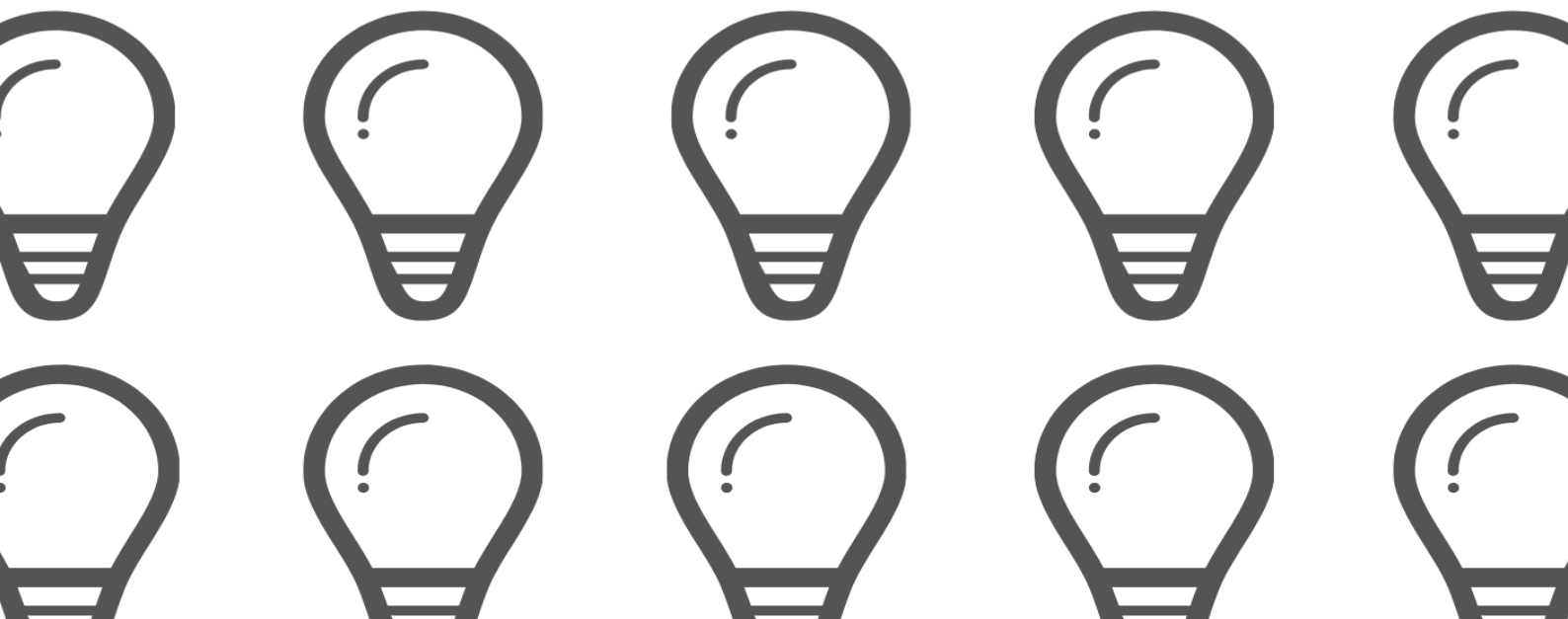


TIAGO DIAS

INICIAÇÃO *EM* *python*™

Começando do ZERO



Sumário

| | | |
|----------|---|-----------|
| | Sumário | 1 |
| | Introdução | 4 |
| 1 | CAPÍTULO - A LINGUAGEM PYTHON | 5 |
| 1.1 | O que é Python? | 5 |
| 1.2 | Por que usar Python? | 6 |
| 1.3 | Ferramentas de Desenvolvimento | 7 |
| 1.3.1 | Editores de Texto | 7 |
| 1.3.2 | IDEs Gratuitas | 7 |
| 1.4 | Resumo do Capítulo | 8 |
| 2 | CAPÍTULO - PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO | 9 |
| 2.1 | Instalando o Python no Windows | 9 |
| 2.1.1 | Adicionando o python no path | 9 |
| 2.1.2 | Instalando o pip | 10 |
| 2.2 | Instalando o Python no Linux | 10 |
| 2.2.1 | Instalação por Gerenciadores de Pacotes | 10 |
| 2.3 | Instalando bibliotecas no Python | 11 |
| 2.3.1 | Bibliotecas no Windows | 12 |
| 2.3.2 | Bibliotecas no Linux | 12 |
| 2.4 | Partiu Python! | 12 |
| 2.5 | Resumo do Capítulo | 12 |
| 3 | CAPÍTULO - CONCEITOS INICIAIS | 14 |
| 3.1 | Executando o Python | 14 |
| 3.2 | Indentação no Python | 17 |
| 3.3 | Imprimindo com Python | 18 |
| 3.4 | Imputando dados com Python | 19 |
| 3.5 | Resumo do Capítulo | 20 |
| 4 | CAPÍTULO - VARIÁVEIS E ESTRUTURAS DE DADOS | 21 |

| | | |
|------------|---|-----------|
| 4.1 | Variáveis | 21 |
| 4.1.1 | Nomeando variáveis | 21 |
| 4.1.2 | Números | 22 |
| 4.1.2.1 | Operações com Números | 22 |
| 4.1.2.1.1 | Operações Aritméticas | 22 |
| 4.1.2.1.2 | Operações Lógicas | 27 |
| 4.1.3 | Booleano | 31 |
| 4.1.3.1 | Operações com Booleano | 31 |
| 4.1.4 | Strings | 33 |
| 4.1.4.1 | Operações com Strings | 33 |
| 4.1.4.1.1 | Métodos de Strings | 33 |
| 4.1.4.1.2 | Expressões regulares | 38 |
| 4.2 | Estruturas de Dados | 40 |
| 4.2.1 | Listas | 40 |
| 4.2.1.1 | Exemplos de Listas: | 40 |
| 4.2.1.2 | Operações com Listas | 40 |
| 4.2.2 | Tuplas | 47 |
| 4.2.2.1 | Exemplos de Tuplas: | 47 |
| 4.2.2.2 | Operações com Tuplas | 47 |
| 4.2.3 | Dicionários | 50 |
| 4.2.3.1 | Exemplos de Dicionários: | 50 |
| 4.2.3.2 | Operações com Dicionários | 50 |
| 4.3 | Resumo do Capítulo | 54 |
| 5 | CAPÍTULO - LOOPS, CONDICIONAIS E FUNÇÕES | 55 |
| 5.1 | Condicionais | 55 |
| 5.1.1 | Condicionais Simples - if | 55 |
| 5.1.1.1 | Exemplo de Condicional Simples | 55 |
| 5.1.2 | Condicionais Compostos - if/else | 56 |
| 5.1.2.1 | Exemplo de Condicional Composto | 56 |
| 5.1.3 | Condicionais aninhados - if/elif/else | 57 |
| 5.1.3.1 | Exemplo de Condicional Aninhado | 57 |
| 5.2 | Loops | 58 |
| 5.2.1 | For | 58 |
| 5.2.1.1 | Exemplo de Loop for | 58 |
| 5.2.2 | While | 59 |

| | | |
|------------|---|-----------|
| 5.2.2.1 | Exemplo de Loop while | 59 |
| 5.2.3 | Continue e Break | 60 |
| 5.2.3.1 | Exemplo de continue | 60 |
| 5.2.3.2 | Exemplo de break | 61 |
| 5.3 | Funções | 62 |
| 5.3.1 | Funções da Biblioteca matplotlib | 62 |
| 5.3.1.1 | Função Pyplot | 62 |
| 5.3.2 | Funções da Biblioteca nltk | 63 |
| 5.3.2.1 | Função word_tokenize | 63 |
| 5.3.2.2 | Função FreqDist | 64 |
| 5.3.3 | Funções da Biblioteca numpy | 65 |
| 5.3.3.1 | Função arange | 65 |
| 5.3.3.2 | Função linspace | 66 |
| 5.3.4 | Funções da Biblioteca wordcloud | 67 |
| 5.3.4.1 | Função WordCloud | 67 |
| 5.4 | Resumo do Capítulo | 70 |
| 6 | CAPÍTULO - TRABALHANDO COM ARQUIVOS DE TEXTO | 72 |
| 6.1 | Lendo um Arquivos de Texto | 72 |
| 6.1.0.1 | Exemplo Lendo um Arquivos de Texto: | 72 |
| 6.2 | Alterando um Arquivos de Texto | 73 |
| 6.2.0.1 | Exemplo Alterando um Arquivos de Texto: | 73 |
| 6.3 | Resumo do Capítulo | 74 |
| 7 | CAPÍTULO - APLICAÇÃO DE CONTAGEM DE PALAVRAS | 75 |
| 7.1 | Aplicação de Contagem de Palavras | 75 |
| 7.2 | Resumo do Capítulo | 79 |
| 8 | CAPÍTULO - APLICAÇÃO DE NUVEM DE PALAVRAS | 80 |
| 8.1 | Aplicação de Nuvem de Palavras | 80 |
| 8.2 | Resumo do Capítulo | 83 |
| | Considerações finais | 84 |
| | REFERÊNCIAS | 85 |

Introdução

Vamos fazer uma viagem na linguagem de programação Python, com o objetivo de tirar o usuário da inércia, levando ele do conhecimento ZERO ao estágio inicial de conhecimento da linguagem Python, ou você usuário que já tem um conhecimento básico na linguagem e quer aprender coisas novas com manipulação de dados com Python.

Quero pegar na sua mão e te levar a outro patamar, com um passo a passo detalhado que vai te proporcionar a criação do seus primeiros códigos Python, ou de novos códigos para você que já é um conhecedor da linguagem.

Vou te apresentar alguns conceitos importantes para entendimento, te ajudar na preparação do seu ambiente de desenvolvimento seja usuário Linux ou Windows, então vamos criar alguns códigos de comandos básicos até evoluir o seu conhecimento para chegar na criação das aplicações em Python.

Este não é um manual completo de tudo que o Python pode fazer, apenas um pontapé inicial para abrir sua mente e te mostrar o potencial dessa ferramenta.

1 CAPÍTULO - A Linguagem Python

Vamos ver alguns conceitos introdutórios para que possamos entender o que é Python e porque utilizar essa linguagem que esta crescendo de maneira incrível.

1.1 O que é Python?

Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. O padrão de facto é a implementação CPython.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

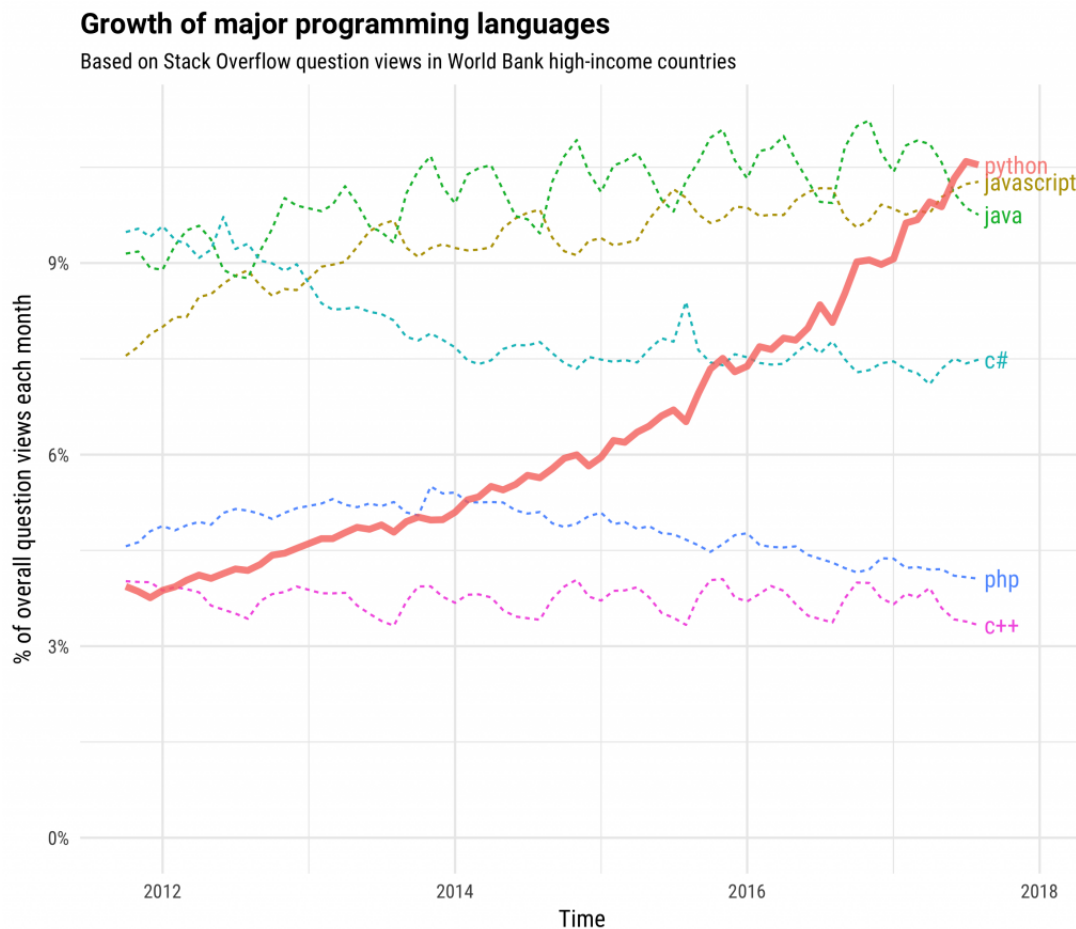
Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é principalmente utilizada para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web. Foi considerada pelo público a 3ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018, e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk.

O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome.

1.2 Por que usar Python?

O crescimento da linguagem ao longo dos últimos anos é um sinal do grande potencial e a conquista com o maior número de usuário na sua utilização, deixando para trás linguagem que dominaram o todo do mercado nos últimos anos. Como podemos ver na Figura 1.

Figura 1 – O Incrível Crescimento do Python



Fonte: <https://stackoverflow.blog/2017/09/06/incredible-growth-python/>

O Stack Overflow monitora a interação de programadores com perguntas e respostas em tópicos como forma de medir a quantidade de usuários e as interações entre eles.

Dessa forma é possível perceber o crescimento da comunidade Python e a evolução exponencial dos desenvolvedores, sendo hoje uma linguagem bastante importante, principalmente para a área de análise de dados.

1.3 Ferramentas de Desenvolvimento

Veja aqui uma listagem de algumas ferramentas para te auxiliar no desenvolvimento python:

1.3.1 Editores de Texto

- ◇ **Notepad++** é um editor de texto e de código fonte de código aberto sob a licença GPL. Suporta várias linguagens de programação rodando sob o sistema Microsoft Windows. O Notepad++ é distribuído como um Software livre.
- ◇ **Sublime Text** é um software multiplataforma de edição de texto, no entanto utilizado por muitos desenvolvedores para editar código-fonte, escrito em linguagem Python. Inicialmente, o programa foi pensado para ser uma extensão do Vim. Sublime Text é um editor de texto, e não uma IDE como alguns dizem.
- ◇ **Atom** é open source e feito pelo Github e com suporte para várias linguagens, dentre elas o Python. É integrado ao Git e Github, sendo possível mexer com o Git e Github através da interface do editor de texto. Ótimo para iniciantes.
- ◇ **Visual Studio Code** é open source e free, desenvolvido pela Microsoft. Suporta inúmeras linguagens de programação.

1.3.2 IDEs Gratuitas

- ◇ **PyCharm community** é desenvolvido pela companhia JetBrains. Esta edição é liberada sob a licença da Apache. É multiplataforma. Essa IDE fornece análise de código, um depurador gráfico, um testador de unidade integrado, integração com sistemas de controle de versão (VCSes), e suporta desenvolvimento de web com Django.
- ◇ **Komodo-Edit** também desenvolvido pela ActiveState o Komodo-Edit é uma excelente opção de editor, bastante rico em recursos tais como autocomplete, calltips, multi-language file support, syntax coloring, syntax checking, Vi emulation, Emacs key bindings e outros.
- ◇ **NetBeans** analogamente ao Eclipse, o NetBeans também oferece suporte ao Python através de plugins.

1.4 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ O que é Python.
- ◇ Porque usar Python.
- ◇ As ferramentas para desenvolvimento Python.

Se não conseguiu entender todos esse conceitos, releia o capítulo, pois vamos precisar desse entendimentos no decorrer dos próximos capítulos. Vamos agora preparar o ambiente de desenvolvimento.

2 CAPÍTULO - Preparação do Ambiente de Desenvolvimento

Vamos ver detalhadamente a instalação do Python e suas bibliotecas que são uma coleção de comandos para simplificar o processo de programação e remover a necessidade de reescrever os comandos mais usados. Eles podem ser usados importando-os no início de um script. Vamos instalar algumas bibliotecas necessárias para o desenvolvimento das nossas aplicações.

2.1 Instalando o Python no Windows

Realizar o download em <https://www.python.org/downloads/> da Release version Python 3.7 e realizar a instalação.

2.1.1 Adicionando o python no path

Para que você consiga rodar o python pela linha de comando é necessário adicioná-lo no path do sistema que pode ser feito seguindo os passos a seguir:

1. Abra o painel de controle
2. Navegue até sistema
3. Selecione as configurações avançadas do sistema
4. Clique em variáveis de ambiente
5. Procure nas variáveis do sistema pela variável "Path"
6. Clique em editar
7. Verifique se o caminho onde está o Python está na variável, caso não exista adicione ao final dos valores separando cada valor com ";"

Abaixo figura mostrando o caminho:

The image consists of two side-by-side screenshots from a Windows 7 desktop, illustrating the steps to add a Python path to the system environment variables. Red boxes and numbers highlight the key actions.

Left Screenshot: Navigating to System Properties

- 1** The address bar in the Control Panel shows the path: **Painel de Controle > Sistema e Segurança > Sistema**.
- 2** The **Sistema** icon in the left sidebar is highlighted.
- The **Propriedades do Sistema** window is open, with the **Avançado** (Advanced) tab selected.
- 4** The **Variáveis de Ambiente...** button at the bottom of the window is highlighted.

Right Screenshot: Editing the Path Variable

- The **Variáveis de Ambiente** (Environment Variables) window is open.
- 5** The **Path** variable under the **Variáveis do sistema** (System variables) section is selected.
- 6** The **Editar...** (Edit...) button for the selected variable is highlighted.
- A zoomed-in view of the **Editar Variável de Sistema** (Edit System Variable) dialog is shown below:
 - Nome da variável:** Path
 - Valor da variável:** `local\Programs\Python\Python37-32\Scripts`

Para que seja possível instalar as bibliotecas do python é necessário que você tenha o pip instalado no sistema. Para isso acesse o cdm (com permissões de administrador) e na linha de comando rode:

2.2 Instalando o Python no Linux

Os gerenciadores de pacotes mais comuns são apt-get (Debian, Ubuntu) e yum (RedHat, CentOS). Caso sua distribuição utilize um gerenciador de pacotes diferente, acesse a página

de downloads do Python.

Apt-get

Para instalar o Python 3.5, digite em um terminal:

```
$ sudo apt-get install python3.5
```

Para instalar o gerenciador de pacotes pip, digite em um terminal:

```
$ sudo apt-get install python-pip
```

```
$ sudo apt-get install python-pip
```

Yum

Para instalar o Python 3.5, digite em um terminal:

```
$ sudo yum install python35
```

Para instalar o gerenciador de pacotes pip, digite em um terminal:

```
$ yum -y install python-pip
```

2.3 Instalando bibliotecas no Python

Segue lista e uma breve explicação das bibliotecas que vamos instalar. Já podemos instalar essas bibliotecas listadas abaixo pois vamos utilizar na criação da nossa aplicação.

1. **NumPy** tradicionalmente, começamos nossa lista com as bibliotecas para aplicativos científicos e o NumPy é um dos principais pacotes nessa área. Ele é destinado ao processamento de grandes matrizes e matrizes multidimensionais, e uma extensa coleção de funções matemáticas de alto nível e métodos implementados possibilitam a execução de várias operações com esses objetos. Essa biblioteca pode vir instalada a depender da versão do Python, podendo não ser necessária a sua instalação.
2. **Matplotlib** é uma biblioteca de baixo nível para criar diagramas e gráficos bidimensionais. Com este pacote, você pode construir gráficos diversos, desde histogramas e gráficos de dispersão a gráficos de coordenadas não cartesianas. Além disso, muitas bibliotecas de plotagem populares são projetadas para trabalhar em conjunto com o matplotlib.
3. **NLTK** é um conjunto de bibliotecas, uma plataforma completa para Processamento de Linguagem Natural. Com a ajuda do NLTK, você pode processar e analisar texto de várias maneiras, aplicar tokenize e tag, extrair informações, etc. O NLTK também é usado para prototipar e construir sistemas de pesquisa.

4. **Wordcloud** (ou nuvem de tags) é uma representação visual dos dados de texto. Exibe uma lista de palavras, a importância de cada exibição mostrada com tamanho ou cor da fonte. Este formato é útil para perceber rapidamente os termos mais importantes. O Python está totalmente adaptado para desenhar esse tipo de representação, graças à biblioteca wordcloud desenvolvida por Andreas Mueller, para a instalação desta biblioteca pode ser necessário a instalação das Ferramentas de Build do Visual Studio versão 14.0 ou superior.

2.3.1 Bibliotecas no Windows

Para instalar as bibliotecas no Windows acesse o cmd e na linha de comando rode:

```
python -m pip install <Nome da Biblioteca>
```

2.3.2 Bibliotecas no Linux

Para instalar as bibliotecas no Linux acesse o cmd e na linha de comando rode:

```
$ sudo pip install -U <Nome da Biblioteca>
```

2.4 Partiu Python!

Agora já sabemos "O que é Python?", "Por que usar Python", as ferramenta de desenvolvimento e preparamos todo o ambiente de desenvolvimento vamos partir para a prática, começar com comando básicos até o desenvolvimento de uma aplicação de análise de dados.

A partir desse ponto, devido a similaridade de execução tanto em Windows quanto em Linux, todos os códigos apresentados como exemplo serão apresentados em Windows.

2.5 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Instalação do Python no Windows e no Linux.
- ◇ Instalação das bibliotecas Python em Windows e Linux.

Se não conseguiu entender todos esse conceitos, releia o capítulo, pois vamos precisar desse entendimentos no decorrer dos próximos capítulos. Vamos agora para o alguns conceitos iniciais.

3 CAPÍTULO - Conceitos Iniciais

Vamos ver detalhadamente como é a execução em Python e as formas possíveis, a indentação da linguagem Python e os comandos de impressão (`print()`) e imputação de dados (`input()`).

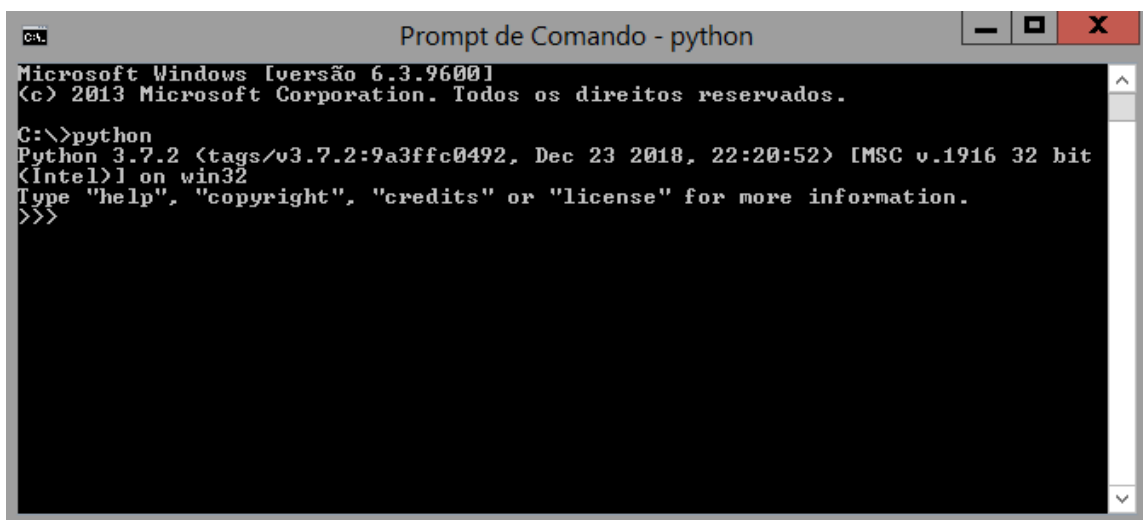
3.1 Executando o Python

Para executar uma linha de comando ou uma aplicação escrita em Python, temos algumas formas para fazer isso, e fica a critério do programador. Vamos ver essas formas execução.

◇ Interpretador Local

Ao fazer a instalação do Python podemos abrir diretamente no cmd do Windows o interpretador Python e executar linhas de comandos escritas na linguagem. Abrindo o cmd digitamos apenas o comando `python` e será executado o interpretador conforme Figura 3, a partir desse momento já podemos inserir a linhas de comando e o interpretador apresentará o resultado.

Figura 3 – Interpretador no Windows

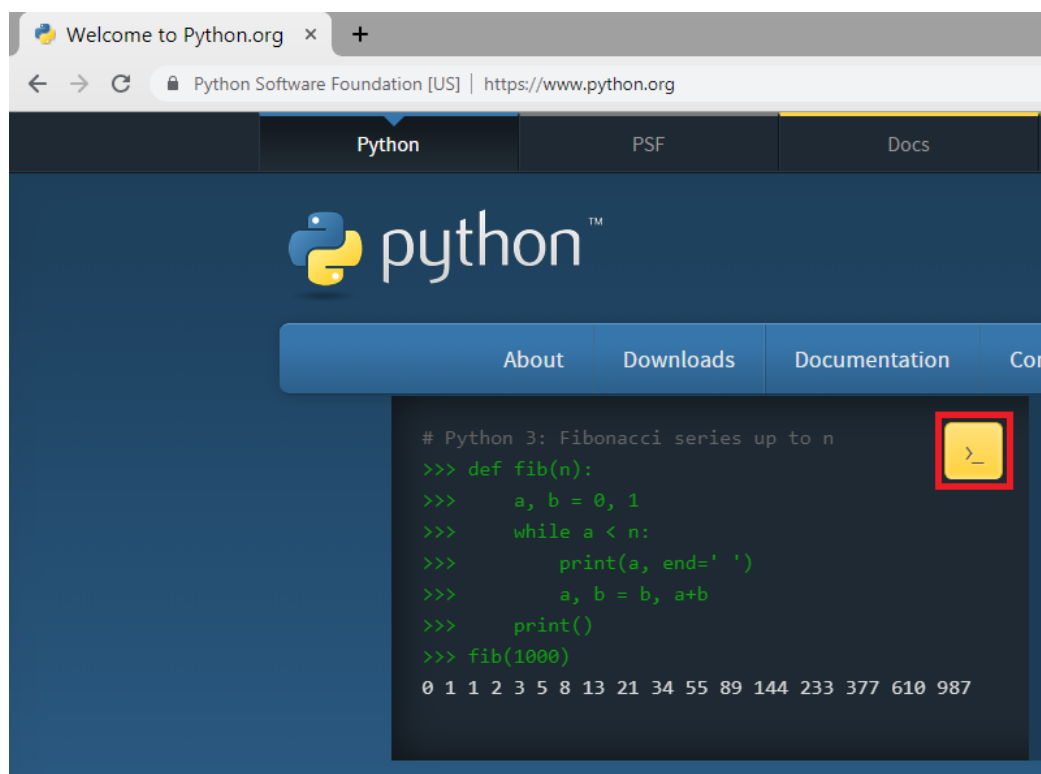


◇ Interpretador Online

No site oficial do Python, é disponibilizado um interpretador online, semelhante ao interpretador local, mas sem necessidade de instalação do Python.

Na Figura 4 podemos ver a tela inicial do site, clicando no item em destaque 'launch interactive Shell'.

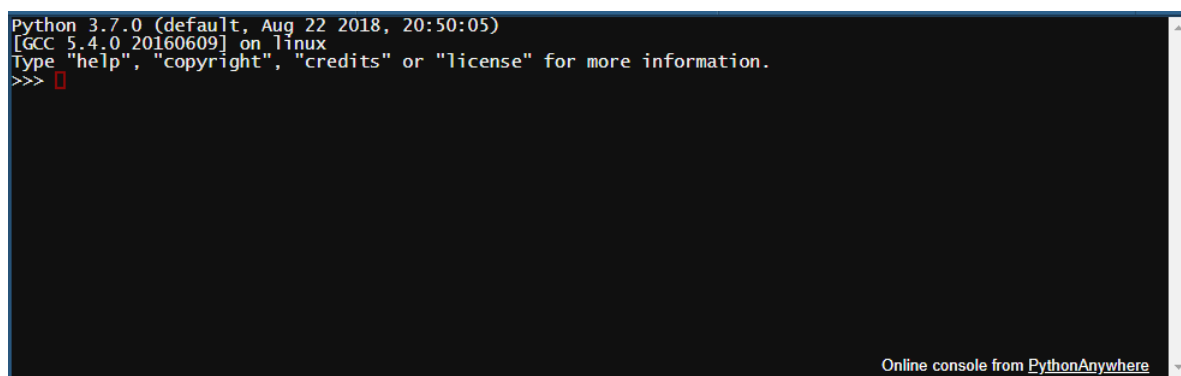
Figura 4 – Interpretador Online



Fonte: <https://www.python.org/>

Então temos um console online do interpretador Python conforme Figura 5.

Figura 5 – Interpretador Online



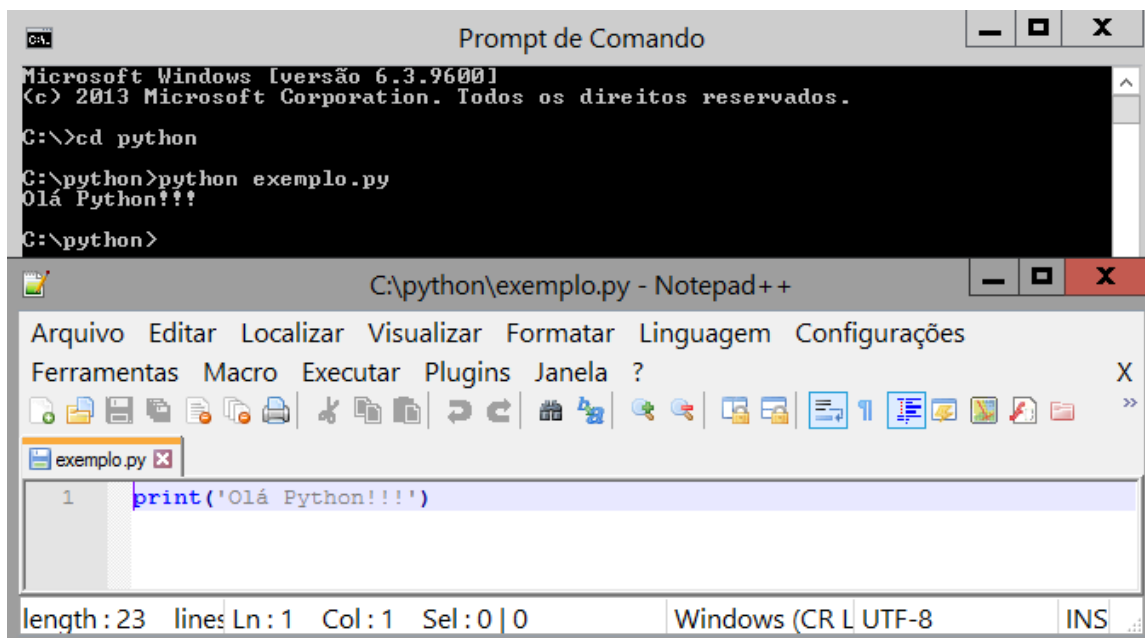
Fonte: <https://www.python.org/>

Dessa forma podemos utilizar o interpretador em qualquer sistema operacional, sem a instalação de programas, apenas com acesso a internet.

◇ Executando Arquivos

Outra forma que podemos utilizar, é a execução de arquivos criados em editores de texto e salvos com extensão '.py'. Basta acessar o diretório onde o arquivo está salvo através do cmd e chamar o interpretador e indicar o arquivo com o comando `python nome_do_arquivo.py`, conforme exemplo na Figura 6.

Figura 6 – Executando Arquivos



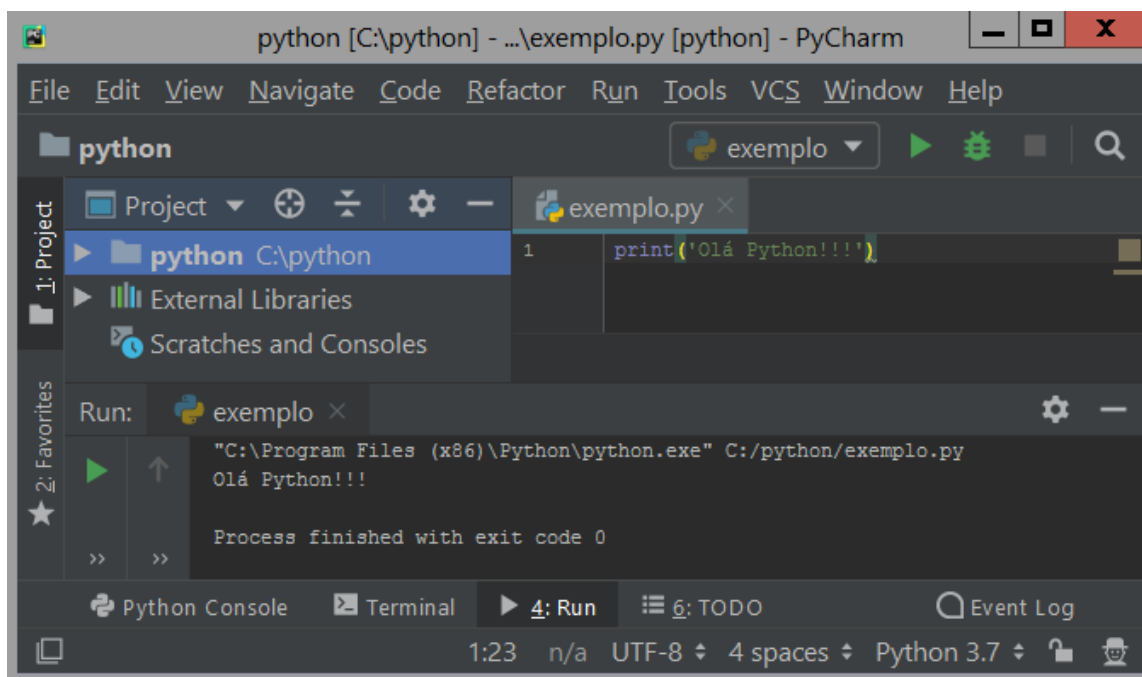
Executar comandos salvos em arquivos facilita a reutilização do código e correção dos códigos sem a reescrita no interpretador sem falar da praticidade de utilização de editores de texto, utilizamos no exemplo o Notepad++.

◇ Utilizando IDEs

A utilização de IDEs pode auxiliar no desenvolvimento de aplicações python devido a várias ferramentas que ajudando o usuário, como indentação do código, identificação de erros de maneira mais fácil com indicação do erro antes da execução e a linha que apresenta o erro, organização no desenvolvimento dentre outras facilidades.

A execução nas IDEs é bem simples normalmente existem, botões, atalhos, execução com depuração, somente com um clique as instruções são executadas.

Figura 7 – Utilizando IDEs



No nosso exemplo na Figura 7 utilizamos o PyCharm community, após configurar o caminho do interpretador Python, basta apenas clicar no botão ou utilizar o atalho para que a aplicação seja executada.

3.2 Indentação no Python

A indentação é uma característica peculiar no Python. Enquanto em outras linguagens os blocos são delimitados explicitamente, em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual, sem delimitadores de blocos.

Python requer uma indentação padronizada enquanto em outras linguagens a indentação não é necessária devido aos delimitadores de blocos, sendo importante apenas para melhor visualização.

Dessa forma a indentação incorreta ou a falta dela pode causar erros na execução ou até mesmo nem compilar o código, apresentando um erro na execução, caso utilize um editor de texto comum, para não haver erros de indentação, configure o editor para a indentação do Python, no caso das IDEs que suportam Python pode auxiliar o uso da função de indentação automática.

Vamos ver abaixo um mesmo trecho de código com a indentação correta e incorreta, os comandos vamos aprender mais para frente, vamos observar a estrutura hierárquica da indentação.

◇ Indentação Incorreta

```
letras = ['a', 'b', 'c'] # Bloco do primeiro nível hierárquico
for n in letras: # Bloco do primeiro nível hierárquico
    if n == 'b': # Bloco do primeiro nível hierárquico
        continue # Bloco do primeiro nível hierárquico
    print( n ) # Bloco do primeiro nível hierárquico
```

Podemos ver que todos os blocos de comando estão no mesmo nível hierárquico, não é possível identificar a hierarquia dos comandos, por isso o Python não consegue executar esse código e apresenta uma mensagem de erro ao usuário.

◇ Indentação Correta

```
letras = ['a', 'b', 'c'] # Bloco do primeiro nível hierárquico
for n in letras: # Bloco do primeiro nível hierárquico
    if n == 'b': # Bloco do segundo nível hierárquico
        continue # Bloco do terceiro nível hierárquico
    print( n ) # Bloco do segundo nível hierárquico
```

Podemos ver no código assim com a indentação correta como é visualmente possível identificar a hierarquia das instruções de comando, dessa forma o interpretador consegue entender o código perfeitamente.

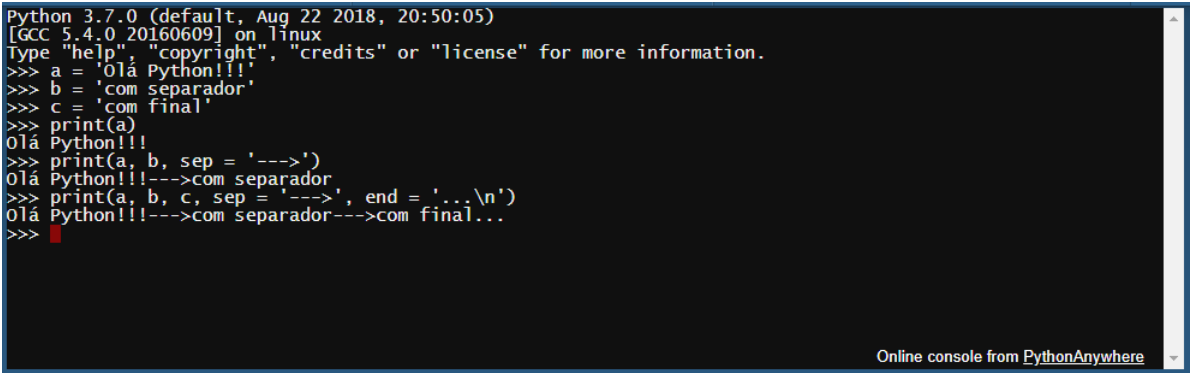
3.3 Imprimindo com Python

A função `print()` imprime os argumentos passados a ela, podendo receber um ou mais argumento, alguns parâmetros, separados por vírgulas dentro de parênteses, exemplo `print(argumento1, argumento1, sep='separador', end='final', file='fluxo_de_saida')`. Os argumentos passados ao final com as palavras chave `sep`, `end` e `file`, tem valores predefinidos, mas podem ser alterados.

- ◇ **sep** - O valor padrão é um espaço em branco, quando dois ou mais argumentos são passados para a função `print` `sep` coloca entre eles um espaço em branco ou um outro valor se passado no parâmetro.
- ◇ **end** - O valor padrão é uma nova linha '
'
n' dessa forma a função `print` adiciona uma nova linha depois de imprimir tudo que lhe foi passado ou podendo ser passado outro valor no parâmetro.
- ◇ **file** - O valor padrão é definido por `sys.stdout`, o fluxo de saída padrão que normalmente é o terminal. Podemos alterar esse valor para especificar um arquivo por exemplo.

Abaixo na Figura 8 vemos um exemplo da função `print`, executado diretamente no interpretador online do Python conforme mostrado anteriormente.

Figura 8 – Imprimindo com Python



```
Python 3.7.0 (default, Aug 22 2018, 20:50:05)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 'Olá Python!!!'
>>> b = 'com separador'
>>> c = 'com final'
>>> print(a)
Olá Python!!!
>>> print(a, b, sep = '--->')
Olá Python!!!--->com separador
>>> print(a, b, c, sep = '--->', end = '...\n')
Olá Python!!!--->com separador--->com final...
>>>
```

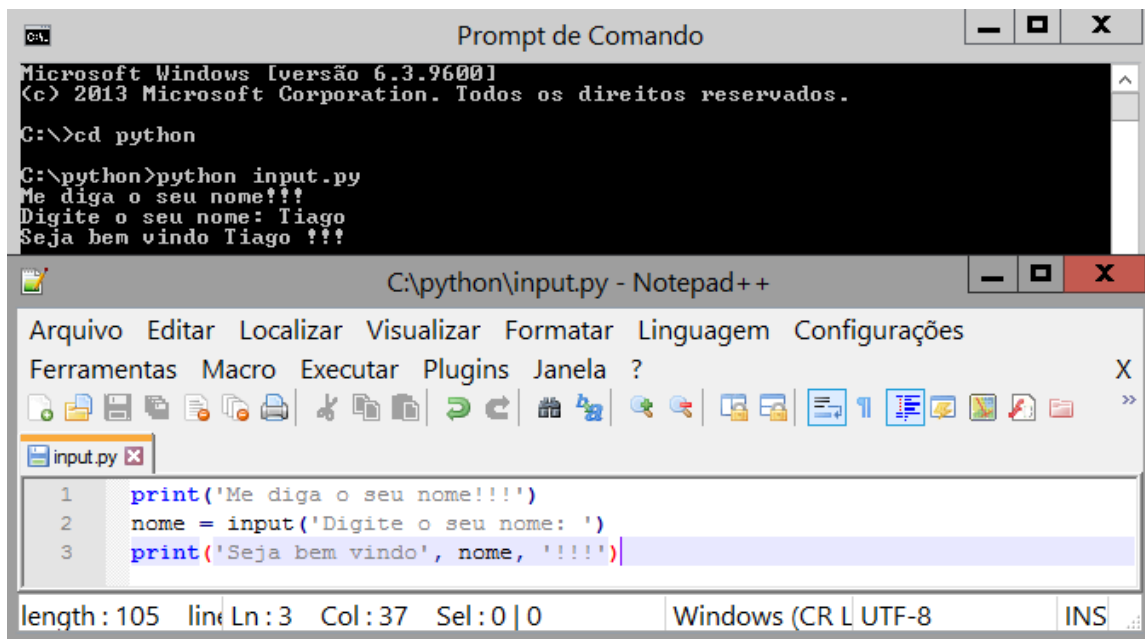
Online console from [PythonAnywhere](#)

3.4 Imputando dados com Python

A função `input()` faz uma interação do programa e espera uma entrada do usuário pelo terminal, basta digitar o valor e tecla enter.

Abaixo na Figura 9 vemos um exemplo da função `input`, executado a partir de arquivo. No arquivo `'input.py'` temos a aplicação, executada através do comando `python input.py`, o interpretador Python executa linha a linha da aplicação.

Figura 9 – Imputando dados com Python



Os dados inseridos pelo usuário na função após atribuído a variável 'nome' pode ser utilizado pelo programa de forma a deixar a aplicação dinâmica, na Figura 9 utilizamos a variável 'nome' na função print para imprimir o nome digitado pelo usuário.

3.5 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Execução em Python.
- ◇ Como indentar códigos no Python.
- ◇ Como imprimir em Python.
- ◇ Como imputar dados em Python.

Se não conseguiu entender todos esses conceitos, releia o capítulo, pois vamos precisar desses entendimentos no decorrer dos próximos capítulos. Vamos agora para o entendimento das variáveis e estruturas de dados.

4 CAPÍTULO - Variáveis e Estruturas de Dados

Vamos começar com conceitos de variáveis avançando o entendimento para chegar na estrutura de dados.

4.1 Variáveis

Assim como em outras linguagens, o Python pode manipular variáveis básicas como strings (palavras ou cadeias de caracteres), inteiros e reais (float) e booleanas (true ou false). Para criá-las, basta utilizar um comando de atribuição, que define seu tipo e seu valor.

4.1.1 Nomeando variáveis

As variáveis podem ser nomeadas de livre vontade do criador, com nomes longos, contendo letras e números, atentando apenas que elas devem necessariamente começar com letras minúsculas.

Também devemos lembrar das palavras reservadas da linguagem, que não podem ser utilizadas para nomear variáveis, as palavras reservadas podem variar, à medida em que a linguagem evolui, segue lista abaixo na Tabela 1.

Tabela 1 – Palavras Reservadas

| | | | | | |
|---------|-------|--------|--------|--------|----------|
| and | as | assert | break | class | continue |
| def | del | elif | else | except | exec |
| finally | for | from | global | if | import |
| in | is | lambda | non | local | not |
| pass | raise | return | try | while | with |
| yield | True | False | None | or | |

Vamos começar com as variáveis numéricas.

4.1.2 Números

Existem alguns tipos de variáveis numéricas e com elas podemos fazer algumas operações básicas. Vamos listar abaixo os tipos numéricos:

- ◇ Inteiro (int)
- ◇ Inteiro preciso (long)
- ◇ Ponto flutuante (float)
- ◇ Número complexo (complex)

Em Python, temos dois tipos de operações básicas com variáveis numéricas, aritmética e lógica, as operações aritméticas são as matemáticas básicas e as lógicas são as operações relacionais (ou comparativas).

4.1.2.1 Operações com Números

Vamos ver exemplos das operações com números:

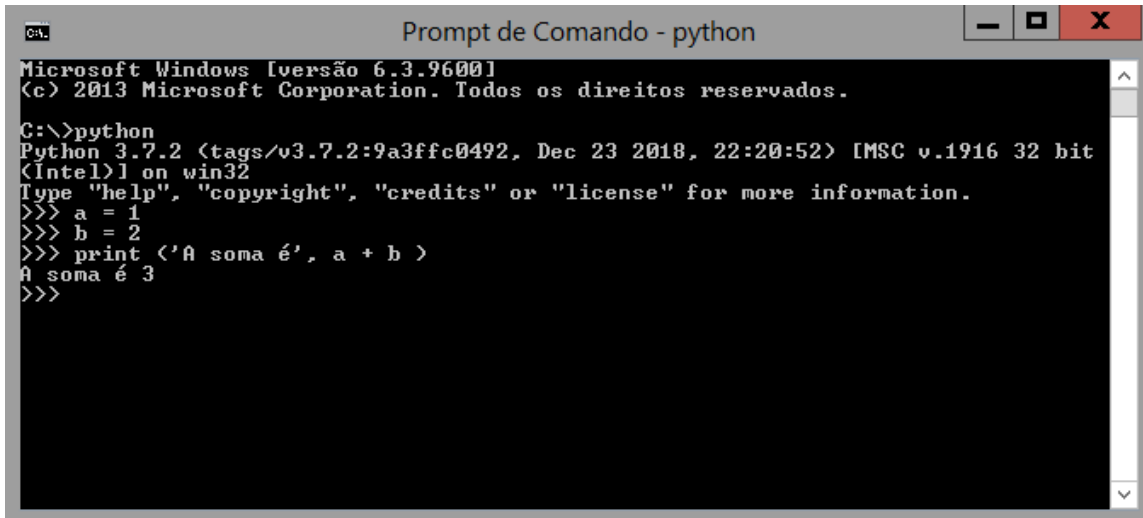
4.1.2.1.1 Operações Aritméticas

◇ Realizando uma Adição

```
a = 1 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'A soma é', a + b ) # Imprime o valor da soma a + b
```

Veja o resultado da soma no interpretador Python na Figura 10.

Figura 10 – Exemplo de Soma



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

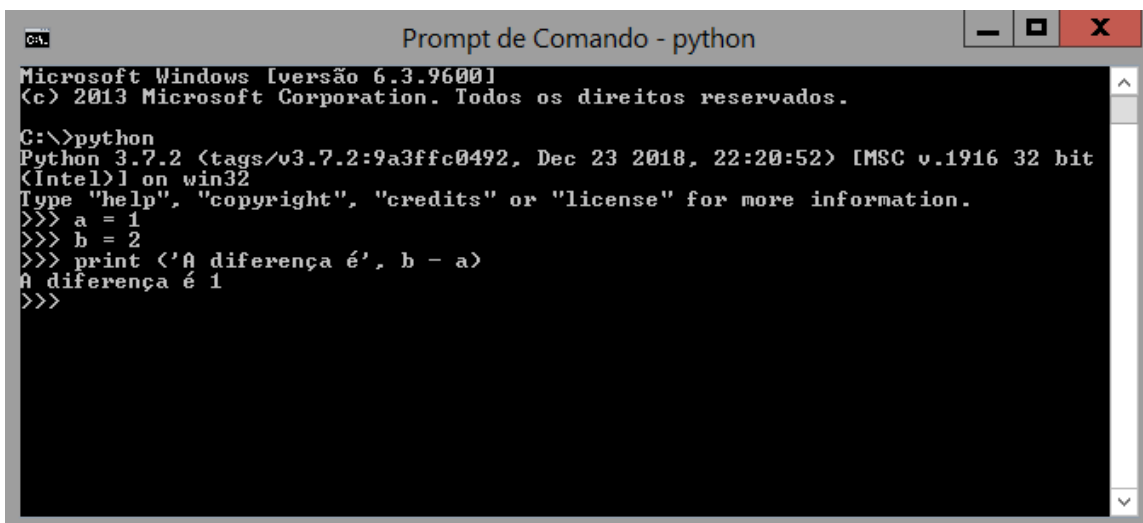
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 1
>>> b = 2
>>> print ('A soma é', a + b )
A soma é 3
>>>
```

◇ Realizando uma Subtração

```
a = 1 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'A diferença é', b - a ) # Imprime o valor da diferença
↪ b - a
```

Veja o resultado da subtração no interpretador Python na Figura 11.

Figura 11 – Exemplo de Diferença



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

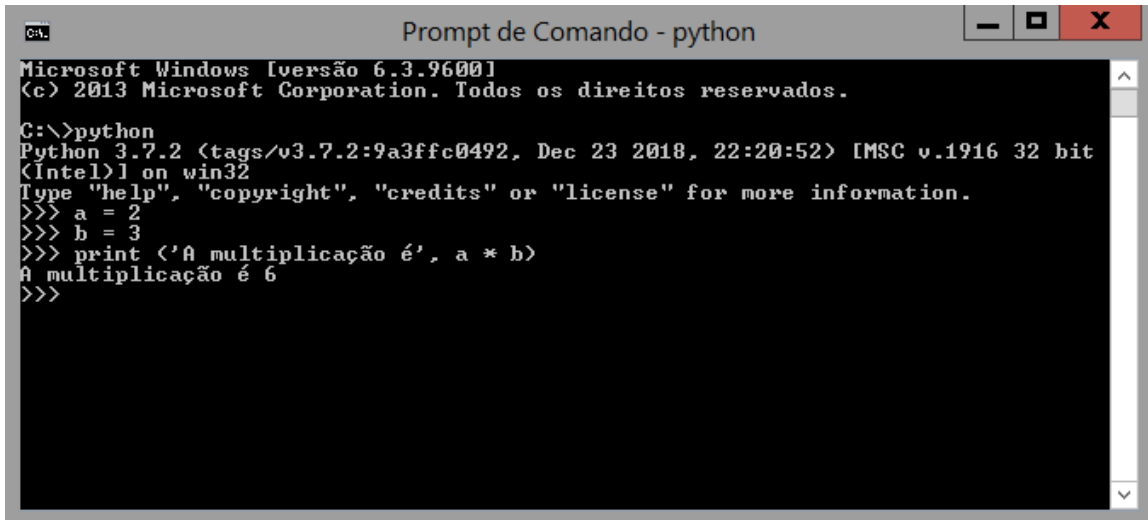
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 1
>>> b = 2
>>> print ('A diferença é', b - a)
A diferença é 1
>>>
```

◇ Realizando uma Multiplicação


```
a = 2 # Atribui um valor na variável 'a'
b = 3 # Atribui um valor na variável 'b'
print ( 'A multiplicação é', a * b ) # Imprime o valor da
↳ multiplicação a * b
```

Veja o resultado da multiplicação no interpretador Python na Figura 12.

Figura 12 – Exemplo de Multiplicação



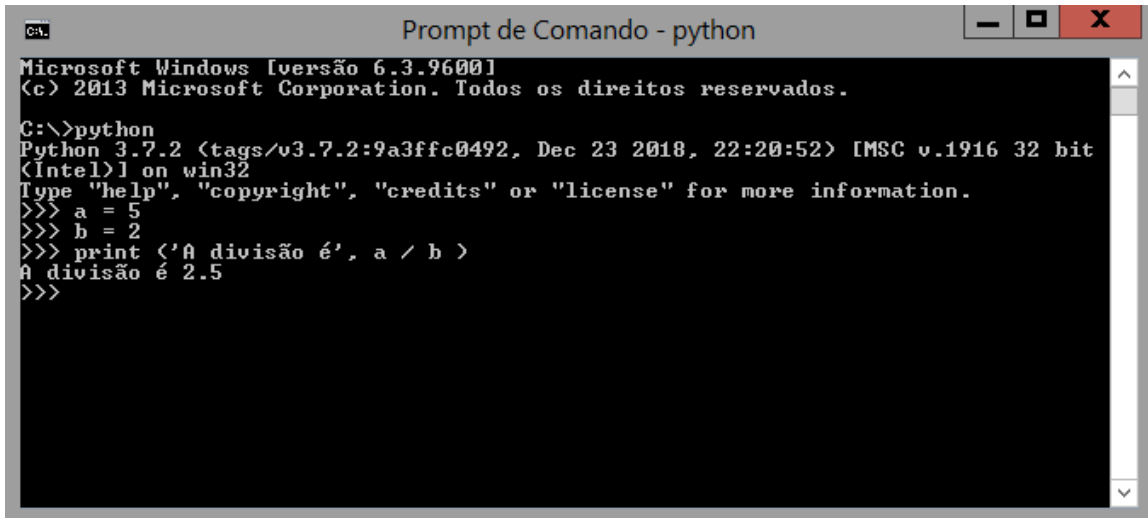
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> b = 3
>>> print (<'A multiplicação é', a * b)
A multiplicação é 6
>>>
```

◇ Realizando uma Divisão

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'A divisão é', a / b ) # Imprime o valor da divisão a /
↳ b
```

Veja o resultado da divisão no interpretador Python na Figura 13, observe que essa é uma divisão com decimal, podemos ter uma divisão apenas com inteiro.

Figura 13 – Exemplo de Divisão



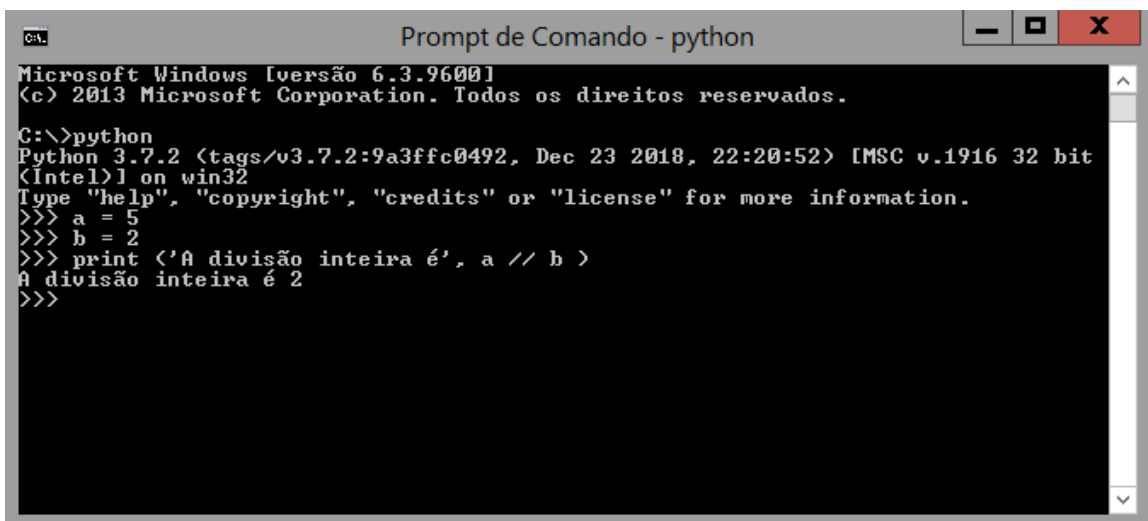
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print ('A divisão é', a / b )
A divisão é 2.5
>>>
```

◇ Realizando uma Divisão inteira

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'A divisão inteira é', a // b ) # Imprime o valor da
↪ divisão inteira a // b
```

Veja o resultado no interpretador Python na Figura 14, essa é uma divisão inteira.

Figura 14 – Exemplo de Divisão Inteira



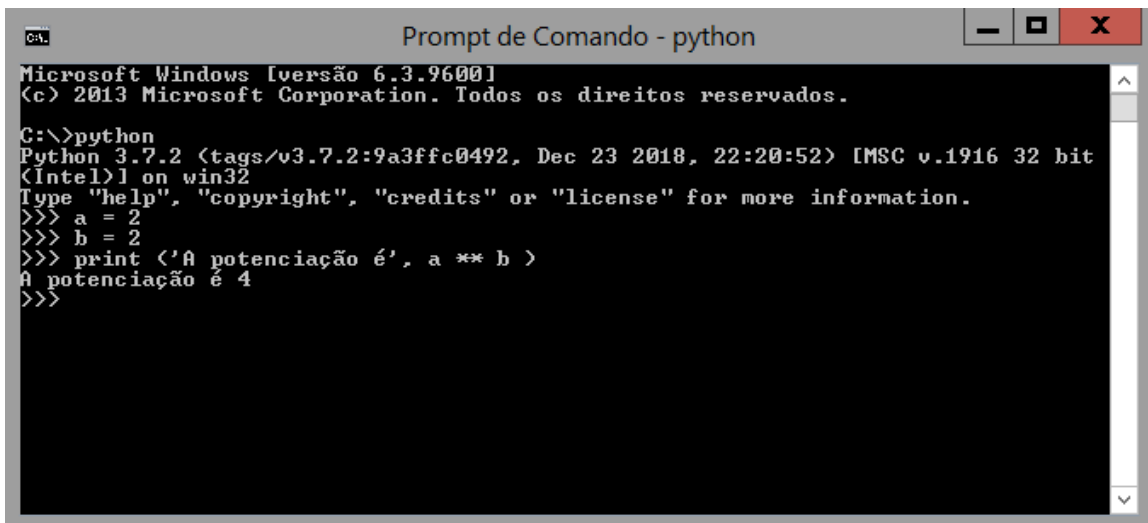
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print ('A divisão inteira é', a // b )
A divisão inteira é 2
>>>
```

◇ Realizando uma Potenciação

```
a = 2 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'A potenciação é', a ** b ) # Imprime o valor da
→ potenciação a ** b
```

Veja o resultado da potenciação no interpretador Python na Figura 15.

Figura 15 – Exemplo de Potenciação



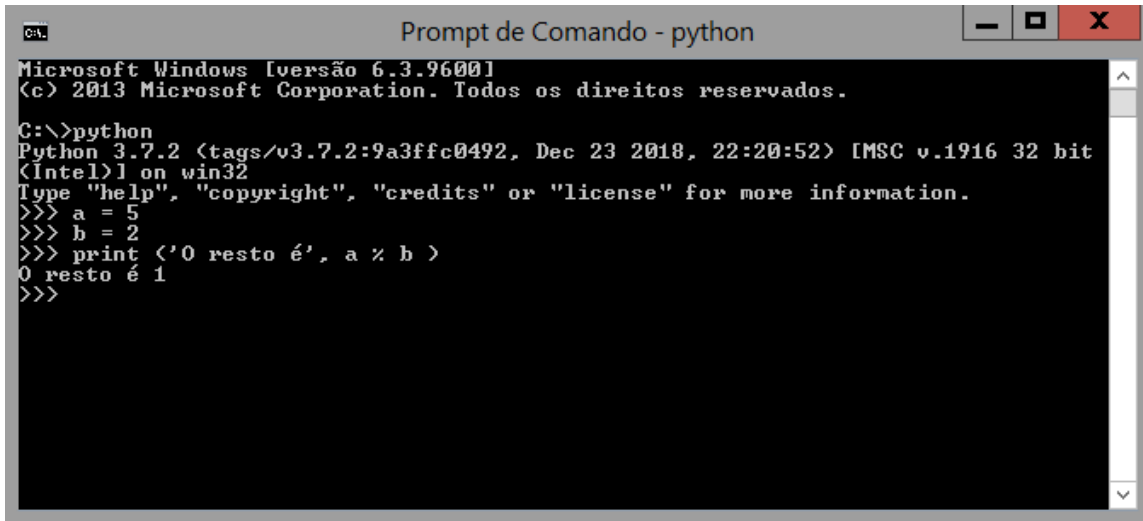
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> b = 2
>>> print ('A potenciação é', a ** b)
A potenciação é 4
>>>
```

◇ Realizando um Resto

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'O resto é', a \% b ) # Imprime o resto a % b
```

Veja o resultado da operação de resto do interpretador Python na Figura 16.

Figura 16 – Exemplo de Resto



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print (<'0 resto é', a % b >)
0 resto é 1
>>>
```

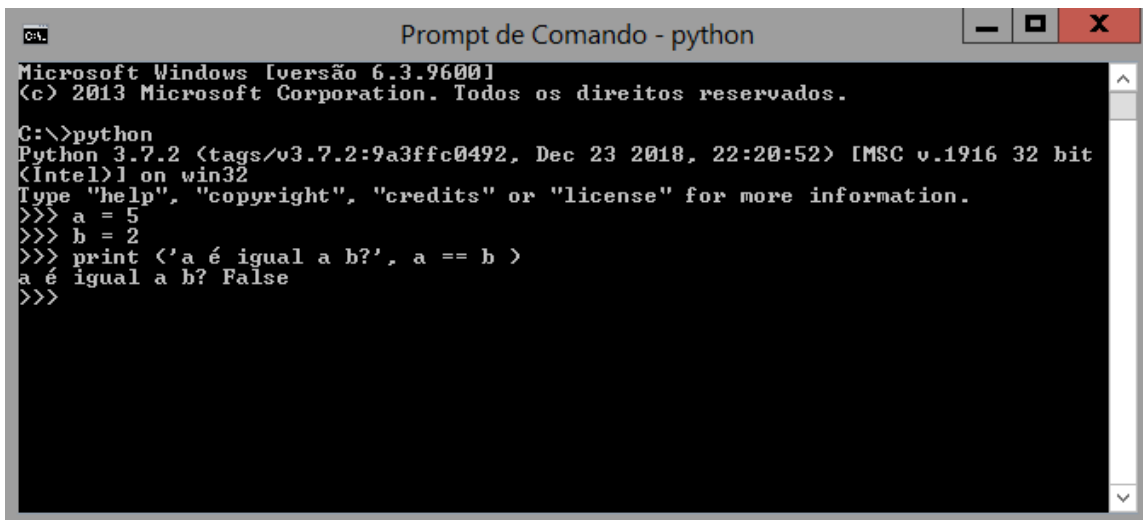
4.1.2.1.2 Operações Lógicas

◇ Relação Igual

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'a é igual a b?', a == b ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 17.

Figura 17 – Exemplo de Igualdade



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

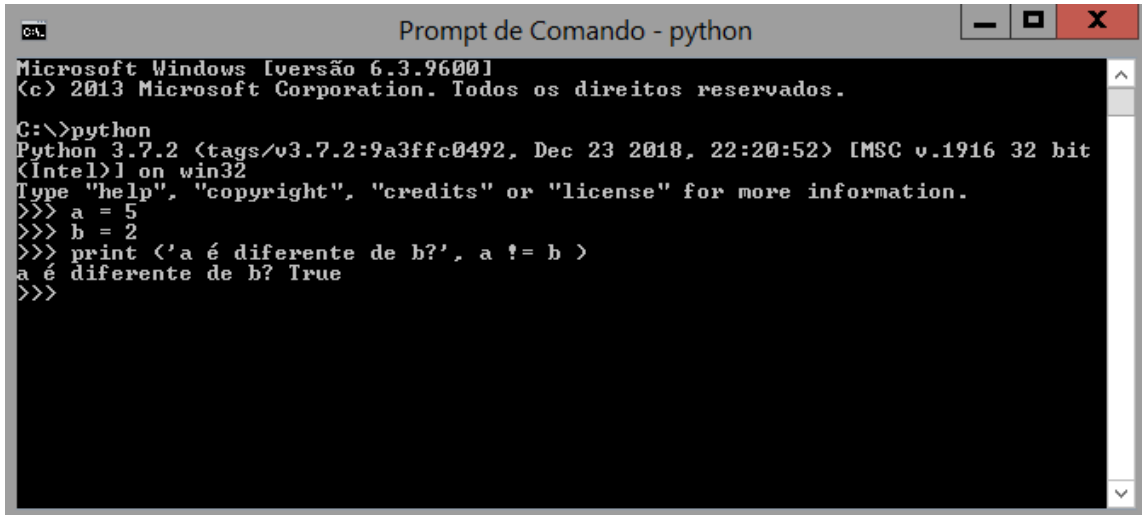
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print (<'a é igual a b?', a == b >)
a é igual a b? False
>>>
```

◇ Relação Diferente

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'a é diferente de b?', a != b ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 18.

Figura 18 – Exemplo de Diferença



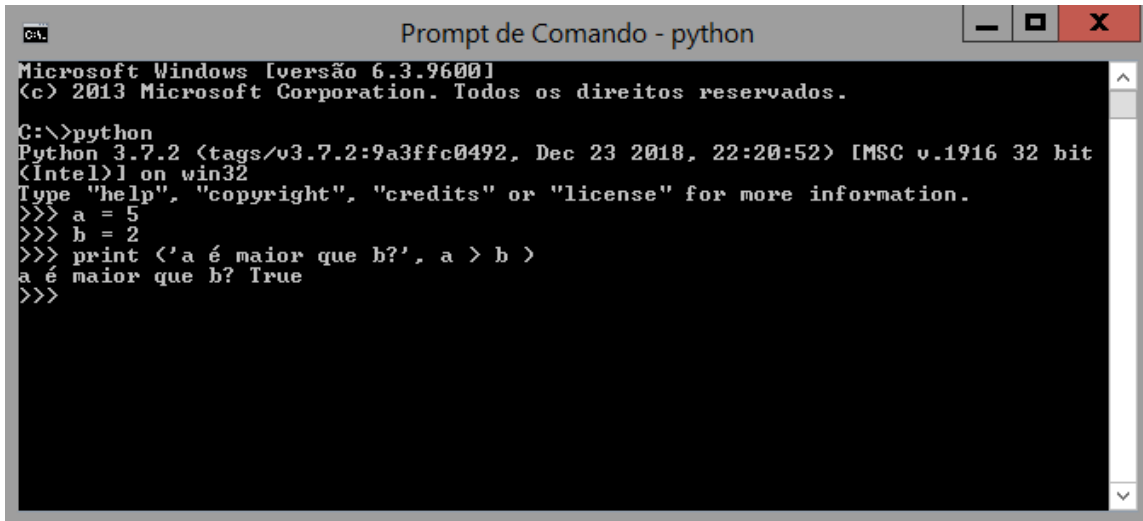
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel> on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print (<'a é diferente de b?>, a != b >)
a é diferente de b? True
>>>
```

◇ Relação Maior que

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'a é maior que b?', a > b ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 19.

Figura 19 – Exemplo de Maior que



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

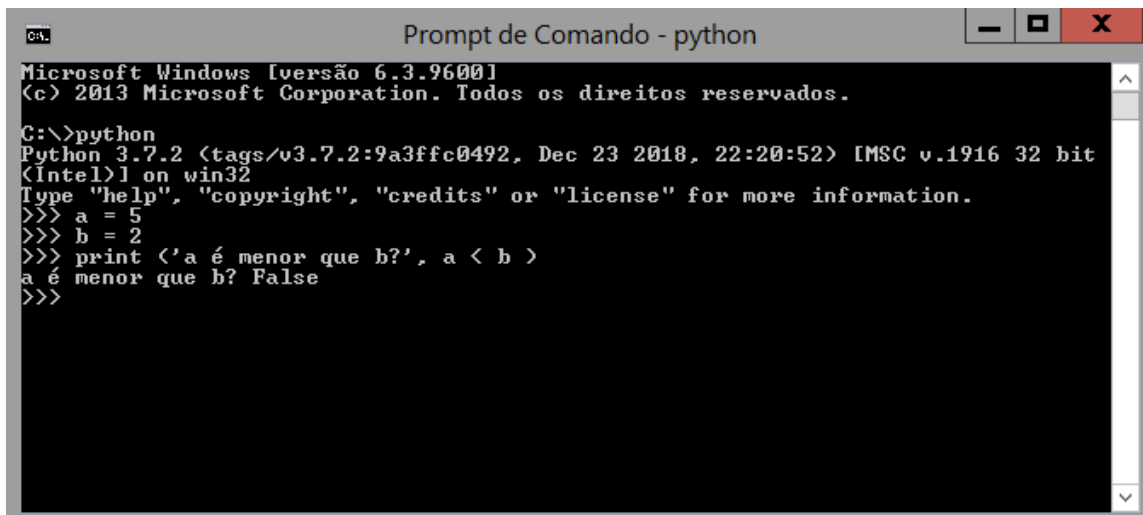
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print ('a é maior que b?', a > b )
a é maior que b? True
>>>
```

◇ Relação Menor que

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'a é menor que b?', a < b ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 20.

Figura 20 – Exemplo de Menor que



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

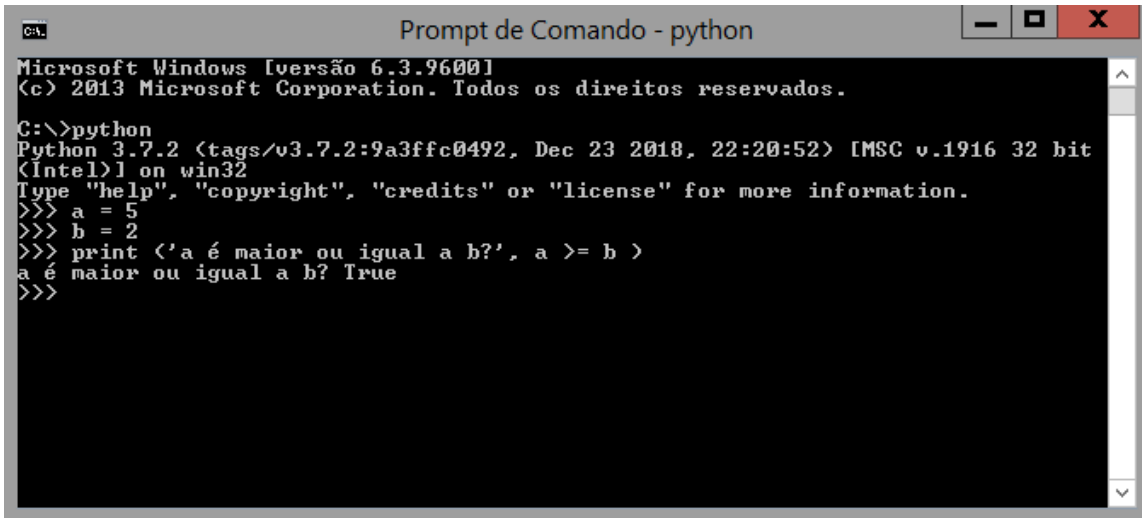
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print ('a é menor que b?', a < b )
a é menor que b? False
>>>
```

◇ Relação Maior ou igual

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'a é maior ou igual a b?', a >= b ) # Imprime o
↪ resultado
```

Veja o resultado no interpretador Python na Figura 21.

Figura 21 – Exemplo de Maior ou igual



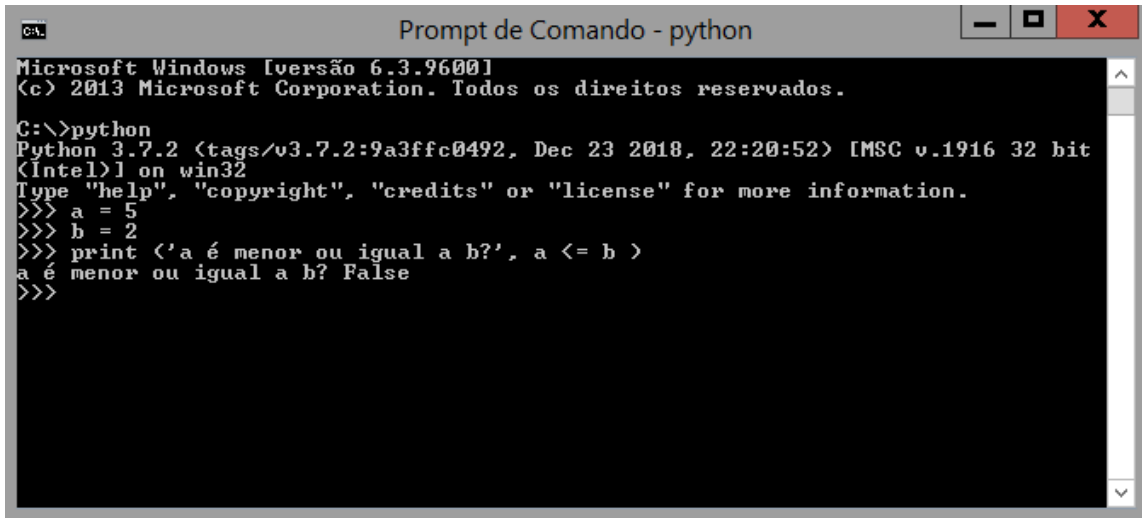
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print ('a é maior ou igual a b?', a >= b )
a é maior ou igual a b? True
>>>
```

◇ Relação Menor ou igual

```
a = 5 # Atribui um valor na variável 'a'
b = 2 # Atribui um valor na variável 'b'
print ( 'a é menor ou igual a b?', a <= b ) # Imprime o
↪ resultado
```

Veja o resultado no interpretador Python na Figura 22.

Figura 22 – Exemplo de Menor ou igual



```
C:\>python
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 2
>>> print ('a é menor ou igual a b?', a <= b )
a é menor ou igual a b? False
>>>
```

4.1.3 Boleano

Para completar as operações lógicas em Python, temos que lembrar da famosa ‘tabela verdade’, onde uma variável pode assumir valor booleano verdadeiro ou falso e permitindo realizar operação lógicas com essas variáveis, sendo úteis para representar o resultado de uma comparação.

4.1.3.1 Operações com Boleano

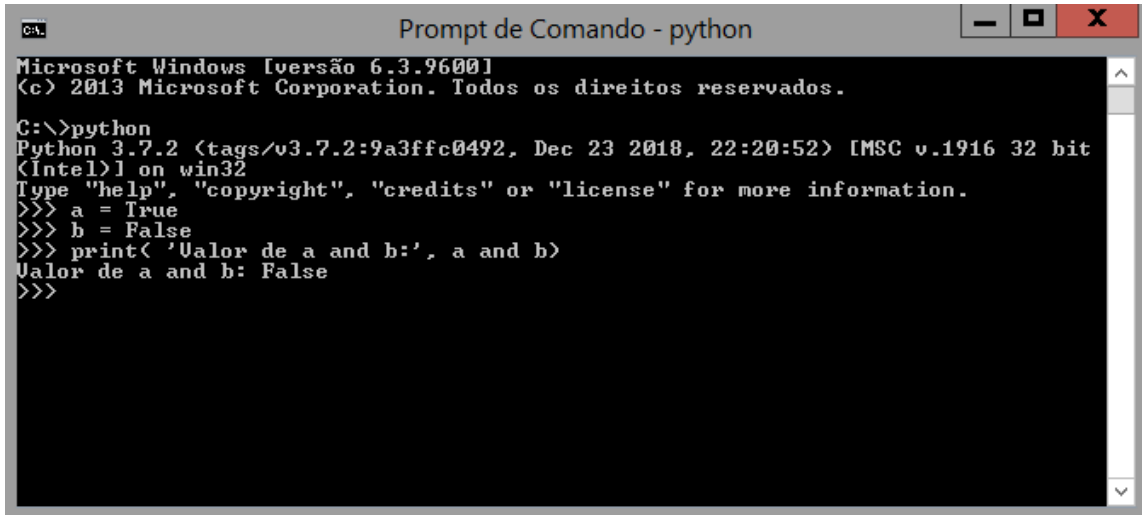
As operações booleanas possíveis são:

◇ Boleano E

```
a = True # Atribui um valor booleano na variável 'a'
b = False # Atribui um valor booleano na variável 'b'
print( 'Valor de a and b:', a and b ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 23.

Figura 23 – Exemplo de Boleano E



```
C:\>python
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

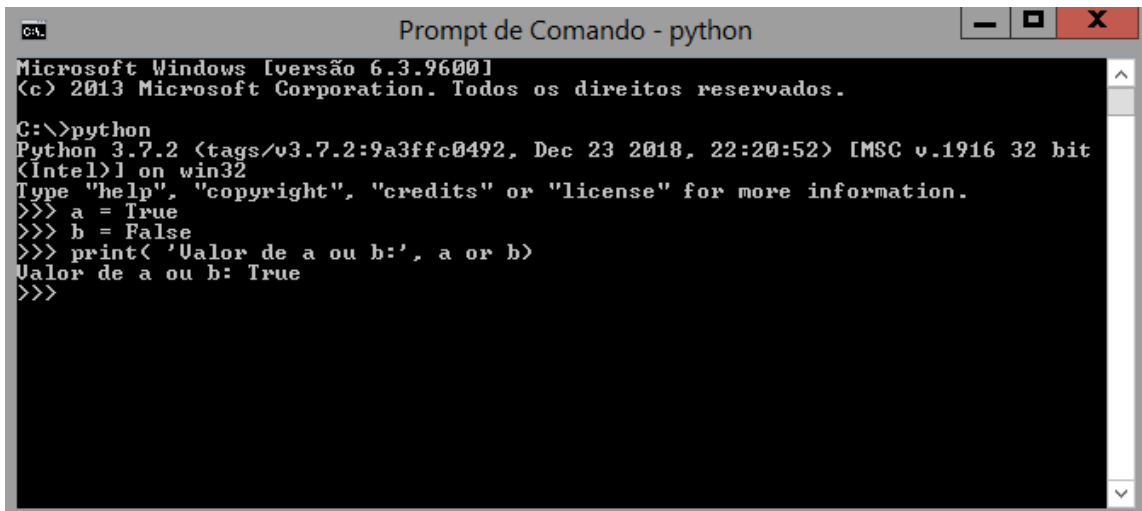
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = True
>>> b = False
>>> print( 'Valor de a and b:', a and b)
Valor de a and b: False
>>>
```

◇ Boleano Ou

```
a = True # Atribui um valor booleano na variável 'a'
b = False # Atribui um valor booleano na variável 'b'
print( 'Valor de a ou b:', a or b ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 24.

Figura 24 – Exemplo de Boleano OR



```
C:\>python
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

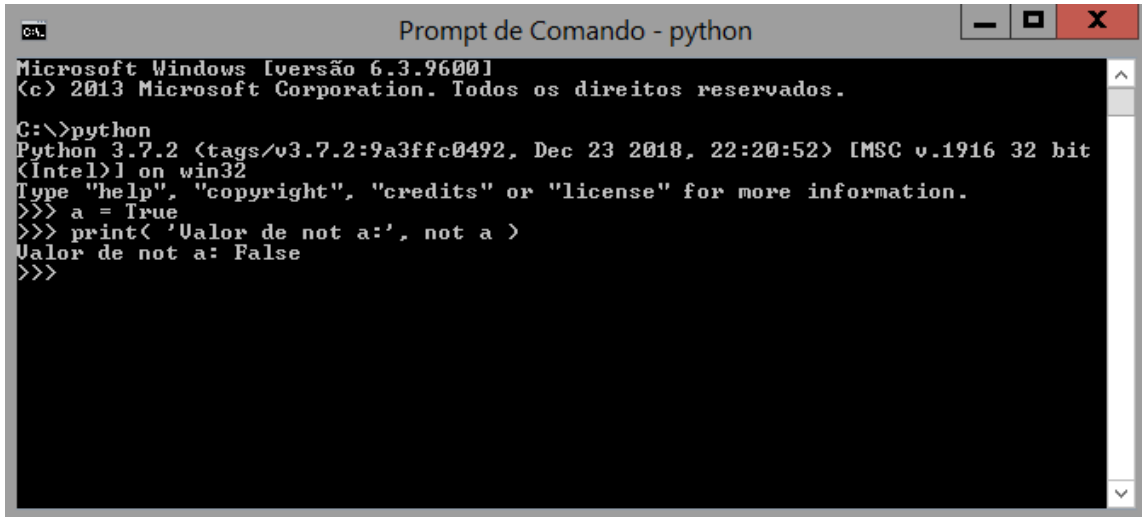
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = True
>>> b = False
>>> print( 'Valor de a ou b:', a or b)
Valor de a ou b: True
>>>
```

◇ Boleano Não

```
a = True # Atribui um valor booleano na variável 'a'
print( 'Valor de not a:', not a ) # Imprime o resultado
```

Veja o resultado no interpretador Python na Figura 25.

Figura 25 – Exemplo de Booleano NOT



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = True
>>> print( 'Valor de not a:', not a )
Valor de not a: False
>>>
```

4.1.4 Strings

Uma string é qualquer sequência de caracteres, que é a menor unidade de todo o texto. Os caracteres são classificados em 3 grupos: numérico, letras do alfabeto e caracteres especiais.

Para o Python, toda string é uma lista imutável, após definida NÃO pode ser alterada, ou seja, você não pode alterar um caractere dentro de uma string. Então, ao fazermos alguma alteração numa determinada parte de uma string, o que de fato ocorrerá é a definição de uma nova string.

4.1.4.1 Operações com Strings

Vamos ver agora as formas de manipulação com strings, utilizando métodos e expressões regulares.

4.1.4.1.1 Métodos de Strings

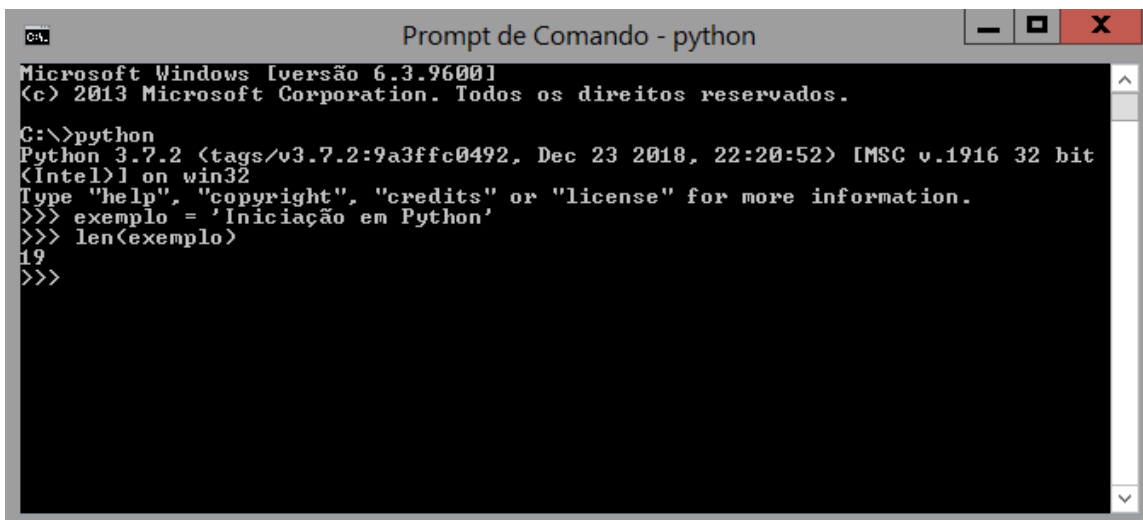
A forma mais simples de manipular strings é através dos métodos que estão dentro das strings. Vamos ver alguns desses métodos a seguir:

- ◇ **Função len** - Usada para encontrar o tamanho de uma string contando espaços e caracteres especiais.

```
exemplo = 'Iniciação em Python' # Criando uma string
len(exemplo) # Contagem do tamanho da string
```

Veja o resultado no interpretador Python na Figura 26, retornando o tamanho da string.

Figura 26 – Exemplo Função len



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

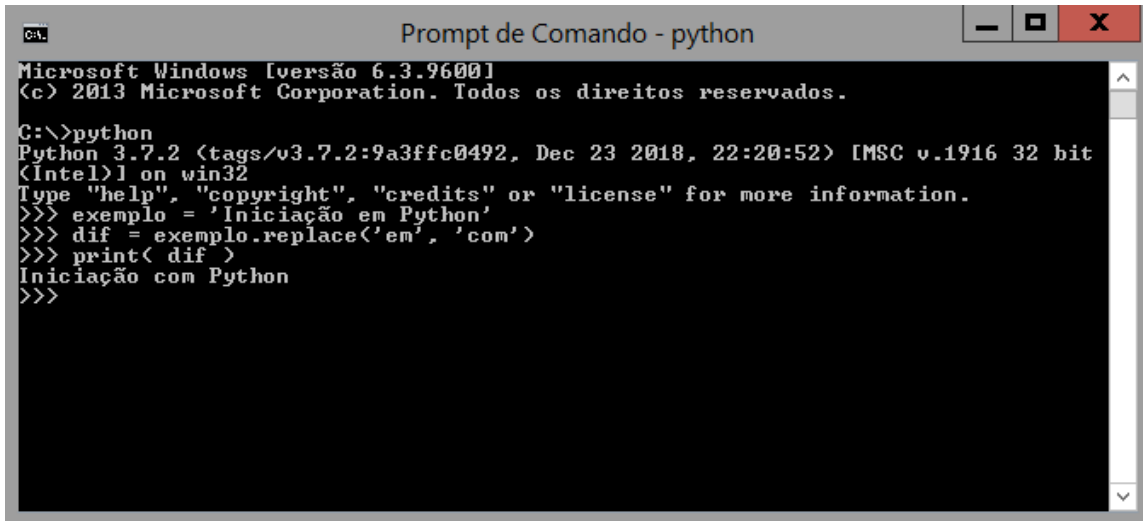
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = 'Iniciação em Python'
>>> len(exemplo)
19
>>>
```

- ◇ **Função replace** - Usada para substituir uma ou mais palavras de uma string.

```
exemplo = 'Iniciação em Python' # Criando uma string
dif = exemplo.replace('em', 'com') # Substituindo palavras
print( dif ) # Imprime a nova string
```

Veja o resultado no interpretador Python na Figura 27 com a impressão da nova string.

Figura 27 – Exemplo Função replace



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

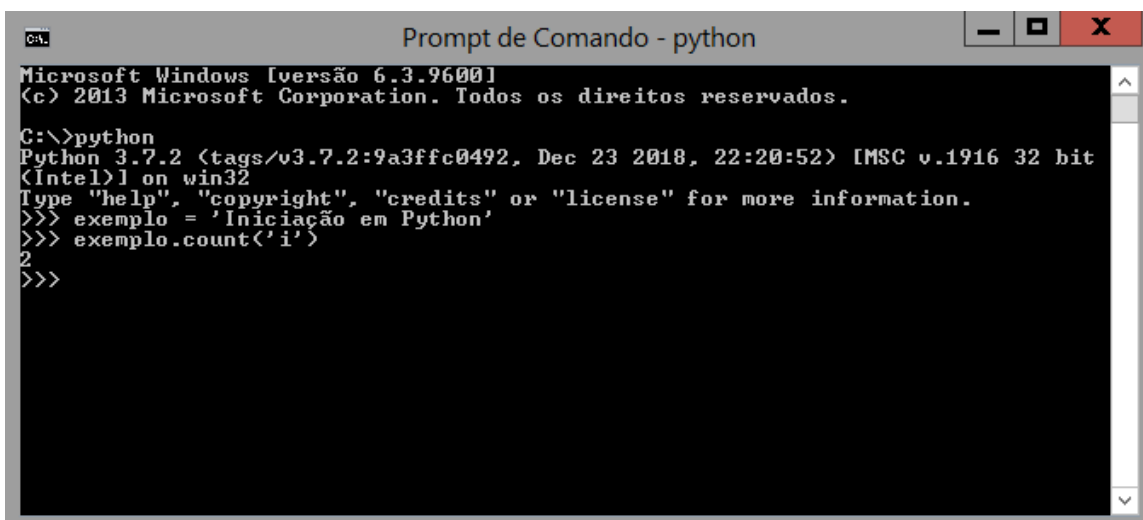
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = 'Iniciacão em Python'
>>> dif = exemplo.replace('em', 'com')
>>> print(dif)
Iniciacão com Python
>>>
```

- ◊ **Função count** - Usada para contar o numero de vezes que a palavra ou carácter aparece na string.

```
exemplo = 'Iniciacão em Python' # Criando uma string
exemplo.count('i') # Contagem do carácter 'i'
```

Veja o resultado no interpretador Python na Figura 28, observe que o caracter 'i' aparece 3 vezes mas a linguagem é case-sensitive (sensível a maiúsculas e minúsculas), por isso apenas 2 na contagem.

Figura 28 – Exemplo Função count



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

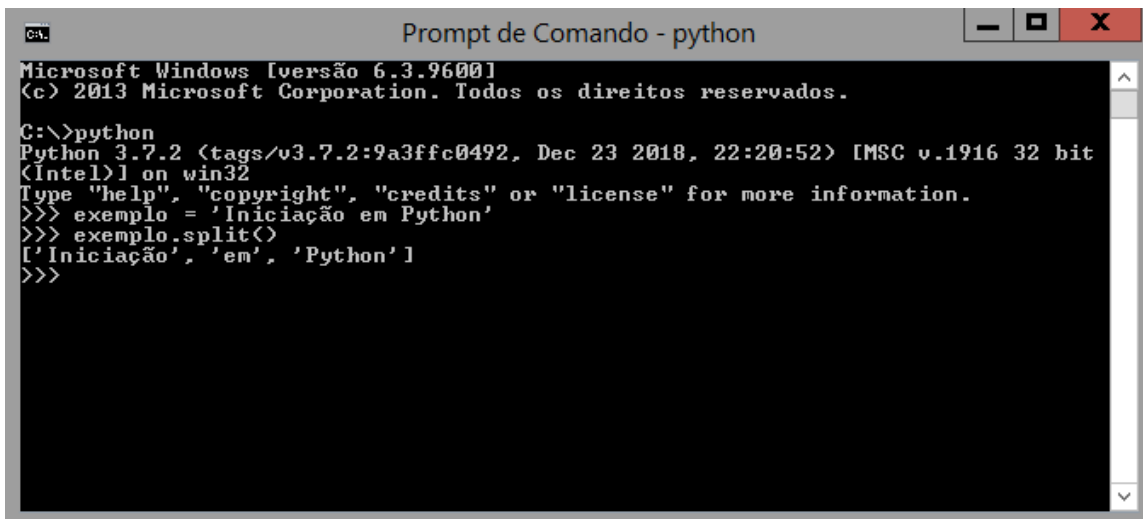
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = 'Iniciacão em Python'
>>> exemplo.count('i')
2
>>>
```

- ◊ **Função split** - Usada para separar uma string.

```
exemplo = 'Iniciação em Python' # Criando uma string
exemplo.split() # Separando um string
```

O resultado será uma lista (veremos o que é uma lista um pouco mais a frente neste capítulo) com cada palavras como itens dessa lista conforme o resultado no interpretador Python na Figura 29.

Figura 29 – Exemplo Função split



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

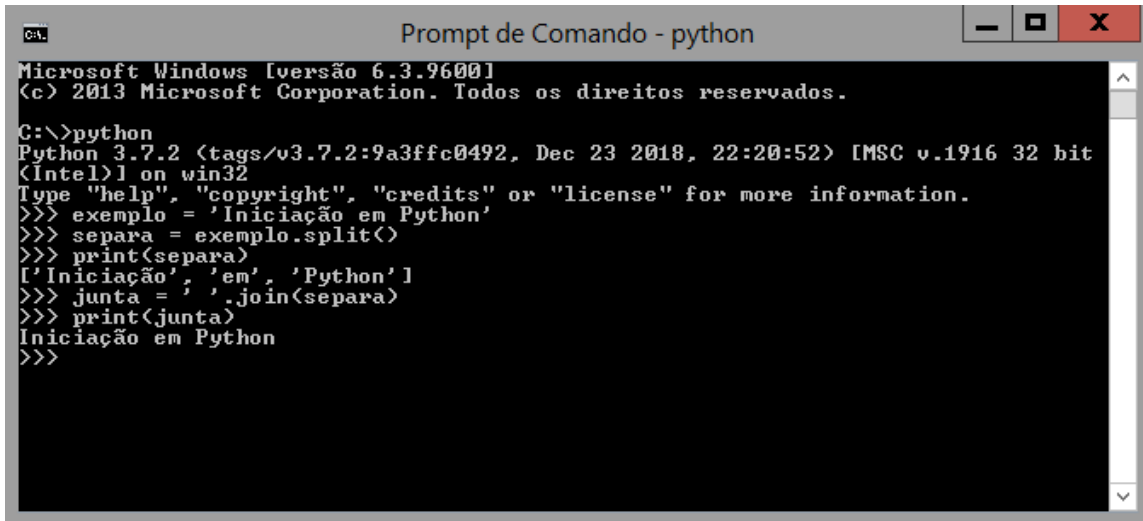
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = 'Iniciação em Python'
>>> exemplo.split()
['Iniciação', 'em', 'Python']
>>>
```

◇ **Função join** - Usada para juntar uma string.

```
exemplo = 'Iniciação em Python' # Criando uma string
separa = exemplo.split() # Separando uma string
print(separa) # Imprimindo a string separa
junta = ' '.join(separa) # Juntando as strings mais o ' '
print(junta) # Imprime a string junta
```

Veja o resultado no interpretador Python na Figura 30, observe que precisamos utilizar uma string para fazer o join, neste caso usei ' '.

Figura 30 – Exemplo Função join



- ◇ **Funções MAIÚSCULAS e minúsculas** - Funções para alterar a caixa de texto das palavras.

upper() - Deixa tudo maiúsculo.

lower() - Deixa tudo minúsculo.

capitalize() - Deixa apenas a primeira letra maiúscula de uma string minúscula.

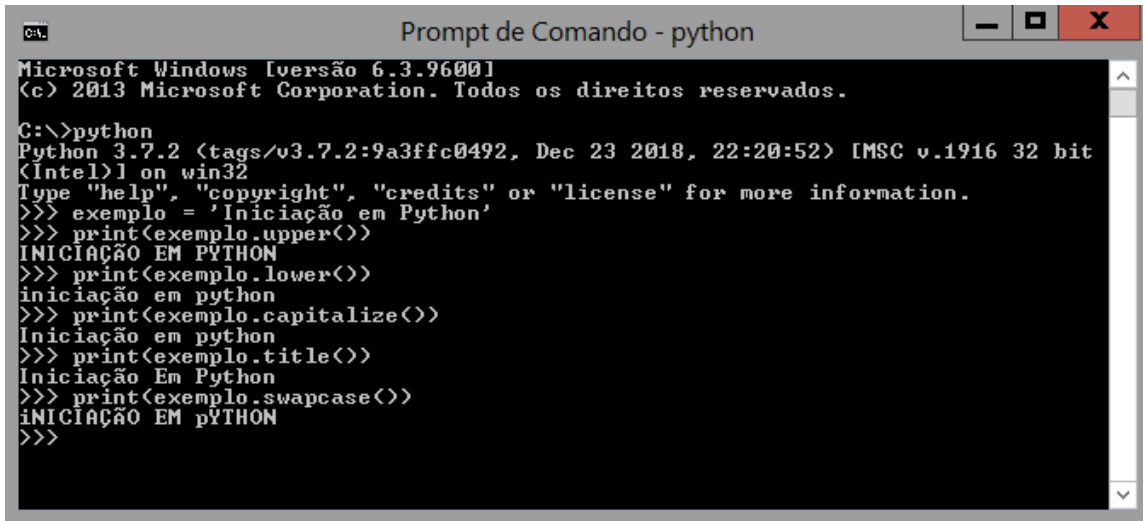
title() - Deixa a primeira letra de cada palavra da string maiúscula.

swapcase() - O que for maiúsculo vira minúsculo e vice-versa

```
exemplo = 'Iniciação em Python' # Criando uma string
print(exemplo.upper()) # Imprime tudo maiúsculo
print(exemplo.lower()) # Imprime tudo minúsculo
print(exemplo.capitalize()) # Imprime apenas a primeira letra
↪ maiúsculo
print(exemplo.title()) # Imprime a primeira letra de cada
↪ palavra maiúsculo
print(exemplo.swapcase()) # Imprime o que for maiúsculo vira
↪ minúsculo e vice-versa
```

Veja o resultado no interpretador Python na Figura 31, com uma string apenas podemos ver todas as alterações da caixa de texto das palavras.

Figura 31 – Exemplo Funções MAIÚSCULAS e minúsculas



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel> on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = 'Iniciação em Python'
>>> print(exemplo.upper())
INICIAÇÃO EM PYTHON
>>> print(exemplo.lower())
iniciação em python
>>> print(exemplo.capitalize())
Iniciação em python
>>> print(exemplo.title())
Iniciação Em Python
>>> print(exemplo.swapcase())
iNiciAção eM pYTHON
>>>
```

4.1.4.1.2 Expressões regulares

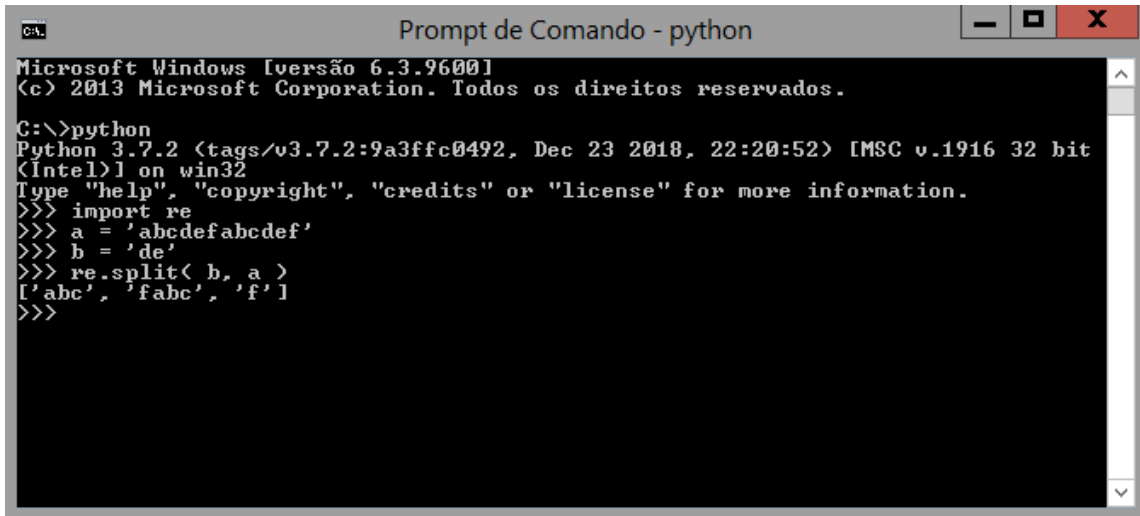
Expressões regulares são uma ferramenta que permitem que padrões sejam "achados" ou "casados" dentro de strings. A biblioteca que cuida de expressões regulares no Python é o `re`, precisamos importar essa biblioteca (como já vimos no CAPÍTULO - Preparação do Ambiente de Desenvolvimento) para realizar as expressões regulares, vamos lá:

- ◇ `re.split` - Separa a string no(s) ponto(s) onde o valor desejado é encontrado.

```
import re # Importa biblioteca de expressão regulares
a = 'abcdefabcdef' # String a ser verificada
b = 'de' # String que desejo encontrar
re.split( b, a ) # Separa a string a quando encontrar a string
↪ b
```

Veja o resultado no interpretador Python na Figura 32:

Figura 32 – Exemplo ER re.split



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

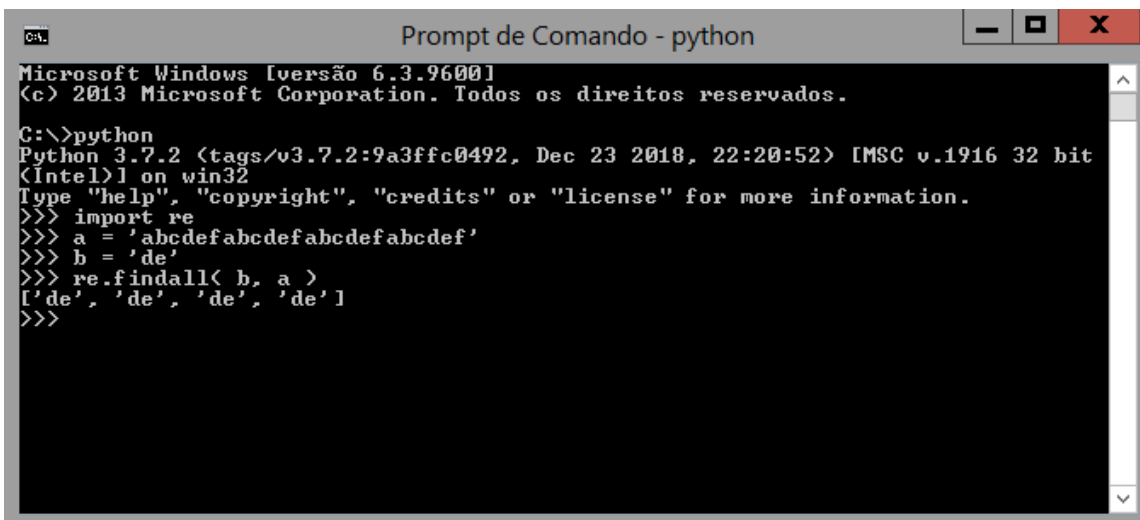
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import re
>>> a = 'abcdefabcdef'
>>> b = 'de'
>>> re.split(b, a)
['abc', 'fab', 'c', 'f']
>>>
```

- ◇ **re.findall** - Retorna uma lista do valor desejado é encontrado a quantidade de vezes que encontrar.

```
import re # Importa biblioteca de expressão regulares
a = 'abcdefabcdefabcdef' # String a ser verificada
b = 'de' # String que desejo encontrar
re.findall(b, a) # Cria uma lista da string b a quantidade de
↳ vezes que encontrar na string a
```

Veja o resultado no interpretador Python na Figura 33:

Figura 33 – Exemplo ER re.findall



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import re
>>> a = 'abcdefabcdefabcdef'
>>> b = 'de'
>>> re.findall(b, a)
['de', 'de', 'de', 'de']
>>>
```


4.2 Estruturas de Dados

As estruturas de dados são os meios utilizados para armazenar e recuperar informações através das variáveis. Em Python existem dois tipos de estruturas de dados, as sequências que são objetos ordenados e finitos (strings, listas e tuplas) e dicionários que é um conjunto de elementos de mapeamentos indexados por chaves. A seguir veremos essas estruturas de dados detalhadamente.

4.2.1 Listas

Uma lista (list) em Python é uma sequência ou coleção ordenada de valores. Cada valor na lista é identificado por um índice. Os valores que formam uma lista são chamados elementos ou itens. Listas são similares a strings, que são uma sequência de caracteres, no entanto, diferentemente de strings, os itens de uma lista podem ser de tipos diferentes.

4.2.1.1 Exemplos de Listas:

- ◇ `numeros = [1, 2, 3]`
- ◇ `letras = ['a', 'b', 'c']`
- ◇ `diversos = ['z', 'a1', 2]`

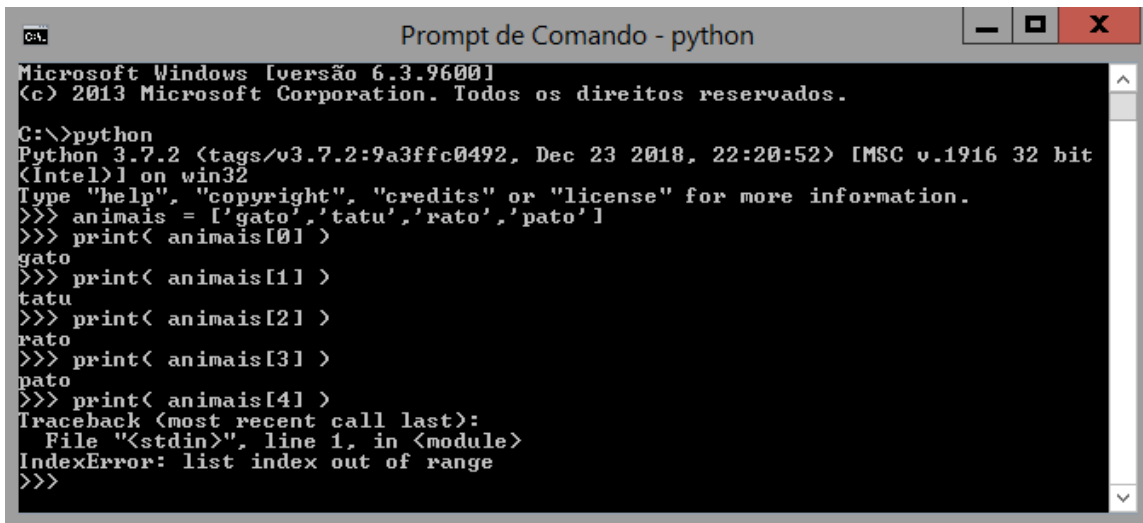
4.2.1.2 Operações com Listas

- ◇ **Índices** - O índice representa a posição do item na lista, contamos o índice da lista a partir do zero, ao acessar um índice inválido é apresentado um erro.

```
animais = ['gato', 'tatu', 'rato', 'pato'] # Lista de strings
print( animais[0] ) # Imprime o primeiro elemento da lista
print( animais[1] ) # Imprime o segundo elemento da lista
print( animais[2] ) # Imprime o terceiro elemento da lista
print( animais[3] ) # Imprime o quarto elemento da lista
print( animais[4] ) # Erro elemento inexistente
```

Veja o resultado no interpretador Python na Figura 34:

Figura 34 – Exemplo com Índices



```

Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> animais = ['gato', 'tatu', 'rato', 'pato']
>>> print( animais[0] )
gato
>>> print( animais[1] )
tatu
>>> print( animais[2] )
rato
>>> print( animais[3] )
pato
>>> print( animais[4] )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>

```

◊ **Função sum** - Soma os elementos de uma lista.

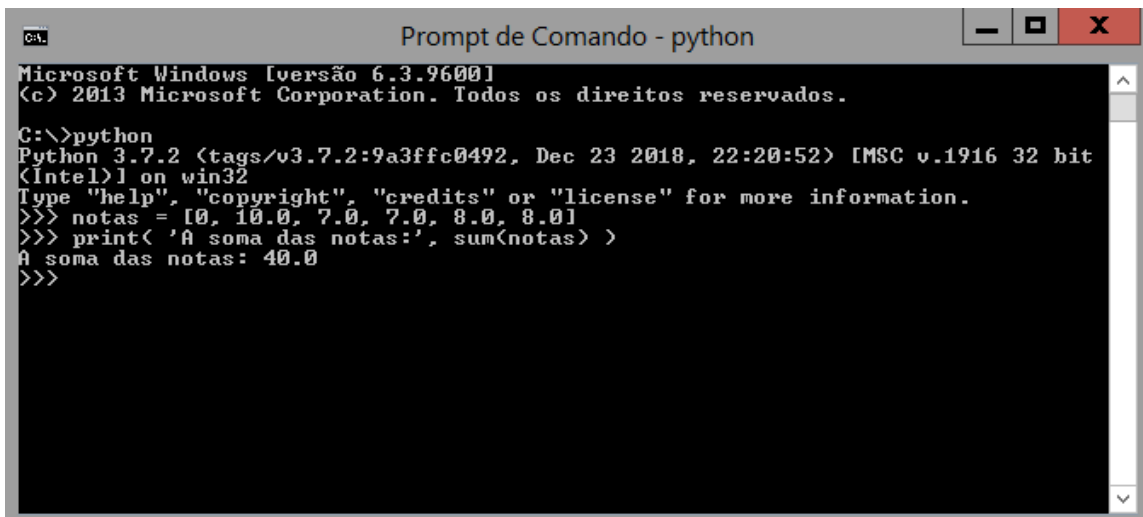
```

notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
print( 'A soma das notas:', sum(notas) ) # Imprime a soma das
↪ notas

```

Veja o resultado no interpretador Python na Figura 35:

Figura 35 – Exemplo Soma da lista



```

Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>> print( 'A soma das notas:', sum(notas) )
A soma das notas: 40.0
>>>

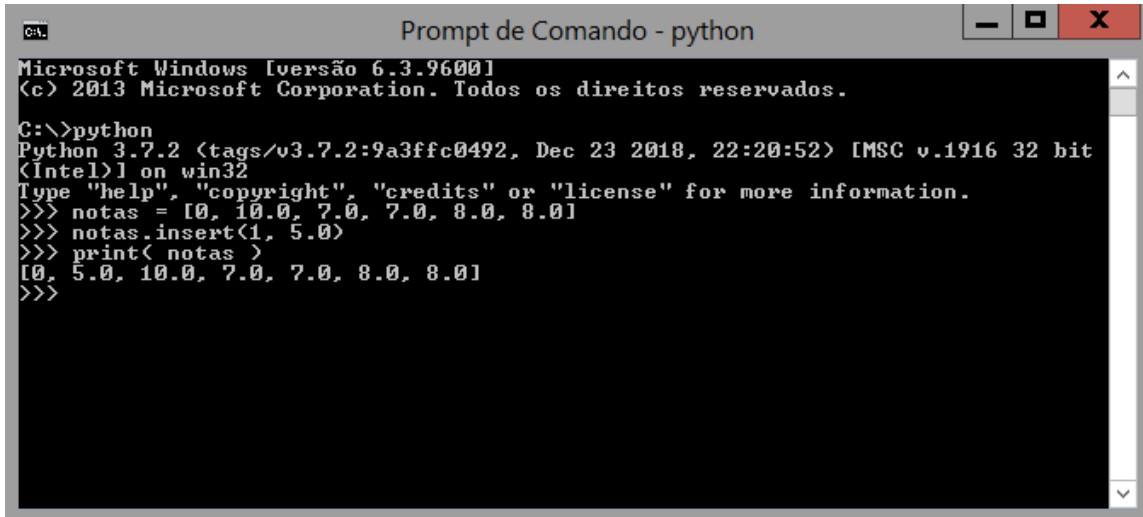
```

◊ **Função insert** - Insere um elemento em uma posição X de uma lista.

```
notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
notas.insert(1, 5.0) # Insere o elemento da lista na posição 1
print( notas ) # Imprime a nova lista de notas
```

Veja o resultado no interpretador Python na Figura 36:

Figura 36 – Exemplo Insere elementos da lista



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

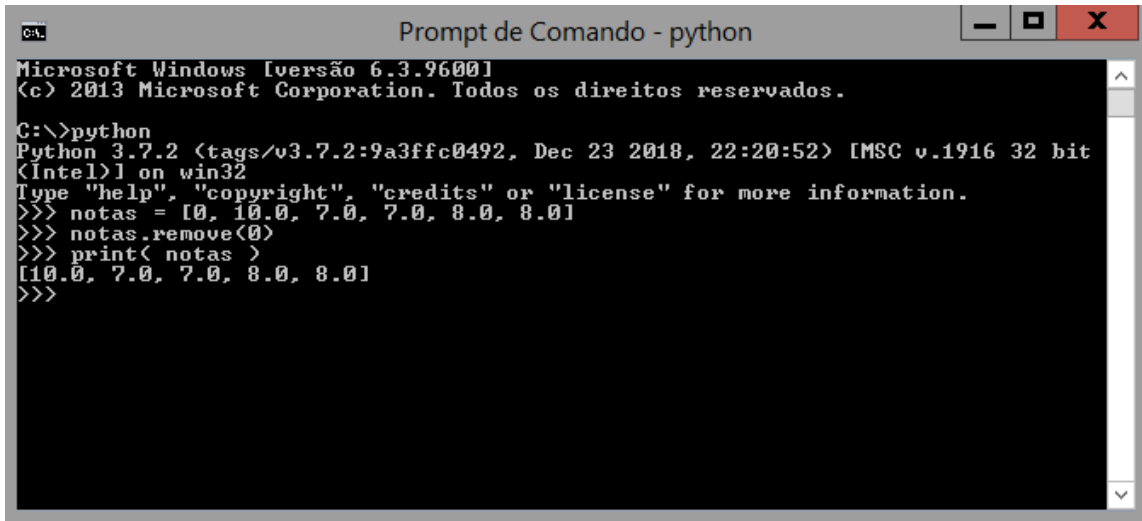
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>> notas.insert(1, 5.0)
>>> print( notas )
[0, 5.0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>>
```

◇ **Função remove** - Remove um elemento de uma lista.

```
notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
notas.remove(0) # Remove o primeiro elemento da lista
print( notas ) # Imprime a nova lista de notas
```

Veja o resultado no interpretador Python na Figura 37:

Figura 37 – Exemplo Remove elementos da lista



◇ Funções Máximo e Mínimo

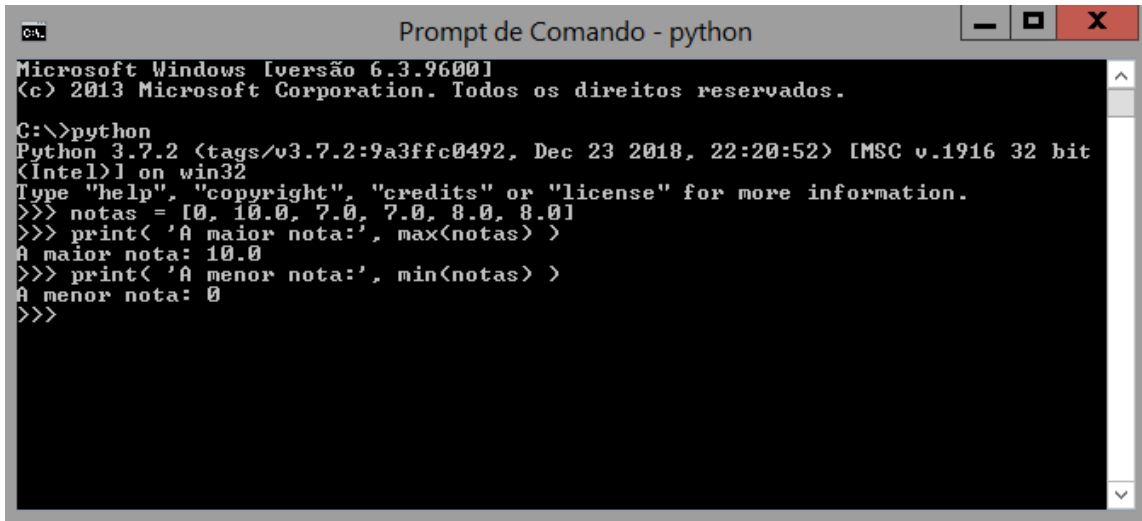
max() - Retorna o maior valor da lista.

min() - Retorna o menor valor da lista.

```
notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
print( 'A maior nota:', max(notas) ) # Imprime a maior nota da
↪ lista
print( 'A menor nota:', min(notas) ) # Imprime a menor nota da
↪ lista
```

Veja o resultado no interpretador Python na Figura 38:

Figura 38 – Exemplo Máximo e Mínimo



```
C:\>python
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>> print( 'A maior nota:', max(notas) )
A maior nota: 10.0
>>> print( 'A menor nota:', min(notas) )
A menor nota: 0
>>>
```

- ◇ Função `sort` e `sorted` - ordena uma lista.

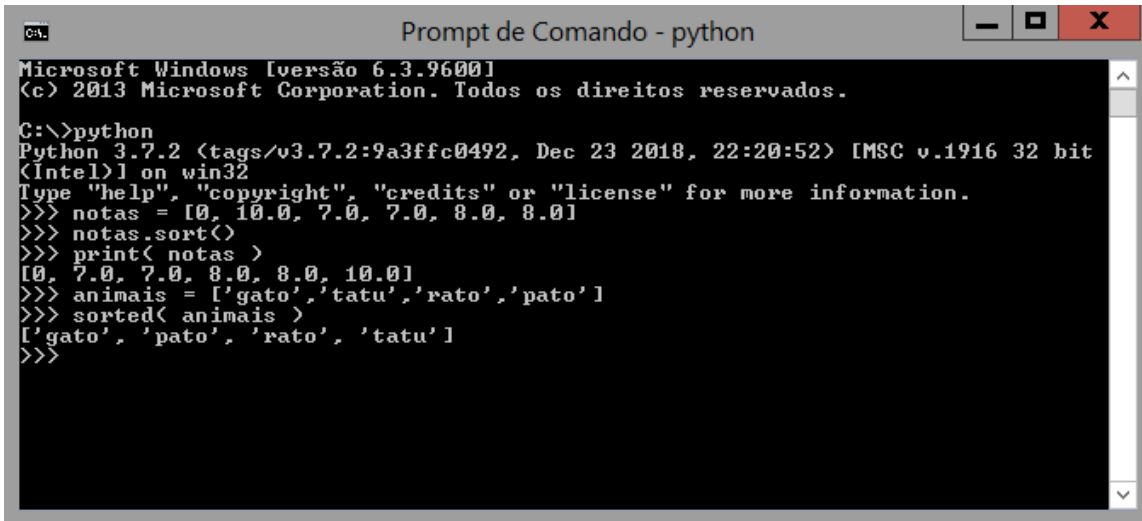
`sort()` - Ordena a lista lista.

`soted()` - Retorna a lista ordenada.

```
notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
notas.sort() # Ordena a lista
print( notas ) # Imprime a nova lista de notas
animais = ['gato', 'tatu', 'rato', 'pato'] # Lista de strings
sorted( animais )
```

Veja o resultado no interpretador Python na Figura 39:

Figura 39 – Exemplo Ordenação



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

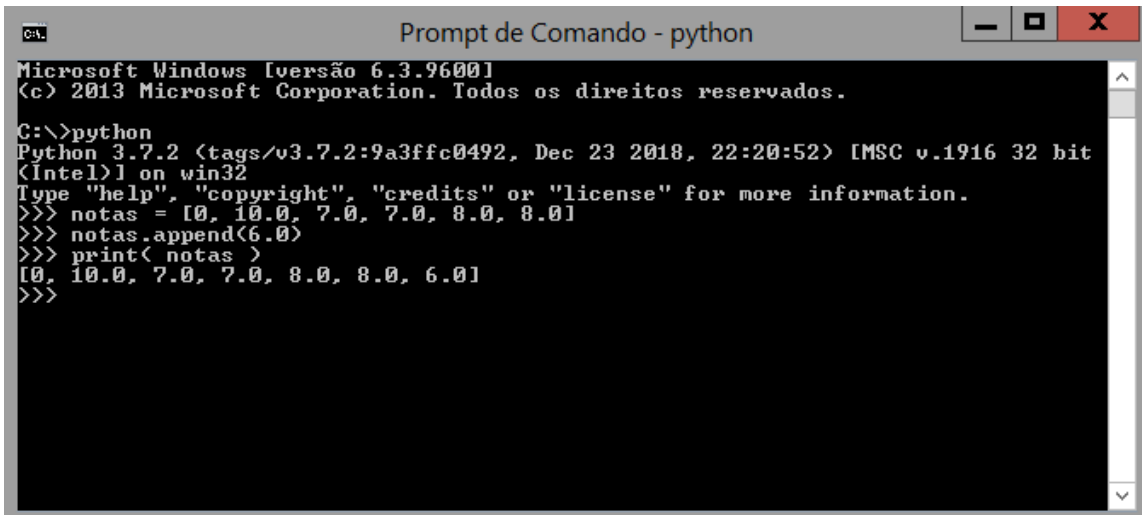
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>> notas.sort()
>>> print( notas )
[0, 7.0, 7.0, 8.0, 8.0, 10.0]
>>> animais = ['gato', 'tatu', 'rato', 'pato']
>>> sorted( animais )
['gato', 'pato', 'rato', 'tatu']
>>>
```

- ◊ **Função append** - Insere um item no final da lista.

```
notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
notas.append(6.0) # Insere na lista
print( notas ) # Imprime a nova lista de notas
```

Veja o resultado no interpretador Python na Figura 40:

Figura 40 – Exemplo append



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

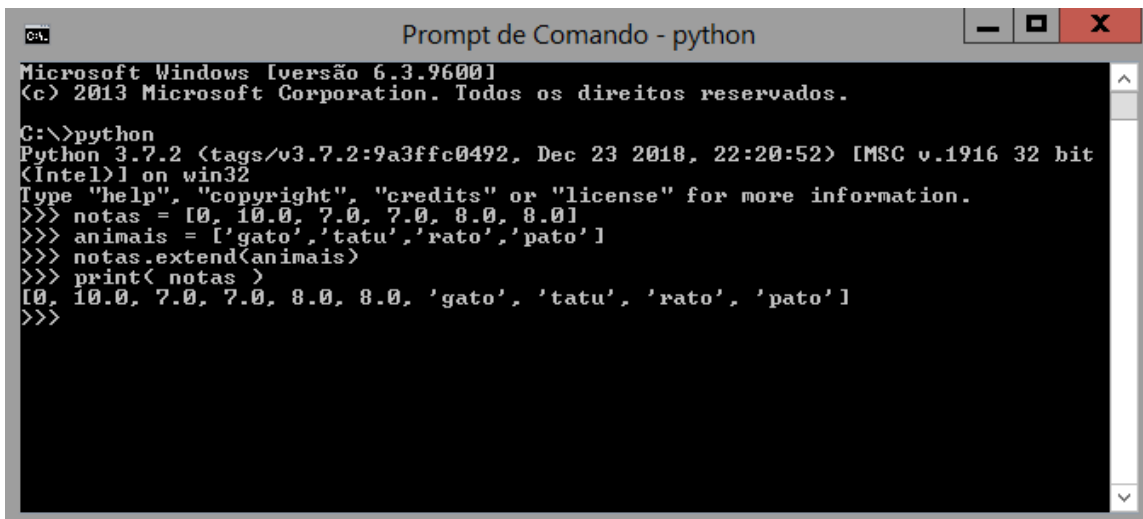
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>> notas.append(6.0)
>>> print( notas )
[0, 10.0, 7.0, 7.0, 8.0, 8.0, 6.0]
>>>
```

- ◊ **Função extend** - “Mescla” duas listas, fazendo com que passe a existir apenas uma, com todos os elementos.

```
notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0] # Lista de valores
animais = ['gato', 'tatu', 'rato', 'pato'] # Lista de strings
notas.extend(animais) # Mescla as listas
print( notas ) # Imprime a nova lista de notas
```

Veja o resultado no interpretador Python na Figura 41:

Figura 41 – Exemplo extend



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

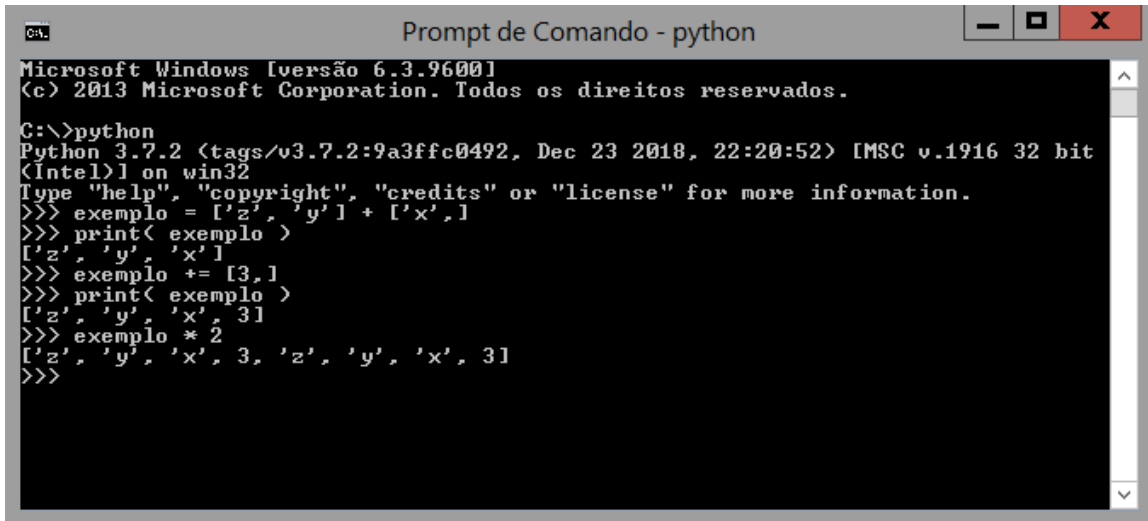
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> notas = [0, 10.0, 7.0, 7.0, 8.0, 8.0]
>>> animais = ['gato', 'tatu', 'rato', 'pato']
>>> notas.extend(animais)
>>> print( notas )
[0, 10.0, 7.0, 7.0, 8.0, 8.0, 'gato', 'tatu', 'rato', 'pato']
>>>
```

◇ Operações com Listas

```
exemplo = ['z', 'y'] + ['x',] # Soma listas
print( exemplo ) # Imprime a lista após a soma
exemplo += [3,] # Soma listas
print( exemplo ) # Imprime a lista após a nova soma
exemplo * 2 # Retorna Multiplicação da lista
```

Veja o resultado no interpretador Python na Figura 42:

Figura 42 – Exemplo Operações com Listas



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = ['z', 'y'] + ['x', 1]
>>> print(exemplo)
['z', 'y', 'x', 1]
>>> exemplo += [3, 1]
>>> print(exemplo)
['z', 'y', 'x', 3, 1]
>>> exemplo * 2
['z', 'y', 'x', 3, 'z', 'y', 'x', 3]
>>>
```

4.2.2 Tuplas

Uma tupla (tuple), como uma lista, é uma sequência de itens de qualquer tipo. Entretanto, diferentemente de listas, tuplas são imutáveis. Sintaticamente, uma tupla é uma sequência de valores separadas por uma vírgula.

4.2.2.1 Exemplos de Tuplas:

- ◇ Alguem = ('João', 'Salvador', 1989)
- ◇ seq = (1, 2, 3)

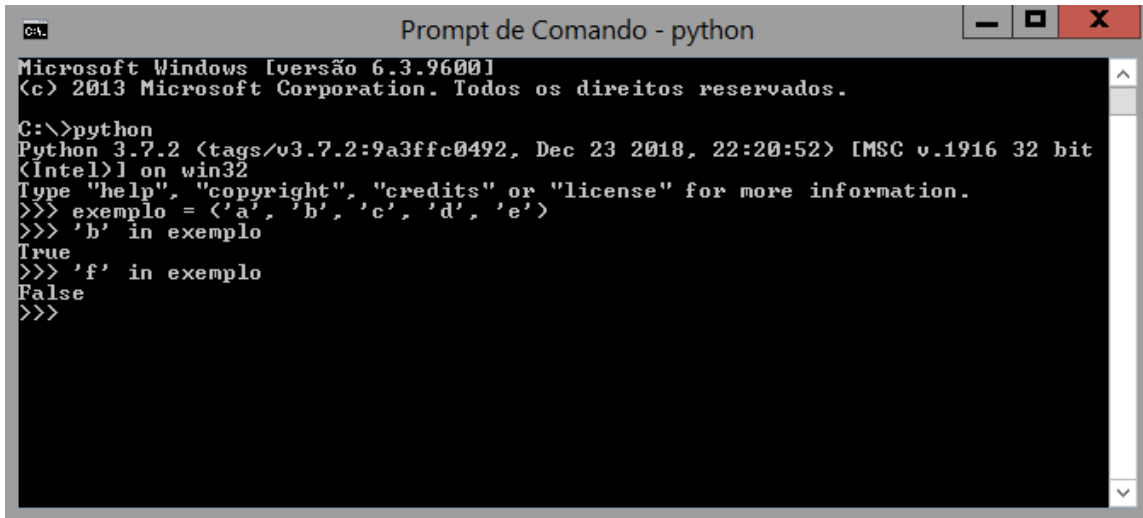
4.2.2.2 Operações com Tuplas

- ◇ **Elemento em uma tupla** - Encontrar elemento em uma tupla.

```
exemplo = ('a', 'b', 'c', 'd', 'e') # Tupla de strings
'b' in exemplo # procura o elemento 'b' na tupla
'f' in exemplo # procura o elemento 'f' na tupla
```

Veja o resultado no interpretador Python na Figura 43:

Figura 43 – Exemplo Elemento em uma tupla



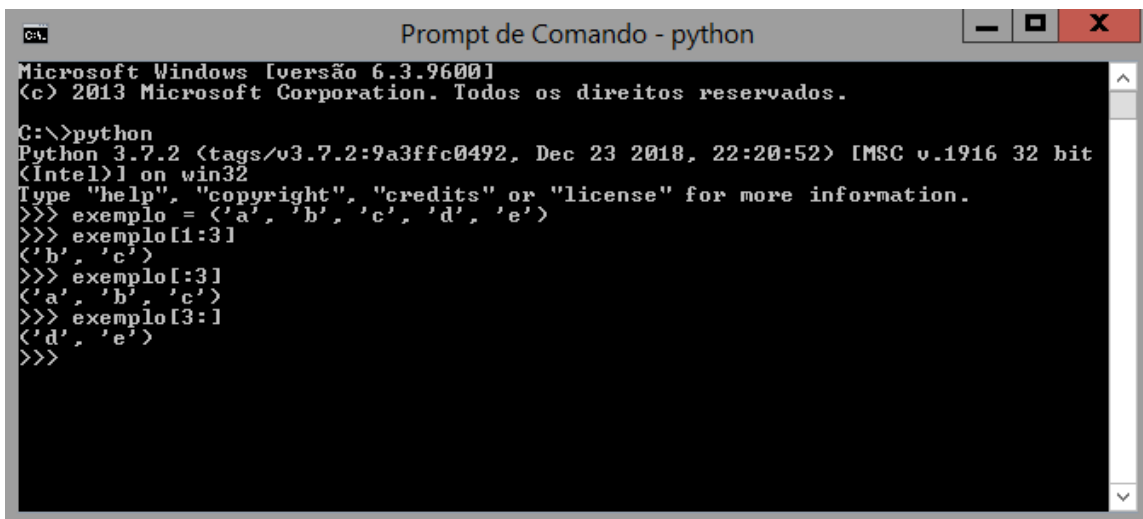
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = ('a', 'b', 'c', 'd', 'e')
>>> 'b' in exemplo
True
>>> 'f' in exemplo
False
>>>
```

◇ **Slicing em uma tupla** - Retorna uma parte dos elemento de uma tupla.

```
exemplo = ('a', 'b', 'c', 'd', 'e') # Tupla de strings
exemplo[1:3] # Retorna elementos do índice 1 até o índice 2 na
    ↳ tupla
exemplo[:3] # Retorna elementos do inicio até o índice 2 na
    ↳ tupla
exemplo[3:] # Retorna elementos do índice 3 até o final na
    ↳ tupla
```

Veja o resultado no interpretador Python na Figura 44:

Figura 44 – Exemplo Slicing em uma tupla



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = ('a', 'b', 'c', 'd', 'e')
>>> exemplo[1:3]
('b', 'c')
>>> exemplo[:3]
('a', 'b', 'c')
>>> exemplo[3:]
('d', 'e')
>>>
```

◇ Métodos index e count

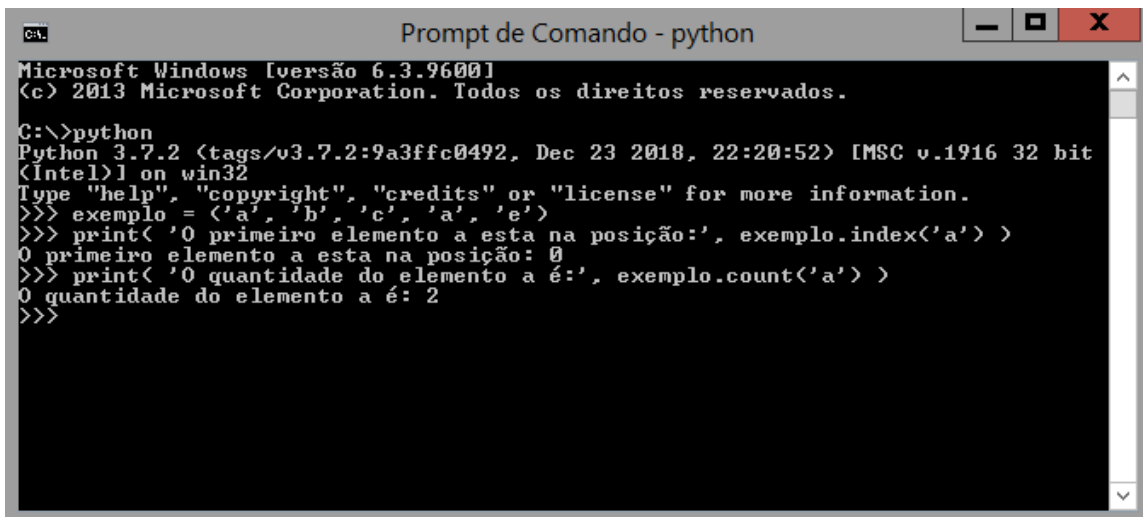
index() - Retorna a posição da primeira ocorrência do elemento na tupla.

count() - Retorna a quantidade de elementos na tupla.

```
exemplo = ('a', 'b', 'c', 'a', 'e') # Tupla de strings
print( 'O primeiro elemento a esta na posição:',
    ↪ exemplo.index('a') ) # Retorna a posição da primeira
    ↪ ocorrência do elemento 'a' na tupla
print( 'O quantidade do elemento a é:', exemplo.count('a') ) #
    ↪ Retorna a quantidade do elemento 'a' na tupla
```

Veja o resultado no interpretador Python na Figura 45:

Figura 45 – Exemplo Métodos index e count



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

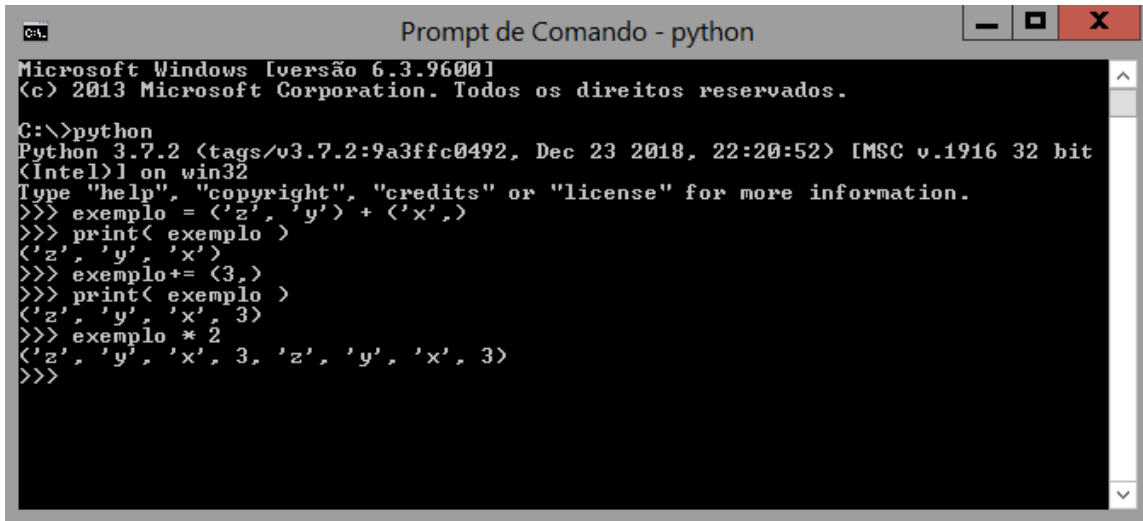
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = ('a', 'b', 'c', 'a', 'e')
>>> print( 'O primeiro elemento a esta na posição:', exemplo.index('a') )
O primeiro elemento a esta na posição: 0
>>> print( 'O quantidade do elemento a é:', exemplo.count('a') )
O quantidade do elemento a é: 2
>>>
```

◇ Operações com Tuplas

```
exemplo = ('z', 'y') + ('x',) # Soma tuplas
print( exemplo ) # Imprime a tupla após a soma
exemplo += (3,) # Soma tuplas
print( exemplo ) # Imprime a tupla após a nova soma
exemplo * 2 # Retorna multiplicação da tupla
```

Veja o resultado no interpretador Python na Figura 46:

Figura 46 – Exemplo Operações com Tuplas



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exemplo = ('z', 'y') + ('x',)
>>> print( exemplo )
('z', 'y', 'x')
>>> exemplo+= (3,)
>>> print( exemplo )
('z', 'y', 'x', 3)
>>> exemplo * 2
('z', 'y', 'x', 3, 'z', 'y', 'x', 3)
>>>
```

4.2.3 Dicionários

Dicionário é um tipo diferente de coleção. Ele é um tipo de mapeamento nativo do Python. Um mapa é uma coleção associativa desordenada. A associação, ou mapeamento, é feita a partir de uma chave, que pode ser qualquer tipo imutável, para um valor, que pode ser qualquer objeto de dados do Python.

4.2.3.1 Exemplos de Dicionários:

- ◇ `dic = { 'a' : 1, 'b' : 2, 'c' : 3 }`
- ◇ `dicionario = { 'letra1' : a, 'letra2' : b, 'letra3' : c }`

4.2.3.2 Operações com Dicionários

◇ Funções do Dicionário

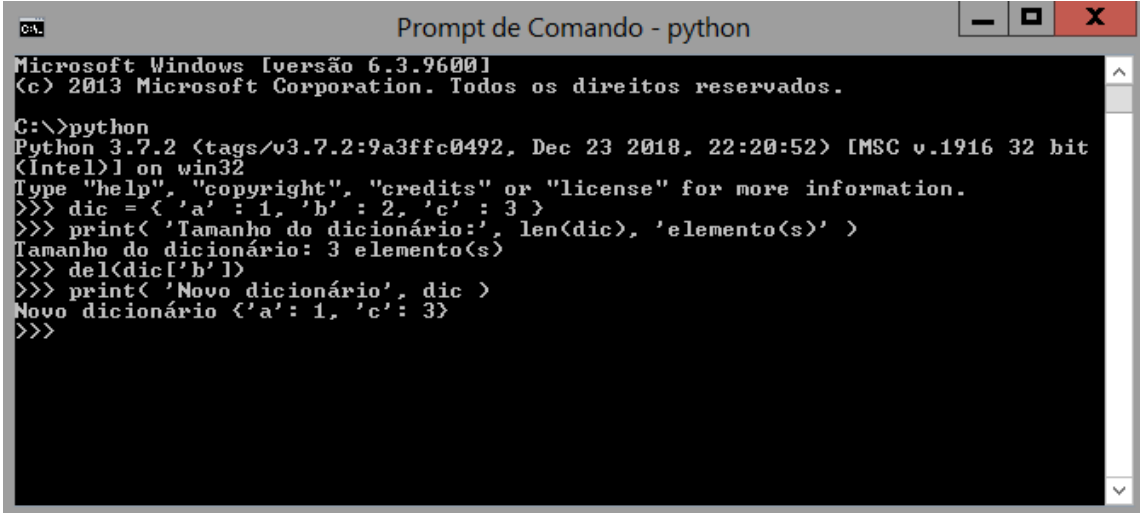
`len()` - Quantidade de elementos contidos do dicionário.

`del()` - Deleta elementos no dicionário.

```
dic = { 'a' : 1, 'b' : 2, 'c' : 3 } # Cria o dicionário
print( 'Tamanho do dicionário:', len(dic), 'elemento(s)' ) #
↳ Imprime a quantidade de elementos do dicionário
del(dic['b']) # Deleta o elementos do dicionário chave 'b'
print( 'Novo dicionário', dic ) # Imprime o novo dicionario
```

Veja o resultado no interpretador Python na Figura 47:

Figura 47 – Exemplo Funções do Dicionário



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dic = { 'a' : 1, 'b' : 2, 'c' : 3 }
>>> print( 'Tamanho do dicionário:', len(dic), 'elemento(s)' )
Tamanho do dicionário: 3 elemento(s)
>>> del(dic['b'])
>>> print( 'Novo dicionário', dic )
Novo dicionário {'a': 1, 'c': 3}
>>>
```

◇ Verificação do Dicionário

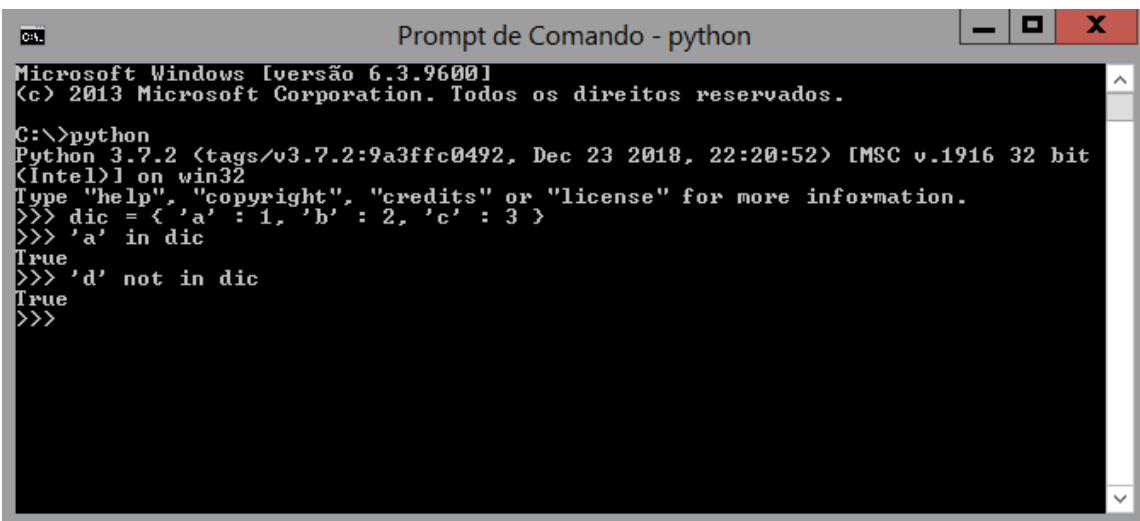
in - Verifica se o item está no dicionário pela chave.

not in - Verifica se o item não está no dicionário pela chave.

```
dic = { 'a' : 1, 'b' : 2, 'c' : 3 } # Cria o dicionário
'a' in dic # Verifica se a chave 'a' está no dicionário
'd' not in dic # Verifica se a chave 'd' não está no dicionário
```

Veja o resultado no interpretador Python na Figura 48:

Figura 48 – Exemplo Verificação do Dicionário



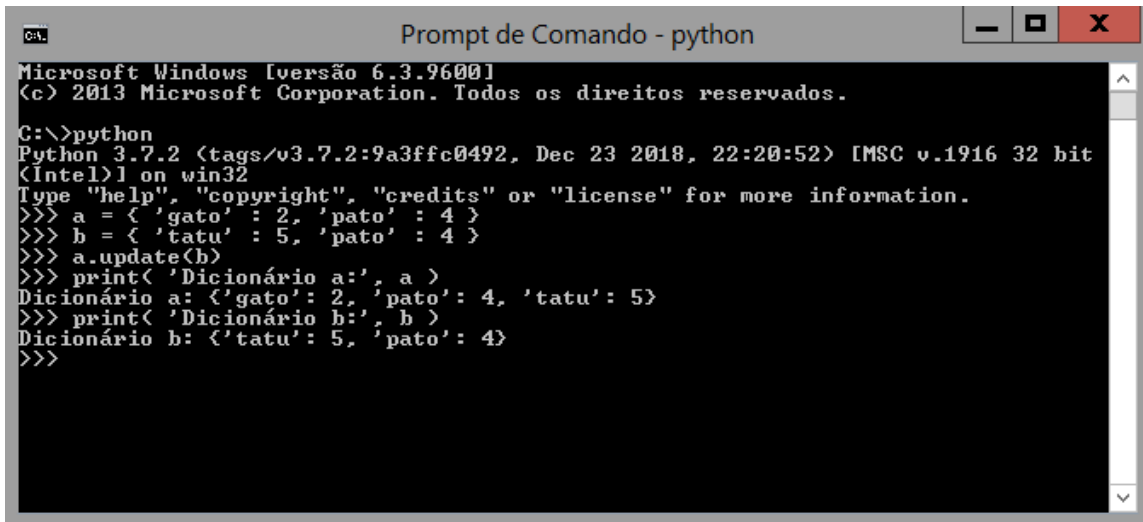
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dic = { 'a' : 1, 'b' : 2, 'c' : 3 }
>>> 'a' in dic
True
>>> 'd' not in dic
True
>>>
```

◇ Mesclando Dicionários

```
a = { 'gato' : 2, 'pato' : 4 } # Cria o dicionário 'a'
b = { 'tatu' : 5, 'pato' : 4 } # Cria o dicionário 'b'
a.update(b) # Mescla o dicionário 'b' com o 'a'
print( 'Dicionário a:', a ) # Imprime o novo dicionário 'a'
print( 'Dicionário b:', b ) # Imprime o dicionário 'b'
```

Veja o resultado no interpretador Python na Figura 49:

Figura 49 – Exemplo Mesclando Dicionários



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = { 'gato' : 2, 'pato' : 4 }
>>> b = { 'tatu' : 5, 'pato' : 4 }
>>> a.update(b)
>>> print( 'Dicionário a:', a )
Dicionário a: {'gato': 2, 'pato': 4, 'tatu': 5}
>>> print( 'Dicionário b:', b )
Dicionário b: {'tatu': 5, 'pato': 4}
>>>
```

◇ Convertendo Dicionário em Lista

keys() - Retorna a lista de chaves de um dicionário.

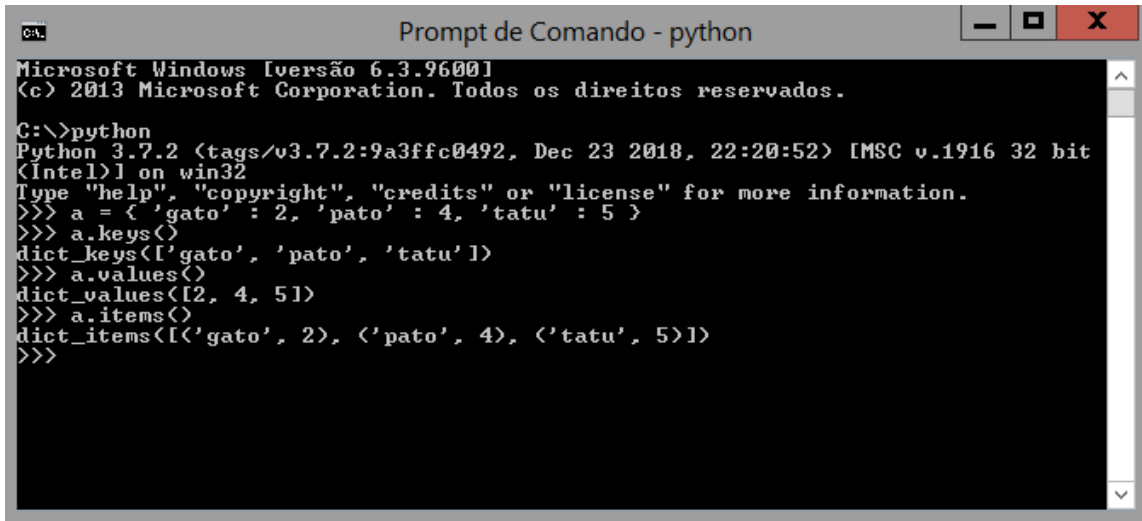
values() - Retorna a lista de valores de um dicionário.

items() - Retorna a lista de lista de tuplas, composta por chave e o valor de um dicionário.

```
a = { 'gato' : 2, 'pato' : 4, 'tatu' : 5 } # Cria o dicionário
↪ 'a'
a.keys() # Retorna a lista de chaves do dicionário 'a'
a.values() # Retorna a lista de valores do dicionário 'a'
a.items() # Retorna a lista de tuplas(chave, valor) do
↪ dicionário 'a'
```

Veja o resultado no interpretador Python na Figura 50:

Figura 50 – Exemplo Convertendo Dicionário em Lista



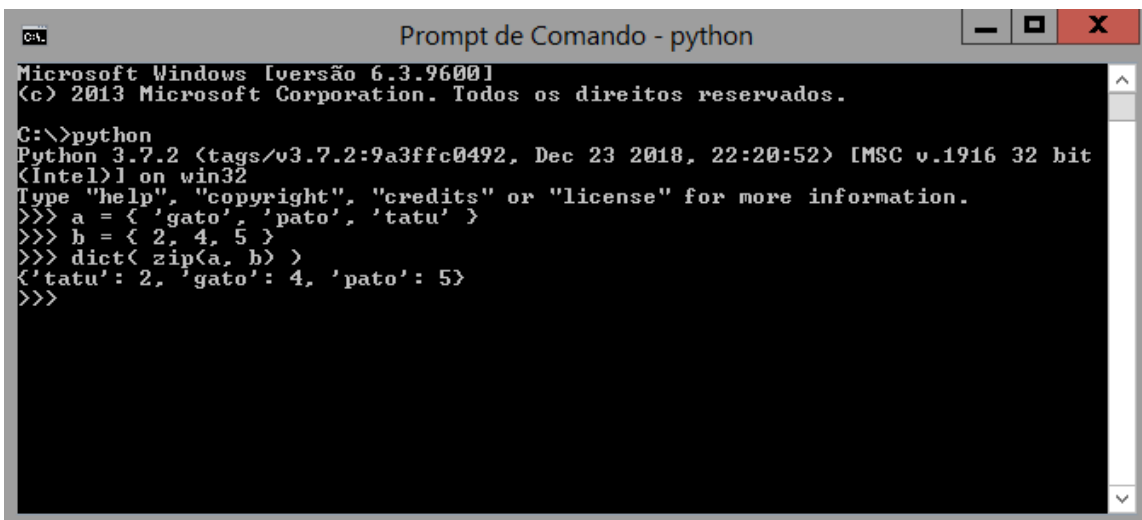
```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = { 'gato' : 2, 'pato' : 4, 'tatu' : 5 }
>>> a.keys()
dict_keys(['gato', 'pato', 'tatu'])
>>> a.values()
dict_values([2, 4, 5])
>>> a.items()
dict_items([('gato', 2), ('pato', 4), ('tatu', 5)])
>>>
```

◇ Convertendo Lista em Dicionário

```
a = { 'gato', 'pato', 'tatu' } # Cria a lista 'a'
b = { 2, 4, 5 } # Cria a lista 'b'
dict( zip(a, b) ) # Retorna um dicionário 'a' como chave e 'b'
                  ↪ como valores
```

Veja o resultado no interpretador Python na Figura 51:

Figura 51 – Exemplo Convertendo Lista em Dicionário



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [ 'gato', 'pato', 'tatu' ]
>>> b = [ 2, 4, 5 ]
>>> dict( zip(a, b) )
{'tatu': 2, 'gato': 4, 'pato': 5}
>>>
```

4.3 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Conceito de variáveis dos tipos numéricas, strings e booleanas.
- ◇ Trabalhar com variáveis numéricas, strings e booleanas.
- ◇ Conceito das estruturas de dados listas, tuplas e dicionários.
- ◇ Trabalhar com listas, tuplas e dicionários.

Se não conseguiu entender todos esse conceitos, releia o capítulo, pois vamos precisar desse entendimentos no decorrer dos próximos capítulos. Vamos agora entender o que é condicionais, loops e funções.

5 CAPÍTULO - Loops, Condicionais e Funções

Vamos para um nível mais avançado, agora precisamos de conceitos de lógicas para entender os condicionais, loops e funções.

5.1 Condicionais

Os Operadores Condicionais possuem uma ou mais condições que são verificadas e retornam o valor Verdadeiro ou Falso. Podendo haver condicionais simples, compostos e aninhados, vamos ver cada tipo de condicional a seguir.

5.1.1 Condicionais Simples - if

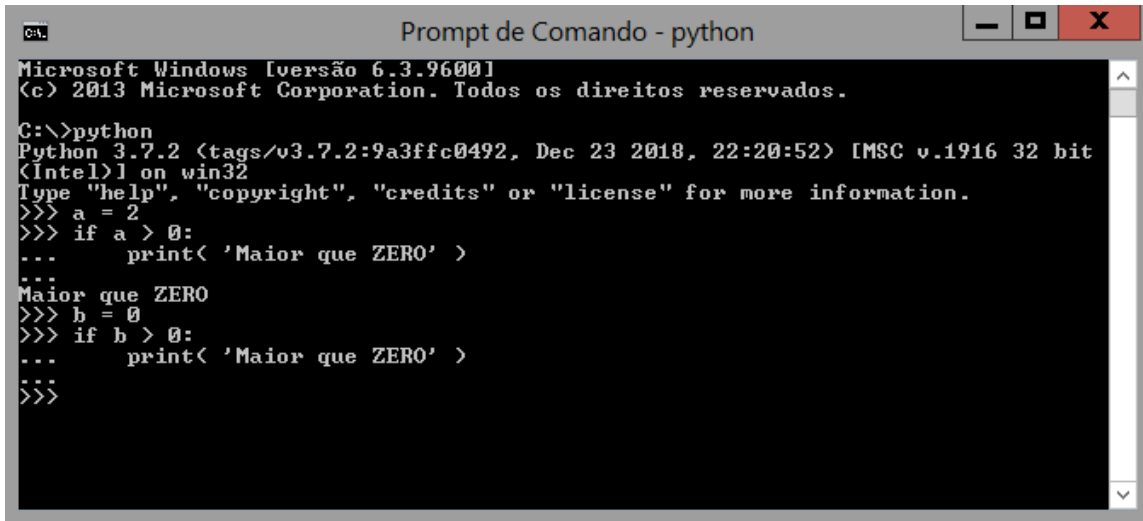
A Estrutura Condicional Simples executa uma determinada ação somente quando a condição for verdadeira, caso a condição seja falsa passa para a próxima instrução do programa.

5.1.1.1 Exemplo de Condicional Simples

```
a = 2 # Atribui o valor 2 a variável 'a'
if a > 0: # Verifica a condição 'a' maior que 0
    print( 'Maior que ZERO' ) # Imprime se 'a' for maior que 0
b = 0 # Atribui o valor 0 a variável 'b'
if b > 0: # Verifica a condição 'b' maior que 0
    print( 'Maior que ZERO' ) # Imprime se 'b' for maior que 0
```

Veja o resultado no interpretador Python na Figura 52 e observe que para a variável 'a' a condição foi verdadeira então executou a impressão 'Maior que ZERO', já para a variável 'b' a condição foi falsa então ele não fez nada.

Figura 52 – Exemplo de Condicional Simples



```

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel> on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> if a > 0:
...     print( 'Maior que ZERO' )
...
Maior que ZERO
>>> b = 0
>>> if b > 0:
...     print( 'Maior que ZERO' )
...
>>>

```

5.1.2 Condicionais Compostos - if/else

A Estrutura Condicional Composta executa uma determinada ação quando a condição for verdadeira e outra ação quando a condição for falsa.

5.1.2.1 Exemplo de Condicional Composto

```

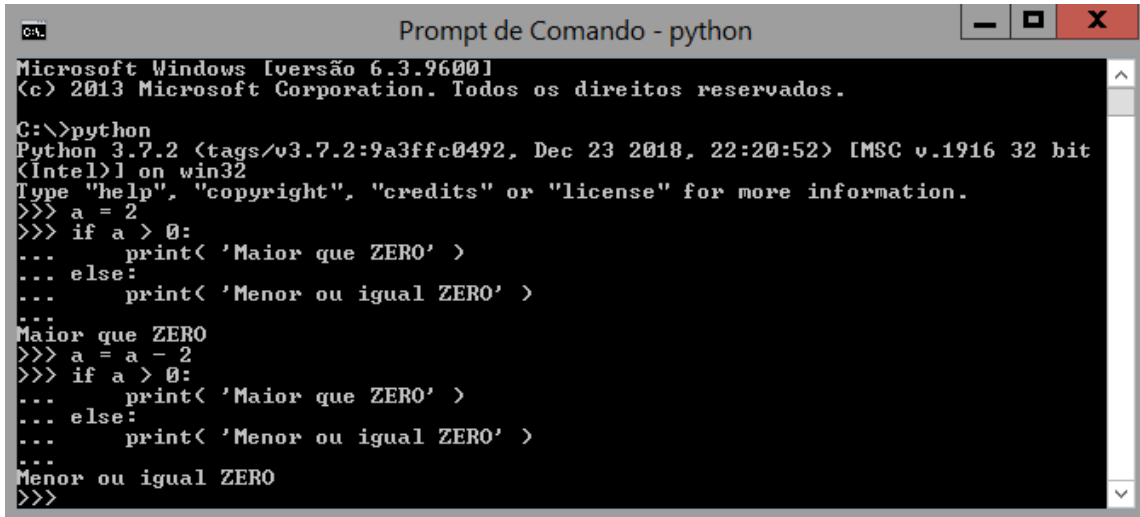
a = 2 # Atribui o valor 2 a variável 'a'
if a > 0: # Verifica a condição 'a' maior que 0
    print( 'Maior que ZERO' ) # Imprime se 'a' for maior que 0
else: # Verifica a condição diferente de 'a' maior que 0
    print( 'Menor ou igual ZERO' ) # Imprime se 'a' for menor ou igual a
    ↪ 0
a = a - 2 # Agora subtrai 2 da variável 'a'
if a > 0: # Verifica a condição 'a' maior que 0
    print( 'Maior que ZERO' ) # Imprime se 'a' for maior que 0
else: # Verifica a condição diferente de a maior que 0
    print( 'Menor ou igual ZERO' ) # Imprime se 'a' for menor ou igual a
    ↪ 0

```

Observando resultado no interpretador Python na Figura 53 podemos entender perfeitamente o funcionamento do condicional composto, num primeiro momento a variável 'a' está com o valor 2 então ele imprime a condição do if 'Maior que ZERO' já no segundo

momento ao subtrair 2 da variável 'a' ela fica com 0, então ele imprime a condição do else 'Menor ou igual ZERO'.

Figura 53 – Exemplo de Condicional Composto



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> if a > 0:
...     print( 'Maior que ZERO' )
... else:
...     print( 'Menor ou igual ZERO' )
...
Maior que ZERO
>>> a = a - 2
>>> if a > 0:
...     print( 'Maior que ZERO' )
... else:
...     print( 'Menor ou igual ZERO' )
...
Menor ou igual ZERO
>>>
```

5.1.3 Condicionais aninhados - if/elif/else

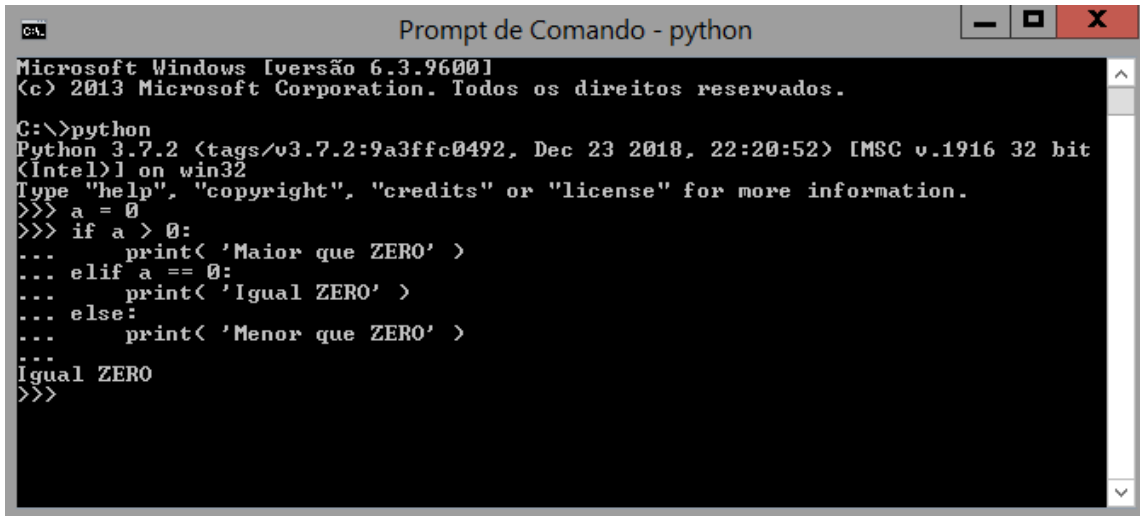
As estruturas Condicionais Aninhadas são várias condições em cascatas, ou seja, condições dentro de condições. Executando apenas o ação para a condição verdadeira.

5.1.3.1 Exemplo de Condicional Aninhado

```
a = 0 # Atribui o valor 0 a variável 'a'
if a > 0: # Verifica a condição 'a' maior que 0
    print( 'Maior que ZERO' ) # Imprime se 'a' for maior que 0
elif a == 0: # Verifica a condição 'a' igual a 0
    print( 'Igual ZERO' ) # Imprime se 'a' for igual a 0
else: # Verifica a condição diferente de 'a' maior ou igual a 0, ou
    ↪ seja, menor que 0
    print( 'Menor que ZERO' ) # Imprime se 'a' for menor que 0
```

Veja o resultado no interpretador Python na Figura 54, onde devido a variável 'a' ter o valor 0, então ele imprime a condição do elif 'Igual ZERO'.

Figura 54 – Exemplo de Condicional Aninhado



```
C:\>python
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>1 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 0
>>> if a > 0:
...     print( 'Maior que ZERO' )
... elif a == 0:
...     print( 'Igual ZERO' )
... else:
...     print( 'Menor que ZERO' )
...
Igual ZERO
>>>
```

5.2 Loops

Vamos utilizar loops quando uma mesma instrução ou conjunto de instruções precisa ser repetida várias vezes seguidas. O laço de repetição, como também é conhecido, permite executar um bloco repetidas vezes, enquanto uma dada condição é atendida.

Em Python, temos os loops for e while. O for percorre os itens de uma coleção e, para cada um deles, executa um bloco de instruções, já o while, executa um bloco de instruções várias vezes enquanto uma condição é atendida. Vamos ver exemplos para ficar mais claro o entendimento.

5.2.1 For

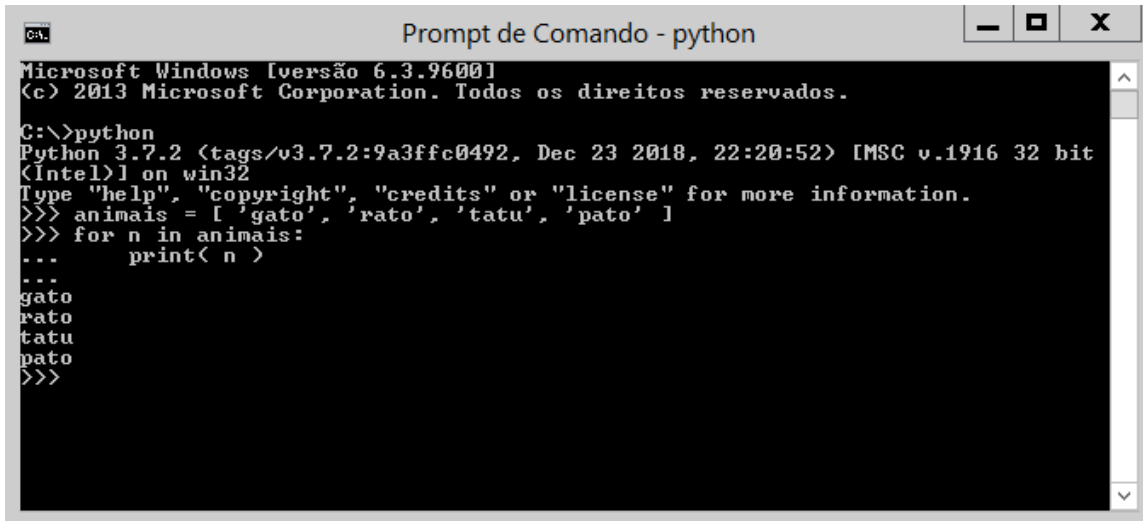
O comando for permite percorrer sequências previamente conhecidas.

5.2.1.1 Exemplo de Loop for

```
animais = [ 'gato', 'rato', 'tatu', 'pato' ] # Criação de uma lista
for n in animais: # Loop for n vezes na lista animais
    print( n ) # Imprime as n vezes do loop for
```

Veja o resultado no interpretador Python na Figura 55, onde foi impresso os 4 itens da lista, um por vez a cada interação do loop.

Figura 55 – Exemplo Loop for



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> animais = [ 'gato', 'rato', 'tatu', 'pato' ]
>>> for n in animais:
...     print( n )
...
gato
rato
tatu
pato
>>>
```

5.2.2 While

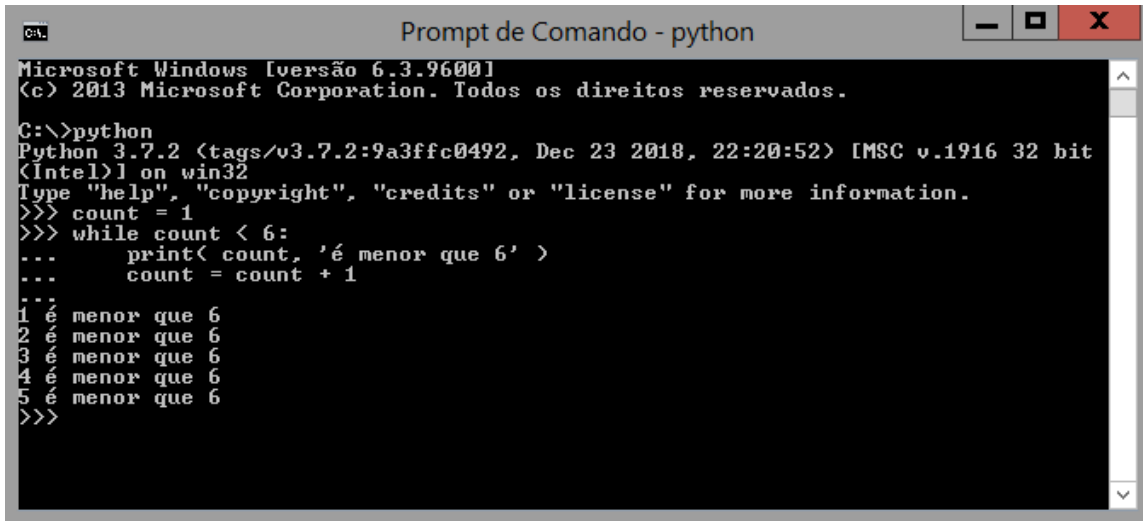
O comando while permite repetir instruções enquanto uma condição for verdadeira.

5.2.2.1 Exemplo de Loop while

```
count = 1 # Atribui valor 1 a variável 'count'
while count < 6: # Loop while enquanto 'count' menor que 6
    print( count, 'é menor que 6' ) # Imprime enquanto 'count' for menor
    ↪ que 6
    count = count + 1 # Incremente 1 ao 'count' em cada loop while
```

Veja o resultado no interpretador Python na Figura 56 que enquanto o valor da variável 'count' for menor que 6, o bloco de instruções vai ser executado, imprimindo o contador na mensagem e somando + 1 a cada interação do loop até não satisfazer mais a condição.

Figura 56 – Exemplo Loop while



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel> on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> count = 1
>>> while count < 6:
...     print( count, 'é menor que 6' )
...     count = count + 1
...
1 é menor que 6
2 é menor que 6
3 é menor que 6
4 é menor que 6
5 é menor que 6
>>>
```

5.2.3 Continue e Break

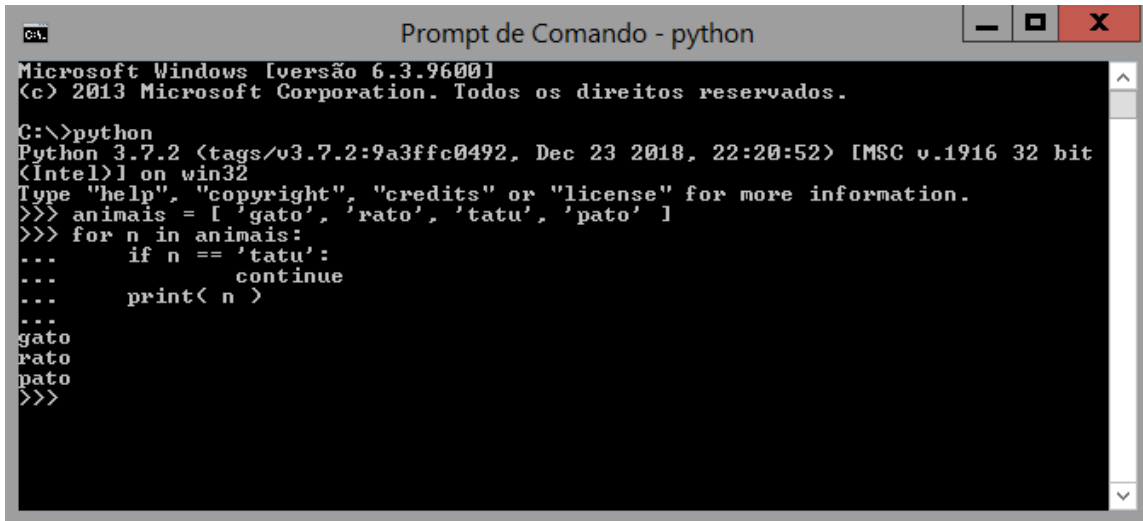
Para ajudar e otimizar o controle dos loops temos os comando continue e break, que já trazem em seus nomes basicamente as suas funções. Para iniciar imediatamente a próxima volta do loop, use o comando continue e para encerrar imediatamente o loop, use o comando break, vamos ver os exemplos.

5.2.3.1 Exemplo de continue

```
animais = [ 'gato', 'rato', 'tatu', 'pato' ] # Criação de uma lista
for n in animais: # Loop for n vezes na lista animais
    if n == 'tatu': # Verifica a condição n igual a 'tatu'
        continue # Passa para o próximo loop for nesse ponto
    print( n ) # Imprime as n vezes do loop for
```

Ao utilizar o continue dentro de uma condição, no caso a palavra 'tatu', quando esse item é encontrado na lista ele passa para o próximo item sem executar as instruções após, no nosso caso a impressão desse item, imprimindo apenas os outros animais da lista, como podemos na Figura 57.

Figura 57 – Exemplo Loop com continue



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

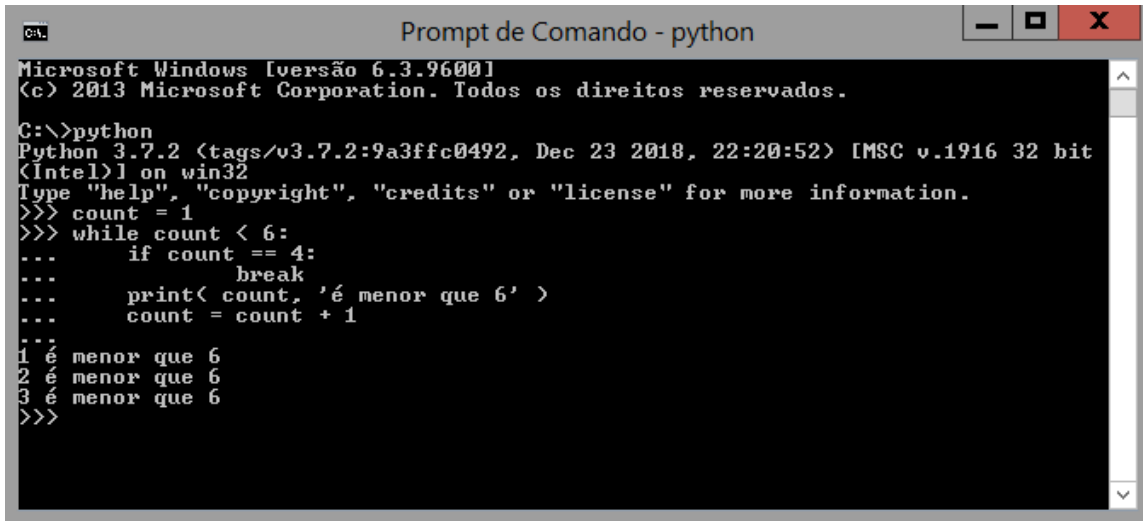
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> animais = [ 'gato', 'rato', 'tatu', 'pato' ]
>>> for n in animais:
...     if n == 'tatu':
...         continue
...     print( n )
...
gato
rato
pato
>>>
```

5.2.3.2 Exemplo de break

```
count = 1 # Atribui valor 0 a variável 'count'
while count < 6: # Loop while enquanto 'count' menor que 6
    if count == 4: # Verifica a condição n igual a '4'
        break # Para a execução do loop while nesse ponto
    print( count, 'é menor que 6' ) # Imprime enquanto for menor que 6
    count = count + 1 # Incrementa 1 ao contador em cada loop while
```

Ao utilizar o break dentro de uma condição, quando a mesma é atendida ele interrompe a execução do while naquele exato momento, imprimindo apenas 3 interações do while, como podemos ver na Figura 58:

Figura 58 – Exemplo Loop com break



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel> on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> count = 1
>>> while count < 6:
...     if count == 4:
...         break
...     print( count, 'é menor que 6' )
...     count = count + 1
...
1 é menor que 6
2 é menor que 6
3 é menor que 6
>>>
```

5.3 Funções

Em Python, uma função tem um nome e é uma sequência de comandos que executa alguma tarefa. A sua principal finalidade é nos ajudar a organizar, padronizar, simplificar programas em pedaços que correspondam a como imaginamos uma solução do problema, para evitar que todo o que utilizam a linguagem criem tudo do zero. Vamos ver exemplos de algumas funções muito interessantes.

5.3.1 Funções da Biblioteca matplotlib

5.3.1.1 Função Pyplot

A função Pyplot está disponível na biblioteca matplotlib que deve ser importada para utilizar a função. O conjunto de funções disponível em matplotlib.pyplot permite criação de gráficos de vários tipos, desenha linhas na área do gráfico, personaliza com títulos, legendas entre outras. Vamos ver alguns exemplos a seguir.

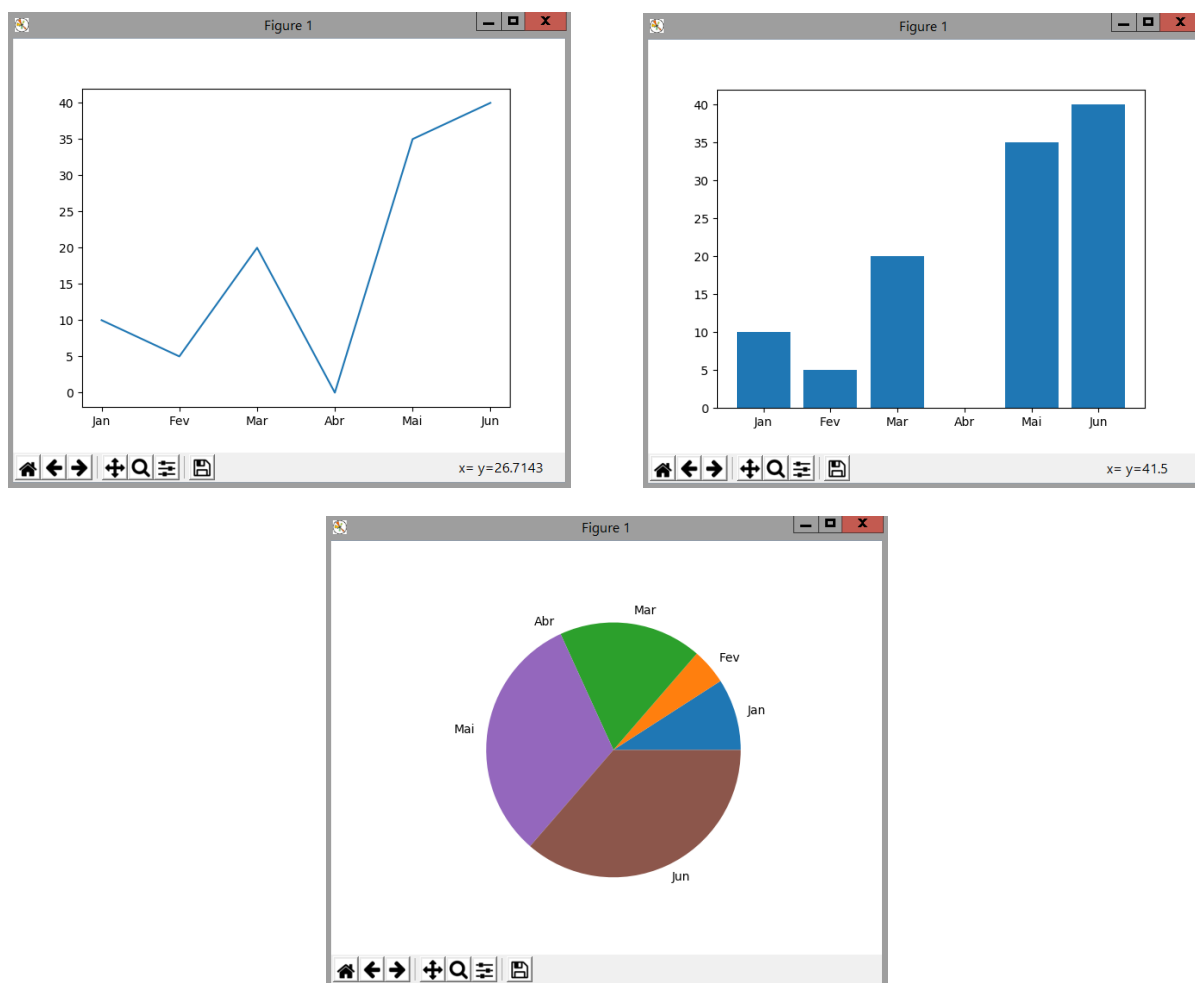
Gráficos pyplot - Impressão de gráficos de linha, barras e pizza.

```
import matplotlib.pyplot as plt # Importa biblioteca
meses = ['Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun'] # Lista de strings
valores = [10, 5, 20, 0, 35, 40] # Lista de valores
plt.plot(meses, valores) # Plota um gráfico de linhas
plt.show() # Exibe o gráfico gerado acima
```

```
plt.bar(meses, valores) # Plota um gráfico de barras
plt.show() # Exibe o gráfico gerado acima
plt.pie(valores, labels=meses) # Plota um gráfico de pizza
plt.show() # Exibe o gráfico gerado acima
```

Abaixo na Figura 59, podemos ver a plotagem de três modelos de gráficos bastante comuns.

Figura 59 – Gráficos pyplot



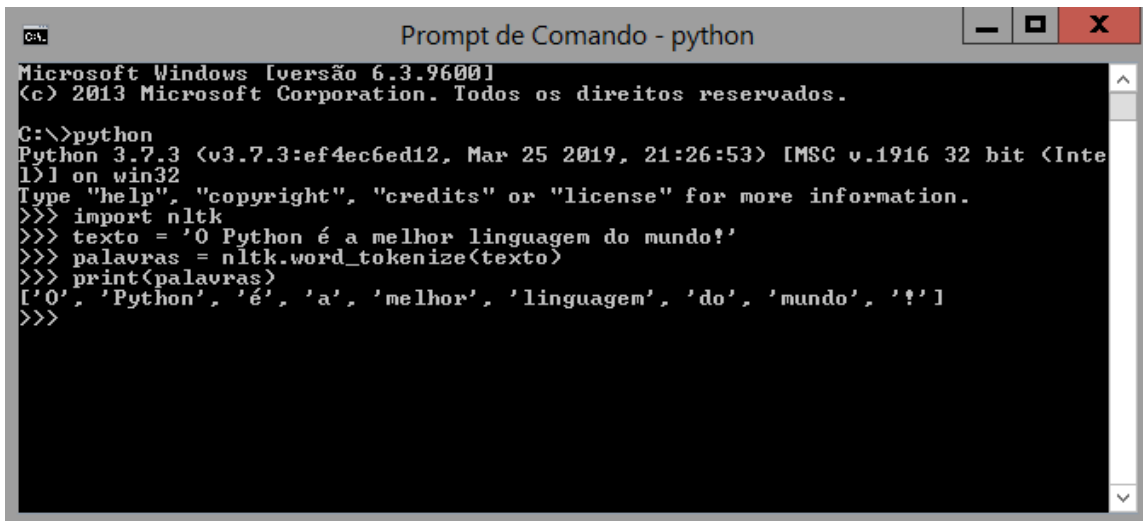
5.3.2 Funções da Biblioteca nltk

5.3.2.1 Função word_tokenize

A função `word_tokenize` esta disponível na biblioteca `nltk`, permite separar palavras de um texto, ou seja, pegar cada palavra de uma sentença.


```
import nltk # Importa a biblioteca
texto = 'O Python é a melhor linguagem do mundo!' # String
palavras = nltk.word_tokenize(texto) # Função para separar palavras
print(palavras) # Imprime a lista de palavras criada
```

Figura 60 – Função word_tokenize



```
C:\>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> texto = 'O Python é a melhor linguagem do mundo!'
>>> palavras = nltk.word_tokenize(texto)
>>> print(palavras)
['O', 'Python', 'é', 'a', 'melhor', 'linguagem', 'do', 'mundo', '!']
>>>
```

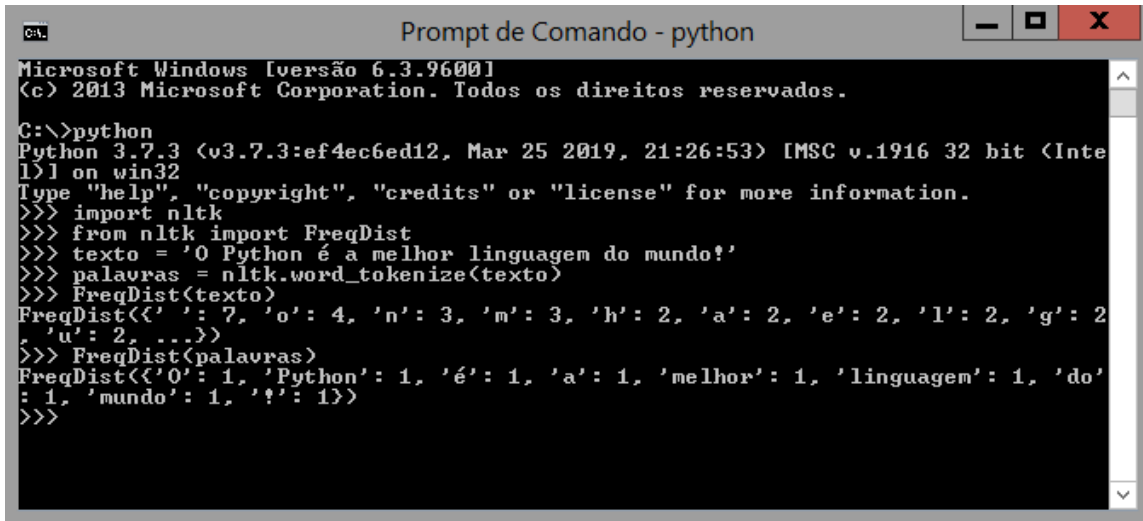
Podemos observar na Figura 60, que a função word_tokenize retorna uma lista contendo cada palavra incluindo os caracteres especiais e excluindo os espaços.

5.3.2.2 Função FreqDist

A função FreqDist está disponível na biblioteca nltk, permite retornar a frequência dos caracteres de um texto ou a frequência das palavras de uma lista, caso o texto tenha sido separado em palavras com o uso da função word_tokenize por exemplo.

```
import nltk # Importa a biblioteca
from nltk import FreqDist # Importa a biblioteca
texto = 'O Python é a melhor linguagem do mundo!' # String
palavras = nltk.word_tokenize(texto) # Função para separar palavras
FreqDist(texto) # Retorna a frequência dos caracteres da string
FreqDist(palavras) # Retorna a frequência das palavras da lista
```

Figura 61 – Função FreqDist



```

C:\>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> from nltk import FreqDist
>>> texto = 'O Python é a melhor linguagem do mundo!'
>>> palavras = nltk.word_tokenize(texto)
>>> FreqDist(texto)
FreqDist<(' ': 7, 'o': 4, 'n': 3, 'm': 3, 'h': 2, 'a': 2, 'e': 2, 'l': 2, 'g': 2, 'u': 2, ...)>
>>> FreqDist(palavras)
FreqDist<('O': 1, 'Python': 1, 'é': 1, 'a': 1, 'melhor': 1, 'linguagem': 1, 'do': 1, 'mundo': 1, '!': 1)>
>>>

```

Podemos observar na Figura 61, que na primeira chamada da função `FreqDist(texto)` ele conta a frequência de todos os caracteres inclusive espaços e caracteres especiais, já na segunda chamada da função `FreqDist(palavras)` é contada a frequência após uma separação de palavras o que exclui o espaço da contagem.

5.3.3 Funções da Biblioteca numpy

5.3.3.1 Função arange

A função `arange` esta disponível na biblioteca `numpy`, permite a criação de arranjos com valores uniformemente espaçados dentro de um determinado intervalo. Podendo ser definido um ponto de partida, um ponto de parada e o tamanho de incremento. Observe que o ponto de parada não será incluído na saída da matriz.

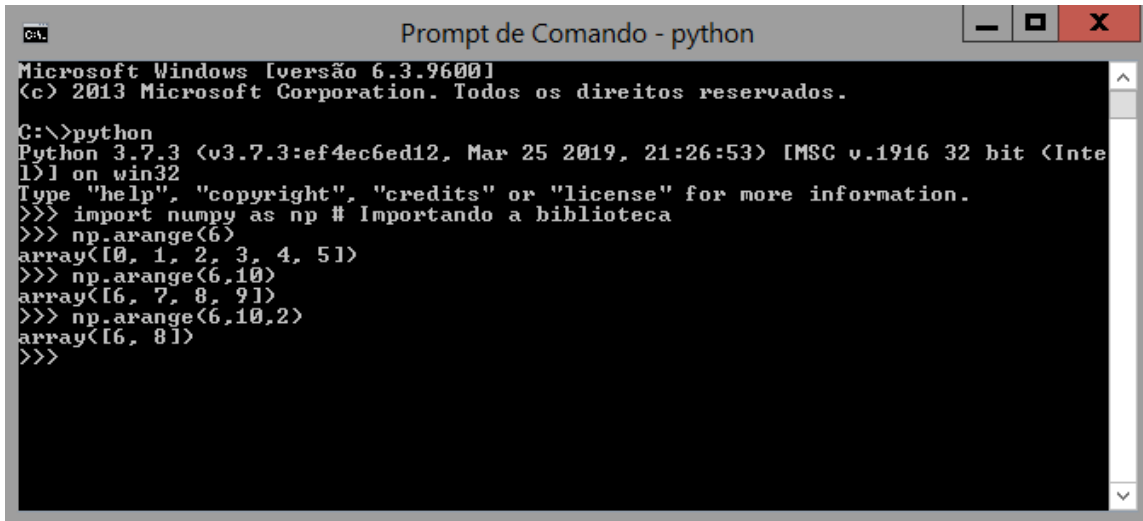
Vamos ver exemplos de como utilizar a função `arange` que possui os seguintes parâmetros `arange(inicio, fim, incremento)`.

```

import numpy as np # Importando a biblioteca
np.arange(6) # Função arange apenas com o parâmetro final
np.arange(6, 10) # Função arange com parâmetros inicial e final
np.arange(6, 10, 2) # Função arange com parâmetros inicial, final e
↪ incremento

```

Figura 62 – Função arange



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np # Importando a biblioteca
>>> np.arange(6)
array([0, 1, 2, 3, 4, 5])
>>> np.arange(6,10)
array([6, 7, 8, 9])
>>> np.arange(6,10,2)
array([6, 8])
>>>
```

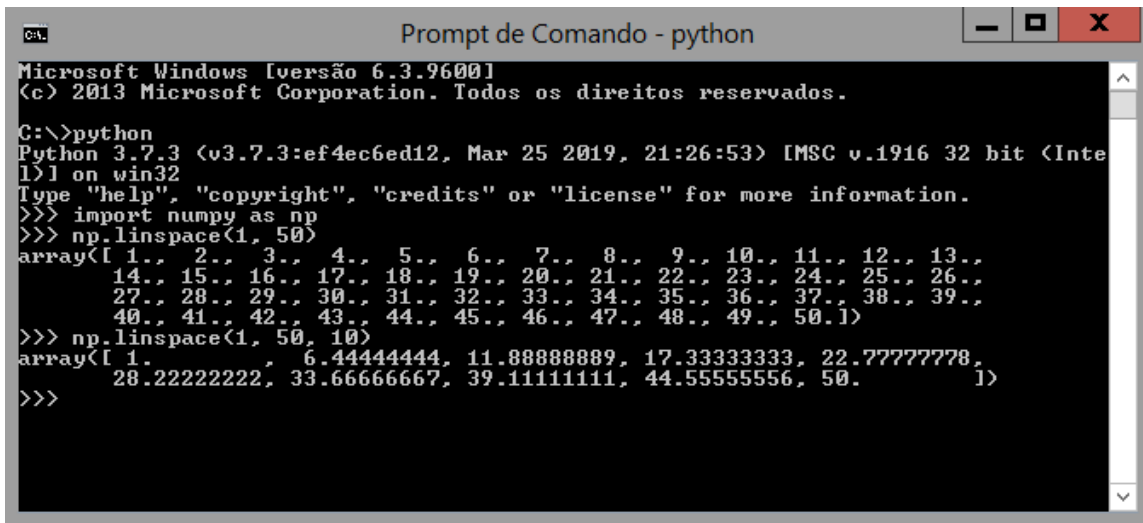
Podemos observar na Figura 62, que os parâmetros inicial (valor 0) e incremento (valor 1) são default caso não sejam informados, e o parâmetro final não é incluído na saída.

5.3.3.2 Função linspace

A função linspace esta disponível também na biblioteca numpy, permite a criação de arranjos com valores espaçados de modo uniforme em um intervalo. Dado o início, o fim e a quantidade de valores, linspace irá distribuí-los uniformemente para você em uma matriz.

```
import numpy as np # Importando a biblioteca
np.linspace(1, 50) # Função linspace com parâmetros inicial e final
np.linspace(1, 50 ,10) # Função linspace com parâmetros inicial, final e
↳ quantidade
```

Figura 63 – Função linspace



```

C:\>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> np.linspace(1, 50)
array([ 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., 32., 33., 34., 35., 36., 37., 38., 39., 40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50.])
>>> np.linspace(1, 50, 10)
array([ 1.,  6.44444444, 11.88888889, 17.33333333, 22.77777778, 28.22222222, 33.66666667, 39.11111111, 44.55555556, 50.])
>>>

```

Podemos observar na Figura 63, que os parâmetros quantidade (valor 50) é default caso não sejam informados, e o parâmetro final incluído na saída.

5.3.4 Funções da Biblioteca wordcloud

5.3.4.1 Função WordCloud

A função WordCloud esta disponível da biblioteca wordcloud, permite a criação de uma nuvem de palavras a partir de uma string, vamos ver um exemplo a seguir.

◊ **Nuvem de Palavras Simples** - Exemplo de uma nuvem de palavras simples.

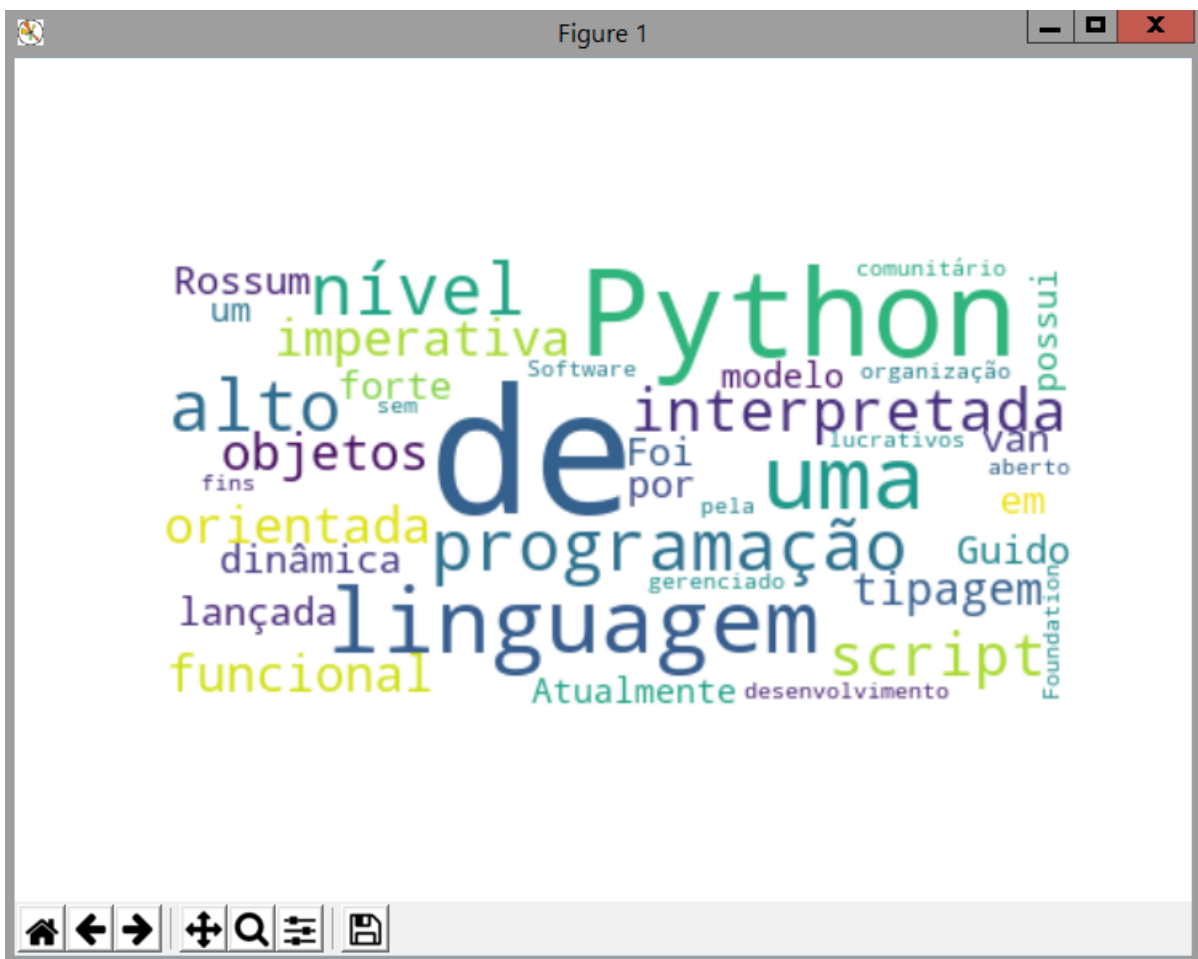
```

from wordcloud import WordCloud # Importa biblioteca
import matplotlib.pyplot as plt # Importa biblioteca
texto = 'Python é uma linguagem de programação de alto nível,
↪ interpretada, de script, imperativa, orientada a objetos,
↪ funcional, de tipagem dinâmica e forte. Foi lançada por
↪ Guido van Rossum em 1991. Atualmente possui um modelo de
↪ desenvolvimento comunitário, aberto e gerenciado pela
↪ organização sem fins lucrativos Python Software Foundation.'
↪ # String
wc = WordCloud(background_color="white", max_words=1000,
↪ contour_width=3, contour_color='firebrick') # parâmetros
↪ para plotagem

```

```
wc.generate(texto) # Gera a plotagem
plt.figure(figsize=[20,10]) # Configura Tamanho
plt.imshow(wc, interpolation='bilinear') # Parametro de exibição
plt.axis("off") # Parametro de exibição
plt.show() # Exibe a nuvem de palavras
```

Figura 64 – Função wordcloud

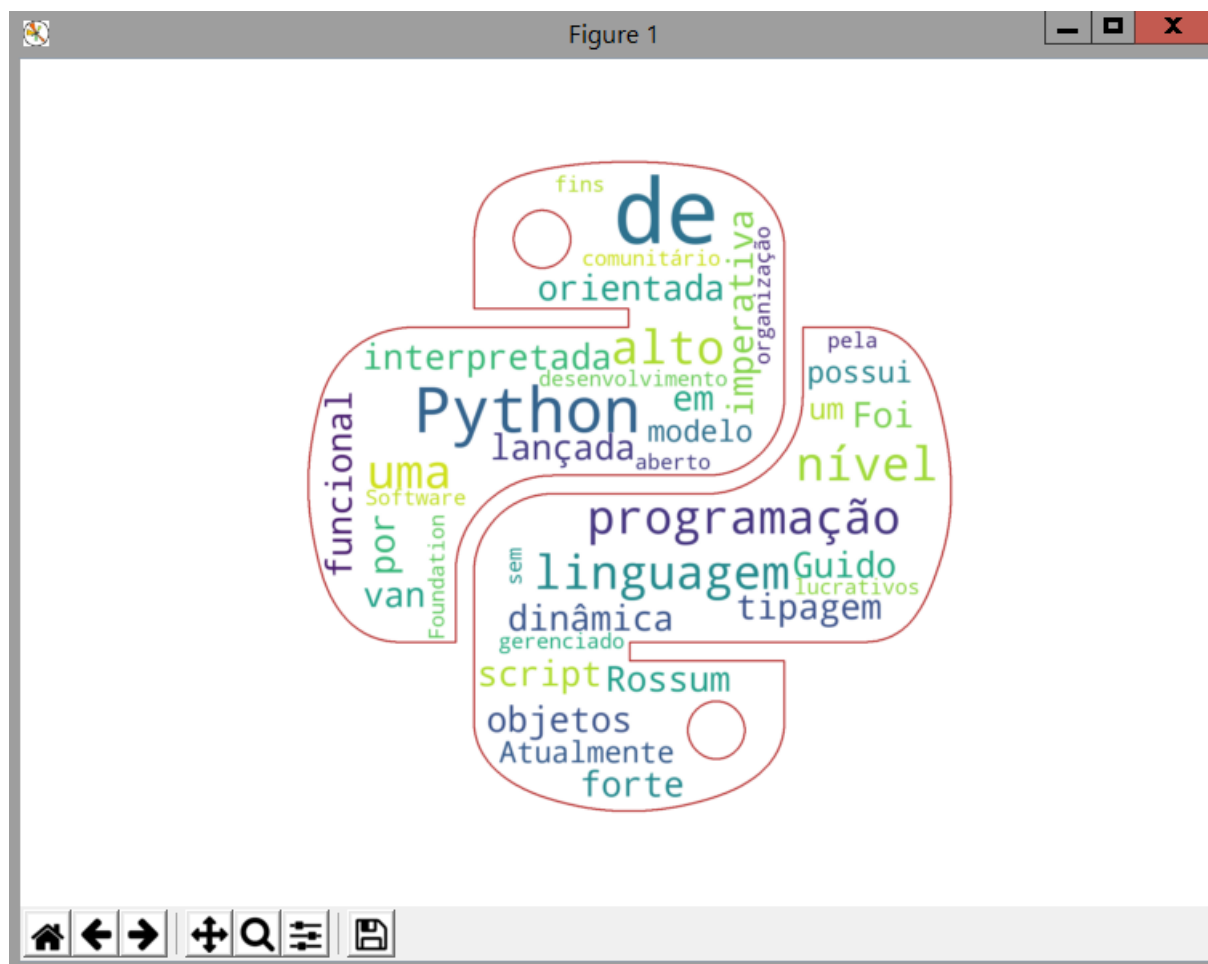


Podemos observar na Figura 64 o resultado do exemplo da nuvem de palavras da forma mais simples possível.

- ◊ **Nuvem de Palavras em Figuras** - Exemplo de uma nuvem de palavras criada dentro de uma imagem.

```
from wordcloud import WordCloud # Importa biblioteca
import matplotlib.pyplot as plt # Importa biblioteca
```

```
import nltk # Importa biblioteca
import numpy as np # Importa biblioteca
from PIL import Image # Importa biblioteca
texto = 'Python é uma linguagem de programação de alto nível,
↳ interpretada, de script, imperativa, orientada a objetos,
↳ funcional, de tipagem dinâmica e forte. Foi lançada por
↳ Guido van Rossum em 1991. Atualmente possui um modelo de
↳ desenvolvimento comunitário, aberto e gerenciado pela
↳ organização sem fins lucrativos Python Software Foundation.'
↳ # String
python = np.array(Image.open('C:\python\python.png')) #
↳ Seleciona a imagem para plotagem
wc = WordCloud(background_color="white", max_words=1000,
↳ mask=python, contour_width=3, contour_color='firebrick') #
↳ parâmetros para plotagem
wc.generate(texto) # Gera a plotagem
wc.to_file('C:\python\python_wordcloud.png') # Salva uma cópia
↳ na nuvem de palavras
plt.figure(figsize=[20,10]) # Configura Tamanho
plt.imshow(wc, interpolation='bilinear') # Parametro de exibição
plt.axis("off") # Parametro de exibição
plt.show() # Exibe a nuvem de palavras
```



Podemos observar na Figura 65 o resultado do exemplo da nuvem de palavras com a plotagem em uma figura.

5.4 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Os tipos de condicionais do Python.
- ◇ Como usar os condicionais.
- ◇ Os tipo de loops do Python.
- ◇ Como fazer loops.

- ◇ O que são funções em Python.
- ◇ Como usar as funções.

Se não conseguiu entender todos esse conceitos, releia o capítulo, pois vamos precisar desse entendimento no decorrer dos próximos capítulos. Vamos agora aprender a trabalhar com arquivos de texto.

6 CAPÍTULO - Trabalhando com Arquivos de Texto

Agora é hora de aprender algumas operações com arquivos de texto, para que possamos manipular através do Python. Já temos algumas funções para facilitar o nosso trabalho, é o que veremos a seguir.

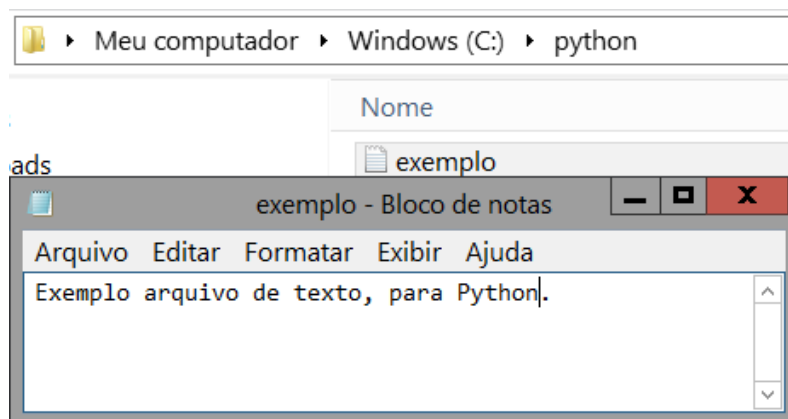
6.1 Lendo um Arquivos de Texto

Para abrir um arquivo e ler o seu conteúdo podemos utilizar a função `open()`. Para visualizar o conteúdo desse arquivo de texto devemos utilizar o método `read()` que retorna o conteúdo inteiro de um arquivo em um string. Vamos ver no exemplo abaixo:

6.1.0.1 Exemplo Lendo um Arquivos de Texto:

Na Figura 66 está o arquivo criado para o nosso exemplo, dá para visualizar seu conteúdo e o caminho onde está salvo, vamos precisar dessa informação para o nosso exemplo.

Figura 66 – Exemplo Arquivo Texto

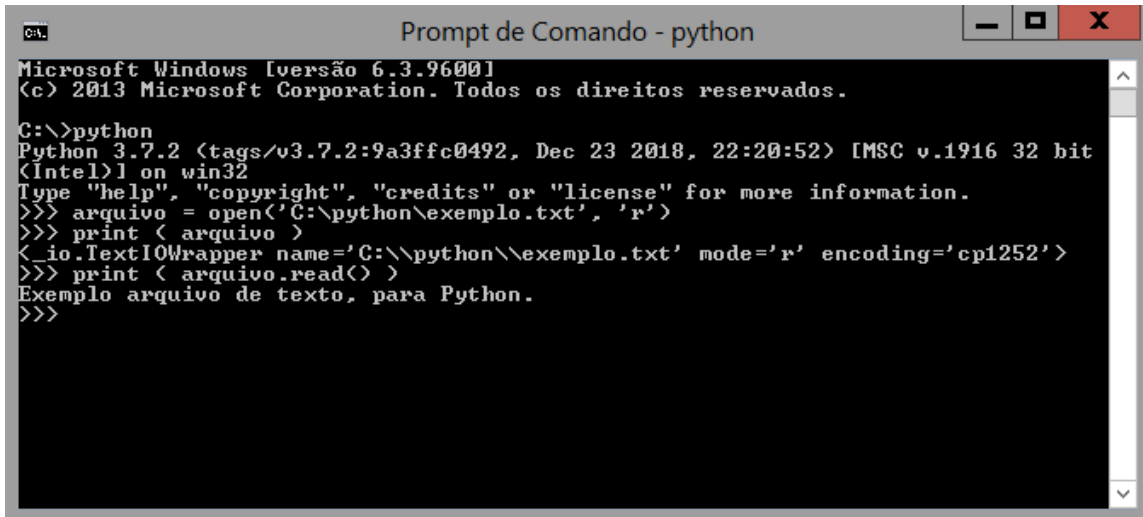


```
arquivo = open('C:\python\exemplo.txt', 'r') # Função para abertura de
→ arquivo de texto, ('Caminho do arquivo', Parâmetro de leitura')
print ( arquivo ) # Imprime a variável que guarda as informações do
→ arquivo aberto
```

```
print ( arquivo.read() ) # Função que seleciona a string dentro do  
↪ arquivo
```

Veja o resultado no interpretador Python na Figura 67, no final da figura podemos observar a impressão do mesmo texto do arquivo apresentado na Figura 66.

Figura 67 – Exemplo Leitura Arquivo de Texto



```
C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> arquivo = open('C:\python\exemplo.txt', 'r')
>>> print ( arquivo )
<_io.TextIOWrapper name='C:\python\exemplo.txt' mode='r' encoding='cp1252'>
>>> print ( arquivo.read() )
Exemplo arquivo de texto, para Python.
>>>
```

6.2 Alterando um Arquivos de Texto

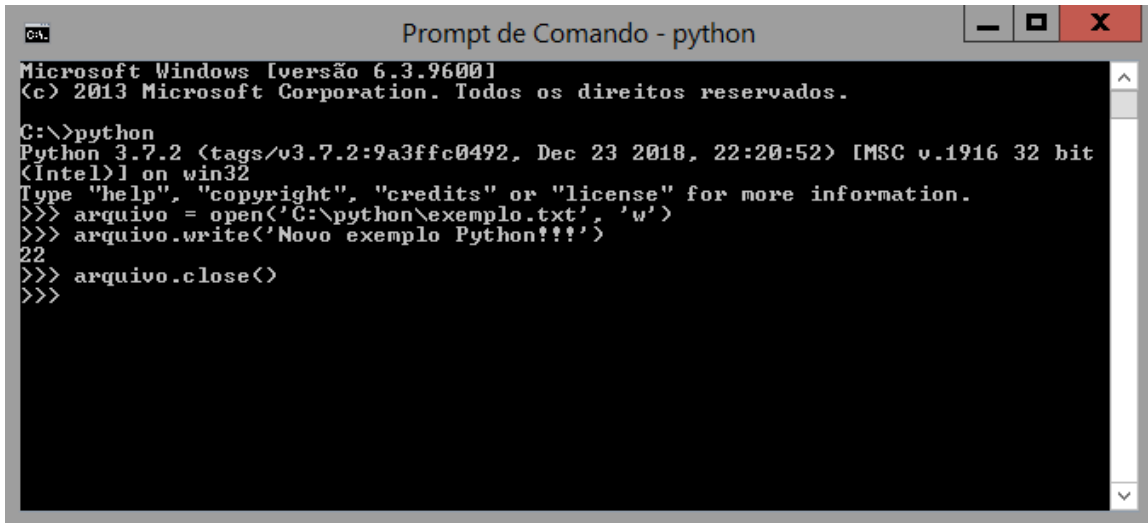
Para alterar o seu conteúdo também utilizamos a função `open()`, mas com o parâmetro de escrita, dessa forma podemos gravar conteúdo neste arquivo. Após a manipulação temos que fechar o arquivo e utilizamos a função `close()` para gravar a alteração realizada. Vamos utilizar o mesmo arquivo da Figura 66 e fazer uma alteração no seu conteúdo que está no exemplo abaixo:

6.2.0.1 Exemplo Alterando um Arquivos de Texto:

```
arquivo = open('C:\python\exemplo.txt', 'w') # Função para abertura de  
↪ arquivo de texto, ('Caminho do arquivo', Parâmetro de leitura)  
arquivo.write('Novo exemplo Python!!!') # Escreve a nova string no  
↪ arquivo aberto  
arquivo.close() # Fecha e salva o conteúdo no arquivo
```

Veja o resultado no interpretador Python na Figura 68, a utilização do parâmetro 'w' na função `open()` é que vai permitir a escrita no arquivo.

Figura 68 – Exemplo Escrevendo Arquivo de Texto

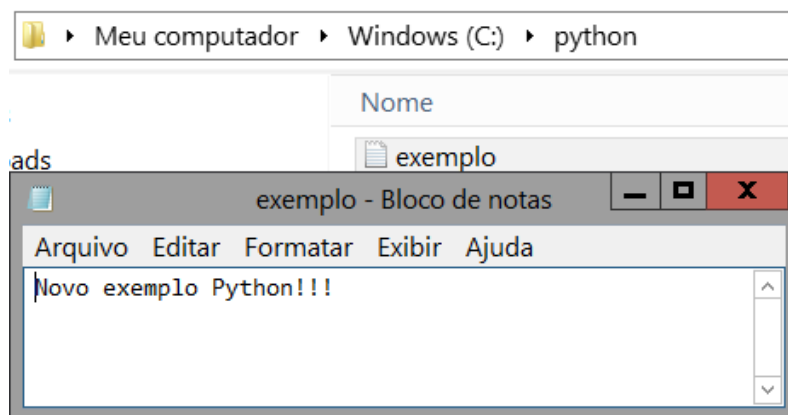


```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.

C:\>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> arquivo = open('C:\python\exemplo.txt', 'w')
>>> arquivo.write('Novo exemplo Python!!!')
>>> arquivo.close()
>>>
```

Abaixo na Figura 69 está o arquivo alterado com o seu novo conteúdo, alteração feita conforme a Figura 68.

Figura 69 – Arquivo de Texto Alterado



6.3 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Manipular arquivos de texto com Python.

Se não conseguiu entender todos esse conceitos, releia o capítulo, pois vamos precisar desse entendimentos no decorrer dos próximos capítulos. Agora já temos os conhecimentos necessários para construir nossas primeiras aplicações em Python, vamos lá.

7 CAPÍTULO - Aplicação de Contagem de Palavras

Pronto! Agora já temos nossa base de conhecimento do Python para construir nossa primeira aplicação, a mesma vai realizar a contagem de palavra repetidas em um ou vários textos.

Para exemplificar vamos resolver o seguinte problema, contar a quantidade das palavras relevantes mais repetidas em textos que contem a integra dos discursos (posse no congresso, entrega da faixa e conferência de economia em Davos) do atual presidente brasileiro Jair Bolsonaro baixados no site do Planalto (<http://www2.planalto.gov.br/acompanhe-o-planalto/discursos/>).

Para chegar a solução precisamos:

1. Ler arquivos de textos com a integra dos discursos
2. Retirar as palavras irrelevantes
3. Plotar um gráfico com a repetição das palavras

Vamos entender o código passo a passo e ver o resultado.

7.1 Aplicação de Contagem de Palavras

Vamos começar criando um arquivo com extensão py aqui vou chamar de (contagem_de_palavras.py), criado no mesmo diretórios onde estão os arquivos de texto dos discursos para facilitar o entendimento e deixar o código mais simples, quando fizermos a leitura do arquivo passaremos apenas o nome do arquivo pois o caminho é o mesmo do arquivo criado (contagem_de_palavras.py). Depois do arquivo criado vamos começar a escrever nossa aplicação no arquivo criado e vou comentar cada linha de comando para não deixar nenhuma dúvida.

```
# importando bibliotecas
from wordcloud import WordCloud
```

```
import matplotlib.pyplot as plt
import nltk
import numpy as np
from nltk import FreqDist
from collections import Counter

# declarando variaveis tipo lista vazias
txt = []
top = []
# declarando variavel tipo lista com valores
discursos = ['congresso.txt', 'entrega_faixa.txt', 'davos.txt']

# loop for para selecionar cada discurso
for d in discursos:
    # criando uma lista das palavras a serem removidas
    lista = ['-', '"', '.', ',', '!', ':', ';', 'de', 'um', 'que',
             'ao', 'da', 'mais', 'em', 'mas', 'dos', 'das', 'acima', 'tudo',
             'na', 'do', 'e', 'a', 'o', 'as', 'os', 'como', 'com', 'esta',
             'no', 'nos', 'meu', 'minha', 'para', 'me', 'pelo', 'à', 'é',
             'ú', 'Com', 'O', 'Que', 'Um', 'Uma', 'aqui', 'aos', 'se', 'sua',
             'suas', 'às', '2005', '28', 'A', 'As', 'E', 'Os', 'sem',
             'isso', 'uma', 'por', 'pela', 'até']
    # realizando abertura e leitura de cada arquivo de texto
    texto = open(d, 'r').read()
    # convertendo a string em um lista de palavras
    palavras = nltk.word_tokenize(texto)

    # loop for em lista das palavras a serem removidas
    for x in lista:
        # loop para verificar todas as palavras do discurso
        while x in palavras:
            # removendo a lista de palavras
            palavras.remove(x)

# juntando os discursos
```

```
txt.extend(palavras)

# contador de palavras com mais de 8 repetições
contador = Counter(txt)
# loop for da quantidade de palavras do discurso
for key, value in contador.items():
    # iniciando uma variavel para contar as repetições
    z = 1
    # condicional if do numero de repetições
    if value > 8:
        # nova lista com mais de 8 repetições
        while z <= value:
            # função para acrescentar itens a nova lista
            top.append(key)
            # incrementando o contador
            z = z + 1

# contando as maiores frequencias
frequencia = FreqDist(top)
# dicionario as palavras e suas frequencias
chave = frequencia.keys()
# posicao do grafico
posicao = np.arange(len(chave))
# plotando de acordo com o dicionario
contagem = frequencia.values()
# tamanho da figura
plt.figure(figsize=(16,9))
# paramentros de plotagem
plt.bar(posicao, contagem, align='center', alpha=0.4)
# legenda eixo x
plt.xticks(posicao, chave)
# legenda eixo y
plt.ylabel('Frequencia das palavras')
# titulo do grafico
plt.title('Palavras do Bolsonaro em discursos oficiais')
```

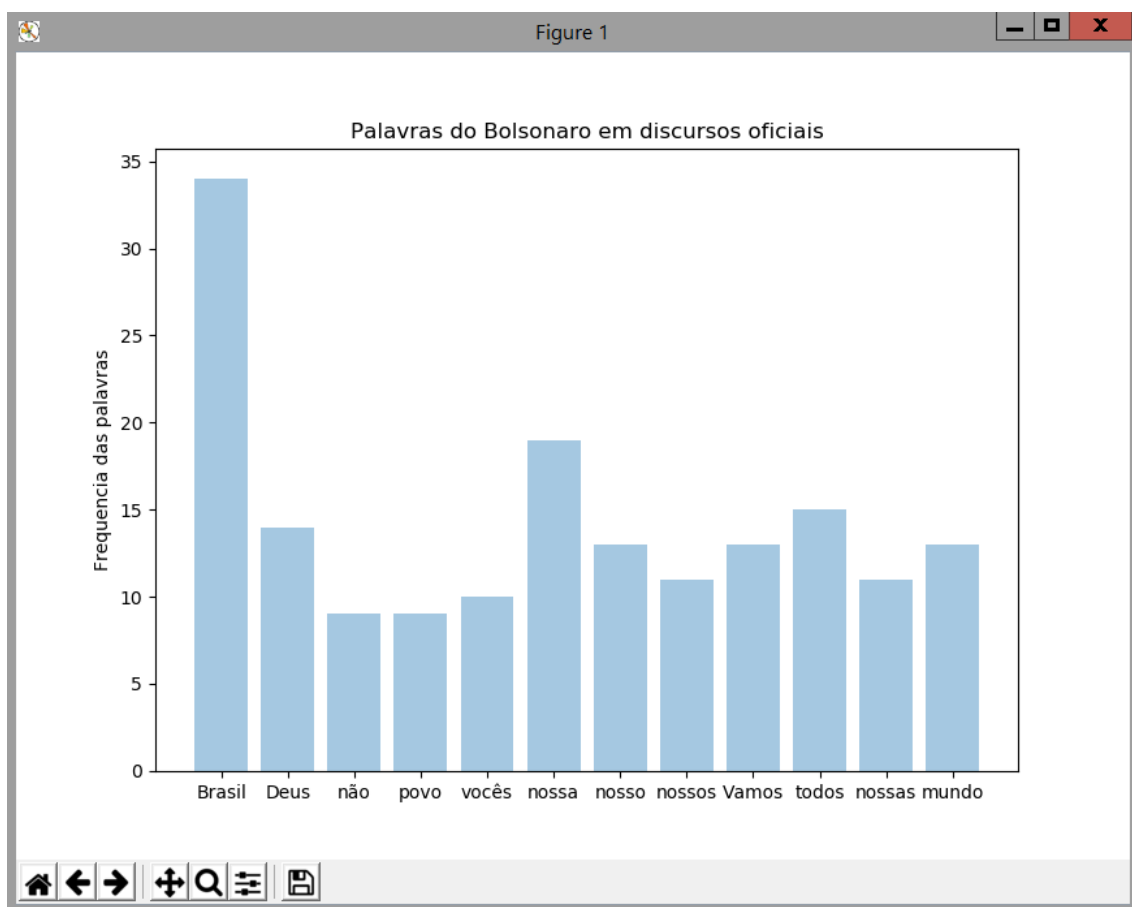
```
# função para plotar o grafico  
plt.show()
```

Agora que finalizamos, vamos salvar o arquivo e executar pelo cmd do Windows na pasta onde esta salvo o arquivo com o comando:

```
python <Nome do Arquivo>
```

Então chegamos ao resultado apresentado na Figura 70.

Figura 70 – Contagem de Palavras



Podemos identificar no gráfico de barras a intensidade de cada palavra no discurso, no nosso caso apenas as palavras repetidas mais de 8 vezes, para que o gráfico fique visualmente bom, pois com a quantidade de palavras distintas nos discursos ficariam muitas barras e nosso objetivo é ver as mais repetidas.

Essa é uma análise interessante que pode ser estendida a outros contextos, por exemplo as palavras mais citadas numa rede social, a contagem de uma palavra em específico, vai

depender da necessidade ou criatividade de cada um.

7.2 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Criação e execução de uma aplicação Python de contagem de palavras.

Se não conseguiu entender todos esse conceitos, releia o capítulo, pois vamos precisar desse entendimento no decorrer dos próximos capítulos. Agora que já construímos nossa primeira aplicação vamos ver mais uma para fixar todos os nossos conhecimentos adquiridos.

8 CAPÍTULO - Aplicação de Nuvem de Palavras

Agora vamos resolver o seguinte problema, criar uma nuvem de palavras relevantes aproveitando os textos utilizados no exemplo anterior que contem a integra dos discursos (posse no congresso, entrega da faixa e conferência de economia em Davos) do atual presidente brasileiro Jair Bolsonaro baixados no site do Planalto (<http://www2.planalto.gov.br/acompanhe-o-planalto/discursos/>).

Para chegar a solução precisamos:

1. Ler arquivos de textos com a integra dos discursos
2. Retirar as palavras irrelevantes
3. Plotar uma nuvem de palavras

Vamos entender o código passo a passo e ver o resultado.

8.1 Aplicação de Nuvem de Palavras

Vamos novamente começar criando um arquivo com extensão py aqui vou chamar de (nuvem_de_palavras.py), criado no mesmo diretórios onde estão os arquivos de texto dos discursos para facilitar o entendimento e deixar o código mais simples, quando fizermos a leitura do arquivo passaremos apenas o nome do arquivo pois o caminho é o mesmo do arquivo criado (nuvem_de_palavras.py). Depois do arquivo criado podemos começar a escrever nossa aplicação no arquivo, vou comentar cada linha de comando para melhor entendimento de cada ação.

```
# importando bibliotecas
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import nltk

# declarando variavel tipo lista vazias
```

```
txt = []
# declarando variavel tipo lista com valores
discursos = ['congresso.txt', 'entrega_faixa.txt', 'davos.txt']

# loop for para selecionar cada discurso
for d in discursos:
    # criando uma lista das palavras a serem removidas
    lista = ['-', '"', '.', ',', '!', ':', ';', 'de', 'um', 'que',
             'ao', 'da', 'mais', 'em', 'mas', 'dos', 'das', 'acima', 'tudo',
             'na', 'do', 'e', 'a', 'o', 'as', 'os', 'como', 'com', 'esta',
             'no', 'nos', 'meu', 'minha', 'para', 'me', 'pelo', 'à', 'é',
             'ú', 'Com', 'O', 'Que', 'Um', 'Uma', 'aqui', 'aos', 'se', 'sua',
             'suas', 'às', '2005', '28', 'A', 'As', 'E', 'Os', 'sem',
             'isso', 'uma', 'por', 'pela', 'até']
    # realizando abertura e leitura de cada arquivo de texto
    texto = open(d, 'r').read()
    # convertendo a string em um lista de palavras
    palavras = nltk.word_tokenize(texto)

    # loop for em lista das palavras a serem removidas
    for x in lista:
        # loop para verificar todas as palavras do discurso
        while x in palavras:
            # removendo a lista de palavras
            palavras.remove(x)

    # juntando os discursos
    txt.extend(palavras)

# convertendo a lista de palavras em string
resultado = ', '.join(txt)
# paramentros de plotagem
wordcloud = WordCloud(max_font_size=200, width = 1024, height =
    ↳ 728).generate(resultado)
# tamanho da figura
```

Agora que finalizamos, vamos salvar o arquivo e executar pelo cmd do Windows na pasta onde esta salvo o arquivo com o comando:

Então chegamos ao resultado apresentado na Figura 71.

Figure 1

Frequência das palavras do Bolsonaro

The word cloud displays various terms, with the largest and most frequent words being 'Brasil', 'nossa', 'nosso', 'tudo', 'vamos', 'presidente', 'mundo', 'segurança', 'vida', 'grande', 'nós', 'governo', 'família', 'não', 'você', 'país', 'desafio', 'liberdade', 'trabalho', 'economia', 'política', 'reforma', 'estou', 'interesse', 'desenvolvimento', 'juntos', 'essa', 'brasileira'. Other visible words include 'compromisso', 'nacional', 'defesa', 'internacionais', 'precisa', 'esse', 'dia', 'tem', 'Minha', 'entre', 'será', 'mercado', 'valores', 'também', 'Estado', 'missão', 'Casa', 'história', 'queremos', 'momento', 'primeiro', 'confiança', 'bem', 'nação', 'liberdade', 'desafio', 'interesse', 'desenvolvimento', 'juntos', 'essa', 'brasileira', 'trabalho', 'economia', 'política', 'reforma', 'estou', 'interesse', 'desenvolvimento', 'juntos', 'essa', 'brasileira'.

Podemos identificar na nuvem de palavras gerada através do tamanho de cada palavras as mais repetidas, quanto maior a palavra mais vezes repetidas, e tirar diversas conclusões de acordo com a análise desejada.

Se lembrarmos do gráfico de barras temos as 12 palavras como as maiores aqui na nossa nuvem, o que realmente deveria ocorrer, mas como nesse exemplo não limitamos as palavras repetidas mais de 8 vezes apenas, então temos todas as palavras pronunciadas no discurso da mais repetida a menos repetida.

A nuvem de palavras tem uma análise visual mais amigável e mais interessante que o gráfico de palavras que digamos ser mais frio e direto, são análises similares da mesma informação e vai depender do que for proposto para que possamos apresentar a resposta.

8.2 Resumo do Capítulo

Neste capítulo aprendemos os seguintes itens:

- ◇ Criação e execução de uma aplicação Python de nuvem de palavras.

Se não conseguiu entender todos esse conceitos, releia o capítulo correspondente a sua dúvida, pois chegamos ao fim da nossa jornada, espero que tenha conseguindo conseguir compreender todos os capítulos. Se chegamos até aqui e conseguimos entender e praticar todos os assuntos abordados, podemos ter uma noção do poder do Python para análise de dados, agora e seguir praticando e buscar novos desafios.

Considerações finais

Se chegamos até aqui não é um fim! Mas sim um começo, agora que sabemos o potencial da linguagem Python e o que ela pode agregar na manipulação de dados principalmente, o nosso limite é a imaginação.

Tenha certeza que só vimos a pontinha do iceberg. O Python é um mundo a ser explorado, espero que tenha atingido o objetivo da iniciação de tirar você da inércia, do zero, te apresentando um pouquinho do que o Python é capaz de fazer!

Referências

FOUNDATION, P. S. *Python*. 2019. Disponível em: <<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>>. Acesso em: 03 Abril 2019. Nenhuma citação no texto.

ROBINSON, D. *The Incredible Growth of Python*. 2017. Disponível em: <<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>>. Acesso em: 28 Março 2019. Nenhuma citação no texto.