibido abaixo:

```
    {'Sociologia': 0, 'História': 0, 'Geografia': 0}
    ('Sociologia', 0)
    ('História', 0)
    ('Geografia', 0)
```

Acrescentando organização ao nosso dicionário disciplinas:

```
    # Métodos do Dictionary


disciplinas = {}.fromkeys(['Sociologia', 'História',
'Geografia'], 0)

    print(disciplinas)
    # Output: {'Sociologia': 0, 'História': 0, 'Geografia':
0}


    for i in disciplinas.items():
        print(i)

    print(list(sorted(disciplinas.keys())))
    # Output: ['Geografia', 'História', 'Sociologia']
```

Ao executar o código acima, o resultado é exibido abaixo:

```
{'Sociologia': 0, 'História': 0, 'Geografia': 0}
('História', 0)
('Geografia', 0)
('Sociologia', 0)
['Geografia', 'História', 'Sociologia']
```

Compreensão de dicionário (Dictionary Comprehension)

A compreensão de dicionário é uma maneira elegante e concisa de criar um novo dicionário a partir de um iterável no Python. A compreensão do dicionário consiste em um par de expressões (chave: valor) seguido de uma declaração *for* dentro de chaves { }.

Observe o exemplo abaixo – onde estamos utilizando um novo dicionário que se chama *quadrado* - para criar um dicionário com cada item sendo um par de número e seu quadrado.

```
# Compreensão de Dicionário (Dictionary Comprehension)
quadrado = {x: x*x for x in range(8)}

print(quadrado)
```

Ao executar o código acima, o resultado é exibido abaixo:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6:36, 7:49}
```

Este código é equivalente a:

```
quadrado = {}
for x in range(8):
    quadrado[x] = x*x
print(quadrado)
```

Saída:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6:36, 7:49}
```

A compreensão dicionário *(Dictionary Comprehension)* pode opcionalmente conter mais declarações *for* ou *if*.

Uma declaração opcional *if* pode filtrar itens para formar o novo dicionário.

Observe o exemplo para criar um dicionário apenas com itens ímpares.

```
# Dictionary Comprehension com a condicional if
quadrados_impares = {x: x*x for x in range(11) if x % 2 == 1}

print(quadrados_impares)
```

saida

```
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

Outras operações de dicionário - Teste de associação ao dicionário

Podemos testar se a key está em um dicionário ou não, usando o operador in. Observe que o teste de associação é apenas para Keys e não para values. Para executar este teste vamos utilizar o primeiro dicionário quadrado que criamos anteriormente:

```
# Membership Test para Dictionary Keys
quadrado = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# saida: True
print(1 in quadrado)

# saida: True
print(2 not in quadrado)

# membership test somente para key não para value
# saida: False
print(49 in quadrado)
```

Ao executar o código acima, o resultado é exibido abaixo:

```
True
True
False
```

Iterando através de um dicionário

Podemos percorrer cada chave em um dicionário usando um loop *for*. Observe o exemplo abaixo:

```
# Iterando através do Dictionary
quadrados = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in quadrado:
    print(quadrado[i])
```

ao executar o código o resultado é esse:

```
1
9
25
49
81
```

Conversão de tipo de dados

Às vezes, pode ser necessário realizar conversões entre os tipos internos. Para converter entre tipos, basta usar os nomes dos tipos como uma função.

Existem várias funções internas para executar a conversão de um tipo de dados para outro. Essas funções retornam um novo objeto que representa o valor convertido.

- int (x [, base]) : Converte x em um número inteiro. A base especifica a base se x é uma sequência. x é a base do número
- float (x) : Converte x em um número de ponto flutuante. complexx (real [, imag]) : Cria um número complexo. str (x) : Converte o objeto x em uma representação de sequência.
- repr (x) : Converte o objeto x em uma cadeia de expressão. eval (str) : Avalia uma string e retorna um objeto. tuple (s) : Converte s em uma tupla. list (s) : Converte s em uma lista. set (s) : Converte s em um conjunto.
- dict (d) : Cria um dicionário. d deve ser uma sequência de tuplas (chave, valor). frozenset (s) : Converte s em um conjunto congelado (frozen set).
- chr (x) : Converte um número inteiro em um caractere. unichr (x) : Converte um número inteiro em um caractere Unicode.
- ord (x) : Converte um único caractere em seu valor inteiro.
- hex (x) : Converte um número inteiro em uma sequência hexadecimal. out (x) : Converte um número

Tomada de decisão (if...else)

O que é a declaração if ... else em Python?

A tomada de decisão é necessária quando queremos executar um código apenas se uma determinada condição for atendida.

A declaração if…elif…else é usada no Python para tomada de decisão.

Sintaxe da instrução if Python if

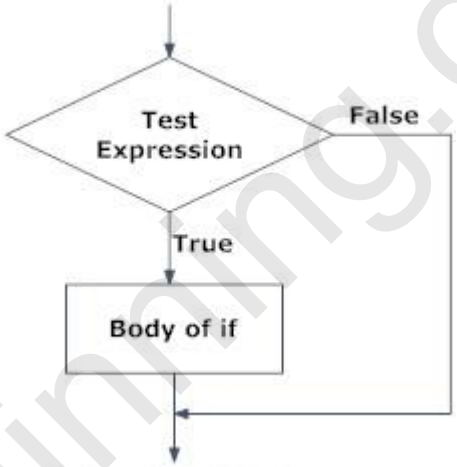
```
if test declaration:
    code suite(c)
```

Aqui, o programa avalia as intruções `test expression` e executará (c) apenas se a `test expression` for *True*.

Se *test expression* for *False*, a instrução/ões (c) não será(ão) executada(s). Em Python, o corpo da instrução *if* é indicado pelo recuo (indentação). O corpo começa com um recuo e a primeira linha sem indentação marca o fim.

Python interpreta valores diferentes de zero como *True*. *None* e 0 são interpretados como *False*.

Fluxograma de instrução Python if



Exemplo:

```
    #Se o número for positivo, a mensagem será exibida de forma adequada

    num = 25
    if num > 0:
        print(num, "é um número positivo.")
    print("Esse texto sempre será exibido. Não faz parte do
corpo da estrutura de decisão.")

    num = -31
    if num > 0:
        print(num, " é um número positivo.")
    print("Esse texto sempre será exibido. Não faz parte do
corpo da estrutura de decisão."
```

Quando o programa, acima, é executado a saída será:

```
     25 é um número positivo
     Esse texto sempre será exibido. Não faz parte do corpo da
estrutura de decisão.
     Esse texto sempre será exibido. Não faz parte do corpo da
estrutura de decisão.
```

No exemplo acima, num > 0 é a expressão de teste *(test expression).*

O corpo de if é executado apenas se for avaliado como *True*. Quando a variável *num* é igual a *25,* a expressão de teste *(test expression)* é verdadeira e as instruções dentro do corpo da declaração *if* são executadas.

Se a variável *num* for igual a -31, a expressão de teste *(test expression)* será falsa e as instruções dentro do corpo da declaração *if* serão ignoradas.

A declaração *print()* fica fora do bloco *if* (sem indentação). Portanto, é executado independentemente da expressão de teste *(test expression).*

Instrução Python if ... else

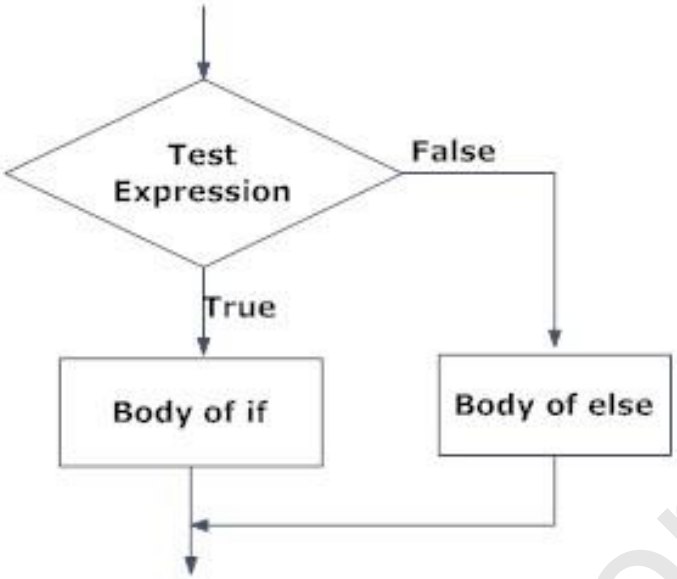
Sintaxe de if ... else

```
if test expression:
    Body of if
else:
    Body of else
```

A instrução *if..else* avalia a `test expression` e executará o corpo if somente quando a condição de teste for *True*.

Se a condição for *False*, o corpo da instrução *else* é executado. O recuo (indentação) é usado para separar os blocos.

Python if..else Fluxograma



Exemplo:

```
     # O programa a seguir verifica se o número é positivo
ou negativo
     # após a verificação exibe a mensagem apropriada
     num = 52

     # você pode testar alterando as declarações com duas
novas variáveis como
     # mostrado abaixo
     # num = -43
     # num = 0

     if num >= 0:
         print("Positivo ou Zero")
     else:
         print("Número Negativo")
```

Resultado

```
     Positivo ou Zero
```

No exemplo acima, quando *num* é igual a *52*, a expressão de teste (*test expression*) é verdadeira e o corpo de *if* é executado e o corpo da instrução *else* é ignorado.

Se *num* for igual a *-43*, por exemplo, a expressão de teste (*test expression*) será falsa e o corpo de *else* será executado, assim o corpo de *if* passa a ser ignorado.

Se *num* for igual a *0*, a expressão de teste é verdadeira (*test expression*) e o corpo de *if* é executado e o corpo de *else* é ignorado.

Instrução Python if ... elif ... else

Sintaxe de if ... elif ... else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

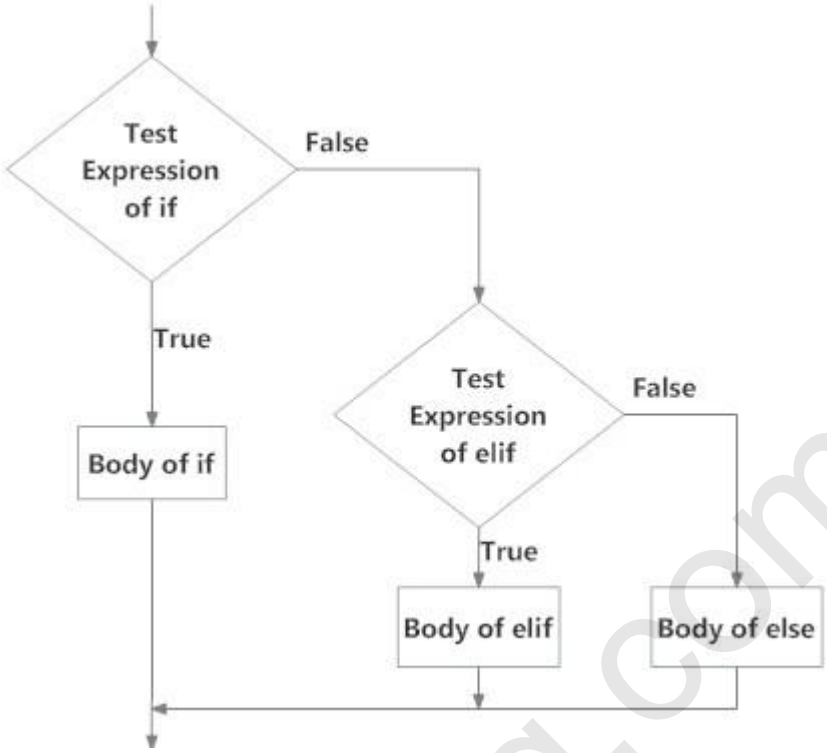
A declaração *elif* é uma “abreviação” da declaração *else if*. Nos permite verificar várias expressões.

Se a condição *if* for *False*, ele verifica a condição do próximo bloco *elif* e assim sucessivamente.

Se todas as condições forem *False*, o corpo de *else* é executado.

Apenas um bloco entre os vários blocos *if...elif...else* é executado de acordo com a condição.O bloco *if* pode ter apenas um bloco *else*. Mas podem existir vários blocos *elif*.

Fluxograma de if ... elif ... else



Exemplo *if ... elif ... else*

```
''' 'No programa abaixo
verificamos se o número é positivo ou
negativo ou zero e
exibimos a mensagem apropriada ''' '
num = 78.9

# você pode testar alterando as declarações com duas
novas variáveis como
# mostrado abaixo

# num = -560.2
# num = 0

if num > 0:
    print("Numero positivo")
elif num == 0:
    print("Zero")
else:
    print("Numero Negativo")
```

Resultado

```
Numero positivo
```

Quando a variável *num* é positiva o valor da expressão do resultado é:

Número positivo é impresso.

Se *num* for igual a 0 o valor da expressão do resultado é:

Zero

Se *num* for negativo o valor da expressão do resultado é:

Número Negativo.

Instruções aninhadas *if*

Podemos ter uma declaração *if...elif...else* dentro de outra decalração *if...elif...else.* Chamamos isso de "*aninhamento"* de declarações.

Qualquer número dessas instruções pode ser "*aninhada"* - uma dentro da outra. O recuo (indentação) é a única maneira de descobrir o nível de *"aninhamento*". Eles podem ficar confusos, portanto devem ser usados com muito cuidado – use, apenas, se for estritamente necessário.

Exemplo

```
    '' 'No programa abaixo declaramos um número
    E verificamos se este número é positivo ou
    negativo ou zero. Na sequência, exibimos a mensagem
apropriada
    Desta vez, usamos a declaração if aninhada
    Altere o valor da variável para um numero negativo e
depois para o numero zero. Veja o resultado para cada uma
das saídas.'' '
    num = 1000.0
    if num >= 0:
        if num == 0:
            print("Número zero")
        else:
            print("Número positivo")
    else:
```

```
        print(" Número negativo")
```

Resultado do primeiro teste:

```
Número positivo
```

Resultado do segundo teste:

```
Número negativo
```

Resultado do terceiro teste:

```
Número zero
```

Loops

O que é loop *for* em Python?

O loop for no Python é usado para iterar sobre uma sequência (lista , tupla , string) ou outros objetos iteráveis. A iteração sobre uma sequência é chamada de travessia.

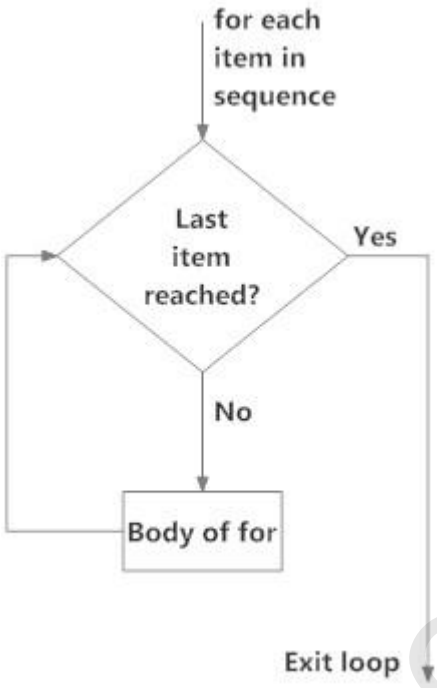
Sintaxe do loop *for*

```
for val in sequence:
  corpo do loop for
```

Aqui *val* é a variável que leva o valor do item dentro da sequência em cada iteração.

O loop continua até chegarmos ao último item da sequência. O corpo do loop ***for*** é separado do restante do código usando indentação.

Fluxograma do Loop *for*



Exemplo

```
    # o programa abaixo encontra a soma de todos os números
armazenados em uma lista
    # Lista de números

    nums = [18, 30, 21, 9, 43, 29, 32, 2, 110]

    # variável para armazenar a soma
    soma = 0

    # interar a lista de numeros
    for val in nums:
     soma = soma+val

    print("O resultado da soma é", soma)
```

Ao executar o programa, a saída será:

```
    O resultado da soma é 294
```

A função range ()

Podemos gerar uma sequência de números usando a função range(), por exemplo: **range(10) irá gerar números de 0 a 9 (10 números).**

Também podemos definir o tamanho de ***início, parada e passo*** da seguinte forma: ***range(start, stop, step_size).*** O padrão *step_size é 1* se não for fornecido.

O objeto *range* é "preguiçoso" em certo sentido, porque não gera todos os números que "contidos" quando o criamos. No entanto, não é um iterador uma vez que suporta as operações in, len e __getitem__.

Essa função não armazena todos os valores na memória; seria ineficiente. Assim, ele lembra o início, a parada, o tamanho da etapa e gera o próximo número em movimento.

Para forçar esta função para saída de todos os itens, podemos usar a função list().

O exemplo a seguir nos mostra esse uso:

```
print(range(11))

print(list(range(11)))

print(list(range(1, 9)))

print(list(range(1, 40, 4)))
```

Resultado

```
range(0, 11)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 5, 9, 13, 17, 21, 25, 29, 33, 37]
```

Podemos usar a função *range()* em loops for para iterar através de uma sequência de números. Ele pode ser combinado com a função *len()* para iterar através de uma sequência usando indexação.

Observe o exemplo abaixo:

```
    #o programa abaixo itera usando indexação

    mstyle = ['classica', 'jazz', 'blues']

    # itera a lista usando indexação
    for i in range(len(mstyle)):
      print("Gosto de ouvir alguns gêneros musicais como:
", mstyle[i])
```

Ao executar o programa, a saída será:

```
    Gosto de ouvir alguns gêneros musicais como: classica
    Gosto de ouvir alguns gêneros musicais como: jazz
    Gosto de ouvir alguns gêneros musicais como: blues
```

loop *for* usando *else*

Um loop *for* também pode ter um bloco *else* opcional . O bloco *else* é executado se a iteração sobre os itens na sequência usados no loop for forem “esgotados”.

A palavra-chave break pode ser usada para parar um loop for. Nesses casos, a parte *else* é ignorada.

Portanto, o bloco *else* de um loop *for* é executada se nenhuma interrupção ocorrer.

Abaixo, o exemplo para ilustra isso.

```
    items = [0, 1, 5]

    for i in items:
```

```
        print(i)
    else:
        print("Não há mais itens restantes.")
```

Ao executar o programa, a saída será:

```
0
1
5
Não há mais itens restantes.
```

Aqui, o loop *for* imprime itens da lista até o loop esgotar. Quando o loop for esgotado, ele executa o bloco de código dentro de *else* e imprime "Não há mais itens restantes."

Esta declaração *for...else* pode ser usada com a palavra-chave *break* para executar o bloco *else* apenas quando *break* não foi executada.

Observe o exemplo abaixo:

```
    # programa para exibir as notas dos alunos
    student_name = 'Goham'

    registers = {'Goten': 8.5, 'Trunks': 5.0, 'Majin Boo': 9.0}

    for student in registers:
        if student == student_name:
            print(registers[student])
            break
    else:
        print('Não há registros para este nome.')
```

Ao executar o programa, a saída será:

```
Não há registros para este nome.
```

Loop while

Loops são usados na programação para repetir um bloco de código específico – como observado, acima, estudando o loop *for*. Neste próximo tema, estudaremos o loop *while* implementado com Python.

O que é o loop *while* no Python?

O loop *while* é usado para iterar sobre um bloco de código, desde que a expressão de teste (condição) seja verdadeira.

Geralmente ***usamos*** esse loop ***quando não sabemos o número de vezes para iterar*** previamente.

Sintaxe do loop while em Python

```
while test_expression:
    Body of while
```

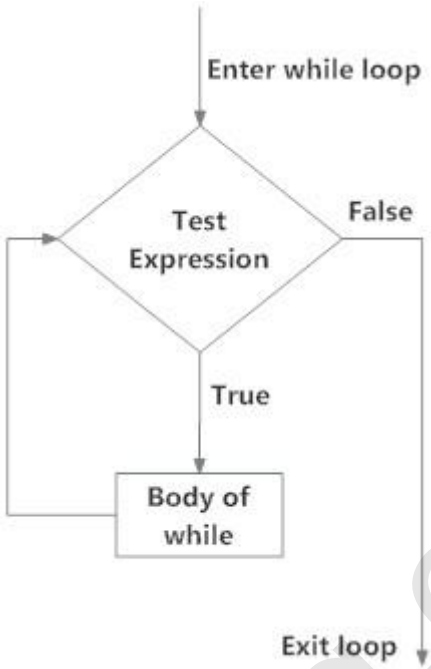
No loop *while*, a expressão de teste *(`test_expression`)* é verificada, em primeira instância. O corpo do loop é inserido apenas se a avaliação da expressão de teste *(`test_expression`)* for *True*. Após uma iteração, a expressão de teste *(`test_expression`)* é verificada novamente. Esse processo continua até que a expressão de teste *(`test_expression`) seja False.*

Em Python, o corpo do loop *while* é determinado por meio de indentação.

O corpo começa com recuo e a primeira linha sem recuo marca o fim.

Python interpreta qualquer valor diferente de zero como *True*. None e 0 (zero) são interpretados como *False*.

Fluxograma do loop While



Exemplo:

```
# Programa para adicionar números naturais
# soma = 1+2+3+...+n

n = 15

# inicializa a soma e o contador
soma = 0
x = 1

while x <= n:
    soma = soma + x
    x = x+1     # atualiza o contador

# exibe a soma na tela
print("O resultado da soma é", soma)
```

Ao executar o programa, a saída mostrada será:

```
O resultado da soma é 120
```

No programa acima, a expressão de teste *(test_expression)* será *True* desde que nossa variável de contador *i* seja menor ou igual a n (15, em nosso programa).

Precisamos aumentar o valor da variável do contador no corpo do loop. Isso é muito importante (e principalmente esquecido). Não fazer isso resultará em um loop infinito (loop sem fim).

Finalmente, o resultado é exibido.

Loop while com else

O loop *while* com *else* assume as mesmas características adotadas para o loop *for*, loops while também podem ter um bloco *else* opcional .

O bloco *else* é executado se a condição no loop while for avaliada como *False*.

Assim, o loop *while* pode ser finalizado com uma instrução *break*. Nesses casos, o bloco else é ignorado. Portanto, o bloco *else* de um loop *while* é executado se nenhuma interrupção (break) ocorrer e a condição for falsa.

Observe o exemplo abaixo:

```
'''Exemplo para ilustrar
o usa da declaração
com loop while'''

count = 0

while count < 3:
    print("Dentro do loop")
    count = count + 1
else:
    print("Dentro do else")
```

Saida:

```
Dentro do loop
```

```
Dentro do loop
Dentro do loop
Dentro do else
```

Aqui, usamos uma variável de contador para imprimir a string *"Dentro do loop"* três vezes.Na quarta iteração, a condição em while torna-se *False*. Portanto, o bloco *else* passa a ser executado.

Declarações break e continue

Esse tema aborda o uso das declarações *break* e *continue* implementados utilizando Python. Elas são usadas para alterar o fluxo de um loop.

Para que serve break e continue no Python?

Em Python, as instruções *break* e *continue* podem alterar o fluxo de um loop normal.

Os loops iteram sobre um bloco de código até que a expressão de teste *(test_expression)* seja falsa, mas às vezes desejamos terminar a iteração atual ou mesmo o loop inteiro sem verificar a expressão de teste *(test_expression)*.

As instruções *break* e *continue* são usadas para se construir esse tipo de cenário.

Declaração break

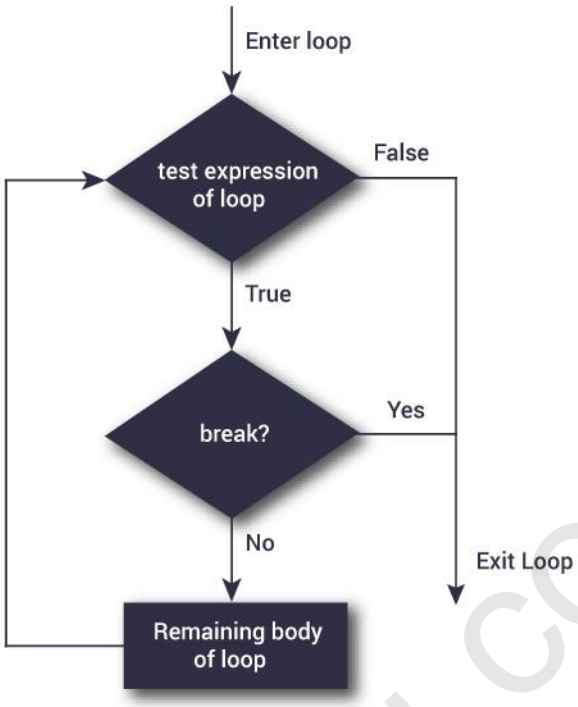
A declaração *break* finaliza o loop que a contém. O controle do programa flui para a instrução imediatamente após o corpo do loop.

Se a declaração *break* estiver dentro de um loop aninhado (loop dentro de outro loop), encerrará o loop que estiver numa posição mais interna.

Sintaxe da declaração break

```
break
```

Fluxograma



O funcionamento da instrução break em loop for e while é mostrado abaixo.

```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

Exemplo: uso da declaração *break*

```
# Uso do break statement dentro do loop
```

```
for i in "uma frase":
    if i == "f":
        break
    print(i)

print("Fim")
```

Este é o resultado do programa executado, acima:

```
u
m
a

Fim
```

Neste programa, iteramos através da string *“uma frase”.* Verificamos se a letra é ***f***, assim que o código encontra essa letra, rompe-se o ciclo. Portanto, vemos em nossa produção que todas as letras até ***f*** são impressas. Depois disso, o loop termina.

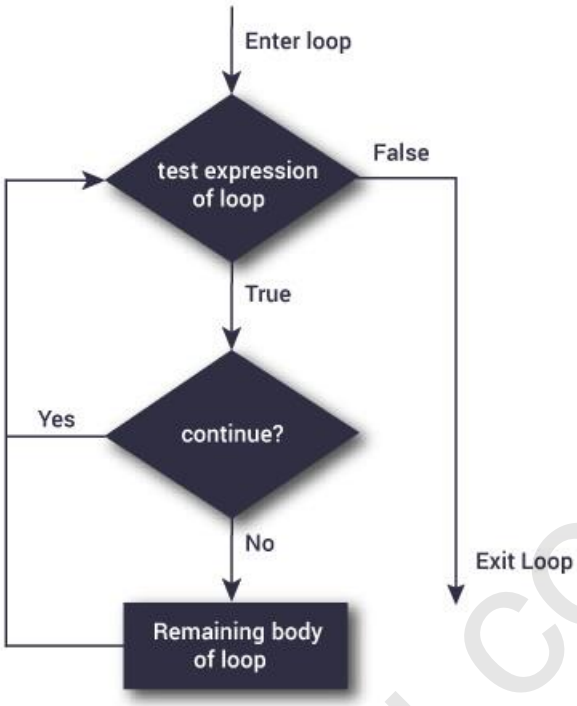
Declaração continue

A declaração *continue* é usada para ignorar o restante do código dentro de um loop apenas para a iteração atual. O loop não termina, mas continua com a próxima iteração.

Sintaxe da declaração continue

```
continue
```

Fluxograma da declaração *continue*



O funcionamento da instrução continue no loop for e while é mostrado

```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

abaixo.

Exemplo:

```
    # Programa que mostra o uso do continue statement dentro
do loop
```

```
for i in "uma frase":
    if i == "f":
        continue
    print(i)

print("Fim")
```

O resultado da execução do código acima é:

```
u
m
a

r
a
s
e
Fim
```

O programa acima é o mesmo do exemplo anterior. Aqui, a instrução *break* foi substituída por *continue*.

Continuamos com o loop, se a string for ***f***, a letra é omitida e o restante do bloco é executado Portanto, vemos em nossa saída que todas as letras foram impressas, exceto ***f***.

Date & Time

Um programa Python pode lidar com data e hora de várias maneiras. Converter entre formatos de data é uma tarefa comum para computadores. Os módulos de hora e calendário do Python ajudam a rastrear datas e horas.

O que é Tick?

Intervalos de tempo são números de ponto flutuante (floats) em unidades de segundos. Instantes específicos no tempo são expressos em segundos desde as 00:00 de 1º de janeiro de 1970 (época).

Existe um módulo *time* popular disponível no Python, que fornece funções para trabalhar com horários e para converte-los entre representações. A função *time.time ()* retorna a hora atual do sistema em ticks desde as 00:00 de 1º de janeiro de 1970 (época) até este momento.

Exemplo

```
import time      # necessário para utilizarmos o time module.

     ticks = time.time()
     print ("Numeros de ticks desde 12:00am, 1 de janeiro de
1970:", ticks)
```

Isso produziria um resultado da seguinte maneira:

```
Numeros de ticks desde 12:00am, 1 de janeiro de 1970:
7186862.73399
```

A aritmética de datas é fácil de fazer com ticks. No entanto, as datas anteriores à época indicada *(1 de janeiro de 1970)* não podem ser representadas nesse formato. Datas no futuro distante também não podem ser representadas dessa maneira - o ponto de corte é em algum momento de 2038 para UNIX e Windows.

O que é o TimeTuple?

Muitas das funções de tempo do Python lidam com o tempo como uma tupla de 9 números, como mostrado abaixo :

Índice	Campo	Valores
0 0	4 dígitos para ano	2020
1	Mês	1 a 12
2	Dia	1 a 31

3	Hora	0 a 23
4	Minuto	0 a 59
5	Segundo	0 a 61 (60 ou 61 para segundos em anos bissextos)
6	Dia da semana	0 a 6 (0 é segunda-feira)
7	Dia do ano	1 a 366 (dia juliano)
8	Horário de verão	-1, 0, 1, -1 significa que a biblioteca usada (time) determina o horário de verão

A tupla acima é equivalente à estrutura **struct_time** . Essa estrutura possui os seguintes atributos:

Índice	Atributos	Valores
0 0	tm_year	2020
1	tm_mon	1 a 12
2	tm_mday	1 a 31
3	tm_hour	0 a 23
4	tm_min	0 a 59

5	tm_sec	0 a 61 (60 ou 61 para segundos em anos bissextos)
6	tm_wday	0 a 6 (0 é segunda-feira)
7	tm_yday	1 a 366 (dia juliano)
8	tm_isdst	-1, 0, 1, -1 significa que a biblioteca determina o horário de verão

módulo *time*

Estes são as funções do módulo *time* – mencionado no início do nosso passo-a-passo

Função com Descrição
time.altzone O deslocamento do fuso horário local do horário de verão, em segundos a oeste do UTC, se um estiver definido. Isso passa a ser negativo se o fuso horário local do horário de verão estiver a leste do UTC (como na Europa Ocidental, incluindo o Reino Unido). Use-o somente se a luz do dia for diferente de zero.
time.asctime ([tupletime]) Aceita uma tupla de tempo e retorna uma sequência legível de 24 caracteres, como 'Ter 11 de dezembro, 18:07:14 2008'.
time.clock( ) Retorna o tempo atual da CPU como um número de ponto flutuante de segundos. Para medir os custos computacionais de diferentes abordagens, o valor de time.clock é mais útil que o de time.time ().

time.ctime ([segundos])

Como asctime (localtime (segundos)) e sem argumentos é como asctime ()

time.gmtime ([segundos])

Aceita um instante expresso em segundos desde a época e retorna uma tupla de tempo t com a hora UTC. Nota: t.tm_isdst é sempre 0

time.localtime ([segundos])

Aceita um instante expresso em segundos desde a época e retorna uma tupla de tempo t com a hora local (t.tm_isdst é 0 ou 1, dependendo se o horário de verão se aplica a segundos instantâneos pelas regras locais).

time.mktime (tupletime)

Aceita um instante expresso como uma tupla de tempo no horário local e retorna um valor de ponto flutuante com o instante expresso em segundos desde a época.

time.sleep (segundos)

Suspende o encadeamento de chamada por segundos.

time.strftime (fmt [, tupletime])

Aceita um instante expresso como uma tupla de tempo no horário local e retorna uma sequência que representa o instante conforme especificado pela sequência fmt.

time.strptime (str, fmt = '% a% b% d% H:% M:% S% Y')

Analisa str de acordo com o formato string fmt e retorna o instante no formato de tupla de tempo.

time.time ()

Retorna o instante de hora atual, um número de ponto flutuante de segundos desde a época.

time.tzset () Redefine as regras de conversão de horário usadas pelas rotinas da biblioteca. A variável de ambiente TZ especifica como isso é feito.

Existem dois atributos importantes disponíveis no módulo de tempo. Eles são:

Atributo com Descrição
time.timezone O atributo *time.timezone* é o deslocamento em segundos do fuso horário local (sem DST) do UTC (> 0 nas Américas; <= 0 na maior parte da Europa, Ásia, África).
time.tzname O atributo time.tzname é um par de sequências dependentes do código do idioma, que são os nomes do fuso horário local sem e com o horário de verão, respectivamente.

Obtendo hora atual – com sctruct_time

Para converter um instante de tempo de *segundos após o* valor do ponto flutuante *da época* em uma tupla de tempo, passe o valor do ponto flutuante para uma função (por exemplo, hora local) que retorna uma tupla de tempo com todos os nove itens válidos.

```
import time

localtime = time.localtime(time.time())
```

```
print ("Hora atual local :", localtime)
```

Isso produziria o seguinte resultado: *(considerando o momento do teste)*

```
Hora atual local : time.struct_time(tm_year=2020, tm_mon=5, tm_mday=14, tm_hour=18, tm_min=50, tm_sec=5, tm_wday=3, tm_yday=135, tm_isdst=0)
```

Obtendo hora formatada

Você pode formatar a qualquer momento, conforme sua exigência, mas um método simples para obter horas em um formato legível é **asctime ()**:

```
import time

localtime = time.asctime( time.localtime(time.time()) )
print ("Hora atual local :", localtime)
```

Isso produziria o seguinte resultado -

```
Hora atual local : Thu May 14 18:51:41 2020
```

O módulo do *calendar*

O módulo de calendário fornece funções relacionadas ao calendário, incluindo funções para imprimir um calendário de texto para um determinado mês ou ano.

Por padrão, o calendário assume segunda-feira como o primeiro dia da semana e domingo como o último. Para alterar isso, chame a função calendar.setfirstweekday ().

Aqui está uma lista de funções disponíveis com o módulo de *calendário* :

Função com Descrição

calendar.calendar (ano, w = 2, l = 1, c = 6)

Retorna uma sequência multilinha com um calendário para o ano do ano formatado em três colunas separadas por espaços em c. w é a largura em caracteres de cada data; cada linha tem comprimento 21 * w + 18 + 2 * c. l é o número de linhas para cada semana.

calendar.firstweekday ()

Retorna a configuração atual para o dia da semana que começa a cada semana. Por padrão, quando o calendário é importado pela primeira vez, é 0, o que significa segunda-feira.

calendar.isleap (ano)

Retorna True se ano é um ano bissexto; caso contrário, False.

calendar.leapdays (y1, y2)

Retorna o número total de dias bissextos nos anos dentro do intervalo (y1, y2).

calendar.month (ano, mês, w = 2, l = 1)

Retorna uma sequência multilinha com um calendário para mês mês do ano ano, uma linha por semana mais duas linhas de cabeçalho. w é a largura em caracteres de cada data; cada linha tem comprimento 7 * w + 6. l é o número de linhas para cada semana.

calendar.monthcalendar (ano, mês)

Retorna uma lista de listas de entradas. Cada sub-lista indica uma semana. Dias fora do mês mês do ano ano são definidos como 0; os dias do mês são definidos para o dia do mês 1 ou superior.

calendar.monthrange (ano, mês)

Retorna dois números inteiros. O primeiro é o código do dia da semana para o primeiro dia do mês mês no ano ano; o segundo é o número de dias no mês. Os códigos dos dias da semana são de 0 (segunda-feira) a 6 (domingo); os números dos meses são de 1 a 12.

calendar.prcal (ano, w = 2, l = 1, c = 6)

Como imprimir calendar.calendar (ano, w, l, c).

calendar.prmonth (ano, mês, w = 2, l = 1)

Como imprimir calendar.month (ano, mês, w, l).

calendar.setfirstweekday (weekday)

Define o primeiro dia de cada semana como código de dia da semana dia da semana. Os códigos dos dias da semana são de 0 (segunda-feira) a 6 (domingo).

calendar.timegm (tupletime)

O inverso de time.gmtime: aceita um instante de tempo na forma de tupla de tempo e retorna o mesmo instante que um número de segundos de ponto flutuante desde a época.

calendar.weekday (ano, mês, dia) Retorna o código do dia da semana para a data especificada. Os códigos dos dias da semana são de 0 (segunda-feira) a 6 (domingo); os números dos meses são de 1 (janeiro) a 12 (dezembro).

Obtendo calendário com um único mês

O módulo de calendário oferece uma ampla variedade de métodos para que possamos operar as datas; podemos trabalhar com calendários anuais e mensais. Aqui, imprimimos um calendário para um determinado mês (dezembro de 2019):

```
import calendar

calendário_mes = calendar.month(2019, 12)
print ("este é o calendário para o mês de dezembro:")
print (calendário_mes)
```

a execução do programa acima produziria o seguinte resultado:

```
Este é o calendário para o mês de dezembro:

    December 2019

Mo Tu We Th Fr Sa Su

                   1

 2  3  4  5  6  7  8

 9 10 11 12 13 14 15

16 17 18 19 20 21 22
```

```
23 24 25 26 27 28 29

30 31
```

Obtendo calendário de um ano inteiro

No próximo, vamos imprimir o calendário com todos os meses para o ano de 2019:

```
import calendar

calendário_ano = calendar.calendar(2019)
print ("este é o calendário com os meses do ano de 2019:")
print (calendário_ano)
```

a execução do programa acima produziria o seguinte resultado:

```
Este é o calendário com os meses do ano de 2019:

                                  2019


      January                   February                  March

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su

1  2  3  4  5  6                  1  2  3                  1  2  3

7  8  9 10 11 12 13       4  5  6  7  8  9 10       4  5  6  7  8  9 10

14 15 16 17 18 19 20      11 12 13 14 15 16 17      11 12 13 14 15 16 17

21 22 23 24 25 26 27      18 19 20 21 22 23 24      18 19 20 21 22 23 24

28 29 30 31               25 26 27 28               25 26 27 28 29 30 31


      April                     May                       June

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su

1  2  3  4  5  6  7            1  2  3  4  5                     1  2

8  9 10 11 12 13 14       6  7  8  9 10 11 12       3  4  5  6  7  8  9
```

```
15 16 17 18 19 20 21      13 14 15 16 17 18 19      10 11 12 13 14 15 16
22 23 24 25 26 27 28      20 21 22 23 24 25 26      17 18 19 20 21 22 23
29 30                     27 28 29 30 31            24 25 26 27 28 29 30


       July                    August                  September

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
1  2  3  4  5  6  7                1  2  3  4                         1
8  9 10 11 12 13 14       5  6  7  8  9 10 11       2  3  4  5  6  7  8
15 16 17 18 19 20 21      12 13 14 15 16 17 18       9 10 11 12 13 14 15
22 23 24 25 26 27 28      19 20 21 22 23 24 25      16 17 18 19 20 21 22
29 30 31                  26 27 28 29 30 31         23 24 25 26 27 28 29
30


     October                   November                  December

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
1  2  3  4  5  6                   1  2  3                      1
7  8  9 10 11 12 13       4  5  6  7  8  9 10       2  3  4  5  6  7  8
14 15 16 17 18 19 20      11 12 13 14 15 16 17       9 10 11 12 13 14 15
21 22 23 24 25 26 27      18 19 20 21 22 23 24      16 17 18 19 20 21 22
28 29 30 31               25 26 27 28 29 30         23 24 25 26 27 28 29
30 31
```

Função (Function)

Uma função é um bloco de código reutilizável organizado que é usado para executar uma única ação relacionada. As funções fornecem melhor modularidade para seu aplicativo e um alto grau de reutilização de código.

Como você já sabe, o Python oferece muitas funções internas, como print (), etc., mas você também pode criar suas próprias funções. Essas funções são chamadas *de funções definidas pelo usuário.*

Definindo uma Função

Você pode definir funções para fornecer a funcionalidade necessária. Aqui estão regras simples para definir uma função no Python.

- Os blocos de funções começam com a palavra-chave **def** seguida pelo nome da função e parênteses (()).
- Quaisquer parâmetros ou argumentos de entrada devem ser colocados entre parênteses. Você também pode definir parâmetros dentro desses parênteses.

Não é obrigatório passar argumentos para uma função

- A primeira declaração de uma função pode ser uma declaração opcional - a string de documentação da função ou *docstring* .
- O bloco de código em todas as funções começa com dois pontos (:) e é recuado.
- A declaração return [expression] sai de uma função, opcionalmente retornando uma expressão ao chamador. Uma declaração de retorno sem argumentos é igual a return None.

Sintaxe

```
def nomedafuncao( parametros ):
"function_docstring"
function_suite
return [expressao]
```

Por padrão, os parâmetros têm um comportamento posicional e você precisa informá-los na mesma ordem em que foram definidos.

Exemplo

A função a seguir pega uma string como parâmetro de entrada e a imprime na tela padrão.

```
def printme( str ):
      "Isso imprime uma string passada nessa função"
print (str)     return
```

Chamando uma função

Definir apenas uma função fornece um nome, especifica os parâmetros que devem ser incluídos na função e estrutura os blocos de código.

Uma vez finalizada a estrutura básica de uma função, você pode executá-la chamando-a de outra função ou diretamente do prompt do Python. A seguir está o exemplo para chamar a função printme () -

```
# A definição de função  def
printme( str ):
   " Isso imprime uma string passada nessa função"
print (str)    return

# Agora você pode chamar a função printme
printme("Esta é a primeira chamada para a função definida
pelo usuário!")
printme("Novamente segunda chamada para a mesma função")
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
Esta é a primeira chamada para a função definida pelo
usuário!
Novamente segunda chamada para a mesma função
```

Passagem por referência vs valor

Todos os parâmetros (argumentos) na linguagem Python são passados por referência. Isso significa que, se você alterar a que um parâmetro se refere dentro de uma função, a alteração também será refletida novamente na função de chamada. Por exemplo -

```
def changeme( mylist ):
   " Isso imprime uma string passada nessa função"    print
("Valores dentro da função antes da alteração:", mylist)
   mylist[2]=50
   print ("Valores dentro da função após alteração: ",mylist)
return

# Agora é possivel chamar a função changeme
mylist = [10,20,30] changeme( mylist )
print ("Valores fora da função: ", mylist)
```

Aqui, estamos mantendo a referência do objeto passado e acrescentando valores no mesmo objeto. Portanto, isso produziria o seguinte resultado –

```
Valores dentro da função antes da alteração: [10, 20,
30]
Valores dentro da função após alteração:  [10, 20, 50]
Valores fora da função:  [10, 20, 50]
```

Há mais um exemplo em que o argumento está sendo passado por referência e a referência está sendo substituída dentro da função chamada.

```
def changeme( mylist ):
   " Isso altera uma lista passada para esta função "
mylist = [1,2,3,4] # Isso ajudaria nova referência em
mylist
   print ("Valores dentro da função: ", mylist)
return

# Agora é possivel chamar a função

mylist = [10,20,30] changeme(
mylist )
print ("Valores fora da função: ", mylist)
```

O parâmetro *mylist* é local para a função changeme. Alterar minha lista dentro da função não afeta *minha lista* . A função não realiza nada e, finalmente, isso produziria o seguinte resultado -

```
Valores dentro da função:  [1, 2, 3, 4]
Valores fora da função:  [10, 20, 30]
```

Argumentos de Função

Você pode chamar uma função usando os seguintes tipos de argumentos formais -

- Argumentos obrigatórios
- Argumentos de palavras-chave
- Argumentos padrão
- Argumentos de tamanho variável

Argumentos obrigatórios

Argumentos necessários são os argumentos passados para uma função na ordem posicional correta. Aqui, o número de argumentos na chamada de função deve corresponder exatamente à definição da função.

Para chamar a função *printme ()* , você definitivamente precisa passar um argumento, caso contrário, *ocorrerá* um erro de sintaxe da seguinte maneira:

```
# A definição de função def
printme( str ):
   " Isso imprime uma string passada nessa função"
print str    return;

# Agora é possivel chamar a função printme  printme()
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
     Traceback (most recent call last):
        line 11, in <module>
printme();
     TypeError: printme() takes exactly 1 argument (0 given)
```

Argumentos de palavras-chave

Os argumentos de palavra-chave estão relacionados às chamadas de função. Quando você usa argumentos de palavra-chave em uma chamada de função, o chamador identifica os argumentos pelo nome do parâmetro.

Isso permite ignorar argumentos ou colocá-los fora de ordem, porque o intérprete Python pode usar as palavras-chave fornecidas para combinar os valores com os parâmetros. Você também pode fazer chamadas de palavraschave para a função *printme ()* das seguintes maneiras -

```
     def printme( str ):
        "Isso imprime uma string passada nessa função"
print (str)    return

     # Agora é possivel chamar a função printme
printme( str = "Minha nova string")
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
     Minha nova string
```

O exemplo a seguir fornece uma imagem mais clara. Observe que a ordem dos parâmetros não importa.

```
# A definição de função def
printinfo( nome, idade ):
   " Isso imprime uma string passada nessa função"
print ("Nome: ", nome)    print ("Idade ", idade)
return

# Agora é possivel chamar a função printinfo  printinfo(
idade = 31, nome = "Saul Goodman" )
```

Quando o código acima é executado, ele produz o seguinte resultado

```
Nome:  Saul Goodman
Idade: 31
```

Argumentos padrão

Um argumento padrão é um argumento que assume um valor padrão se um valor não for fornecido na chamada de função para esse argumento. O exemplo a seguir fornece uma idéia dos argumentos padrão, ele imprime a idade padrão se não for aprovada -

```
# A definição de função def
printinfo( nome, idade = 35 ):
   " Isso imprime uma string passada nessa
função"    print ("Nome: ", nome)    print
("Idade ", idade)    return

# Agora é possivel chamar a função printinfo  printinfo(
idade = 31, name = "Saul Goodman" ) printinfo( nome =
"Walter White" )
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Name:  Saul Goodman
Age  31
Name:  Walter White
Age  35
```

Argumentos de tamanho variável

Pode ser necessário processar uma função para mais argumentos do que você especificou ao definir a função. Esses argumentos são chamados argumentos de *tamanho variável* e não são nomeados na definição da função, diferentemente dos argumentos obrigatórios e padrão.

A sintaxe para uma função com argumentos de variável que não é de palavrachave é esta -

```
def functionname([formal_args,] *var_args_tuple ):
```

```
"function_docstring"
function_suite
return [expression]
```

Um asterisco (*) é colocado antes do nome da variável que contém os valores de todos os argumentos de variáveis que não são palavras-chave. Essa tupla permanece vazia se nenhum argumento adicional for especificado durante a chamada de função. A seguir, um exemplo simples:

```
# A definição de função def
printinfo( arg1, *vartuple ):
   " Isso imprime argumentos variáveis passados "
print ("A saída é: ")     print (arg1)
       for var in
vartuple:
      print (var)
return

# Agora é possivel chamar a função
printinfo printinfo( 10 ) printinfo( 70,
60, 50 )
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
A saida é:
10
A saida é:
70
60
50
```

As funções *anônimas*

Essas funções são chamadas anônimas porque não são declaradas da maneira padrão usando a palavra-chave *def*. Você pode usar a palavrachave *lambda* para criar pequenas funções anônimas.

- Os formulários Lambda podem receber qualquer número de argumentos, mas retornam apenas um valor na forma de uma expressão. Eles não podem conter comandos ou várias expressões.
- Uma função anônima não pode ser uma chamada direta para impressão porque lambda requer uma expressão Ou funções
- As funções do Lambda têm seu próprio espaço para nome local e não podem acessar variáveis além daquelas na lista de parâmetros e no espaço para nome global.
- Embora pareça que os lambda são uma versão de uma linha de uma função, eles não são equivalentes às instruções embutidas em C ou C ++, cujo objetivo é passar a alocação da pilha de funções durante a chamada por motivos de desempenho.

Sintaxe

A sintaxe das funções *lambda* contém apenas uma única instrução, que é a seguinte:

```
lambda [arg1 [,arg2,.....argn]]:expression
```

A seguir, é apresentado o exemplo para mostrar como a forma de função *lambda* funciona:

```
     # A definição de função
     soma = lambda arg1, arg2: arg1 + arg2
     # Agora é possível chamar a função soma como uma
função print ("Valor do total da soma : ", soma( 10, 20 ))
print ("Valor do total da soma : ", soma( 20, 20 ))
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Valor do total da soma:  30
Valor do total da soma:  40
```

A declaração de *retorno*

A declaração return [expression] sai de uma função, opcionalmente retornando uma expressão ao chamador. Uma declaração de retorno sem argumentos é igual a return None.

Todos os exemplos acima não estão retornando nenhum valor. Você pode retornar um valor de uma função da seguinte maneira -

```
# A definição de função def
soma( arg1, arg2 ):
   # Adicionar os dois parâmetros e retornando-os."
total = arg1 + arg2
   print ("Dentro da função : ", total)
return total

# Agora é possível chamar a função soma  total
= soma( 10, 20 )
print ("Fora da função : ", total )
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
     Dentro da função :
30 Fora da função :  30
```

Escopo das Variáveis

Todas as variáveis em um programa podem não estar acessíveis em todos os locais desse programa. Isso depende de onde você declarou uma variável.

O escopo de uma variável determina a parte do programa em que você pode acessar um identificador específico. Existem dois escopos básicos de variáveis no Python:

- Variáveis globais
- Variáveis locais

Variáveis globais vs. locais

Variáveis definidas dentro de um corpo de função têm um escopo local e aquelas definidas fora têm um escopo global.

Isso significa que variáveis locais podem ser acessadas apenas dentro da função em que são declaradas, enquanto variáveis globais podem ser acessadas em todo o corpo do programa por todas as funções. Quando você chama uma função, as variáveis declaradas dentro dela são colocadas no escopo. A seguir, um exemplo simples:

```
total = 0   # Essa é uma variável global.
# A definição de função def
soma( arg1, arg2 ):
   # # Adicionar os dois parâmetros e retornando-os."
total = arg1 + arg # Aqui total é uma variável local
print ("Dentro da função local total : ", total)    return
total

# Agora é possível chamar a função soma soma(
10, 20 )
print ("Fora da função global total : ", total )
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
Dentro da função local total:  30
Fora da função global total:  0
```

Files I/O (Manipulação de arquivos)

Neste passo abordaremos todas as funções básicas de I / O disponíveis no Python, a partir da versão 3.

Imprimir na tela

A maneira mais simples de produzir resultados é usar a instrução *print* , na qual você pode passar zero ou mais expressões separadas por vírgulas. Essa função converte as expressões que você passa em uma string e grava o resultado na saída padrão da seguinte maneira -

```
print ("Se os fatos não se encaixam na teoria, modifique os fatos.")
```

Isso produz o seguinte resultado de saida

```
Se os fatos não se encaixam na teoria, modifique os fatos.
```

Leitura da entrada do teclado

O Python 2 possui duas funções internas para ler dados da entrada padrão, que por padrão vem do teclado. Essas funções são **input ()** e **raw_input ()**

No Python 3, a função raw_input () está obsoleta. Além disso, as funções input () lêem os dados do teclado como string, independentemente de estarem entre aspas (" ou "") ou não.

A função de entrada

A função **input ([prompt])** é equivalente a raw_input, exceto pelo fato de assumir que a entrada é uma expressão válida do Python e retornar o resultado avaliado para você.

```
      x = input("Insira Algo:")
      Insira Algo:10

      x
      '10'

      x = input("Insira Algo:")
      Isira Algo:'10' # dados inseridos tratados como string com
ou sem aspas ''
      x
      "'10'"
```

Abrindo e fechando arquivos

Até agora, você estava lendo e gravando na entrada e saída padrão. Agora, veremos como usar arquivos de dados reais.

O Python fornece funções e métodos básicos necessários para manipular arquivos por padrão. Você pode executar a maior parte da manipulação de arquivos usando um objeto de **arquivo** .

A função open

Antes de poder ler ou gravar um arquivo, você deve abri-lo usando a função open () interna do Python. Essa função cria um objeto de **arquivo** , que seria utilizado para chamar outros métodos de suporte associados a ele.

Sintaxe

```
file object = open(file_name [, access_mode][,
buffering])
```

Aqui estão os detalhes do parâmetro:

- **file_name** - o argumento file_name é um valor de sequência que contém o nome do arquivo que você deseja acessar.
- **access_mode** - O access_mode determina o modo em que o arquivo deve ser aberto, isto é, ler, escrever, acrescentar, etc. Uma lista completa dos possíveis valores é fornecida abaixo na tabela. Este é um parâmetro opcional e o modo de acesso a arquivos padrão é lido (r).
- **buffer** - Se o valor do buffer estiver definido como 0, nenhum buffer será realizado. Se o valor do buffer for 1, o buffer da linha será executado ao acessar um arquivo. Se você especificar o valor do buffer como um número inteiro maior que 1, a ação do buffer será executada com o tamanho do buffer indicado. Se negativo, o tamanho do buffer é o padrão do sistema (comportamento padrão).

Aqui está uma lista dos diferentes modos de abrir um arquivo :

Modo e descrição
r Abre um arquivo apenas para leitura. O ponteiro do arquivo é colocado no início do arquivo. Este é o modo padrão.
rb Abre um arquivo para leitura apenas em formato binário. O ponteiro do arquivo é colocado no início do arquivo. Este é o modo padrão.
r + Abre um arquivo para leitura e gravação. O ponteiro do arquivo colocado no início do arquivo.
rb + Abre um arquivo para leitura e gravação em formato binário. O ponteiro do arquivo colocado no início do arquivo.
W Abre um arquivo apenas para gravação. Substitui o arquivo se o arquivo existir. Se o arquivo não existir, cria um novo arquivo para gravação.
wb Abre um arquivo para gravação apenas em formato binário. Substitui o arquivo se o arquivo existir. Se o arquivo não existir, cria um novo arquivo para gravação.
w + Abre um arquivo para escrever e ler. Substitui o arquivo existente, se o arquivo existir. Se o arquivo não existir, cria um novo arquivo para leitura e gravação.
wb + Abre um arquivo para escrever e ler em formato binário. Substitui o arquivo existente, se o arquivo existir. Se o arquivo não existir, cria um novo arquivo para leitura e gravação.

a Abre um arquivo para anexar. O ponteiro do arquivo está no final do arquivo, se o arquivo existir. Ou seja, o arquivo está no modo de acréscimo. Se o arquivo não existir, ele criará um novo arquivo para gravação.
ab Abre um arquivo para anexar em formato binário. O ponteiro do arquivo está no final do arquivo, se o arquivo existir. Ou seja, o arquivo está no modo de acréscimo. Se o arquivo não existir, ele criará um novo arquivo para gravação.
a + Abre um arquivo para anexar e ler. O ponteiro do arquivo está no final do arquivo, se o arquivo existir. O arquivo é aberto no modo de acréscimo. Se o arquivo não existir, ele criará um novo arquivo para leitura e gravação.
ab + Abre um arquivo para anexar e ler em formato binário. O ponteiro do arquivo está no final do arquivo, se o arquivo existir. O arquivo é aberto no modo de acréscimo. Se o arquivo não existir, ele criará um novo arquivo para leitura e gravação.

O arquivo Object Attributes

Uma vez que um arquivo é aberto e você tem um *arquivo de* objeto, você pode obter várias informações relacionadas a esse arquivo.

Aqui está uma lista de todos os atributos relacionados a um objeto de arquivo:

Atributo e descrição
file.closed Retorna true se o arquivo estiver fechado, false caso contrário.
file.mode Retorna o modo de acesso com o qual o arquivo foi aberto.
File.name Retorna o nome do arquivo.

Nota - o atributo softspace não é suportado no Python 3.x

Exemplo

```
# abrindo o arquivo usando a função open
umArq = open("umArq.txt", "wb")
print ("Nome do arquivo ", umArq.name)
print ("Fechado: ", umArq.closed)
print ("Modo de abertura : ", umArq.mode)
umArq.close()
```

Isso produz o seguinte resultado -

```
Nome do arquivo:  umArq.txt
Fechado  :  False
Modo de Abertura :  wb
```

O método close ()

O método close () de um objeto de arquivo limpa qualquer informação não escrita e fecha o objeto de arquivo, após o qual não é mais possível escrever.

O Python fecha automaticamente um arquivo quando o objeto de referência de um arquivo é reatribuído para outro arquivo. É uma boa prática usar o método close () para fechar um arquivo.

Sintaxe

```
fileObject.close();
```

Exemplo

```
# abrindo o arquivo usando a função open
umArq = open("umArq.txt", "wb")
print ("Nome do arquivo ", umArq.name)

# Fechando o arquivo aberto
umArq.close()
```

Isso produz o seguinte resultado -

```
Nome do arquivo:  umArq.txt
```

Lendo e gravando arquivos

O objeto file fornece um conjunto de métodos de acesso para facilitar nossa vida. Veríamos como usar os métodos **read ()** e **write ()** para ler e gravar arquivos.

O método write ()

O método **write ()** grava qualquer string em um arquivo aberto. É importante observar que as seqüências de caracteres Python podem ter dados binários e não apenas texto.

O método write () não adiciona um caractere de nova linha ('\ n') ao final da string

Sintaxe

```
fileObject.write(string);
```

Aqui, o parâmetro passado é o conteúdo a ser gravado no arquivo aberto.

Exemplo

```
# abrindo o arquivo usando a função open
umArq = open("umArq.txt", "w")
umArq.write(" Se, a princípio, a ideia não é absurda,\n
então não há esperança para ela.\n")
# Fechando o arquivo aberto
umArq.close()
```

O método acima criaria o arquivo foo.txt e gravaria o conteúdo fornecido nesse arquivo e, finalmente, fecharia o arquivo. Se você abrir esse arquivo, ele terá o seguinte conteúdo:

```
Se, a princípio, a ideia não é absurda,
então não há esperança para ela.
```

O método read ()

O método **read ()** lê uma string de um arquivo aberto. É importante observar que as seqüências de caracteres Python podem ter dados binários. além dos dados de texto.

Sintaxe

```
fileObject.read([count]);
```

Aqui, parâmetro passado é o número de bytes a serem lidos no arquivo aberto. Esse método começa a ler desde o início do arquivo e, se estiver faltando *contagem* , ele tenta ler o máximo possível, talvez até o final do arquivo.

Exemplo

Vamos pegar um arquivo *foo.txt* , que criamos acima.

```
# abrindo o arquivo usando a função open
umArq = open("umArq.txt", "r+")
str = umArq.read(10)
print ("A leitura da string é : ", str)

# Fechando o arquivo aberto
umArq.close()
```

Isso produz o seguinte resultado:

```
A leitura da string:  Se, a prin
```

Posições de arquivo

O método *tell ()* informa a posição atual no arquivo; em outras palavras, a próxima leitura ou gravação ocorrerá com muitos bytes desde o início do arquivo.

O método de *busca (deslocamento [, de])* altera a posição atual do arquivo. O argumento de **deslocamento** indica o número de bytes a serem movidos. O argumento **from** especifica a posição de referência de onde os bytes devem ser movidos.

Se *from* estiver definido como 0, o início do arquivo será usado como posição de referência. Se estiver definido como 1, a posição atual será usada como posição de referência. Se estiver definido como 2, o final do arquivo será considerado a posição de referência.

Exemplo

Vamos pegar um arquivo *foo.txt* , que criamos acima.

```
# abrindo o arquivo usando a função open
umArq = open("umArq.txt", "r+")
str = umArq.read(10)
print ("A leitura da string é: ", str)

# Verificando a posição atual
position = umArq.tell()
print ("Posição atual do arquivo : ", position)

# Reposicionar o ponteiro no início, novamente
position = umArq.seek(0, 0)
str = umArq.read(10)
print ("Novamente lendo a String : ", str)

# Fechando o arquivo aberto
umArq.close()
```

Isso produz o seguinte resultado

```
A leitura da string:  Se, a prin
Posição atual do arquivo:  10
Novamente lendo a String:  Se, a prin
```

Renomeando e excluindo arquivos

O módulo **OS do** Python fornece métodos que ajudam a executar operações de processamento de arquivos, como renomear e excluir arquivos.

Para usar este módulo, você precisa importá-lo primeiro e depois chamar as funções relacionadas.

O método rename ()

O método **rename ()** usa dois argumentos, o nome do arquivo atual e o novo nome do arquivo.

Sintaxe

```
os.rename(current_file_name, new_file_name)
```

Exemplo

A seguir, é apresentado um exemplo para renomear um arquivo existente *test1.txt*

```
#!/usr/bin/python3
import os

# Renomeie um arquivo de umArq.txt para text2.txt
os.rename( "umArq.txt", "text2.txt" )
```

O método remove ()

Você pode usar o método **remove ()** para excluir arquivos, fornecendo o nome do arquivo a ser excluído como argumento.

Sintaxe

```
os.remove(file_name)
```

Exemplo

A seguir, é apresentado um exemplo para excluir um arquivo existente *test2.txt*:

```
import os

# Excluir arquivo text2.txt
os.remove("text2.txt")
```

Diretórios em Python

Todos os arquivos estão contidos em vários diretórios, e o Python também não tem problemas em lidar com eles. O módulo **os** possui vários métodos que ajudam a criar, remover e alterar diretórios.

O método mkdir ()

Você pode usar o método **mkdir ()** do módulo **os** para criar diretórios no diretório atual. Você precisa fornecer um argumento para esse método, que contém o nome do diretório a ser criado.

Sintaxe

```
os.mkdir("newdir")
```

Exemplo

A seguir, é apresentado um exemplo para criar um *teste de* diretório no diretório atual

```
import os

# Crie um diretório "test"
os.mkdir("MinhaPasta")
```

O método getcwd ()

O método **getcwd ()** exibe o diretório de trabalho atual.

Sintaxe

```
os.getcwd()
```

Exemplo

A seguir, é apresentado um exemplo para fornecer o diretório atual:

```
#!/usr/bin/python3
import os

# Mostra a localização do diretório atual
os.getcwd()
```

O método rmdir ()

O método **rmdir ()** exclui o diretório, que é passado como argumento no método.

Antes de remover um diretório, todo o conteúdo nele deve ser removido.

Sintaxe

```
os.rmdir('dirname')
```

Exemplo

A seguir, é apresentado um exemplo para remover o diretório "/ tmp / test". É necessário fornecer um nome completo do diretório, caso contrário, ele procuraria esse diretório no diretório atual.

```
import os

# remove um diretório de onde você escolher.
os.rmdir( " C:/Users/dir/Desktop/nomedapasta"  )
```

Módulos

Um módulo permite que você organize logicamente seu código Python. O agrupamento do código relacionado em um módulo facilita a compreensão e o uso do código. Um módulo é um objeto Python com atributos nomeados arbitrariamente que você pode vincular e referenciar.

Simplesmente, um módulo é um arquivo que consiste em código Python. Um módulo pode definir funções, classes e variáveis. Um módulo também pode incluir código executável.

Exemplo

O código Python para um módulo chamado *namemodule, por exemplo,* normalmente reside em um arquivo chamado *namemodule.py* . Aqui está um exemplo de um módulo simples, *support.py*

```
def print_func( word ):

   print("Olá : ", word)

   return
```

A declaração de *importação*

Você pode usar qualquer arquivo de origem Python como módulo executando uma instrução de importação em algum outro arquivo de origem Python. A *importação* possui a seguinte sintaxe:

```
import module1[, module2[,... moduleN]
```

Quando o intérprete encontra uma instrução de importação, importa o módulo se o módulo estiver presente no caminho de pesquisa. Um caminho de pesquisa é uma lista de diretórios que o intérprete pesquisa antes de importar um módulo. Por exemplo, para importar o módulo *support.py*, você precisa colocar o seguinte comando na parte superior do script:

```
# Import do module support
import support

# Agora você pode chamar a função definida desse módulo
da seguinte maneira
support.print_func("Lucinda")
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Olá : Lucinda
```

Um módulo é carregado apenas uma vez, independentemente do número de vezes que é importado. Isso evita que a execução do módulo ocorra repetidamente se ocorrerem várias importações.

A declaração from ... import

A instrução *from* do Python permite importar atributos específicos de um módulo para o namespace atual. A *importação from ...* possui a seguinte sintaxe:

```
from modname import name1[, name2[, ... nameN]]
```

Por exemplo, para importar a função fibonacci do módulo fib, use a seguinte instrução -

```
from fib import fibonacci
```

Esta declaração não importa todo o módulo *fib* para o *namespace* atual; apenas introduz o item *fibonacci* do módulo *fib* na tabela de símbolos global do módulo de importação.

```
# módulo números Fibonacci

def fib(n): # retorna a série de Fibonacci até n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result
```

Agora é preciso fazer a importação do módulo para poder utiliza-lo. Observe a instrução abaixo:

```
from fib import fib

fib(100)
```

ao executar o código acima, o resultado é exibido da seguinte forma:

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

A declaração from ... import *

Também é possível importar todos os nomes de um módulo para o namespace atual usando a seguinte instrução de importação -

*from modname import **

Isso fornece uma maneira fácil de importar todos os itens de um módulo para o namespaceatual; no entanto, essa declaração deve ser usada com moderação.

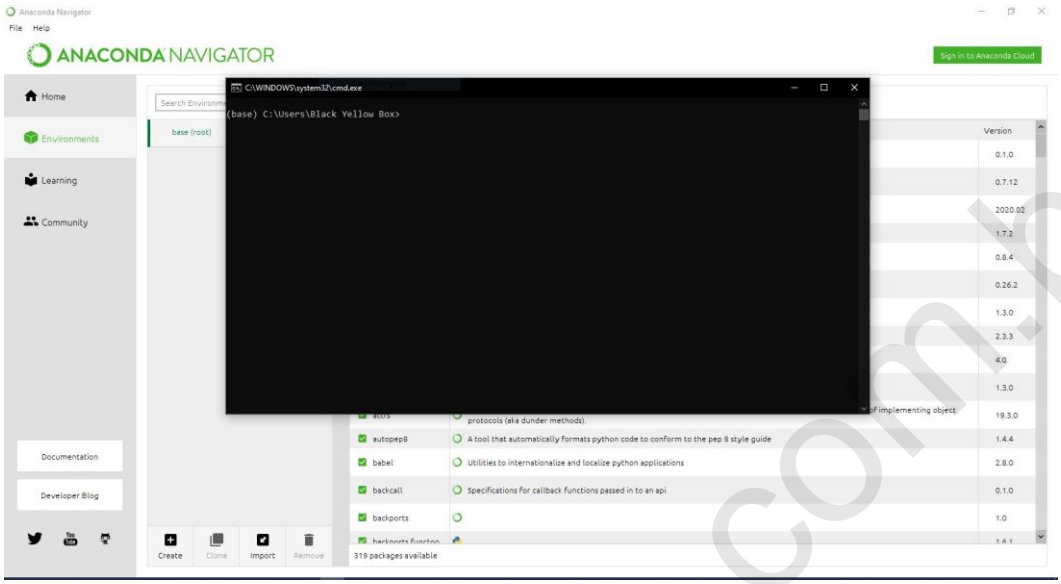
Executando módulos como scripts

Dentro de um módulo, o nome do módulo (como uma sequência) está disponível como o valor da variável global __name__. O código no módulo será executado, como se você o importasse, mas com o __name__ definido como "__main__".

Adicione o código – indicado abaixo - ao final do seu módulo.:

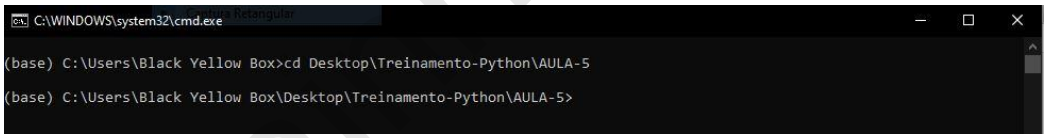
```
# módulo números Fibonacci



def fib(n): # retorna a série de Fibonacci até n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result

# este é o trecho a ser adicionado
if __name__ == "__main__":
    f = fib(100)
    print(f)
```

Na sequencia, abra o prompt de comando do Anaconda, como mostrado abaixo – através do Anaconda Navigator, como mostrado abaixo:



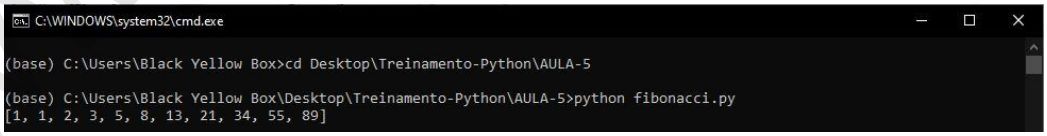
É preciso seguir o caminho até que você esteja exibindo, através do prompt, o caminho onde se encontra o arquivo. Como exibido abaixo:

C:\Users\seu_usuario\Desktop\pasta_que_contem_o_arquivo>:



Agora, basta executar o seguinte comando python:

C:\Users\seu_usuario\Desktop\pasta_que_contem_o_arquivo>python fibonacci.py. Observe a imagem abaixo:



O resultado é mostrado logo após a instrução e chamada.

Localizando Módulos

Quando você importa um módulo, o intérprete Python procura pelo módulo nas seguintes seqüências :

- O diretório atual.

- Se o módulo não for encontrado, o Python pesquisará cada diretório na variável de shell PYTHONPATH.
- Se tudo mais falhar, o Python verifica o caminho padrão. No UNIX, esse caminho padrão é normalmente / usr / local / lib / python /.

O caminho de pesquisa do módulo é armazenado no módulo do sistema sys como a variável **sys.path** . A variável sys.path contém o diretório atual, PYTHONPATH, e o padrão dependente da instalação.

A variável *PYTHONPATH*

O PYTHONPATH é uma variável de ambiente, consistindo em uma lista de diretórios. A sintaxe de PYTHONPATH é a mesma da variável de shell PATH.

Aqui está um PYTHONPATH típico de um sistema Windows:

```
set PYTHONPATH = c:\python34\lib;
```

E aqui está um PYTHONPATH típico de um sistema UNIX -

```
set PYTHONPATH = /usr/local/lib/python
```

Namespaces e Escope

Variáveis são nomes (identificadores) que são mapeados para objetos. Um *namespace* é um dicionário de nomes de variáveis (key) e seus objetos correspondentes (value).

Uma instrução Python pode acessar variáveis em um *namespace local* e no *namespace global* . Se uma variável local e uma global tiverem o mesmo nome, a variável local sombreará a variável global.

Cada função possui seu próprio *namespace* local. Os métodos de classe seguem a mesma regra de escopo que as funções comuns.

Python faz suposições informadas sobre se as variáveis são locais ou globais. Ele assume que qualquer variável atribuída a um valor em uma função é local.

Portanto, para atribuir um valor a uma variável global dentro de uma função, você deve primeiro usar a instrução global.

A declaração *global VarName* diz ao Python que *VarName* é uma variável global. O Python para de procurar a variável no espaço de nomes local.

Por exemplo, definimos uma variável *Money* no namespace global. Dentro da função *M oney*, atribuímos um valor a *Money* , portanto, o Python assume *Money* como uma variável loc al. No entanto, acessamos o valor da variável local *Money* antes de defini-la, portanto, um Unbo undLocalError é o resultado (`UnboundLocalError: local variable 'Money' refer enced before assignment`

).

Remover o comentário da instrução global corrige o problema.

```
     Money = 2000
     def AddMoney():
# Remova o comentário da seguinte linha para corrigir o
código:
        # global Money
        Money = Money + 1
     print (Money)
     AddMoney()
     print (Money)
     ######################################################
# Removendo o comentário
     Money = 2000
     def AddMoney():
       # Remova o comentário da seguinte linha para corrigir o
código:
        global Money
        Money = Money + 1
     print (Money)
     AddMoney()
     print (Money)
```

O resultado é:

```
2000

2001
```

A função dir ()

A função interna dir () retorna uma lista classificada de cadeias contendo os nomes definidos por um módulo.

A lista contém os nomes de todos os módulos, variáveis e funções definidas em um módulo. A seguir, um exemplo simples -

```
# Importando o module math interno

import math



content = dir(math)

print (content)
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
['__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'tau', 'trunc']
```

Aqui, a variável de cadeia especial *__name__* é o nome do módulo e *__file__* é o nome do arquivo no qual o módulo foi carregado.

As funções globals () e locals ()

As funções *globals ()* e *locals ()* podem ser usadas para retornar os nomes nos namespaces globais e locais, dependendo do local de onde são chamados.

Se locals () for chamado de dentro de uma função, ele retornará todos os nomes que podem ser acessados localmente a partir dessa função.

Se globals () for chamado de dentro de uma função, ele retornará todos os nomes que podem ser acessados globalmente a partir dessa função.

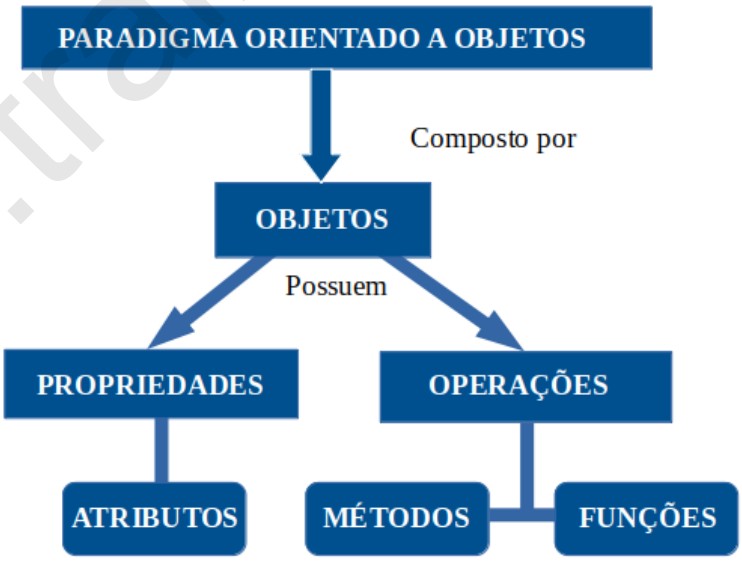
O tipo de retorno de ambas as funções é dicionário. Portanto, os nomes podem ser extraídos usando a função keys ().

Orientação a Objetos com Python

Par falarmos de orientação a objetos é importate que iniciemos com a definição de paradigma:

- Paradigma é a forma que vamos programar e executar o nosso software.
- Paradigma procedural: é baseado no conceito de chamados e procedimentos conhecidos como rotinas, funções, métodos.
- Paradigma OO: composto por objetos que possuem propriedades que são.

Vamos observar o diagrama abaixo – ele ilustra o paradigma de orientação a objetos:



Vamos pensar numa abordagem para enterdermos melhor como funciona o paradigma de programção orientada a objetos:

Premissa: construir um carro com foco em algumas funcionalidades básicas, por exemplo:

- frear
- acelerar
- trocar de marcha

Por que partir de uma premissa com esse contexto? A reposta é que o paradigma de programação orientada a objetos possui – por essência – a intenção de aproximar o desenvolvimento de softwares daquilo que conhecemos como mundo real.

Abaixo, podemos observar algumas das vantagens de se programar a partir do paradigma de Orientação a objetos:

Confiabilidade - o isolamento entre as partes de cada “pedaço” de um sistema que, a partir de uma alteração não, necessariamente, afeta outras parte do mesmo sistema; além disso, os componentes do projeto podem ser desenvolvidos paralelamente.

Facil de manter - permite maior facilidade nos processo de atualização de código.

Extensível – é possivel “acoplar” novas funcionalidades ao sistema a partir dos objetos já criados.Se seguirmos o contexto da criação e um carro poderíamos entender da seguinte forma: criamos uma classe Carro para representar o carro. Dessa forma é possível utilizar essa mesma classe para criar um sistema que possa trabalhar com venda de carros; novamente, podemos utilizar para criar um sistema que funcione para agendar tarefas para reparo de carros – oficinas mecânicas. Entre outros projetos.

Classe e Objeto

O que é um objeto? É algo material ou abstrato que pode ser percebido pelos sentidos e descrito por meio das suas caracteristicas, comportamentos e estado atual (status) Por exemplo “o carro”:

Podemos descrever o carro da seguinte forma:

- características: marca, fabricante(montadora), cor; considerando este cenário, entendemos que as características enumeradas são atributos do carro – por exemplo: cor = amarelo, branco, azul
- comportamentos: vamos considerar as funcionalidades que citamos acima: acelerar, frear, trocar de marcha
- estado atual: é representado pelos valores dos atributos naquele momento que é analisado.

Então, entendemos que é possível ter um modelo de carro e, esse modelo, pode possuir dieferentes características e funcionalidades. Esso modelo – molde, formato - chamamos de classe.

CARRO = OBJETO MOLDE = CLASSE

Os métodos pertencem a uma classe métodos = acelerar, frear, trocar a marcha, aumentar velocidade; através dos métodos é possível alterar os atributos do objeto.

Todo objeto “nasce” a partir de uma classe, a partir de um molde/modelo

A classe define quem são os atributos e métodos comuns que serão compartilhados por um objeto.

Instanciar o que é isso?

Quando temos uma classe e queremos gerar um objeto a partir dela então você faz o instanciamento.

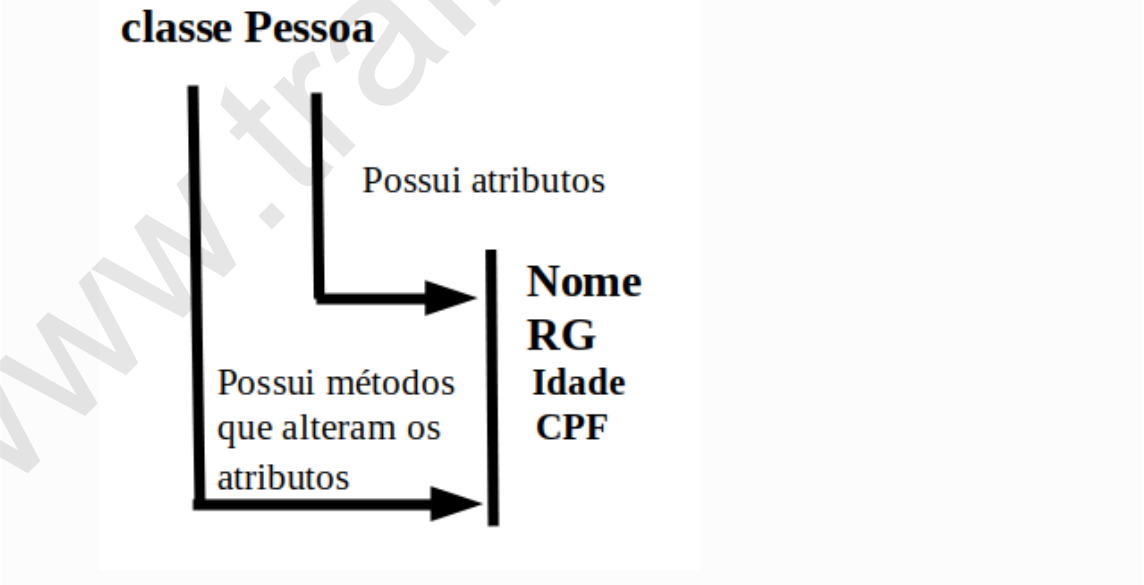
Instanciar é gerar um objeto a partir de uma classe:

c = Carro() –> c passa a ser a instância da Classe Carro

O que é uma classe?

Classe é um termo usado para um ***tipo de objeto***. As classes descrevem os objetos A classe serve como modelo, como molde para armazenar informações e realizar tarefas.

Uma classe pode conter vários elementos



Podemos ter várias classes, os objetos dessas classes são instanciados de forma que a execução do programa é vista como um conjunto de objetos relacionados que se comunicam enviando mensagens uns para outros.

No contexto Python

Python é uma linguagem orientada a objetos desde sua criação. Por esse motivo, é fácil criar e usar classes e objetos. Este passo ajuda você a se tornar um especialista no uso do suporte à programação orientada a objetos do Python.

Visão geral da terminologia OOP

- **Classe** (Class)- Um protótipo definido pelo usuário para um objeto que define um conjunto de atributos que caracterizam qualquer objeto da classe. Os atributos são membros de dados (variáveis de classe e variáveis de instância) e métodos, acessados via notação de ponto.
- **Variável de classe (Class Variable)** - uma variável compartilhada por todas as instâncias de uma classe. Variáveis de classe são definidas dentro de uma classe, mas fora de qualquer método da classe. Variáveis de classe não são usadas com tanta freqüência quanto as variáveis de instância.
- **Membro de dados** (data member) - Uma variável de classe ou variável de instância que contém dados associados a uma classe e seus objetos.
- **Sobrecarga de função (function Overloading)** - A atribuição de mais de um comportamento a uma função específica. A operação executada varia de acordo com os tipos de objetos ou argumentos envolvidos.
- **Variável de instância** (instance variable) - uma variável que é definida dentro de um método e pertence apenas à instância atual de uma classe.
- **Herança (inheritance)** - A transferência das características de uma classe para outras classes derivadas dela.
- **Instância (instance)** - Um objeto individual de uma determinada classe. Um objeto obj que pertence a uma classe Circle, por exemplo, é uma instância da classe Circle.
- **Instanciação** (instantition) - A criação de uma instância de uma classe.
- **Método (method)** - Um tipo especial de função que é definida em uma definição de classe.
- **Objeto (object)** - uma instância exclusiva de uma estrutura de dados definida por sua classe. Um objeto compreende membros de dados (variáveis de classe e variáveis de instância) e métodos.
- **Sobrecarga de operador (operator overloading)** - A atribuição de mais de uma função a um operador específico.

Criando Classes

A instrução de *classe* cria uma nova definição de classe. O nome da classe segue imediatamente a palavra-chave *classe* seguida por dois pontos da seguinte maneira -

```
class ClassName:

   'Optional class documentation string'

   class_suite
```

- A classe possui uma sequência de documentação, que pode ser acessada via *ClassName .__ doc__* .
- O *class_suite* consiste em todas as instruções do componente que definem os membros da classe, atributos de dados e funções.

Exemplo

A seguir está o exemplo de uma classe Python simples

```
class Employee:
   'Classe base comum para todos os funcionários'
   empCount = 0

   def __init__(self, nome, salario):
      self.nome = nome
      self.salario = salario
      Employee.empCount += 1

   def displayCount(self):
     print ("Total de funcionários %d" %
Employee.empCount)

   def displayEmployee(self):
      print ("Nome : ", self.nome,  ", Salario: ",
self.salario)
```

- A variável *empCount* é uma variável de classe cujo valor é compartilhado entre todas as instâncias dessa classe. Isso pode ser acessado como *Employee.empCount* de dentro da classe ou fora dela.
- O primeiro método *__init__ ()* é um método especial, chamado construtor de classe ou método de inicialização que o Python chama quando você cria uma nova instância dessa classe.

Você declara outros métodos de classe, como funções normais, com a exceção de que o primeiro argumento para cada método é *próprio* . Python adiciona o argumento *próprio* à lista para você; você não precisa incluí-lo quando chama os métodos.

Criando objetos de instância (Instance Objects)

Para criar instâncias de uma classe, você a chama usando o nome da classe e passa os argumentos que o método *__init__* aceita.

```
#Cria o primeiro objeto da classe Employee
empreg1 = Employee("Madalena", 6000)
# Cria o segundo objeto da classe Employee
empreg2 = Employee("Bento", 5000)
```

Acessando atributos

Você acessa os atributos do objeto usando o operador de ponto com objeto. A variável de classe seria acessada usando o nome da classe da seguinte maneira

```
empreg1.displayEmployee()
empreg2.displayEmployee()
print ("Total de funcionários %d" % Employee.empCount)
```

Aqui, a classe implementada completamente:

```
class Employee:
   'Classe base comum para todos os funcionários'
   empCount = 0

   def __init__(self, nome, salario):
      self.nome = nome
      self.salario = salario
      Employee.empCount += 1

   def displayCount(self):
     print ("Total de funcionários %d" %
Employee.empCount)

   def displayEmployee(self):
      print ("Nome : ", self.nome,  ", Salario: ",
self.salario)


# Cria o primeiro objeto da classe Employee
empreg1 = Employee("Madalena", 6000)
# Cria o segundo objeto da classe Employee
empreg2 = Employee("Bento", 5000)
empreg1.displayEmployee()
empreg2.displayEmployee()
print ("Total de funcionários %d" % Employee.empCount)
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Nome :  Madalena ,Salario:  6000
Nome :  Bento ,Salario:  5000
Total de funcionários 2
```

Em vez de usar as instruções normais para acessar atributos, você pode usar as seguintes funções:

- O **getattr (obj, nome [, padrão])** - para acessar o atributo do objeto.

- O **hasattr (obj, nome)** - para verificar se um atributo existe ou não.
- O **setattr (obj, nome, valor)** - para definir um atributo. Se o atributo não existir, ele será criado.
- O **delattr (obj, nome)** - para excluir um atributo.

```
    hasattr(emp1, 'salario')        # Retorna verdadeiro se o
atributo 'salário' existir
    getattr(emp1, ' salario')          # Retorna o valor do
atributo 'salário'
    setattr(emp1, ' salario', 7000)  # Define o atributo
'salário' em 7000
    delattr(emp1, ' salario')      # Exclue atributo 'salário'
```

Atributos de classe incorporados

Toda classe Python continua seguindo os atributos internos e eles podem ser acessados usando o operador de ponto como qualquer outro atributo -

- **__dict__** - Dicionário que contém o espaço para nome da classe.
- **__doc__** - sequência de documentação da classe ou nenhuma, se não definida.
- **__name__** - Nome da classe.
- **__module__** - Nome do módulo no qual a classe está definida. Este atributo é "__main__" no modo interativo.
- **__bases__** - Uma tupla possivelmente vazia que contém as classes base, na ordem em que ocorrem na lista de classes base.

Para a classe acima, vamos tentar acessar todos esses atributos:

```
    class Employee:
       'Classe base comum para todos os funcionários'
       empCount = 0

       def __init__(self, nome, salario):
          self.nome = nome
          self.salario = salario
          Employee.empCount += 1

       def displayCount(self):
         print ("Total de funcionários %d" %
Employee.empCount)

       def displayEmployee(self):
          print ("Nome : ", self.nome,  ", Salario: ",
self.salario)
    # Cria o primeiro objeto da classe Employee
    empreg1 = Employee("Madalena", 6000)
    # Cria o primeiro objeto da classe Employee
    empreg2 = Employee("Bento", 5000)
    print ("Employee.__doc__:", Employee.__doc__)
    print ("Employee.__name__:", Employee.__name__)
```

```
print ("Employee.__module__:", Employee.__module__)
print ("Employee.__bases__:", Employee.__bases__)
print ("Employee.__dict__:", Employee.__dict__ )
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
Employee.__doc__: Common base class for all employees

Employee.__name__: Employee

Employee.__module__: __main__

Employee.__bases__: ()

Employee.__dict__:       {'__module__':       '__main__',
'displayCount':

<function displayCount at 0xb7c84994>, 'empCount': 2,

'displayEmployee':    <function    displayEmployee    at
0xb7c8441c>,

'__doc__': 'Common base class for all employees',

'__init__': <function __init__ at 0xb7c846bc>}
```

Destruindo objetos (coleta de lixo)

O Python exclui objetos desnecessários (tipos internos ou instâncias de classe) automaticamente para liberar espaço na memória. O processo pelo qual o Python recupera periodicamente blocos de memória que não estão mais em uso é denominado Garbage Collection.

O garbage collection do Python é executado durante o processo de execução do programa e acionado quando a contagem de referência de um objeto atinge zero. A contagem de referência de um objeto é alterada conforme o número de aliases que apontam para ele.

A contagem de referência de um objeto aumenta quando é atribuído um novo nome ou colocado em um contêiner (lista, tupla ou dicionário). A contagem de referência do objeto diminui quando é excluída com *del* , sua referência é reatribuída ou sua referência fica fora do escopo. Quando a contagem de referência de um objeto chega a zero, o Python o coleta automaticamente.

```
a = 40      # Cria o objeto <40>
b = a       # Aumenta ref. contagem de <40>
c = [b]     # Aumenta ref. contagem <40>

del a       # Diminui ref. contagem de <40>
b = 100     # Diminui ref. contagem de <40>
c[0] = -1   # Diminui ref. contagem de <40>
```

Você normalmente não notará quando o coletor de lixo destrói uma instância órfã e recupera seu espaço. No entanto, uma classe pode implementar o método especial __*del* __ *()* , chamado destruidor, que é chamado quando a instância está prestes a ser destruída. Este método pode ser usado para limpar quaisquer recursos que não sejam de memória usados por uma instância.

Exemplo

Esse “destruidor” __del __ () imprime o nome da classe de uma instância que está prestes a ser destruída:

```
class Point:
   def __init__( self, x=0, y=0):
      self.x = x
      self.y = y
   def __del__(self):
      class_name = self.__class__.__name__
      print (class_name, "destruido")

pt1 = Point()
pt2 = pt1
pt3 = pt1
print id(pt1), id(pt2), id(pt3) # imprime os ids dos
objetos
del pt1
del pt2
del pt3
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
3083401324 3083401324 3083401324
```

```
Point destroyed
```

Nota - Idealmente, você deve definir suas classes em um arquivo separado e importá-las no seu arquivo de programa principal usando a instrução *import* .

Herança de Classe (Class Inheritance)

Em vez de começar do zero, você pode criar uma classe derivando-a de uma classe preexistente, listando a classe pai entre parênteses após o novo nome da classe.

A classe filho herda os atributos de sua classe pai e você pode usá-los como se eles fossem definidos na classe filho. Uma classe filha também pode substituir membros e métodos de dados do pai.

Sintaxe

Classes derivadas são declaradas como sua classe pai; no entanto, uma lista de classes base para herdar é fornecida após o nome da classe -

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
   'Sequência opcional de documentação da classe'
   class_suite
```

Exemplo

```
class Parent:        # define a parent class
   parentAttr = 100
   def __init__(self):
      print ("Chamando o construtor pai (parent)")

   def parentMethod(self):
      print ('Chamando o método pai (parente)')

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print ("Atributo Pai (Parent) :",
Parent.parentAttr)

class Child(Parent): # defina a child class
   def __init__(self):
      print ("Chamando o construtor filho(child")
```

```
        def childMethod(self):
           print ('Chamando o método filho (child)')

     c = Child()           # instancia de child
     c.childMethod()       # child chama seu método
     c.parentMethod()      # chama o método do parent
     c.setAttr(200)        # chama novamente o método do
parent
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
Chamando o Construtor Filho

Chamando método filho

Chamando o método pai

Atributo pai: 200
```

De maneira semelhante, você pode conduzir uma classe de várias classes principais da seguinte maneira:

```
     class A:         # define a class A
     .....

     class B:          # define a class B
     .....

     class C(A, B):    # subclass de A e B
     .....
```

Você pode usar as funções issubclass () ou isinstance () para verificar um relacionamento de duas classes e instâncias.

- A função booleana **issubclass (sub, sup)** retorna true se a subclasse fornecida **sub** é de fato uma subclasse da superclasse **sup** .
- A função booleana **isinstance (obj, Class)** retorna true se *obj* for uma instância da classe *Class* ou uma instância de uma subclasse de Class

Métodos de substituição (Overriding Methods)

Você sempre pode substituir seus métodos de classe pai. Uma razão para substituir os métodos dos pais é porque você pode querer funcionalidades especiais ou diferentes em sua subclasse.

Exemplo

```
      class Parent:        # define a parent class
         def myMethod(self):
            print ('Chamando o método pai (parente)')

      class Child(Parent): # define child class
         def myMethod(self):
            print ('Chamando o método filho (child)')

      c = Child()          # instancia de child
      c.myMethod()         # child chama método
substituído(overriden)
```

Quando o código acima é executado, ele produz o seguinte resultado:

```
Chamando o método filho (child)
```

Métodos de Sobrecarga de Base (Overloading Methods)

A tabela a seguir lista algumas funcionalidades genéricas que você pode substituir em suas próprias classes -

Método, Descrição e Sample Call
__init__ (self [, args ...]) Construtor (com qualquer argumento opcional) Exemplo de chamada: *obj = className (args)*
__del __ (self) Destruidor, exclui um objeto Exemplo de chamada: *del obj*
__repr __ (self) Representação de string avaliada Exemplo de chamada: *repr (obj)*

__str __ (self) Representação de string imprimível Exemplo de chamada: *str (obj)*
__cmp__ (self, x) Comparação de objetos Exemplo de chamada: *cmp (obj, x)*

Operadores de sobrecarga (Overloadind Operators)

Suponha que você tenha criado uma classe Vector para representar vetores bidimensionais, o que acontece quando você usa o operador mais para adicioná-los? Provavelmente o Python gritará com você.

No entanto, você pode definir o método *__add__* em sua classe para executar a adição de vetores e, em seguida, o operador mais se comportaria conforme a expectativa

Exemplo

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

Quando o código acima é executado, ele produz o seguinte resultado -

```
Vector(7,8)
```

Expressões Regulares (python regex)

Importância das Expressões Regulares

Expressões regulares são normalmente o meio padrão para limpar e tratar dados nessas ferramentas. Seja extração de partes específicas de textos de páginas html, obtenção de informação sobre dados do Twitter or preparação de dados para mineração de textos – Expressões Regulares são a melhor aposta para resolver todas essas questões.

Dada sua aplicabilidade, faz sentido conhecer bem Expressões Regulares e saber como usá-las de modo apropriado.

O que são Expressões Regulares e como são usadas?

Dito de forma simples, expressão regular é uma sequência de caracteres usados principalmente para encontrar e substituir padrões numa string ou num arquivo. Como dito anteriormente, a maioria das linguagens de programação como Python, Perl, R, Java e muitas outras suportam regex. Então, aprender regex ajuda de múltiplas maneiras (mais sobre isso depois).

Expressões regulares usam dois tipos de caracteres:

- Meta-caracteres: como o nome sugere, esses caracteres têm significado especial, similar ao * numa carta coringa.
- Literais: como a, b, 1, 2…

Em Python, temos o módulo "**re**" que ajuda com as expressões regulares. Você precisa primeiro importar a biblioteca **re** para então poder usar regex no Python.

```
import re
```

Os usos mais comuns de regex são:

- Buscar uma string (search e match)
- Achar uma string (findall)
- Quebrar uma string em sub strings (split)
- Substituir parte de uma string (sub)

Vamos olhar os métodos que a biblioteca ***re*** fornece para desempenhar essas atividades.

métodos de Expressões Regulares

O pacote ***re*** fornece múltiplos métodos para rodar queries numa string. Aqui segue os métodos mais comumente usados, os quais iremos aplicar:

1. re.match()
2. re.search()
3. re.findall()
4. re.split()
5. re.sub()
6. re.compile()

re.match(padrão, string)

Esse método encontra equivalência se ela ocorrer no início da string. Por exemplo, chamar *match()* na string 'AV Analytics AV' e buscar por um padrão 'AV' vai retornar uma equivalência. Contudo, se buscarmos por 'Analytics', não encontrará um padrão equivalente. Vamos ver funcionando no Python.

Encontra a primeira ocorrência na primeira posição

Código

```
import re

result = re.match(r'AV', 'AV Analytics Vidhya AV')
print (result)
```

```
<re.Match object; span=(0, 2), match='AV'>
```

Acima, mostra que o padrão equivalente foi encontrado. Para retornar a string correspondente vamos usar o método group(). Use 'r' no começo da string padrão para designar ao Python uma string raw.

```
import re

result = re.match(r'AV', 'AV Analytics Vidhya AV')
print (result.group(0))
```

```
Resultado:
AV
```

Vamos agora buscar por ‘Analytics’ na string de exemplo. Aqui vemos que a string raw não começa com ‘AV’ e por isso não deveria retornar nenhum resultado. Vamos ver o que obtemos:

Código

```
import re

result = re.match(r'Analytics', 'AV Analytics Vidhya AV')
print(result)
```

```
Resultado:
None
```

Existem métodos como start() e end() para definir a posição de começo e de fim padrão buscado na string.

Código

```
import re

result = re.match(r'AV', 'AV Analytics Vidhya AV')
print(result.start())
print (result.end())
```

```
Resultado:
0
2
```

Acima podemos ver a posição de começo e de fim do padrão ‘AV’ na string; isso ajuda muito enquanto manipulamos strings.

re.search(padrão, string)

É similar a match() mas não nos restringe a encontrar equivalência apenas no começo da string. Diferente de métodos anteriores, aqui a busca pelo padrão “Analytics” irá retornar resultado positivo.

Encontra a primeira ocorrência em qualquer posição

Código

```
import re

result = re.search(r'Analytics', 'AV Analytics Vidhya AV')
print(result.group(0))
```

```
Resultado:
Analytics
```

Aqui podemos ver que o método search() consegue encontrar um padrão em qualquer posição da string mas que somente retorna a primeira ocorrência do padrão de busca.

re.findall(padrão, string)

É útil obter uma lista de todos os padrões encontrados. Não há restrições em buscar do começo ou do fim. Se usarmos o método findall para buscar por 'AV' numa dada string, irá retornar amboas ocorrências de AV. Quando efetuar buscas numa string, recomendo usar re.findall() sempre, funciona como ambas re.search() e re.match().

Código

```
import re

result = re.findall(r'AV', 'AV Analytics Vidhya AV')
print (result)
```

```
Resultado:
['AV', 'AV']
```

re.split(padrão, string)

Este método ajuda a dividir a string pelas ocorrências do padrão dado.

Código

```
import re

result=re.split(r'y','Analytics')
print (result)
```

```
Resultado:
['Anal', 'tics']
```

Acima, dividimos a a string “Analytics” por “y”. O método split() tem um argumento chamado “maxsplit”. Seu valor padrão é zero. Nesse caso, faz o máximo de divisões possíveis, mas se dermos um valor para maxsplit, ele dividirá a string. Vamos ver o exemplo abaixo:

Código

```
import re

result=re.split(r'i','Analytics Vidhya')
print(result)
```

```
Resultado: ['Analyt', 'cs V', 'dhya'] #Realizou todas as divisões possíveis para o padrão "i".

result=re.split(r'i','Analytics Vidhya', maxsplit=1)

print(result)
```

```
Resultado:
['Analyt', 'cs Vidhya']
```

Aqui você pode notar que fixamos maxsplit em 1. E o resultado é que só tem 2 valores enquanto o primeiro exemplo tem 3 valores.

re.sub(padrão, string)

Em determinados momentos é útil buscar um padrão de string e substituí-lo por uma nova sub-string. Se o padrão não for encontrado, a string é retornada sem mudanças.

Código

```
import re
```

```
result=re.sub(r'da India','do Mundo','AV é a maior comunidade de Analytics da India')
print(result)
```

```
Resultado:
'AV é a maior comunidade de Analytics do Mundo'
```

re.compile(padrão, string)

Podemos combinar uma expressão regular com objetos de padrões, que pode ser usado para encontrar padrões. É útil também para buscar um padrão sem ter que re-escrevê-lo.

Código

```
import re

pattern=re.compile('AV')
result=pattern.findall('AV Analytics Vidhya AV')
print (result)
result2=pattern.findall('AV é a maior comunidade de Analytics da India')
print (result2)
```

```
Resultado:
['AV', 'AV']
['AV']
```

Revendo os principais métodos

Até agora, observamos vários métodos de expressões regulares usando padrões constantes (caracteres fixos). Todavia, e se não tivermos que buscar um padrão constante e quisermos retornar conjuntos específicos de caracteres (definidos a partir de uma regra) a partir de uma string? Não se deixe intimidar. Isso pode ser facilmente resolvido em se definindo uma expressão com a ajuda

de operadores de padrões (meta-caracteres e caracteres literais). Vejamos os operadores mais comuns.

Quais são os operadores mais comuns?

Expressões regulares podem especificar padrões e não somente caracteres fixos. Aqui temos os operadores mais comumente usados e que ajudam a gerar uma expressão para representar os caracteres requeridos numa string ou num arquivo. É muito usado em scraping de páginas web e mineração de textos para extrair as informações requeridas.

Operadores	Descrição
.	Corresponde a qualquer caractere único, exceto a nova linha "\ n".
?	Corresponde a 0 ou uma ocorrência do padrão, encontrada à esquerda
+	Corresponde a uma ou mais ocorrências do padrão, encontradas à esquerda
*	Corresponde a 0 ou mais ocorrências do padrão, encontradas à esquerda
\w	Corresponde a um caracter alfanumérico
\W	Corresponde a um caracter não-alfanumérico
\d	Encontra dígitos [0-9]
\D	Encontra não-dígitos
\s	Corresponde com caracter único de espaço em branco (espaço, nova linha, retorno, tab, from)
\S	Corresponde a qualquer caracter que não seja espaço em branco
\b	limite entre palavra e não-palavra
\B	Oposto de \b
[..]	Corresponde com qualquer caracter único nos colchetes e [^…] corresponde a qualquer caracter único fora dos colchetes
[^…]	Corresponde a qualquer caracter único fora dos colchetes
\	Usado para caracteres de significado especial como \. para corresponder a um período ou \+ para sinal +
^ e $	^ e $ correspondem ao início e final da string, respectivamente
{n, m}	Encontra pelo menos n e no máximo m ocorrências da expressão precedente, se escrevermos com {,m} então irá retornar pelo menos qualquer mínima ocorrência até no máximo m da expressão precedente
a | b	Corresponde a a ou b
()	Agrupa expressões regulares e retorna o texto correspondente
\t, \n, \r	Corresponde a tab, nova linha, retorno

Para mais detalhes sobre meta-caracteres "(","),"|" e outros detalhes, você pode ir para esse link (https://docs.python.org/2/library/re.html).

Agora, vamos entender sobre os operadores através dos exemplos a seguir.

Alguns exemplos de Expressões Regulares

Premissa 1: Retornar a primeira palavra de uma dada string

Solução: Extrair cada caracter (usando “\w”)

Código

```
import re

result=re.findall(r'.','AV is largest Analytics community of India')
print(result)
```

```
Resultado:
['A', 'V', ' ', 'i', 's', ' ', 'l', 'a', 'r', 'g', 'e', 's', 't', ' ', 'A', 'n', 'a', 'l', 'y', 't', 'i', 'c', 's', ' ', 'c', 'o', 'm', 'm', 'u', 'n', 'i', 't', 'y', ' ', 'o', 'f', ' ', 'I', 'n', 'd', 'i', 'a']
```

Acima, o espaço também foi extraído, então para evitar isso use “\w” ao invés de “.”.

Código

```
import re

result=re.findall(r'\w','AV is largest Analytics community of India')
print(result)
```

```
Resultado:
['A', 'V', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't', 'A', 'n', 'a', 'l', 'y', 't', 'i', 'c', 's', 'c', 'o', 'm', 'm', 'u', 'n', 'i', 't', 'y', 'o', 'f', 'I', 'n', 'd', 'i', 'a']
```

Solução-2 Extrair cada palavra usando “*” or “+”

Código

```
import re

result=re.findall(r'\w*','AV is largest Analytics community
of India')
print (result)
```

```
Resultado:
['AV', '', 'is', '', 'largest', '', 'Analytics', '', 'community', '',
'of', '', 'India', '']
```

Novamente, retorna espaço como uma palavra porque "*" retorna zero ou mais correspondências do padrão à esquerda. Agora vamos remover os espaços com "+".

Código

```
import re

result=re.findall(r'\w+','AV is largest Analytics community
of India')
print (result)
```

```
Resultado:
['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']
```

Solução-3 Extrair cada palavra usando "^"

Código

```
import re

result=re.findall(r'^\w+','AV is largest Analytics community
of India')
print (result)
```

```
Resultado:
['AV']
```

Se usarmos “$” ao invés de “^”, irá retornar a palavra do final da string. Vamos ver.

Codigo

```
import re

result=re.findall(r'\w+$','AV is largest Analytics community
of India')
print (result)
```

```
Resultado:
['India']
```

Problema 2: Retornar os dois primeiros caracteres de cada palavra

Solução 1- Extrair 2 caracteres consecutivos de cada palavra, excluindo espaços e usando “\w”

Código

```
import re

result=re.findall(r'\w\w','AV is largest Analytics community
of India')
print (result)
```

```
Resultado:
['AV', 'is', 'la', 'rg', 'es', 'An', 'al', 'yt', 'ic', 'co', 'mm', 'un',
'it', 'of', 'In', 'di']
```

Solução 2- extrair 2 caracteres consecutivos dos constantes no início de cada palavra e usando “\b”

Código

```
import re
```

```
result=re.findall(r'\b\w.','AV is largest Analytics community of India')
print (result)
```

```
Resultado:
['AV', 'is', 'la', 'An', 'co', 'of', 'In']
```

Problema 3: Retornar o domínio de emails

Para explicar de modo simples, seguiremos uma abordagem passo a passo:

Solução 1- Extrair todos os caracteres depois de “@”

Código

```
import re

result=re.findall(r'@\w+','abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com, first.test@rest.biz')
print (result)
```

```
Resultado:
['@gmail', '@test', '@analyticsvidhya', '@rest']
```

Acima, você pode ver que a parte “.com” não foi extraída. Para adicioná-la, use o código abaixo.

```
import re

result=re.findall(r'@\w+.\w+','abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com, first.test@rest.biz')
print (result)
```

```
Resultado:
['@gmail.com', '@test.in', '@analyticsvidhya.com', '@rest.biz']
```

Solução 2- Extrair somente o domínio usando “()”

Código

```
import re

result=re.findall(r'@\w+.(\w+)','abc.test@gmail.com,
xyz@test.in, test.first@analyticsvidhya.com,
first.test@rest.biz')
print (result)
```

```
Resultado:
['com', 'in', 'com', 'biz']
```

Problema 4: Retornar a data de uma string

Aqui usaremos “\d” para extrair os dígitos.

Solução

Código

```
import re

result=re.findall(r'\d{2}-\d{2}-\d{4}','Amit 34-3456 12-05-
2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009')
print (result)
```

```
Resultado:
['12-05-2007', '11-11-2011', '12-01-2009']
```

Se quiser extrair apenas o ano, parênteses novamente irá resolver.

Código

```
import re

result=re.findall(r'\d{2}-\d{2}-(\d{4})','Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009')
print (result)
```

```
Resultado:
['2007', '2011', '2009']
```

Problema 5: Retornar todas as palavras de uma string que se iniciam com vogais

Solução 1- Retorna cada palavra

Código

```
import re

result=re.findall(r'\w+','AV is largest Analytics community of India')
print (result)
```

```
Resultado:
['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']
```

Solução 2- Retorna palavras começando com vogais, usando '[]'

Código

```
import re

result=re.findall(r'[aeiouAEIOU]\w+','AV is largest Analytics community of India')
print (result)
```

```
Output:
['AV', 'is', 'argest', 'Analytics', 'ommunity', 'of', 'India']
```

Acima, você pode notar que retornou “argest” e “ommunity”. Para eliminar essas duas, precisamos usar “\b” como fronteira de palavra.

Solução 3

Código

```
import re

result=re.findall(r'\b[aeiouAEIOU]\w+','AV is largest Analytics community of India')
print (result)
```

```
Resultado:
['AV', 'is', 'Analytics', 'of', 'India']
```

De modo similar, podemos extrair palavras cujos inícios são com constantes usando “^” dentro dos colchetes.

Código

```
import re

result=re.findall(r'\b[^aeiouAEIOU]\w+','AV is largest Analytics community of India')
print (result)
```

```
Resultado:
[' is', ' largest', ' Analytics', ' community', ' of', ' India']
```

Acima, você pode notar que retornou palavras começando com espaço. Para tirá-las do resultado, inclua espaço nos colchetes.

Código

```
import re

result=re.findall(r'\b[^aeiouAEIOU  ]\w+','AV  is  largest
Analytics community of India')
print (result)
```

```
Resultado:
['largest', 'community']
```

Problema 6: Validar um número de telefone (número deve ter 10 dígitos e começar com 8 ou 9)

Temos uma lista de números de telefone na lista “li” e iremos validar os números usando regex

Solução

Código

```
import re

li=['9999999999','999999-999','99999x9999']
for val in li:
   if re.match(r'[8-9]{1}[0-9]{9}',val) and len(val) == 10:
      print ('sim')
   else:
      print ('não')
```

```
Resultado:
sim
não
não
```

Problema 7: Dividir uma string com múltiplos delimitadores

Solução

Código

```
import re

line = 'asdf fjdk;afed,fjek,asdf,foo' # String has multiple delimiters (";",",","," ").
result= re.split(r'[;,\s]', line)
print (result)
```

```
Resultado:
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

Podemos usar também o método re.sub() para substituir esses múltiplos delimitadores por apenas um como 'espaço'.

Código

```
import re

line = 'asdf fjdk;afed,fjek,asdf,foo'
result= re.sub(r'[;,\s]',' ', line)
print (result)
```

```
Resultado:
asdf fjdk afed fjek asdf foo
```

Python webScraping

Webscraping usandoPython e Beautiful Soup

O que é webscraping?
Webscraping é uma técnica de extração de dados; com ela podemos coletar dados de sites. Fazemos a 'raspagem' dos dados que são interessantes para nós.

Existem muitas empresas que utilizam como forma de gerar recursos; agragadores de links, comparadores de preços produtos são exemplo clássicos do usos de tecnicas e programas de webscraping.

Ao se obter um conjunto de dados através de um webscraping podemos armzazená-los em arquivos com formatos distintos, a partir do próprio programa webscrping. Por exemplo:

- salvar em um banco de dados;
- salvar em CSV;
- salvar em XLS;
- salvar numa tabela dentro de um banco de dados NoSQL.

Primeiros passos para se construir uma aplicação Webscraping

Para construirmos um programa webscraping são necessários alguns passos básicos:

- definir a natureza dos dados/informções que queremos pesquisar;
- pesquisar e definir os websites que receberão as pesquisas através do programa webscraping;
- fazer o mapeamento do código html dos sites escolhidos;
- criarmos o script e parametrizar os dados que serão recebidos.

Possuir conheicmento prévio em tenologias relacionadas a websites (a menos html e css)é extremamente util para que seja possivel construir um pragama webscraping.

WebScraping é realmente necessário?

Quando queremos automatizar a busca de um volume de dados/informações siginificatvios dentro de websites ou plataformas de rede socila, por exemplo, um programa webscraping torna-se fortement necessário. Vamos partir da premissa que quermos buscar o preço de um mesmo produto em diferente plataformas de e-commerce: um webscrpaing pode fazer esse trabalho para nós a cada 2 minutos, por exemplo.

Podemos, também, instruir nosso programa webscraping a buscar comentarios - dentro de redes sociais - sobre algum tema especifico. Com algumas linhas de código podemos automatizar esse processo. A unica coisa necessária - depois da implementação do código - é "rodar" nosso script

O que é necessário para criar um programa WebScraping

Algumas ferramentas que já utilizamos para trabalahr com Python:

- distribuição Ancaconda ou a isntalação Python no sistema operacional do nosso computador;
- Jupyter Notebook ou uma IDE de sua prefencia;

Construindo nosso programa WebScraping

Vamos implementar o primeiro webscraping utilizando o método *urlopen()*. Para isso, precisamos - como primeira implementação - importar o módulo urllib.request para dentro de nosso programa

```
from urllib.request import urlopen  # importando o módulo de dependencia
#agora, indicamos a url que queremos inspecionar com nosso webscraping

url = 'https://pythonscraping.com/pages/page1.html'

#implementamos a requisição necessária para capturar o código HTML referente a url que indicamos
codHTML = urlopen(url) # usamos o método urlopen

# aqui, vamos verficar se a captura do contudo HTML funcionou
codHTML.read()
```

```
b'<html>\n<head>\n<title>A Useful Page</title>\n</head>\n<body>\n<h1>An Interesting Title</h1>\n<div>\nLorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eius mod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.\n</div>\n</body>\n</html>\n'
```

O retorno do método urlopen foi: o conteúdo HTML com os inumeros elementos que podemos observar em qualquer estrutura HMTL de um website simple. A partir desse momento, podemos iniciar a extração especifica do conteúdo que queremos. Para fazer o scraping desse conteudo acima usamos o módulo de dependencia *urllib* com o método *urlopen()*. Esse módulo é nativo Python. É possivel, também, usarmos aquilo que é conhecido como projeto "terceirizado" Python. Muitos módulos de dependencia utilizados em python

fazem parte desse grupo. O módulo de dependencia Requests - usados amplamente em projetos webscraping - é um deles.

Utilizando urllib e requests

Requests é um módulo de dependencia "externo" (isso significa que, ao usá-lo, criamos uma dependência para o projeto). A principal caracteristicas do uso do módulo request é sua sintaxe: o código para criarmos um programa webscraping com request é implementado, geralemte, com menos linha de código em relação a módulo urllib. Mesmo cmo menos linhas de código podemos chegar ao mesmo resultado. Isso com que o módulo de dependencia request seja adotado massivamente pela comunida python.

Por sua vez, urllib é, então, tecnicamente, chamada de módulo de dependencia nativo do Python: isso siginifica que a manutenção/ atualização cdo código que faz esse módulo funcionar corretamente recebe a tenção e o trabalho da mesma equipe que trabalha na linguagem no desenvolvimento e aperfeiçoamento da linguagem Python.

No nosso próximo passo, utilizaremos requests e observaremos seu uso. vamos fazer, novamente, o scraping do endereço web indicado no bloco de código anterior.

```
    import requests # modulo de dependencia
    #agora, indicamos a url que queremos inspecionar com n
osso webscraping

    url = 'https://pythonscraping.com/pages/page1.html'

    #implementamos a requisição necessária para capturar o
código HTML referente a url que indicamos
    codHTML = requests.get(url) # usamos o método gete a p
artir da referencia ao módulo dependencia

    # aqui, vamos verficar se a captura do contudo HTML fu
ncionou usando .text ao invés do método read()
    codHTML.text
```

```
    '<html>\n<head>\n<title>A Useful Page</title>\n</head>
\n<body>\n<h1>An Interesting Title</h1>\n<div>\nLorem ipsum
dolor sit amet, consectetur adipisicing elit, sed do eiusmo
```

```
d tempor incididunt ut labore et dolore magna aliqua. Ut en
im ad minim veniam, quis nostrud exercitation ullamco labor
is nisi ut aliquip ex ea commodo consequat. Duis aute irure
dolor in reprehenderit in voluptate velit esse cillum dolor
e eu fugiat nulla pariatur. Excepteur sint occaecat cupidat
at non proident, sunt in culpa qui officia deserunt mollit
anim id est laborum.\n</div>\n</body>\n</html>\n'
```

Manipulando o HTML

BeautifulSoup - WebScraping Magic

BeautifulSoup é um módulo de dependencia massivamente utilizado em Python. Seu uso tem como finalidade principal: **facilitar todo o processo de manipulação que planejamos aplicar ao HTML**. Com BeautifulSoup em Python é possivel extrair dados de HTML e XML de forma muito simples. Para fazer isso acessamos os elementos *"nós" (nodes)* da estrutura do HTML da pagina em que estamos aplicando nosso programa webscraping. Percorrendo as tags HTML ao invés de texto puro, facilitamos imensamente a obentação dos dados que pretendemos extrai do webste pesquisado. Observe que neste passo o pragrama webscraping vai buscar o título do website. Para isso, indicamos, no nosso codigo, o elemento-tag *title* encontrado em qualquer estrtutura HTML de qualquer website simples.

```
    from urllib.request import urlopen
    from bs4 import BeautifulSoup # importando o módulo de
dependencia BeautifulSoup
    url = 'https://www.wikipedia.org/' #inserindo a url pa
ra a pesquisa de dados

    #fazendo a leitura da url indicado com a utilizaçao do
método urlopen()
    codHTML = urlopen(url)

    #utilizando o módulo de dependencia BeautifulSoup para
manipular o HTML da url indicada
    bs = BeautifulSoup(codHTML, 'lxml')

    #chamando a função print para ler o resultado do nosso
program webscraping
    #aqui, indicamos que nosso programa webscraping traga
como resultado
    #a tag <title></title> e o valor associado a ela
    print(bs.title)
```

```
<title>Wikipedia</title>
```

Usamos o elemento *bs* para criar uma instância do módulo de dependencia *BeautifulSoup* para extrair dados da página.

Definição dos métodos find() / find_all()

O método *find()* possui como caracteristica principal **encontrar o primeiro elemento e retorná-lo como resultado.** O método *find_all()* executa uma varredura em todo o documento HTML indicado pela url e retorna todas as ocorrências - que indicamos como parâmetro - encontradas. Vamos utilizar, agora, o método find() para pesquisar e capturar o primeiro elemento-tag 'h1' que ele encontrar.

```
# Utilizando o método find() para capturar apenas o primeiro elemento 'h1' encontrado
bs.find('h1')
```

```
<h1 class="central-textlogo" style="font-size: 1em;" title="Wikipedia">
<img alt="Wikipedia" class="central-featured-logo" height="183" src="portal/wikipedia.org/assets/img/Wikipedia-logo-v2.png" srcset="portal/wikipedia.org/assets/img/Wikipedia-logo-v2@1.5x.png 1.5x, portal/wikipedia.org/assets/img/Wikipedia-logo-v2@2x.png 2x" width="200"/>
<div class="central-textlogo-wrapper">
<div class="central-textlogo__image sprite svg-Wikipedia_wordmark">
Wikipedia
</div>
<strong class="jsl10n localized-slogan" data-jsl10n="slogan">The Free Encyclopedia</strong>
</div>
</h1>
```

Neste proximo passo, continuaremos utilizando BeautifulSoup. Agora, queremos buscar todos os elementos-tags 'h1'. Para esse resultado, vamos implementar o método *find_all()*, componente do módulo de dependencia BeautifulSoup.Observe o código abaixo:

```
    #buscando e exibindo todos os elementos-tags h1 da pág
ina que indicamos atravé da url
    print(bs.find_all('h1'))
```

```
    [<h1 class="central-textlogo" style="font-size: 1em;"
title="Wikipedia">
    <img alt="Wikipedia" class="central-featured-logo" hei
ght="183" src="portal/wikipedia.org/assets/img/Wikipedia-lo
go-v2.png" srcset="portal/wikipedia.org/assets/img/Wikipedi
a-logo-v2@1.5x.png 1.5x, portal/wikipedia.org/assets/img/Wi
kipedia-logo-v2@2x.png 2x" width="200"/>
    <div class="central-textlogo-wrapper">
    <div class="central-textlogo__image sprite svg-Wikiped
ia_wordmark">
    Wikipedia
    </div>
    <strong class="jsl10n localized-slogan" data-jsl10n="s
logan">The Free Encyclopedia</strong>
    </div>
    </h1>]
```

Caso seja necessário encontrar um elemento - utilizando o método *find_all()* podemos optar por dois caminhos:

- utilizar plenamente o método o *find()* - como no passo anteiror;
- passar o argumento limit = 1, ao método *find_all()*

Manteremos o scraping da mesma url mas mudaremos nosso elemento-tag - agora será *link* pesquisado. Vamos observar o código abaixo:

```
    # buscando, através da url indicada, todos os elemento
s-tags 'link' que serão encontrados
    print(bs.find_all('link'))

    #criando uma nova linha
    print('\n')

    #limitando a busca a somente encontar um elemento
    print(bs.find_all('link', limit = 1))

    #criando uma nova linha
    print('\n')

    # exibindo o números de elementos-tags 'h1' encontrado
s
```

```
    print('Número total de elementos-tags link encontrados
= {}'.format(len(bs.find_all('link'))))
```

```
    [<link href="/static/apple-touch/wikipedia.png" rel="a
pple-touch-icon"/>, <link href="/static/favicon/wikipedia.i
co" rel="shortcut icon"/>, <link href="//creativecommons.or
g/licenses/by-sa/3.0/" rel="license"/>, <link href="//uploa
d.wikimedia.org" rel="preconnect"/>]


    [<link href="/static/apple-touch/wikipedia.png" rel="a
pple-touch-icon"/>]


    Número total de elementos-tags link encontrados = 4
```

Nosso programa webscraping capturou todos os elementos-tags 'link' presentes na página hone do website indicado pela url. Observamos que o total de elementos´tags é 4; com o uso do argumento limit = 1 aplicado ao método *find_all* - na terceira instrução do boloc de código - exibimos somente um elemento dos 4 capturados. A instância *bs* que criamos deu a possibilidade a nosso programa webscraping de capturar o elemento alvo e traze-lo para noss implementação.

Usar elementos 'class' e 'id' para fazer extração

Nesse próximo observaremos a extração buscando elementos *class* e *id*. Possivelmente, ao inspecionar qualquer website, encontraremos estes atributos. O módulo de dependencia BeautifulSoup oferece maneiras simples para acessarmos eelementos dentro do website utilizando os dois recursos.

Como inspecionar/mapear elementos de um website para fazer um scraping

A inspeção/mapeamento de uma estrutura de qualquer website possibilita a descoberta do "trajeto" para acessarmos os atributos que desejamos capturar com nossa aplicação webscraping.

Utilizando o software Chrome Borwser procedemos da seguinte forma: clicando com o botão direito do mouse e selecionamos - através do menu suspenso - a opção 'Inspecionar'.

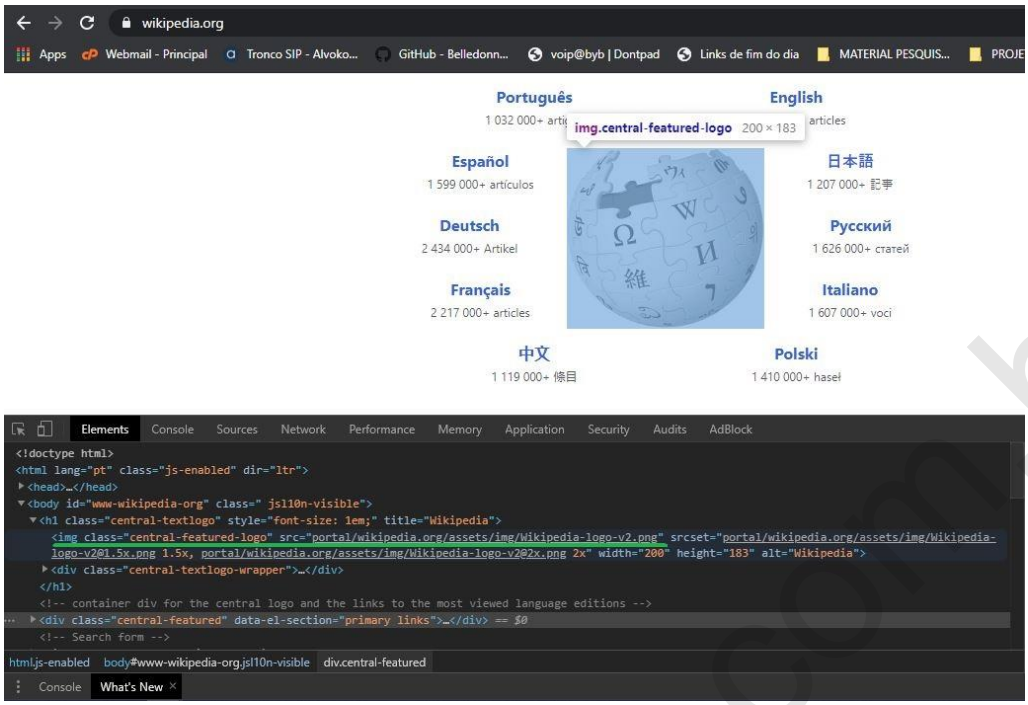
Utilizando o software Mozilla Firefox Borwser procedemos da seguinte forma: clicando com o botão direito do mouse e selecionamos - através do menu suspenso - a opção 'Inspecionar Elemento'.

Segue-se a mesma aboradagem para os demias browsers.

Vamos trabalhar com a inspeção/mapeamento do site https://www.wikipedia.org/

Observando, inspecionando/mapeando o logo do website:

Assim que clicamos em na opção 'Inspecionar' teremos acesso a estrutura HMTL do website. Observe a imagem abaixo:



Na leitura do código HTML acima encontramos uma 'div' com uma classe de nome 'central-featured-logo. Vamos selecioná-la e acessá-la pela nossa aplicação webscraping. Observe o código abaixo

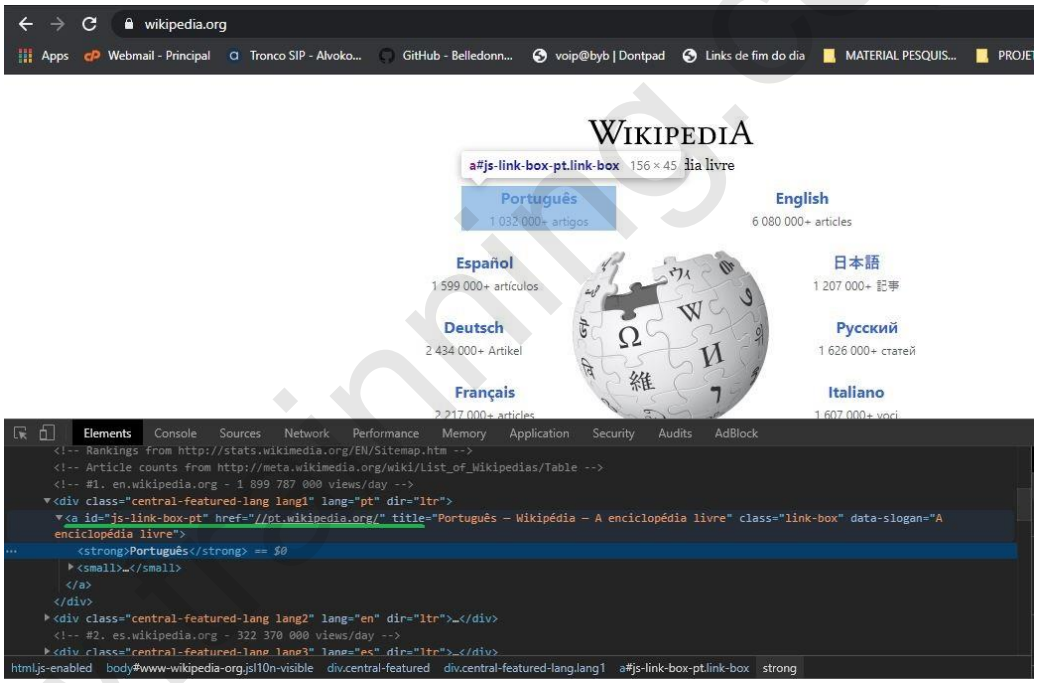
```
    from urllib.request import urlopen  # importando o módulo de dependencia
    from bs4 import BeautifulSoup # importando o módulo de dependencia BeautifulSoup

    #criamos a variavel de nome 'url' e atribuimos o valor com o link do website
    url = 'https://www.wikipedia.org/'

    # acessando e lendo o conteúdo da url utilizando o método urlopen
    codHTML = urlopen(url)

    # craindo a instância do módulo BeautifulSoup para extrair o conteudo desejado
    bs = BeautifulSoup(codHTML, 'lxml')

    # acessando - através do 'trajeto' encontrado através da inspeção/mapeamento - o atributo do logo
    print(bs.find(class_='central-featured-logo'))
```

```
<img alt="Wikipedia" class="central-featured-logo" hei
ght="183" src="portal/wikipedia.org/assets/img/Wikipedia-lo
go-v2.png" srcset="portal/wikipedia.org/assets/img/Wikipedi
a-logo-v2@1.5x.png 1.5x, portal/wikipedia.org/assets/img/Wi
kipedia-logo-v2@2x.png 2x" width="200"/>
```

O resultado é 'None'- se tudo correu bem - pois acessamos a classe com a indicação do elemento 'div'

Selecionar pelo elemento 'id'

Também é possivel capturar o conteudo do website pelo elemento 'id. Na leitura do código HTML acima vamos buscar o elemento para a lingua portuguesa. Vamos selecioná-la e acessá-la pela nossa aplicação webscraping.



Observe o código abaixo:

```
# acessando o elemento 'id' para a lingua protuguesa e exibindo o texto encontrado ao acessard o elemento

print(bs.find(id='js-link-box-pt').text)
```

```
Português
1 032 000+ artigos
```

Acesso utilizando elementos CSS

Seletores CSS é uma outra eficiente para acessar e extrair dados com nossa aplicação webscraping. Podemos considerar alguns exemplos de seeltores CSS para mapearmos os dados que queremos acessar:

- p a: aqui, estamos buscando todas as tags 'a' dentro de um parágrafo
- div p: mapeamos todas os parágrafos (tags 'p') dentro de uma div
- div p span: mapeamos todos os elementos 'span' dentro de um parágrafo (tag 'p') que, por sua vez, estão dentro de uma div (tag 'div')
- table td: todos os elementos 'td' contidos dentro do elemento 'tables'

A ideia é: selecionar/acessar seguindo o hierarqui da estrutura. Para entendermos a arquitetura dos elementos dentro do website basta mapear a árvore-HTML. Assim é possivel encontrar o elemento-alvo e pesquisar os elementos contidos dentro dele. Quanto masi nitida a informação passada ao program webscraping (especifcação das tags) mais fácil será - para nossa aplicação - encontar o dado que buscamos. Para exemplificar o uso de seletores CSS, será utilizado método *select()*, acoplando como argumento o seletor CSS desejado. Observe o código abaixo:

```
    #aqui, selecionaremos todas as tags 'strong' dentro da
s tags 'divs'
    print(bs.select('div strong'))
    print('\n')

    #aqui, selecionaremos as tags 'input' dentro do elemen
to 'forms' que, por sua vez, estão dentro das tags 'divs'
    print(bs.select('div form input'))
```

```
    [<strong class="jsl10n localized-slogan" data-jsl10n="
slogan">The Free Encyclopedia</strong>, <strong>English</st
rong>, <strong>Español</strong>, <strong>日本語</strong>, <s
trong>Deutsch</strong>, <strong>Русский</strong>, <strong>F
rançais</strong>, <strong>Italiano</strong>, <strong>中文</s
trong>, <strong>Português</strong>, <strong>Polski</strong>
, <strong class="jsl10n" data-jsl10n="app-links.title">
    <a class="jsl10n" data-jsl10n="app-links.url" href="ht
tps://en.wikipedia.org/wiki/List_of_Wikipedia_mobile_applic
ations">
    Download Wikipedia for Android or iOS
    </a>
    </strong>]
```

```
[<input name="family" type="hidden" value="wikipedia"/
>, <input id="hiddenLanguageInput" name="language" type="hi
dden" value="en"/>, <input accesskey="F" autocomplete="off"
autofocus="autofocus" dir="auto" id="searchInput" list="sug
gestions" name="search" size="20" type="search"/>, <input n
ame="go" type="hidden" value="Go"/>]
```

Também é possivel utilizar seletores CSS para classes e ids, que são representados por:

- (.) : ao utilizar o elemento . (ponto), o módulo de dependecia BeautifulSoup - assim como CSS - automaticamente pesquisará por classes que forma indicado com este seletor
- (#) : para elementos 'id', o módulo de dependencia BeautifulSouo segue a mesma premissa dos elementos "class"

Observe o código abaixo:

```
    # aqui, acessamos o conteudo do elemento 'class' chama
do footer-sidebar
    print(bs.select('.footer-sidebar'))

    print('\n')

    # aqui, acessamos o conteudo do elemento 'id' da opção
'Português'
    print(bs.select('#js-link-box-pt'))
```

```
    [<div class="footer-sidebar">
    <div class="footer-sidebar-content">
    <div class="footer-sidebar-icon sprite svg-Wikimedia-l
ogo_black">
    </div>
    <div class="footer-sidebar-text jsl10n" data-jsl10n="f
ooter-description">
    Wikipedia is hosted by the <a href="//wikimediafoundat
ion.org/">Wikimedia Foundation</a>, a non-profit organizati
on that also hosts a range of other projects.
    </div>
    </div>
    </div>, <div class="footer-sidebar app-badges">
    <div class="footer-sidebar-content">
    <div class="footer-sidebar-text">
    <div class="footer-sidebar-icon sprite svg-wikipedia_a
pp_tile"></div>
    <strong class="jsl10n" data-jsl10n="app-links.title">
```

```
     <a class="jsl10n" data-jsl10n="app-links.url" href="ht
tps://en.wikipedia.org/wiki/List_of_Wikipedia_mobile_applic
ations">
     Download Wikipedia for Android or iOS
     </a>
     </strong>
     <p class="jsl10n" data-jsl10n="app-links.description">
     Save your favorite articles to read offline, sync your
reading lists across devices and customize your reading exp
erience with the official Wikipedia app.
     </p>
     <ul>
     <li class="app-badge app-badge-android">
     <a href="//play.google.com/store/apps/details?id=org.w
ikipedia&amp;referrer=utm_source%3Dportal%26utm_medium%3Dbu
tton%26anid%3Dadmob" target="_blank">
     <span class="sprite svg-badge_google_play_store">
     </span></a>
     </li>
     <li class="app-badge app-badge-ios">
     <a href="//itunes.apple.com/app/apple-store/id32471523
8?pt=208305&amp;ct=portal&amp;mt=8" target="_blank">
     <span class="sprite svg-badge_ios_app_store">
     </span></a>
     </li>
     </ul>
     </div>
     </div>
     </div>]


     [<a class="link-box" data-slogan="A enciclopédia livre
" href="//pt.wikipedia.org/" id="js-link-box-pt" title="Por
tuguês — Wikipédia — A enciclopédia livre">
     <strong>Português</strong>
     <small><bdi dir="ltr">1 032 000+</bdi> <span>artigos</
span></small>
     </a>]
```

Tratamento de exceções (Handling Exception)

Ao criar e executar uma programa webscraping é possivel que alguns erros ocorram antes, durante ou depois da execução. Abaixo, listamos 3 possibilidades de erro que são consideradas comuns:

- erro em relação ao servidor onde o site está hospedado
- erro na estrutura do códido da aplicação webscraping

- erro de execução quando o link de referencia - por exemplo - foi alterado

Para observados - quando se trata da requisição que nossa apliação webscraping realiza - a validação efetiva a partir do nosso código devemos estrutrá-la com mais algumas instruções. é necessário importar alguns módulos que auxiliarão a execução dessa validação. Observe o código abaixo:

```
    # importando os módulo para realizar o tratamento de e
rro da nossa aplicação webscraping
    from urllib.error import HTTPError
    from urllib.error import URLError
```

podemos criar uma estrutura de validação simples para nossa aplicação webscraping utilzando os statements 'try' / 'except' . Observe o código abaixo:

```
    # tratando o erro de requisição utilizando try/except

    try:
        html = urlopen('https://www.wikipedia.org/')
    except HTTPError as e:
        #se ocorrer algum erro exibimos ele através da cha
mada 'print()'
        print(e)
    except URLError as e:
        #se ocorrer algum erro com a URL exibimos ele atra
vés da chamada 'print()'
        print('O servidor não pode ser encontrado!')
```

Podemos, agora, forçar um erro de URL, uma vez que não encontraremos um erro de servidor (404 ou 503, por exemplo). Assim, podemos observar se o tratamento que implementamos está funcionando.

Vamos observar o código abaixo:

```
    # tratando o erro de requisição utilizando try/except
    try:
        html = urlopen('https://www.wikipedia90.org/')
    except HTTPError as e:
        #se ocorrer algum erro exibimos ele através da cha
mada 'print()'
        print(e)
    except URLError as e:
        #se ocorrer algum erro com a URL exibimos ele atra
vés da chamada 'print()'
        print('O url não pode ser encontrada!')
```

```
O url não pode ser encontrada!
```

WebScraping App

Neste exercício para a criação de uma aplicação webscraping a proposta é: capturar os dados da lista dos 500 melhores filmes disponiveis através da url do website IMDB (website internacionalment conhecido por disponibilizar as internautas reviews, críticas e notícias de filmes, séries e conteudo audiovisual).

Nosso desafio será composto pelas seguintes etapas:

- Importar os móodulos de dependencia necessários
- Validar a URL da lista de filmes que nosssa aplicação vai capturar
- extrair dados de: título, direção e roteirista(s), data de lnçamento do filme e sua classifcação (nota)
- Transferir os dados para um dataset dentro do nosso programa
- Salvar os dados em um arquivo com formato .csv e outro arquivo .xls

Vamos ilustrar cada etapa do procedimento, até que finalizemos alcançado todos os objetivos

Importar os módulos de dependicia

Importaremos um módulo de dependencia em cada célual do nosso notebook e executaremos para ver se, ao importarmos os módulos, não teremos problemas. Observe o código abaixo:

```
# módulo para criarmos o dataframe
import pandas as pd
# módulo para lermos o conteudo trazido pela url
from urllib.request import urlopen
# módulo para tratarmos o erro de requisição HTTP
from urllib.error import HTTPError
# módulo para tratarmos o erro de requisição URL
from urllib.error import URLError
# módulo para extrairmos os dados
from bs4 import BeautifulSoup
```

O módulo d dependencia *'pandas'* possibilita ao nosso programa construir - a partir da obtençaõ de um conjunto de dados - construir um dataframe para armazenar e manipular, posteriormente. O módulo urllib nos possibilita ao nosso programa fazer as requisições ao website; o módulo BeautifulSoup possibilita a extração de dados.

Os módulos foram importados e cada célula com seu respectivo módulo foi executada. se tudo correu bem, até este ponto, precisamos, agoar, validar noss url base e, posteriormente , indicaros elementos que nossa aplicação acessará para extarir os dados. Observe o código abaixo, ele dispõe a url base:

```
#Url Base
url = 'https://www.imdb.com/chart/top/'
```

Agora, vamos implementar o tratamento de erro no nosso programa webscraping. Observe o código abaixo:

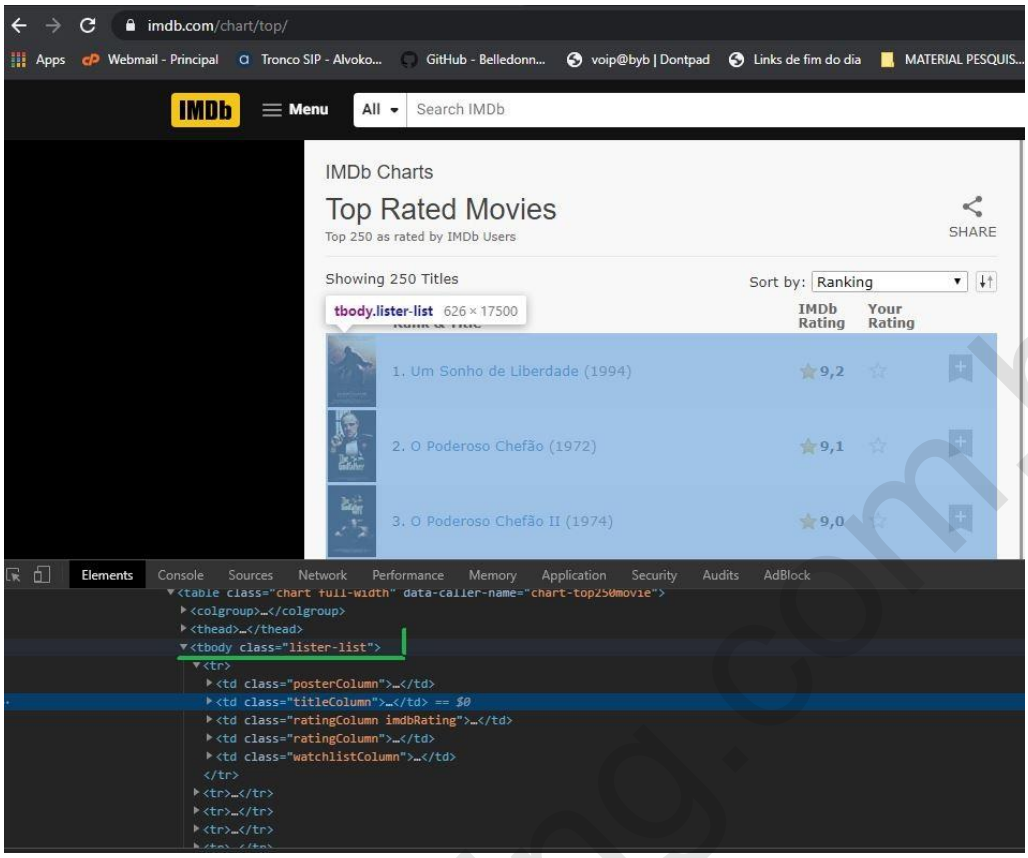
```
#tratando erros da url

try:
    html = urlopen(url)
except HTTPError as e:
    #Erros HTTP
    print(e)
except URLError as e:
    #URL errada
    print('Url não encontrada!')
```

Se não foi encontrado nenhum erro. Seguiremos para o próximo passo.

Vamos observar o HTML - fazer a inspenção acessando link 'https://www.imdb.com/chart/top/' pelo browser. Dessa forma, podemos observar e entender como construir nosso programa webscraping para pesquisar, capturar e extrair cada filme:

Inspecionando/mapeando o website

Os filmes

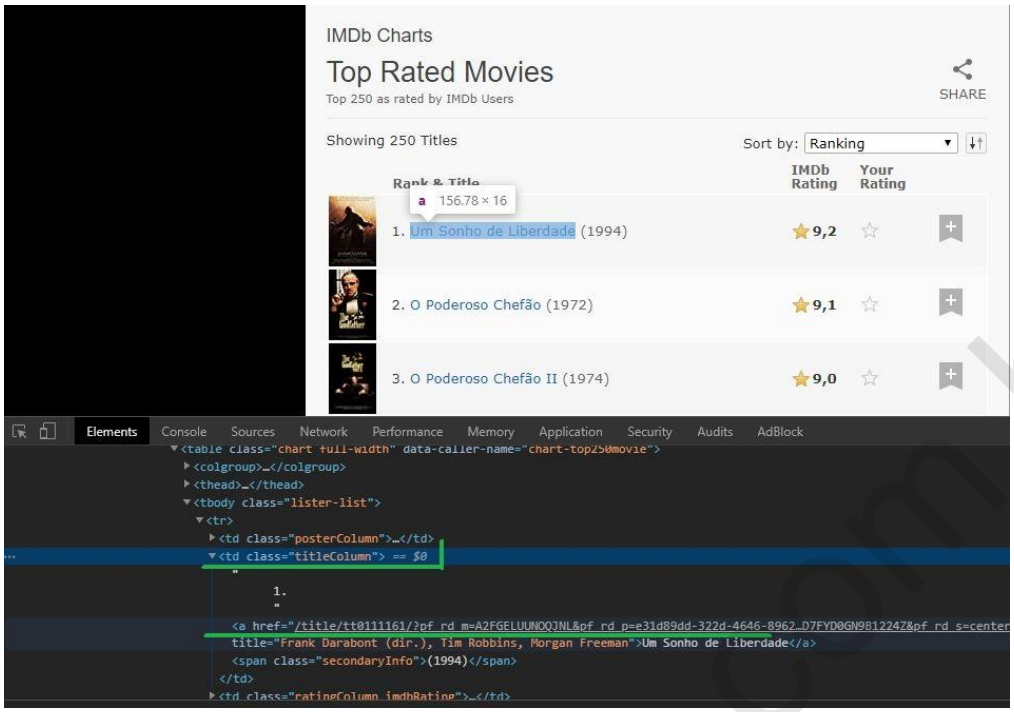


Aqui o procedimento é o seguinte: selecionar todas os elementos 'tr' dentro da tag 'tbody' com o elemento 'class' *lister-list*. Também, criaremos a instância do objeto do módulo de depência BeautifulSoup com a url indicada. Observe o código abaixo:

```
    # criando a instância BeautifulSoup para a extração de dados
    bs = BeautifulSoup(html, 'lxml')
    # selecionando todos os elementos 'tr' dentro do elemento 'lister-list'
    movies = bs.select('.lister-list tr')
```

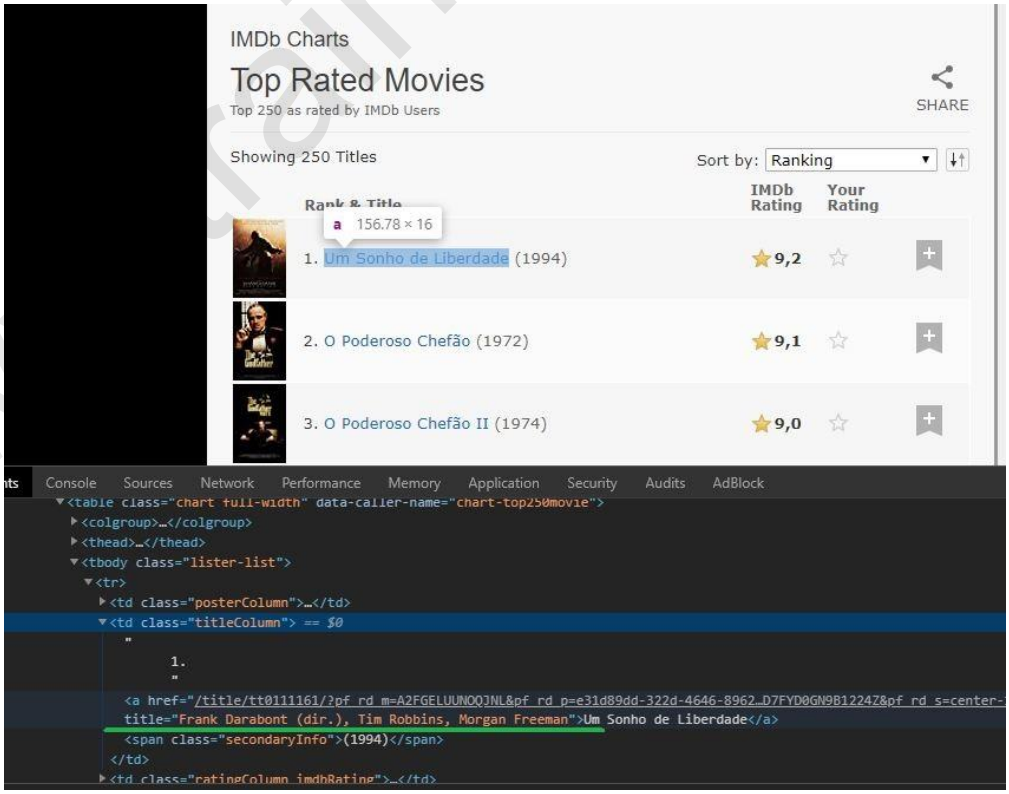
Neste momento vamos voltar ao código HTML da página para observarmos quais são os elementos que nossa aplicação webscraping extrairá.

Título



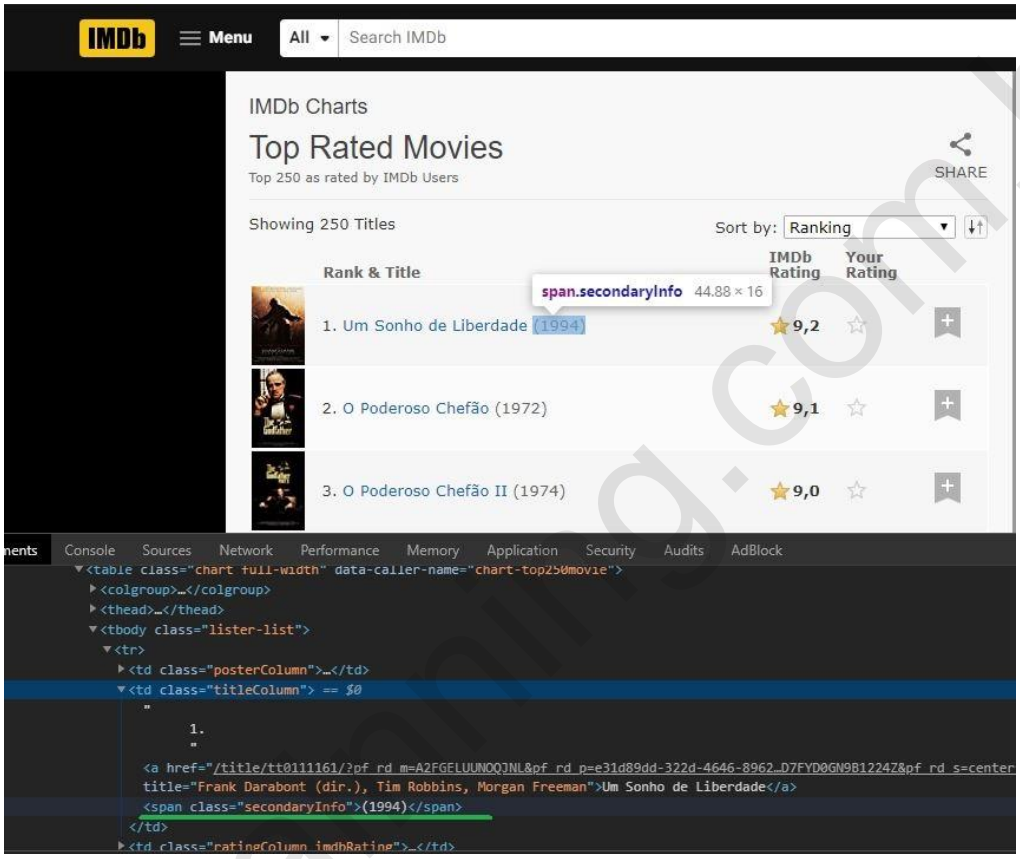
No código HTML acima observamos que para extrairmos o título devemos capturar o elemento a que esta dentro de um elemento 'td' com *'class titleColumn'* e, assim, extrair o texto de dnetro dela.

Diretor(es) / roteirista(s)



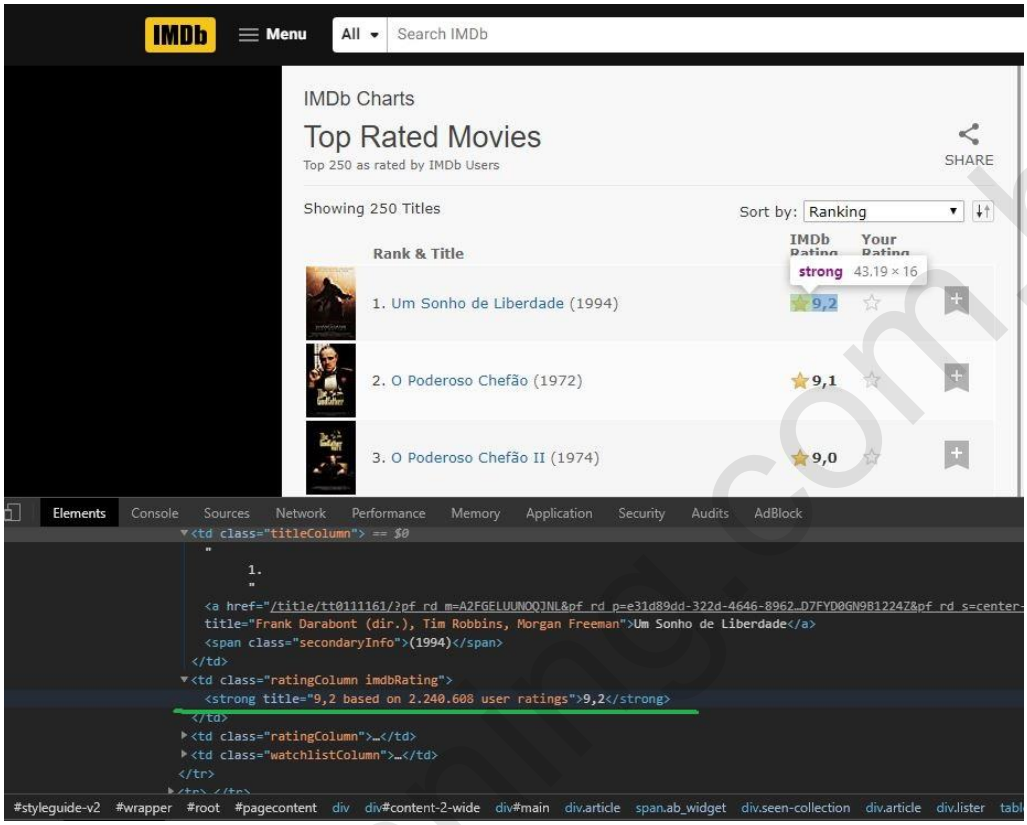
Inspencionando atentamente o código HTML do website observamos que a estrutura dos títulos também implementam o elemento *'title'* que contém o nome do diretor e do roteira. vamos caputrá-los e extraí-los.

Ano do filme



Considerando, a partir da nossa inspeção/mapeamento, dentro do elemento 'titleColumn' , vamos selecionar a tag 'span' que contem o valor com o ano de lançamento do filme.

Classificação do filme / notas (ratings)



Agora, capturamos o elemento 'td' com a *'class imdbRating'* e extrair o texto da 'tag strong' de dentro do elemento 'td'

Voltando ao código da aplicação webscraping

Depois da inspeção/mapeamento realizado a partir do website IMDB - para capturar e extair os dados corretamente - observemos o nosso código:

```
titles = []
directors_writers = []
years = []
ratings = []
```

No bloco de ódigo acima, foram criados algumas listas para serm preenchidas com os dados que serão obtidos através da extração realizada pelo bloco de código que executa o webscraping. Essas listas armazenarão os dadso relacionado a cada atributo inspecionado. Posteriormente, esses datasets construirão um dataframe para exibir o resultado da extração de dados.

No código abaixo, criamos um laço 'for' com a variavel auxiliar 'movie' para iterar dentro de 'movies'. Essa estrtura de código executa a extração para cada elemento que inspecionamos/mapeamos dentro do website IMDB.

Observe o código abaixo:

```
    for movie in movies:
        titles.append(movie.find('td', class_='titleColumn
').find('a').get_text())
        directors_writers.append(movie.find('td', class_='
titleColumn').find('a')['title'])
        years.append(movie.find('td', class_='titleColumn'
).find('span').get_text()[1:5])
        ratings.append(movie.find('td', class_='imdbRating
').find('strong').get_text())
```

Criando um dataframe utilizando a dependencia pandas

Abaixo, o dataframe é criado a partir do resultado da extração de dados obtida com o scraping. Observe o código abaixo:

```
    df = pd.DataFrame({"titulo":titles,
                       "ano de lançamento":years,
                       "classificação": ratings,
                       "diretores e roteiristas":directors
_writers
                      })
```

Agora que craimos um dataframe com os dados obtidos a partir da extração e utilziamos um dicinário para paramtrizar nosso dataframe, vamos exibir as 10 primeiras linhas. Para isso, basta chamar a varaivel 'df' seguida do método *'head()'*. Observe o código abaixo:

```
    df.head(10)
```

	titulo	ano de lançamento	classificação	diretores e roteiristas
0	Um Sonho de Liberdade	1994	9.2	Frank Darabont (dir.), Tim Robbins, Morgan Fre...

	titulo	ano de lançamento	classificação	diretores e roteiristas
1	O Poderoso Chefão	1972	9.1	Francis Ford Coppola (dir.), Marlon Brando, Al...
2	O Poderoso Chefão II	1974	9.0	Francis Ford Coppola (dir.), Al Pacino, Robert...
3	Batman: O Cavaleiro das Trevas	2008	9.0	Christopher Nolan (dir.), Christian Bale, Heat...
4	12 Homens e uma Sentença	1957	8.9	Sidney Lumet (dir.), Henry Fonda, Lee J. Cobb
5	A Lista de Schindler	1993	8.9	Steven Spielberg (dir.), Liam Neeson, Ralph Fi...
6	O Senhor dos Anéis: O Retorno do Rei	2003	8.9	Peter Jackson (dir.), Elijah Wood, Viggo Morte...
7	Pulp Fiction: Tempo de Violência	1994	8.8	Quentin Tarantino (dir.), John Travolta, Uma T...
8	Três Homens em Conflito	1966	8.8	Sergio Leone (dir.), Clint Eastwood, Eli Wallach
9	O Senhor dos Anéis: A Sociedade do Anel	2001	8.8	Peter Jackson (dir.), Elijah Wood, Ian McKellen

Utlizando o método *'to_csv()'* da dependencia 'pandas' podemos salvar o dataframe com esse formato. Observe o código abaixo:

```
df.to_csv('WebScraping-IMDB.csv')
```

Ao executar a instrução acima, você deverá encontrar no mesmo diretório de seu notebbok o arquivo com o nome *'WebScraping-IMDB.csv'* salvo.

Para salvar no formato excel, o procedimento é semelhante. Observe o código abaixo:

```
df.to_excel('WebScrapng-IMDB.xlsx')
```

Executando o código acima você salvará seu dataframe no formato .xlsx

Python GUI(Guide User Interface)

Construindo um editor de texto (aplicativo de exemplo)

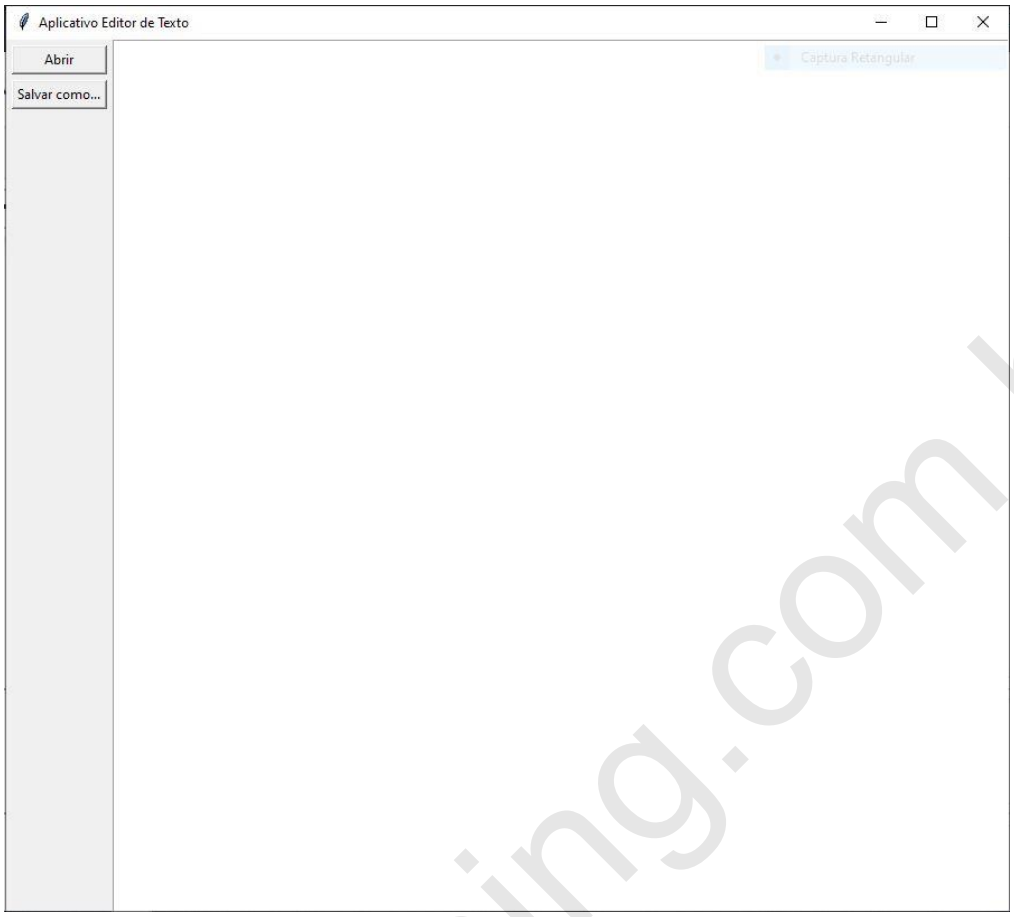
Neste passo, criaremos um aplicativo de editção de texto. Com o aplicativo será possivel criar, abrir, editar e salvar arquivos de texto (formato .txt).

Existem três elementos essenciais no aplicativo:

- Um Buttonwidget chamado btn_open para abrir um arquivo para edição
- Um Buttonwidget chamado btn_save para salvar um arquivo
- Um TextBoxwidget chamado txt_edit para criar e editar o arquivo de texto

Os três widgets serão organizados de forma que os dois botões fiquem no lado esquerdo da janela e a caixa de texto no lado direito. A janela inteira deve ter uma altura mínima de 800 pixels e o elemento *txt_edit* com largura mínima de 800 pixels. Todo o layout deve ser responsivo para que, se a janela for redimensionada, ela também que o elemento *txt_edit* seja redimensionada. A largura do *Frame* deve manter os botões na mesma posição.

Nosso aplicativo deve se parecer com a imagem abaixo:



É possível configurar o layout o gerenciador geometric*.grid()*. O layout contém uma única linha e duas colunas:

- Uma coluna estreita à esquerda para os botões
- Uma coluna mais larga à direita para o editor de texto

Para definir os tamanhos mínimos para a janela e para o elemento *txt_edit*; é possivel definir os parâmetros *minsize* dos métodos *.rowconfigure()* da janela e *.columnconfigure( )* como 800. Para controlar o redimensionamento, você pode definir os parâmetros *weight* desses métodos para 1.

Para colocar os dois botões na mesma coluna, você precisará criar um *widget Frame* chamado *fr_buttons*. De acordo com a imagem acima, os dois botões devem ser organizados verticalmente dentro deste 'frame', com *btn_open* acima do outro botão. Você pode fazer isso com o gerenciador *.grid()* ou *.pack()*.

Agora podemos iniciar a codificação. A primeira etapa é criar todos os widgets necessários. Observe o código abaixo:

```
    import tkinter as tk

    window = tk.Tk()
    window.title("Aplicativo Editor de Texto")

    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window)
    btn_open = tk.Button(fr_buttons, text="Abrir")
    btn_save = tk.Button(fr_buttons, text="Salvar como..."
)
```

O que está occorendo com o código acima:

- Importação do módulo de dependencia tkinter na **linha 1** .
- As **linhas 3 e 4** criam uma nova janela com o título "Aplicativo Editor de Texto".
- As **linhas 6 e 7** definem as configurações de linha (rowconfigure) e coluna (columnconfigure).
- As **linhas 9 a 12** criam os quatro widgets necessários para a 'text box', o 'frame' e os botões abrir e salvar.

Vamos observar a **linha 6**. O parâmetro *minsize* do método *.rowconfigure()* está definido como 800 e o elemento *weight* está definido como 1:

```
window.rowconfigure(0, minsize=800, weight=1)
```

O primeiro argumento é 0, que define a altura da primeira linha como 800 pixels e garante que a altura da linha aumente proporcionalmente à altura da janela. Há apenas uma linha no layout do aplicativo, portanto, essas configurações se aplicam a toda a janela.

Vamos, agora, observar a **linha 7**. Aqui, com o método *.columnconfigure()*, é possível definir os atributos *width* e *weight* da coluna com o índice 1, manteremos 800 para minsize e 1 para weight, respectivamente:

```
window.columnconfigure(1, minsize=800, weight=1)
```

Lembre-se de que os índices de linha e coluna são baseados em zero, portanto, essas configurações se aplicam apenas à segunda coluna. Ao configurar apenas a segunda coluna, a caixa de texto se expandirá e se contrairá naturalmente quando a janela for redimensionada, enquanto a coluna que contém os botões permanecerá com uma largura fixa.

Agora, vamos construir o layout do aplicativo.

Primeiro passo: atribua os dois botões ao frame *fr_buttons* usando o gerenciador *.grid(). Observe o código abaixo*:

```
    btn_open.grid(row=0, column=0, sticky="ew", padx=5, pa
dy=5)
    btn_save.grid(row=1, column=0, sticky="ew", padx=5)
```

Essas duas linhas de código criam uma grade com duas linhas e uma coluna no *frame fr_buttons* , pois ambas, **btn_open** e **btn_save**, têm seus *master atributos* definidos como *fr_buttons*. **btn_open** é colocado na primeira linha e **btn_save** na segunda linha para que **btn_open** apareça acima de **btn_save** no layout.

Ambos, **btn_open** e **btn_save** possuem seus *'sticky' atributos* definidos como *"ew"*, o que força os botões a se expandir horizontalmente nas duas direções e preencher o 'frame' inteiro. Isso garante que os dois botões tenham o mesmo tamanho.

Implementando 5 pixels de preenchimento ao redor de cada botão, definindo os parâmetros *padx* e *pady* como 5. Somente **btn_open** possui preenchimento vertical. Como está no topo, o preenchimento vertical desloca um pouco o botão da parte superior da janela e garante que haja um pequeno espaço entre ele e **btn_save**.

Agora que *fr_buttons* está pronto, você pode configurar o layout da grade para o restante da janela. Vamos observe o código abaixo:

```
fr_buttons.grid(row=0, column=0, sticky="ns")
txt_edit.grid(row=0, column=1, sticky="nsew")
```

Essas duas linhas de código criam uma grade com uma linha e duas colunas para window. Você coloca *fr_buttons* na primeira coluna e *txt_edit* na

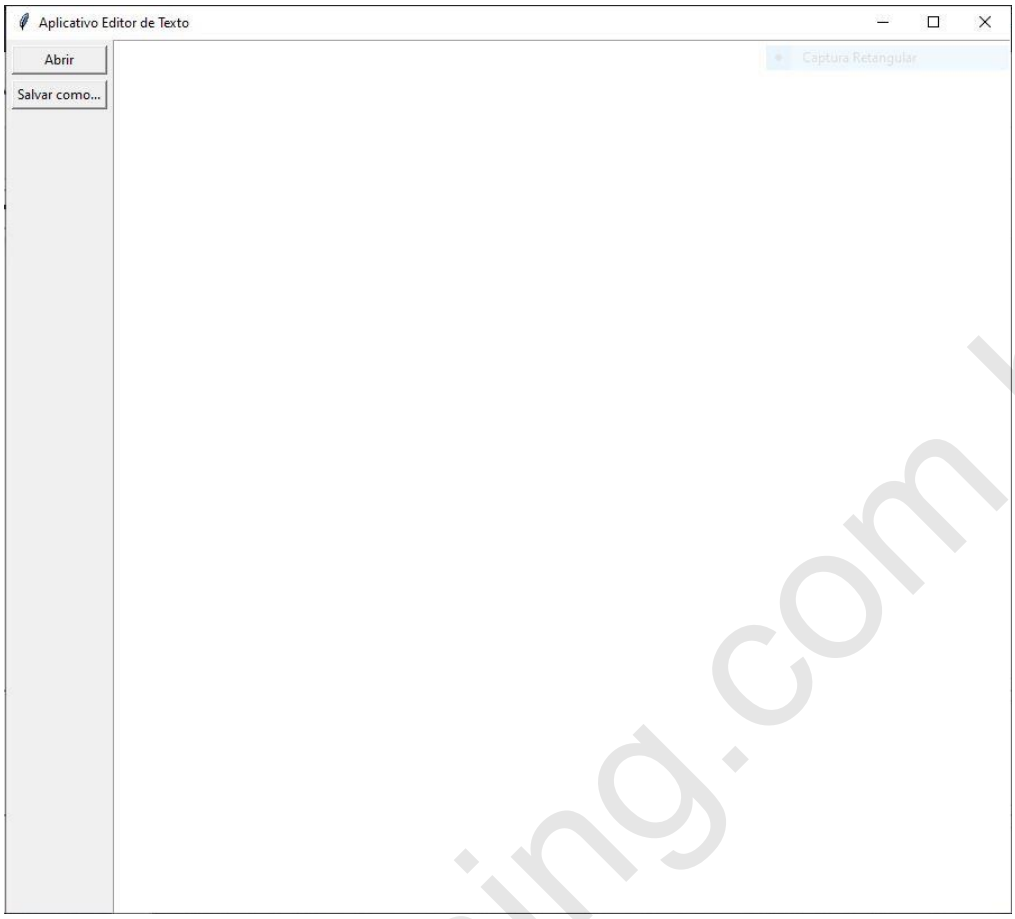
segunda coluna para que *fr_buttons* apareça à esquerda *txt_edit* no layout da janela.

O parâmetro *sticky* for *fr_buttons* é definido como "ns", o que força o 'frame' inteiro a se expandir verticalmente e preencher toda a altura de sua coluna. *txt_edit* preenche toda a célula da grade porque você define o parâmetro *sticky* como "nsew", o que a força a expandir em todas as direções .

Agora que o layout do aplicativo está completo, adicione *window.mainloop()* na parte inferior do programa e salve. Observe o código completo, até aqui:

```
    # código completo até este ponto

    import tkinter as tk

    window = tk.Tk()
    window.title("Aplicativo Editor de Texto")

    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window)
    btn_open = tk.Button(fr_buttons, text="Abrir")
    btn_save = tk.Button(fr_buttons, text="Salvar como..."
)

    btn_open.grid(row=0, column=0, sticky="ew", padx=5, pa
dy=5)
    btn_save.grid(row=1, column=0, sticky="ew", padx=5)

    fr_buttons.grid(row=0, column=0, sticky="ns")
    txt_edit.grid(row=0, column=1, sticky="nsew")

    window.mainloop()
```

Agora execute o arquivo. A seguinte janela é exibida:



Até agora, nosso código obedeceu nossas instruções, ou seja, ainda não é possivel abrir ou salvar um arquivo. Neste próximo passo vamos iniciar a implementação das intruções para o funcionamento dos botões. **btn_open** precisa mostrar uma caixa de diálogo de abertura de arquivo e permitir que o usuário selecione um arquivo. Em seguida, ele precisa abrir esse arquivo e definir o texto *txt_edit* para o conteúdo do arquivo. Abaixo, a função *open_file()* faz exatamente isso. Observe o código:

```
    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
        )
        if not filepath:
            return
        txt_edit.delete("1.0", tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.insert(tk.END, text)
        window.title(f"Aplicativo editor de Texto - {filepath}")
```

Vamos entender aquilo que nosso código está fazendo:

As **linhas 3 a 5** usam a caixa de diálogo *askopenfilename* do tkinter. O módulo *filedialog* exibe uma caixa de diálogo de abertura de arquivo e armazena o caminho de arquivo selecionado *filepath*.

As **linhas 6 e 7** verificam se o usuário fecha a caixa de diálogo ou clica no botão *Cancelar* . Nesse caso, *filepath* será **None** e a função será return chamada sem a execução de nenhum código para ler o arquivo e definir o texto de *txt_edit*.

A **linha 8** limpa o conteúdo atual do *txt_edit* usando o método *.delete()*. As **linhas 9 e 10** abrem o arquivo selecionado e *.read()* "lê" seu conteúdo antes de armazená-lo em *text* como uma sequência.

A **linha 11** atribui a string *text* a ser usada em *txt_edit* usando *.insert()*. A **linha 12** define o título da janela para que ela contenha o caminho do arquivo aberto.

Agora, podemos atualizar o programa para que as chamadas *btn_open* e *open_file()* sempre que for clicado. Primeiro: importe *askopenfilename()* de *tkinter.filedialog* adicionando a seguinte importação à parte superior do programa:

```
    import tkinter as tk
    # adicionar essa importação para o módulo de dependenc
ia
    from tkinter.filedialog import askopenfilename

    window = tk.Tk()
    window.title("Aplicativo editor de Texto")

    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window)
    btn_open = tk.Button(fr_buttons, text="Abrir")
    btn_save = tk.Button(fr_buttons, text="Salvar como..."
)
```

Em seguida, adicione a definição *open_file()* logo abaixo das instruções de importação:

```
    #######adiconar o este bloco##############
    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")]
        )
        if not filepath:
            return
        txt_edit.delete("1.0", tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.insert(tk.END, text)
        window.title(f"Aplicativo editor de Texto - {filep
ath}")

    #######adiconar o este bloco##############

    window = tk.Tk()
    window.title("Aplicativo editor de Texto")

    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window)
    btn_open = tk.Button(fr_buttons, text="Abrir")
    btn_save = tk.Button(fr_buttons, text="Salvar como..."
)
```

Por fim, defina o atributo *command* de *btn_open* como *open_file*:

```
    import tkinter as tk
    from tkinter.filedialog import askopenfilename

    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")]
        )
        if not filepath:
            return
```

```
            txt_edit.delete("1.0", tk.END)
            with open(filepath, "r") as input_file:
                text = input_file.read()
                txt_edit.insert(tk.END, text)
            window.title(f"Aplicativo editor de Texto - {filep
ath}")

        window = tk.Tk()
        window.title("Aplicativo editor de Texto")

        window.rowconfigure(0, minsize=800, weight=1)
        window.columnconfigure(1, minsize=800, weight=1)

        txt_edit = tk.Text(window)
        fr_buttons = tk.Frame(window)
        # a linha de definição indica no enunciado
        btn_open = tk.Button(fr_buttons, text="Abrir", command
=open_file)
        btn_save = tk.Button(fr_buttons, text="Salvar como..."
)
```

Agora, vamos implementar as intruções para *btn_save*. É necessário abrir uma caixa de diálogo **Salvar arquivo** para que o usuário possa escolher onde deseja salvar o arquivo. Usaremos a *asksaveasfilename* do módulo de dependencia *tkinter.filedialog*. Essa função também precisa extrair o texto de *txt_edit* e gravá-lo em um arquivo no local selecionado. Observe o código abaixo:

```
        def save_file():
            """Salvar o arquivo atual como um novo arquivo."""
            filepath = asksaveasfilename(
                defaultextension="txt",
                filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")],
            )
            if not filepath:
                return
            with open(filepath, "w") as output_file:
                text = txt_edit.get("1.0", tk.END)
                output_file.write(text)
            window.title(f"Aplicativo editor de Texto - {filep
ath}")
```

Vamos entender aquilo que o código está executando:

- As **linhas 3 a 6** *usam a asksaveasfilename* para obter o local de salvamento desejado do usuário. O caminho do arquivo selecionado é armazenado na variável *filepath*.
- As **linhas 7 e 8** verificam se o usuário fecha a caixa de diálogo ou clica no botão **Cancelar** . Nesse caso,*filepath* será **None** e a função retornará sem executar nenhum código para salvar o texto em um arquivo.
- A **linha 9** cria um novo arquivo no caminho do arquivo selecionado.
- A **linha 10 extrai o texto *txt_edit* com o método *.get()* e o atribui à variável text.**
- **A linha 11 grava textno arquivo de saída.**
- **A linha 12 atualiza o título da janela para que o novo caminho do arquivo seja exibido no título da janela.**

Agora, é possível atualizar o programa para que btn_save chame save_file() quando for clicado. Importe asksaveasfilename() do módulo de dependencia tkinter.filedialog atualizando a importação na parte superior do seu script. Observe o código abaixo:

```
    import tkinter as tk
    # adicionar essa importação para o módulo de dependenc
ia
    from tkinter.filedialog import askopenfilename, asksav
easfilename

    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")]
        )
        if not filepath:
            return
        txt_edit.delete(1.0, tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.insert(tk.END, text)
        window.title(f"Aplicativo editor de Texto - {filep
ath}")

    window = tk.Tk()
    window.title("Aplicativo editor de Texto")
    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window, relief=tk.RAISED, bd=2)
```

```
    btn_open = tk.Button(fr_buttons, text="Abrir", command
=open_file)
    btn_save = tk.Button(fr_buttons, text="Salavar como...
")

    btn_open.grid(row=0, column=0, sticky="ew", padx=5, pa
dy=5)
    btn_save.grid(row=1, column=0, sticky="ew", padx=5)

    fr_buttons.grid(row=0, column=0, sticky="ns")
    txt_edit.grid(row=0, column=1, sticky="nsew")

    window.mainloop()
```

Em seguida, adicione a definição save_file() logo abaixo da open_file():

```
    import tkinter as tk
    # adicionar essa importação para o módulo de dependenc
ia
    from tkinter.filedialog import askopenfilename, asksav
easfilename

    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")]
        )
        if not filepath:
            return
        txt_edit.delete(1.0, tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.insert(tk.END, text)
        window.title(f"Aplicativo editor de Texto - {filep
ath}")
    ##########################colocar a função save_file a
qui ##########################
    def save_file():
        """Salvar o arquivo atual como um novo arquivo."""
        filepath = asksaveasfilename(
            defaultextension="txt",
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")],
        )
        if not filepath:
            return
```

```
        with open(filepath, "w") as output_file:
            text = txt_edit.get("1.0", tk.END)
            output_file.write(text)
        window.title(f"Aplicativo editor de Texto - {filep
ath}")
    ##########################colocar a função save_file a
qui ##########################
    window = tk.Tk()
    window.title("Aplicativo editor de Texto")
    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window, relief=tk.RAISED, bd=2)
    btn_open = tk.Button(fr_buttons, text="Abrir", command
=open_file)
    btn_save = tk.Button(fr_buttons, text="Salavar como...
")

    btn_open.grid(row=0, column=0, sticky="ew", padx=5, pa
dy=5)
    btn_save.grid(row=1, column=0, sticky="ew", padx=5)

    fr_buttons.grid(row=0, column=0, sticky="ns")
    txt_edit.grid(row=0, column=1, sticky="nsew")

    window.mainloop()
```

Por fim, defina o atributo command de btn_save como save_file:

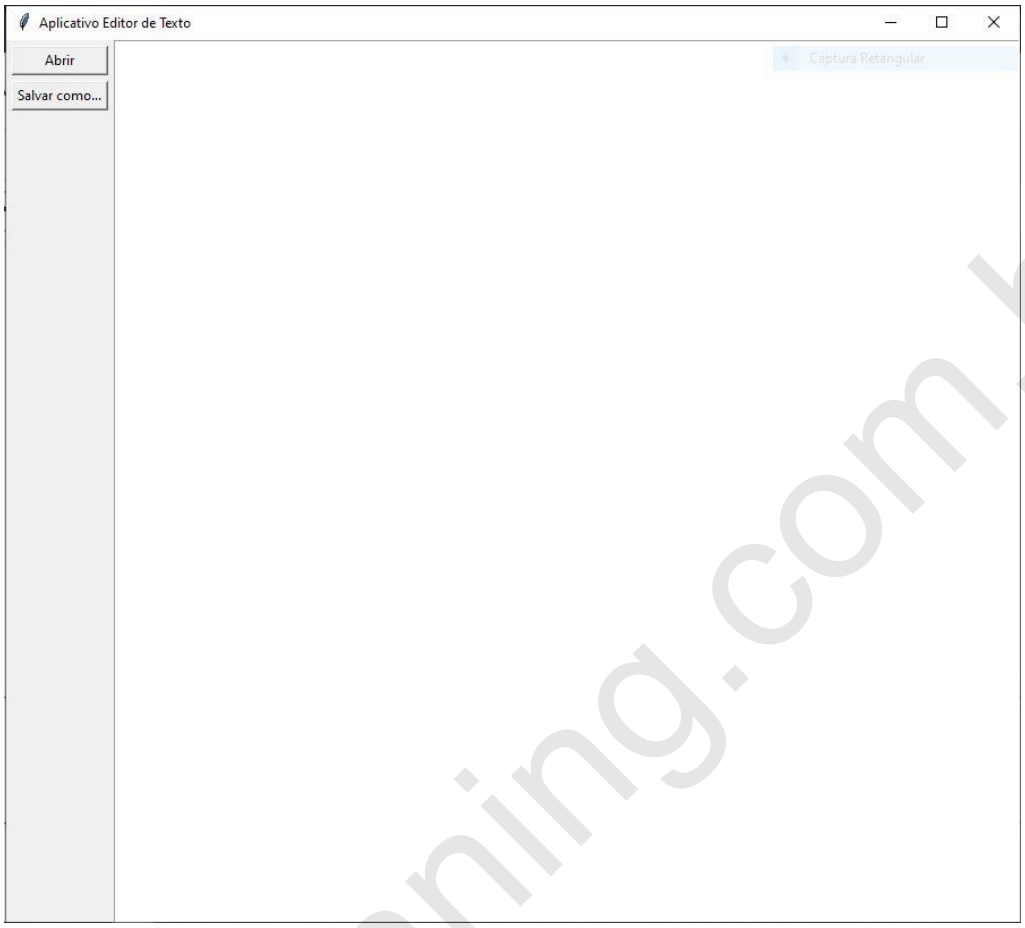
```
    import tkinter as tk
    # adicionar essa importação para o módulo de dependenc
ia
    from tkinter.filedialog import askopenfilename, asksav
easfilename

    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")]
        )
        if not filepath:
            return
        txt_edit.delete(1.0, tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.insert(tk.END, text)
```

```
        window.title(f"Aplicativo editor de Texto - {filep
ath}")

    def save_file():
        """Salvar o arquivo atual como um novo arquivo."""
        filepath = asksaveasfilename(
            defaultextension="txt",
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")],
        )
        if not filepath:
            return
        with open(filepath, "w") as output_file:
            text = txt_edit.get("1.0", tk.END)
            output_file.write(text)
        window.title(f"Aplicativo editor de Texto - {filep
ath}")
    window = tk.Tk()
    window.title("Aplicativo editor de Texto")
    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window, relief=tk.RAISED, bd=2)
    btn_open = tk.Button(fr_buttons, text="Abrir", command
=open_file)

    #### inserção aqui
    btn_save = tk.Button(fr_buttons, text="Salavar como...
",command=save_file)

    btn_open.grid(row=0, column=0, sticky="ew", padx=5, pa
dy=5)
    btn_save.grid(row=1, column=0, sticky="ew", padx=5)

    fr_buttons.grid(row=0, column=0, sticky="ns")
    txt_edit.grid(row=0, column=1, sticky="nsew")

    window.mainloop()
```

Este é nosso aplicativo totalmente funcional. Execute-o para ver o resultado:

```
    import tkinter as tk
    from tkinter.filedialog import askopenfilename, asksav
easfilename
```

```
    def open_file():
        """Abrir um arquivo para edição."""
        filepath = askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")]
        )
        if not filepath:
            return
        txt_edit.delete(1.0, tk.END)
        with open(filepath, "r") as input_file:
            text = input_file.read()
            txt_edit.insert(tk.END, text)
        window.title(f"Simple Text Editor - {filepath}")

    def save_file():
        """Save the current file as a new file."""
        filepath = asksaveasfilename(
            defaultextension="txt",
            filetypes=[("Text Files", "*.txt"), ("All File
s", "*.*")],
        )
        if not filepath:
            return
        with open(filepath, "w") as output_file:
            text = txt_edit.get(1.0, tk.END)
            output_file.write(text)
        window.title(f"Simple Text Editor - {filepath}")

    window = tk.Tk()
    window.title("Simple Text Editor")
    window.rowconfigure(0, minsize=800, weight=1)
    window.columnconfigure(1, minsize=800, weight=1)

    txt_edit = tk.Text(window)
    fr_buttons = tk.Frame(window, relief=tk.RAISED, bd=2)
    btn_open = tk.Button(fr_buttons, text="Open", command=
open_file)
    btn_save = tk.Button(fr_buttons, text="Save As...", co
mmand=save_file)

    btn_open.grid(row=0, column=0, sticky="ew", padx=5, pa
dy=5)
    btn_save.grid(row=1, column=0, sticky="ew", padx=5)

    fr_buttons.grid(row=0, column=0, sticky="ns")
    txt_edit.grid(row=0, column=1, sticky="nsew")

    window.mainloop()
```



Numpy

Bibliotecas Python para Análise de Dados: NumPy

Dentre os principais pacotes que formam o stack de ciência de dados do Python, estão o NumPy, Pandas, Matplotlib, SciPy, Scikit-Learn, Bokeh, StatsModels e o Seaborn. Pretendo mostrar um pouco de cada um deles nesta sequência de posts.

Lembrando que, caso você já utilize o Anaconda, todos os pacotes citados acima já estão inclusos.

NumPy

O NumPy é um pacote voltado para computação matemática, e um dos mais importantes do PyData Stack. Ele oferece as bases matemáticas

necessárias para construção de modelos de deep learning, machine learning e, consequentemente, aplicações de inteligência artificial. É possível utilizar os objetos nativos do NumPy para criação arrays ou matrizes, e assim usufruir das funções matemáticas oferecidas para operações com esses objetos.

Vamos começar importando o NumPy. Também é possível utilizar: from numpy import * . Isso evitará a utilização de np., mas este comando importará todos os módulos do NumPy.

```
# Importando o NumPy
import numpy as np
```

Agora vamos criar uma matriz e utilizar os alguns dos métodos fornecidos pelo NumPy:

```
# Criando uma matriz a partir de uma lista de listas
grupo = [[31,28,89], [90, 54, 69], [12, 55, 77]]
minhaMatriz = np.matrix(grupo)

type(minhaMatriz)
#=> numpy.matrixlib.defmatrix.matrix

# Formato da matriz
np.shape(minhaMatriz)
#=> (3, 3)

print(minhaMatriz.dtype)
#=> int64
```

Também podemos utilizar o NumPy para realizar operações estatísticas, embora tenhamos outros pacotes do Python para isso, como por exemplo, o StatsModel, do qual irei falar na continuação deste post. Vamos ao exemplo:

```
novoArray = np.array([90, 54, 12, 55, 31,28,])


# Calculando a média dos valores
np.mean(novoArray)
#=> 45.0

# Calculando o desvio padrão (variação ou dispersão que existe em relação à média ou valor esperado)
```

```
np.std(novoArray)
#=> 25.099800796022265

# Calculando a variância
np.var(novoArray)
#=> 630.0
```

Referencias Bibliograficas

https://www.tutorialspoint.com/python3/index.htm

https://www.programiz.com/python-programming/if-elif-else

https://www.programiz.com/python-programming/for-loop

https://www.programiz.com/python-programming/while-loop

https://www.programiz.com/python-programming/break-continue

https://www.analyticsvidhya.com/blog/2015/06/regular-expression-python/

https://medium.com/horadecodar/como-fazer-webscraping-com-python-e-beautiful-soup-28a65eee2efd