

Aula 4

Docupedia Export

Author:Goncalves Donathan (CtP/ETS)
Date:25-Aug-2023 21:05

Table of Contents

1 A Linguagem Python	4
1.1 Ambiente Python	4
1.2 Anaconda	6
1.3 Spyder	7
1.4 Seu primeiro programa em Python	12
1.5 Tipos de Variáveis	13
1.6 Tipos numéricos	13
1.7 Nomenclatura de variáveis	14
1.8 Entrada do Usuário	15
1.9 Operadores	16
1.10 Exercício 1 – Transformação Celsius-Fahrenheint	17
1.11 Correção	18
1.12 Strings	18
1.13 Manipulação de string	20
1.14 Fatiamento de strings:	21
1.15 Métodos para strings	23
1.16 Exercício 2 – Abreviação	25
1.17 Correção	25
1.18 Exercício 3 – Nome completo	26
1.19 Correção	27
1.20 Listas	27
1.21 Exercício 4 – Lista de 10 números	32
1.22 Correção	32
1.23 Tuplas	34
1.24 Exercício 5 – Top 10	34

1.25 Correção	35
1.26 Dicionários	35
1.27 Exercício 6 - Cardápio	36
1.28 Correção	38

1 A Linguagem Python

- Linguagem de alto nível;
- Sintaxe simples, fácil de aprender;
- Pode ser usada em Windows, Linux, FreeBSD, Mac OS, etc;
- Suporta programação procedural e orientada a objetos;
- Muitas bibliotecas disponíveis;
- Muito utilizada: Google, Microsoft, Dropbox, NASA, Lawrence Livermore Labs, Industrial Light & Magic;
- Site oficial: <http://www.python.org>

Implementada com um interpretador híbrido:

- o programa é traduzido para um “pseudo” código-máquina (byte-code)
- o byte-code é executado por um interpretador

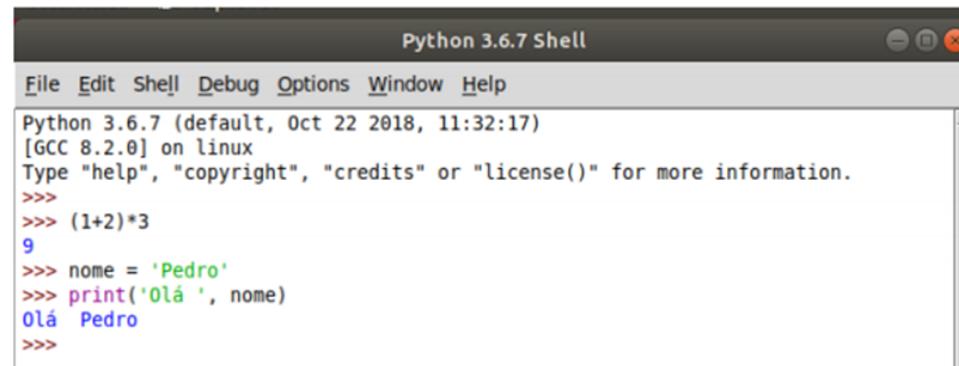
Vantagens: fácil de usar interativamente fácil testar e modificar componentes mais eficiente do que um interpretador clássico.

Desvantagem: não é tão eficiente como uma linguagem compilada (ex: C, C++ ou Fortran)

1.1 Ambiente Python

- Utilização interativa

Executando o interpretador de Python (IDLE) podemos escrever comandos e ver os resultados imediatamente.

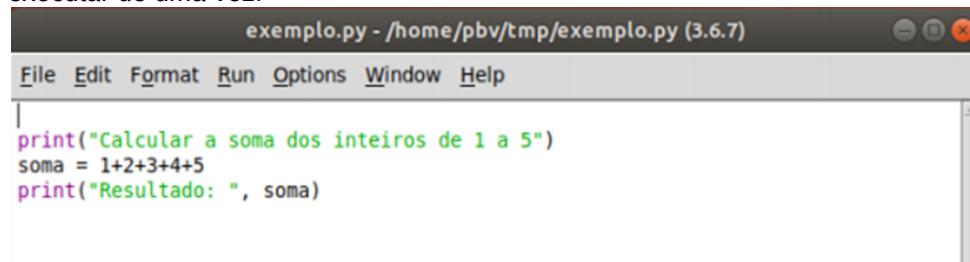


The screenshot shows the Python 3.6.7 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area displays the following Python session:

```
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> (1+2)*3
9
>>> nome = 'Pedro'
>>> print('Olá ', nome)
Olá Pedro
>>>
```

- Utilização com um script

Em alternativa, podemos compor um programa num ficheiro de texto (“script”) e executar de uma vez.



The screenshot shows a terminal window titled "exemplo.py - /home/pbv/tmp/exemplo.py (3.6.7)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the terminal is:

```
print("Calcular a soma dos inteiros de 1 a 5")
soma = 1+2+3+4+5
print("Resultado: ", soma)
```

O ambiente de desenvolvimento IDLE combina: uma janela interativa para comandos (“Python shell”) uma ou mais janelas para ficheiros (“scripts”)

Muitas funções e constantes matemáticas estão disponíveis no módulo *math*. Para usar devemos começar por importar o módulo.

>>> import math

Os nomes
das funções
começam
com prefixo
“math”:

>>>

math.sqrt(2)

raiz quadrada	sqrt
funções trigonométricas	sin, cos, tan
funções trigonométricas inversas	asin, acos, atan, atan2
exponencial e logaritmos	exp, log, log10
e, π	e, pi

1.4142135623730951

>>> math.pi

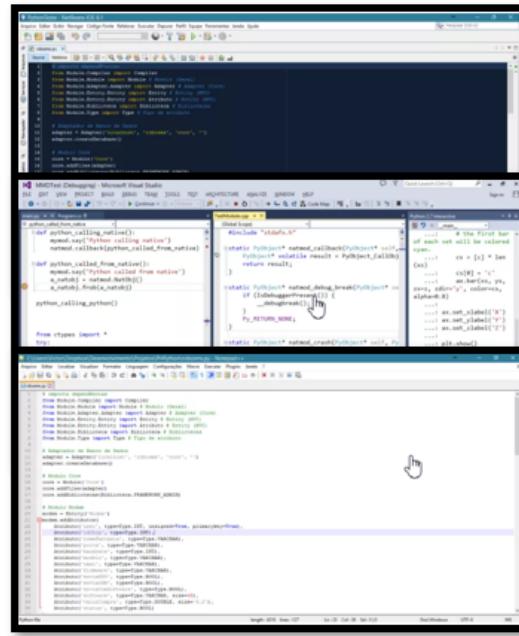
3.141592653589793

Para obter mais informação:

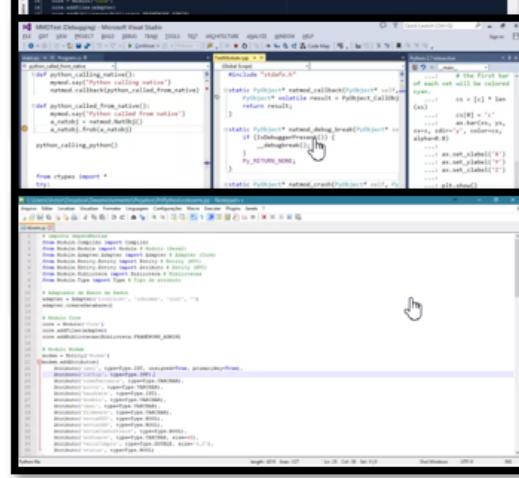
>>> help(math) informação geral

>>> help(math.log) específica sobre uma função

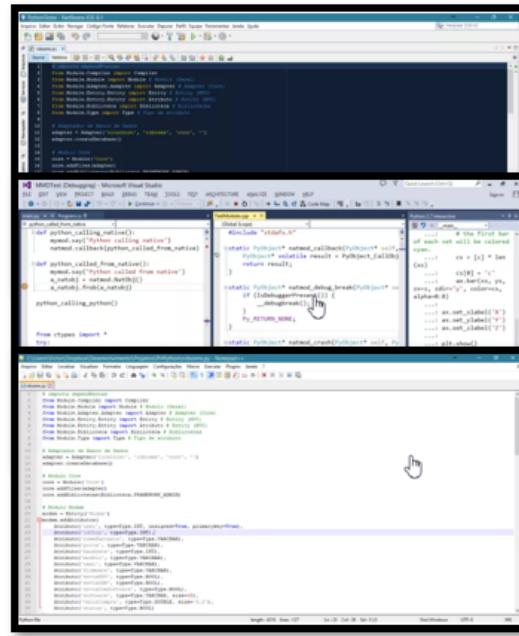
NetBeans



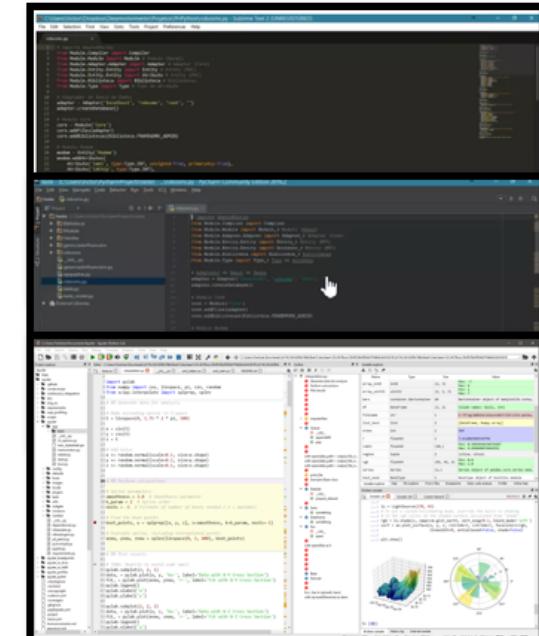
Visual Studio



Notepad++

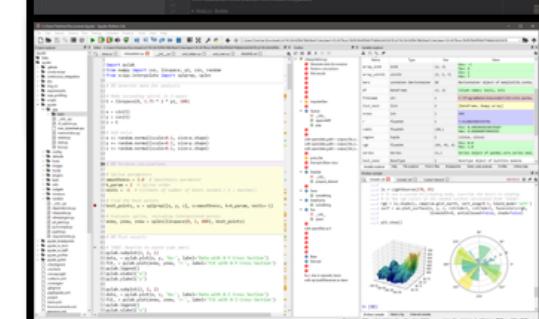


Sublime



PyCharm

Spyder



1.2 Anaconda

- Anaconda é uma distribuição livre e de código aberto das linguagens de programação Python e R. A plataforma visa simplificar o gerenciamento de pacotes. As versões de cada pacote também são gerenciadas pelo sistema.
- Os pacotes open source podem ser instalados individualmente a partir do repositório Anaconda com o comando *conda install* ou usando o comando *pip install* que é instalado com o Anaconda.
- Podemos ter diversos ambientes instalados em uma mesma máquina. Cada pacote terá sua versão do Python e seus pacotes armazenados separadamente. Ao ativarmos um ambiente virtual, os comandos *python*, *pip*, *conda etc* e o caminho de busca dos pacotes são substituídos pelos caminhos do ambiente virtual.

- Após a ativação do ambiente, qualquer comando Python usará as bibliotecas e o interpretador do ambiente virtual.
- A utilização de ambientes virtuais é comum quando desenvolvemos vários projetos paralelamente, de modo que cada projeto tem seu ambiente separado com somente os pacotes necessários instalados.
- É prático e evita inconvenientes como versões de pacotes incompatíveis ou diferentes para cada projeto.

Para criar um ambiente virtual utilizamos

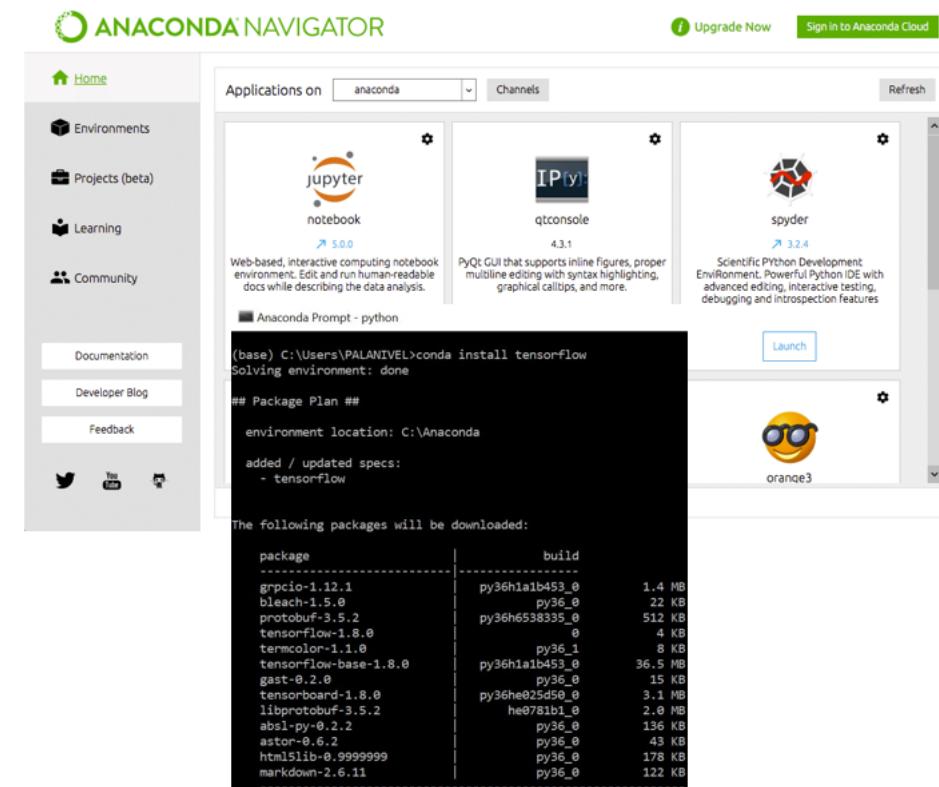
- `conda create -n nome`

Para ativá-lo

- `activate nome`

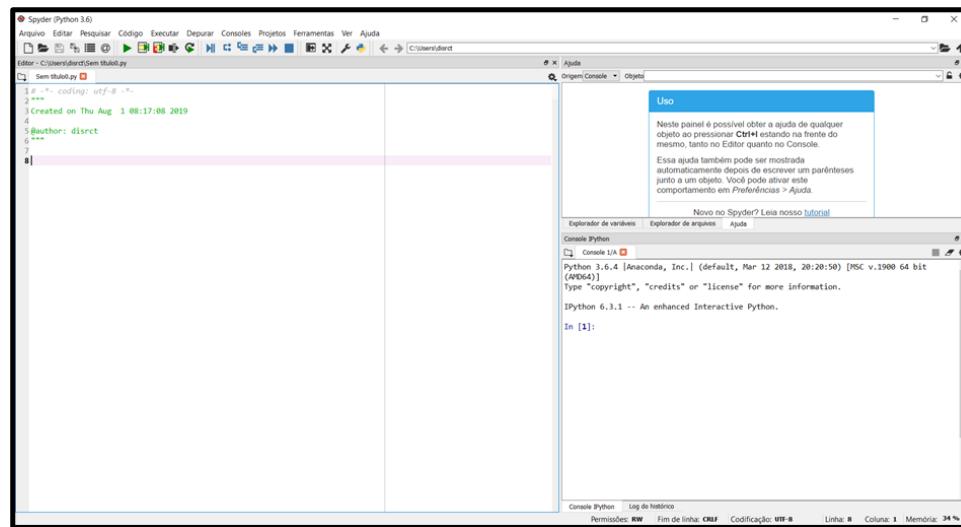
Também podemos listar todos os ambientes instalados utilizando o comando:

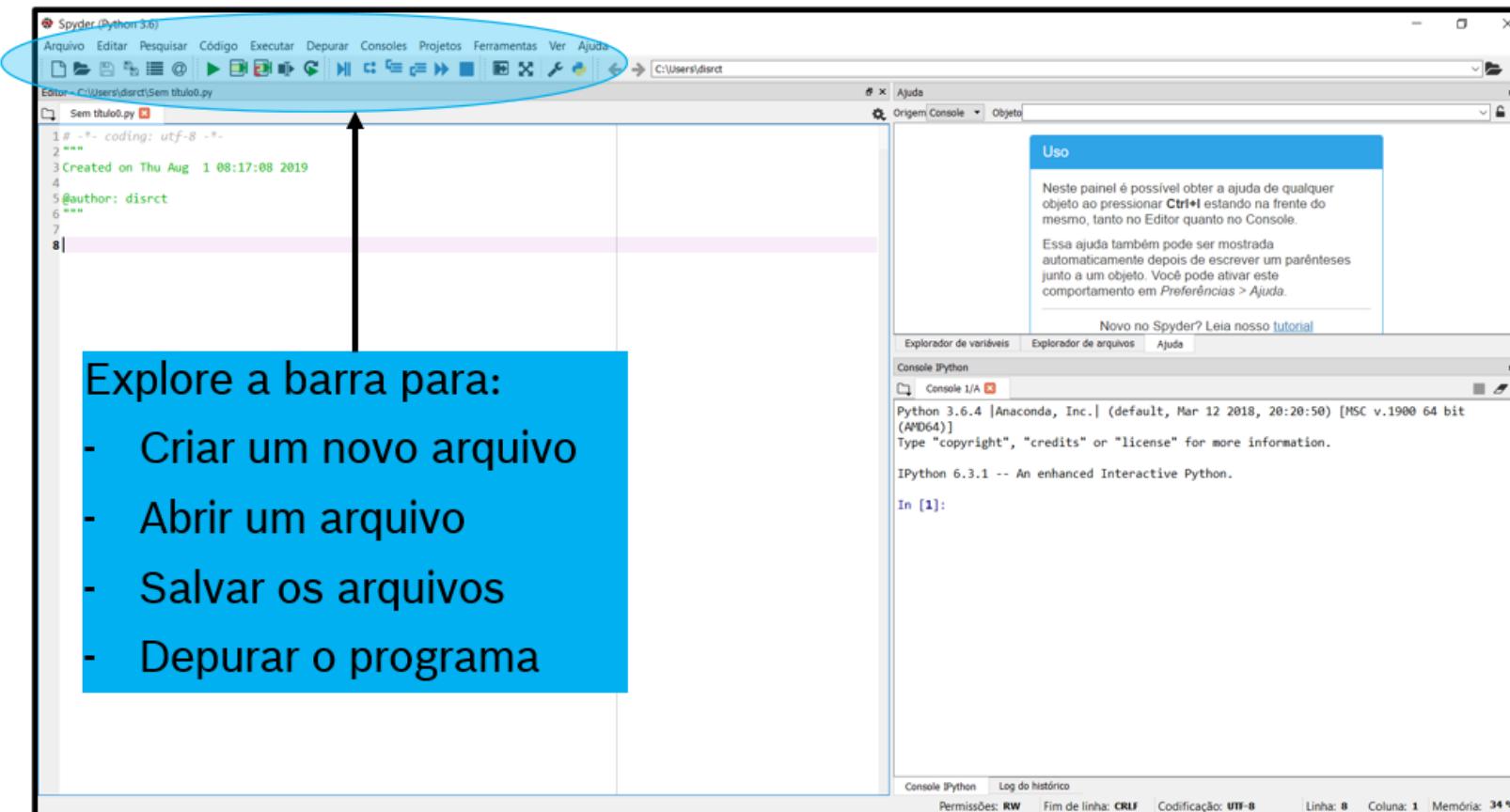
- `conda env list`



1.3 Spyder

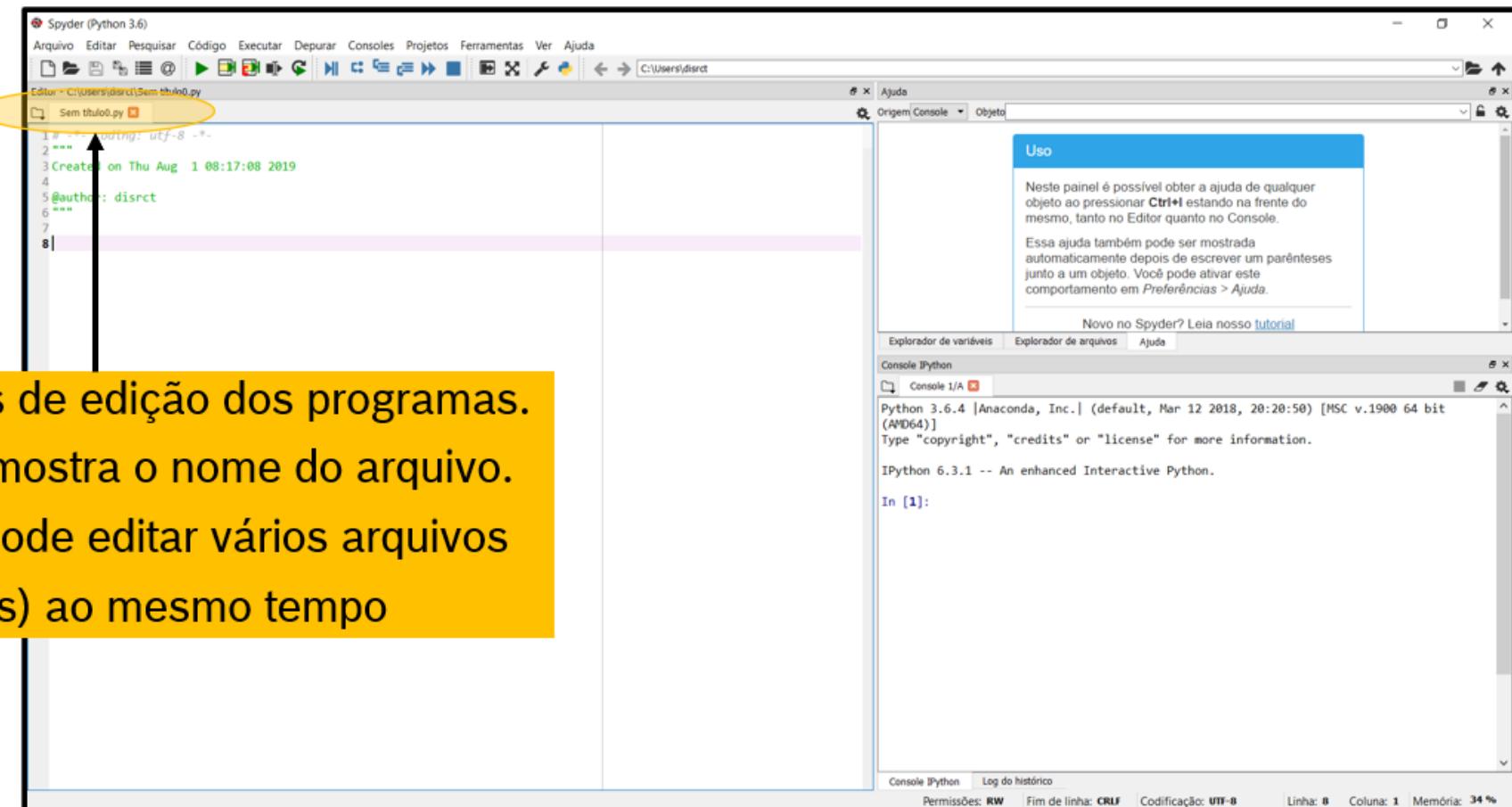
O Spyder é uma interface gráfica, semelhante ao Matlab, que permite a utilização de Python num ambiente interativo, facilitando a edição de scripts, teste, debugging e visualização gráfica;

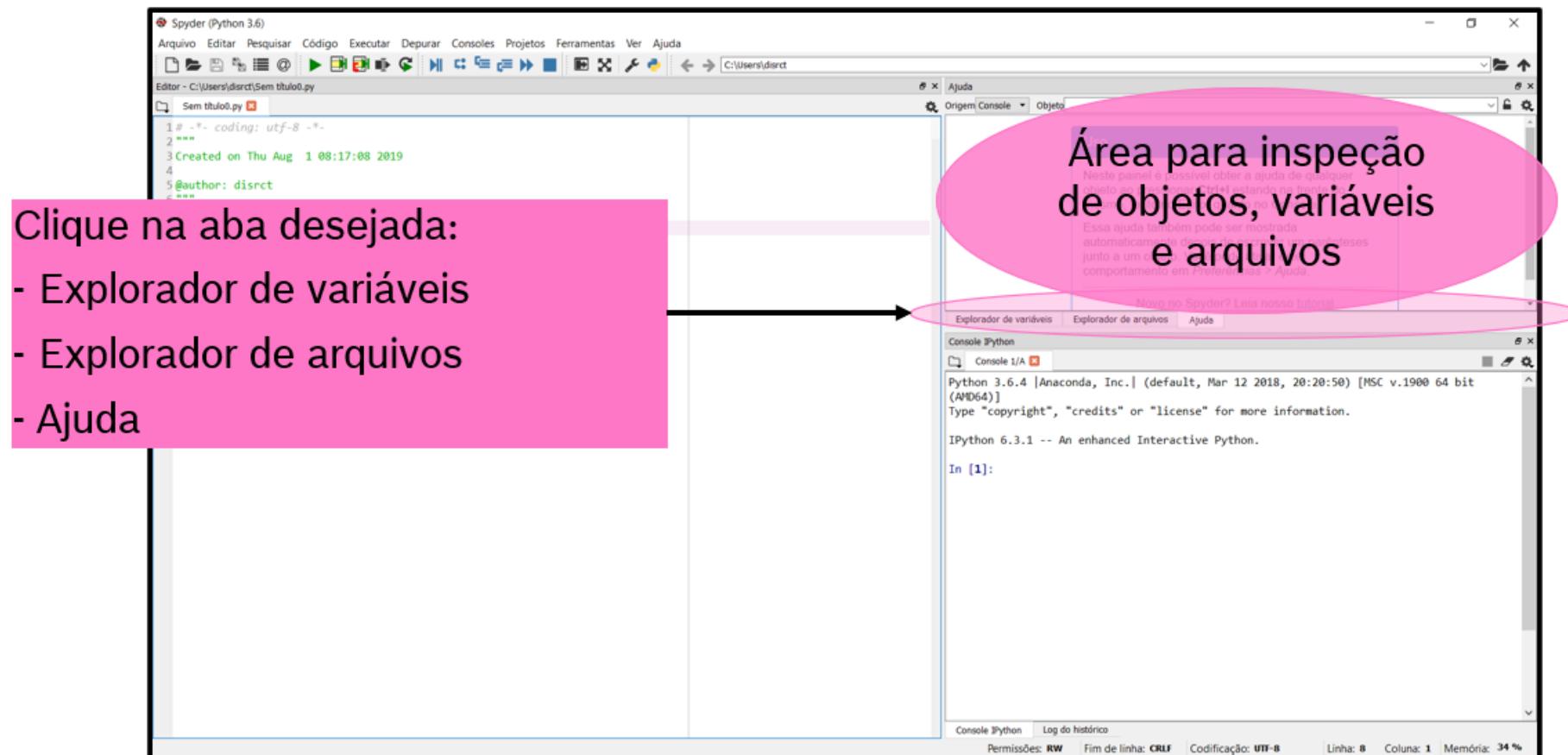


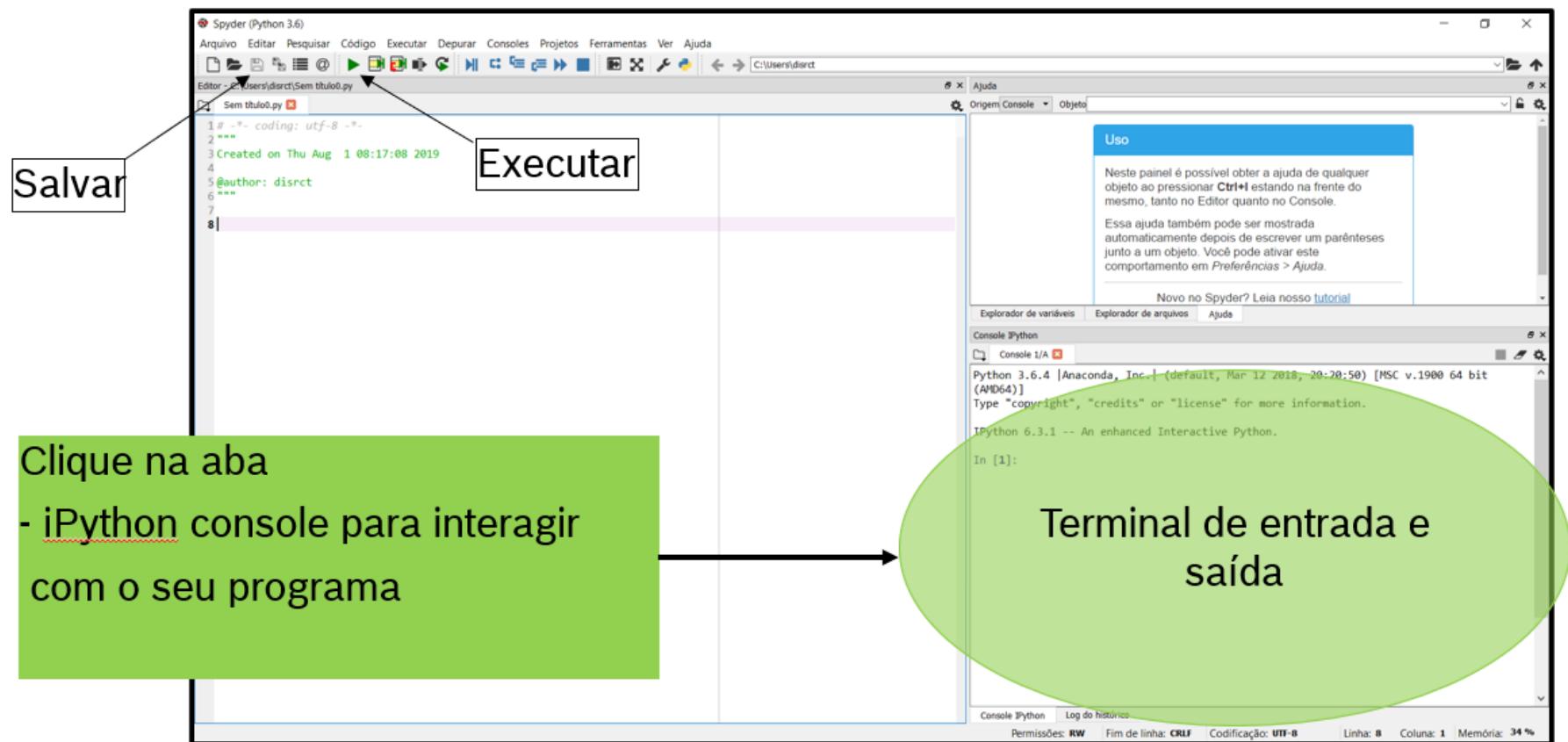


Explore a barra para:

- Criar um novo arquivo
- Abrir um arquivo
- Salvar os arquivos
- Depurar o programa







1.4 Seu primeiro programa em Python

- Toda linguagem de programação consiste em um diálogo entre o homem e uma máquina, então para começarmos a aprender a programar em Python, vamos ver como fazer o computador falar;
- O comando **print** faz com que o computador escreva o que quisermos na tela, por exemplo:

```
>>> print("Vamos programar!")  
Vamos programar!
```

- Note que após o comando, a frase foi escrita entre aspas e entre parênteses;
- Tudo que está dentro do parênteses está englobado pela função **print**, ou seja, será escrito pelo computador;
- As aspas são necessárias pois o objeto que criamos é uma string, um tipo de variável Python, elas podem ser aspas duplas ou simples;
- Se as aspas não forem colocadas, o programa interpretará como um nome de variável, e se essa variável não existir, o programa resultará em erro!

1.5 Tipos de Variáveis

Python possui vários tipos de dados padrão que são usados para definir as operações possíveis e o método de armazenamento para cada um deles. Existem cinco tipos de estrutura de dados padrão em Python:

- **Numérico** (Inteiro, Float, Complexo, Binário)
- **String**
- **Lista**
- **Tupla**
- **Dicionário**

1.6 Tipos numéricos

Os tipos numéricos são quatro:

- inteiro (int)
- ponto flutuante (float)
- booleano (bool)
- complexo (complex)

Em Python, diferente de algumas outras linguagens, não é necessário se fazer um **casting** ao criar uma variável, isso significa que não é preciso dizer ao programa qual será o tipo da variável, somente pelo valor atribuído o interpretador já sabe de que tipo se trata.

```
>>> a = 1          >>> a = 1.0        >>> a = True       >>> a = 4+3j        >>> a = "abc"  
>>> type(a)      >>> type(a)      >>> type(a)      >>> type(a)      >>> type(a)  
<class 'int'>  <class 'float'> <class 'bool'> <class 'complex'> <class 'str'>
```

E elas mudam de tipo dinamicamente. Por exemplo, a variável 'a':

```
>>> a = 1  
>>> type(a)  
<type 'int'>  
>>> a = 1.0  
>>> type(a)  
<type 'float'>  
>>>
```

1.7 Nomenclatura de variáveis

A escolha do nome de suas variáveis é um fator importante para a compreensão e manutenção de seu programa, para nomear uma variável deve-se tomar alguns cuidados, verifique o que é válido na tabela:

Nome	Comentário	Válido
a3	Embora contenha número, nome inicia com uma letra	sim
velocidade	Nome formado com letras	sim
Salario_medio	O “_” auxilia a leitura de nomes grandes	sim
salario	O “” é aceito até mesmo no início	sim

salarioMedio	camelCase. "TenteLerEssaFrase" e "tenteleressafrase"	sim
Salario medio	Nomes de variáveis não podem conter espaços em branco	não
5A	Nomes de variáveis não devem iniciar com números	não

Agora com nossas variáveis criadas e nomeadas, podemos mostrar seu valor ao usuário. Vamos utilizar novamente o comando **print**, mas agora seu parâmetro será o nome da nossa variável.

```
>>> variavel = 10  
>>> print(variavel)  
10
```

Podemos escrever um texto seguido de uma variável, basta separar os dois argumentos por uma vírgula.

```
>>> print("Minha variável é: ",variavel)  
Minha variável é: 10
```

1.8 Entrada do Usuário

A função **input** efetua a leitura de dados inseridos via teclado. Quando o usuário aperta a tecla **ENTER**, a função converte os dados inseridos para o formato **string** automaticamente:

```
>>> nome = input("Digite seu nome: ")  
Digite seu nome: João  
>>> print(nome)  
João
```

Para efetuar a leitura de outros tipos de variáveis (float, int, ...), é necessário converter a entrada do usuário para o tipo desejado.

```
>>> inteiro = int(input("Digite sua idade: "))  
Digite sua idade: 100  
>>> print(inteiro)  
100
```

```
>>> booleano = bool(input("Digite True ou False: "))  
Digite True ou False: True  
>>> print(booleano)  
True
```

1.9 Operadores

Para manipular nossas variáveis, utilizamos os operadores, podendo ser operadores aritméticos ou de comparação, neste caso o resultado é sempre do tipo booleano.

Operadores Aritméticos

Operador	Descrição	Exemplo
+	Soma	5+5=10
-	Subtração	7-2=5

Operadores de comparação

Operador	Descrição	Exemplo
<	Menor que	5 < 10 = True
>	Maior que	7 > 10 = False

*	Multiplicação	$2*2=4$
/	Divisão	$4/2=2$
%	Resto da divisão	$10\%3=1$
**	Potência	$4^{**}2=16$

\leq	Menor ou igual	$1 \leq 1 = \text{True}$
\geq	Maior ou igual	$3 \geq 4 = \text{False}$
\equiv	Igual	$2 \equiv 2 = \text{True}$
\neq	Diferente	$2 \neq 2 = \text{False}$

1.10 Exercício 1 – Transformação Celsius-Fahrenheint

- Peça para o usuário inserir um valor de temperatura em graus Celsius;
- Faça a transformação para Fahrenheint;
- Mostre na tela o novo valor.

Output:

```
Temperatura em Celsius: 25
Temperatura em Fahrenheint: 77.0
```



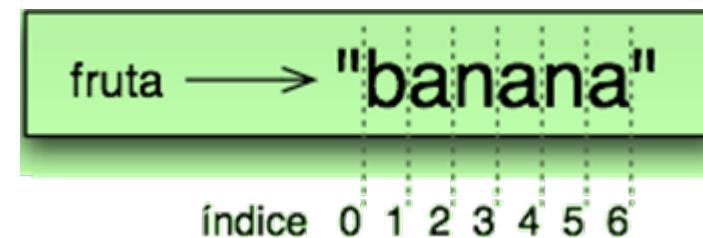
1.11 Correção

```
tempCelsius = int(input("Temperatura em Celsius: "))
tempFahr = (9*tempCelsius)/5 + 32
print("Temperatura em Fahrenheint: ", tempFahr)
```

1.12 Strings

Strings são qualitativamente diferentes dos outros dois tipos porque são feitas de pedaços menores - caracteres. O operador colchete seleciona um único caractere de uma string. Por exemplo:

```
>>> fruta = 'banana'
>>> letra = fruta[0]
>>> print(letra)
b
```



A expressão `fruta[0]` seleciona o caractere número 0 da variável `fruta`, isto é "b". O método `len` retorna o número de caracteres de uma string.

```
>>> len(fruta)  
6
```

O operador + também funciona com strings de uma maneira diferente dos números. Ele funciona concatenando strings, ou seja, juntando duas strings:

```
>>> texto1 = "bana"  
>>> texto2 = "na"  
>>> print(texto1 + texto2)  
banana
```

O operador * também funciona com strings, multiplicando seu conteúdo por um inteiro.

```
>>> print(texto2 * 3)  
nanana
```

Se você tentar colocar aspas em sua string, o programa interpretará que você quer ‘terminar’ a string, para fazer isso basta colocar uma \' antes.

```
>>> print('McDonald's')
      File "<stdin>", line 1
          print('McDonald's')
                  ^

```

```
>>> print('McDonald\'s')
McDonald's
```

Para criar strings com mais de uma linha utilizamos aspas triplas envolvendo o texto desejado.
\n serve como quebra de linha, para printar o caracter \' é preciso escrever '\\'.

```
>>> minha_string = '''linha1
... linha2
... linha3
...
...
>>> print(minha_string)
linha1
linha2
linha3
```

```
>>> print('Quero usar\n o caracter \\')
Quero usar
o caracter \
```

1.13 Manipulação de string

O método **split** separa uma *string* conforme um delimitador, o delimitador padrão é o espaço ''.

```
>>> s = "n o m-e"
>>> s.split(' ')
['n', 'o', 'm-e']
>>> s.split('-')
['n o m', 'e']
```

O método **join** junta cada item da *string* com um delimitador especificado. É o inverso do método `split()`.

```
>>> ','.join("abc")
'a,b,c'
>>> '-'.join(['robert', 'bosch', 'curitiba'])
'robert-bosch-curitiba'
```

1.14 Fatiamento de strings:

O fatiamento é uma ferramenta usada para extrair apenas uma parte dos elementos de uma string.

- `Nome_string[Limite_superior : Limite_inferior]`

```
>>> s = 'python'  
>>> print(s[1:4])  
yth  
>>> print(s[2:])  
thon  
>>> print(s[:])  
python  
>>> print(s[:3])  
pyt
```

Se a nossa string é uma palavra com as letras minúsculas, podemos utilizar o método **upper** para printar a mesma palavra em letras maiúsculas.

```
>>> s = 'python'  
>>> print(s)  
python  
>>> print(s.upper())  
PYTHON
```

Para fazer o contrário utilizamos o método **lower**.

```
>>> s = 'PYTHON'  
>>> print(s)  
PYTHON  
>>> print(s.lower())  
python
```

1.15 Métodos para strings

Método	Descrição
len()	Retorna o tamanho da string
capitalize()	Retorna a string com a primeira letra em maiúscula
count()	Informa quantas vezes um caractere aparece na string
startswith()	Verifica se a string inicia com uma determinada sequência
endswith()	Verifica se a string termina com uma determinada sequência
isalnum()	Verifica se a string possui um conteúdo alfanumérico
isalpha()	Verifica se possui apenas conteúdo alfabético

islower()	Verifica se todas as letras da string são minúsculas
isupper()	Verifica se todas as letras da string são maiúsculas
lower()	Retorna uma cópia da string trocando todas as letras para minúsculo
upper()	Retorna uma cópia da string trocando todas as letras para maiúsculo
swapcase()	Inverte o conteúdo da string
title()	Converte para maiúsculo todas as primeiras letras de cada palavra da string
split()	Transforma a string em uma lista, utilizando os espaços como referência
replace(S1,S2)	Substitui na string o trecho S1 pelo trecho S2
find()	Retorna o índice da primeira ocorrência de um determinado caractere na string. Se o caractere não estiver na string retorna -1.
ljust()	Ajusta a string para um tamanho mínimo, acrescentando espaços à direita.
rjust()	Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda.
center()	Ajusta a string para um tamanho mínimo, acrescentando espaços à direita e à esquerda.
lstrip()	Remove todos os espaços em branco do lado esquerdo da string
rstrip()	Remove todos os espaços em branco do lado direito da string
strip()	Remove todos os espaços em branco da string

1.16 Exercício 2 – Abreviação

- Peça para o usuário inserir um nome;
- Mostre esse nome somente com as 3 primeiras letras;
- As 3 letras devem estar em maiúsculo;

Output:

```
Digite seu nome: Roberto  
ROB
```



1.17 Correção

```
nome = input("Digite seu nome: ")  
abrev = nome[:3]  
print(abrev.upper())
```

1.18 Exercício 3 – Nome completo

- Peça para o usuário inserir seu nome completo com todas os caracteres minúsculos;
- Mostre o nome do usuário com todas as primeiras letras em maiúsculo;
- Print a quantos caracteres o nome do usuário possui

DESAFIO: descubra como colocar a variável no meio do print



Output

```
Insira seu nome completo com letras minúsculas: joao da silva neto  
Joao Da Silva Neto  
Seu nome possui 15 caracteres
```

1.19 Correção

```
nome = input("Insira seu nome completo com letras minúsculas: ")

nomeMaiusculo = nome.title()

nomeContagem = nome.replace(" ", '')

print(nomeMaiusculo, "\nSeu nome possui {} caracteres".format(len(nomeContagem)))
```

1.20 Listas

Uma lista é um conjunto sequencial de valores, onde cada valor é identificado por um índice. Os valores que compõem uma lista são chamados **elementos**. Listas são similares a strings, que são conjuntos ordenados de caracteres, com a diferença que os elementos de uma lista podem possuir qualquer tipo.
Para criar uma nova lista basta envolver os elementos em colchetes [].

```
>>> lista = [10, 1.5, 'True', 'string']
>>> print(lista)
[10, 1.5, 'True', 'string']
```

Uma lista dentro de outra lista é dita estar **aninhada**.

A sintaxe para acessar os elementos de uma lista é a mesma que a sintaxe para acessar os caracteres de uma string. Lembre-se que os índices iniciam em 0.

```
>>> print(lista[2])  
True
```

A função **len** devolve o comprimento de uma lista.

```
>>> print(len(lista))  
4
```

A função **sort** organiza os fatores da lista em ordem crescente alfabética ou numérica, neste caso os elementos devem ser do mesmo tipo.

```
>>> lista = [9,2,6,3,5]  
>>> lista.sort()  
>>> print(lista)  
[2, 3, 5, 6, 9]
```

Para alterar um elemento de sua lista , basta fazer uma atribuição de valor através do índice, por exemplo:

```
>>> lista[0] = 'a'  
>>> print(lista)  
['a', 3, 5, 6, 9]
```

O operador lógico **in** verifica se um elemento é membro de uma sequência.

```
>>> lista = [2,7,9,5]  
>>> 9 in lista  
True  
>>> 1 in lista  
False
```

O operador **+** concatena listas:

```
>>> lista2 = [8,3,6]
>>> print(lista + lista2)
[2, 7, 9, 5, 8, 3, 6]
```

Similarmente, o operador * repete uma lista um número dado de vezes:

```
>>> print(lista * 3)
[2, 7, 9, 5, 2, 7, 9, 5, 2, 7, 9, 5]
```

Existem outros métodos que podemos aplicar às listas. Faça o teste com as funções da tabela abaixo.

Função	Descrição
min()	Retorna o menor valor da lista
max()	Retorna o maior valor da lista
sum()	Retorna a soma dos elementos da lista
append()	Adiciona um novo valor no final da lista
extend()	Insere uma lista no final de outra

del()	Remove um elemento da lista conforme o índice
reverse()	Inverte os elementos da lista

Se queremos modificar uma lista e também manter uma cópia da original, precisamos ter condições de fazer uma cópia da própria lista, não apenas uma referência. Este processo é algumas vezes chamado **clonagem**, para evitar a ambiguidade da palavra “cópia”.

A maneira mais fácil de clonar uma lista é utilizar o operador “:”

Uma vez que variáveis se referem a objetos, se atribuímos uma variável a uma outra, ambas as variáveis se referem ao mesmo objeto. Uma vez que a lista possui dois nomes diferentes, a e b, dizemos que ela está “apelidada” (aliased). Mudanças feitas em um apelido afetam o outro nome, embora este comportamento possa ser útil, ele é às vezes inesperado e indesejado. Em geral, é mais seguro evitar os apelidos quando você está trabalhando com objetos mutáveis.

apelidada

```
>>> a = [1,2,3]
>>> b = a
>>> a[0] = 5
>>> print(b)
[5, 2, 3]
```

clonada

```
>>> a = [1,2,3]
>>> b = a[:]
>>> a[0] = 5
>>> print(b)
[1, 2, 3]
```

1.21 Exercício 4 – Lista de 10 números

- Crie uma lista com 10 valores;
- Peça para o usuário escolher os 10 valores;
- Mostre o tamanho da lista;
- Mostre a lista ordenada;

VALOR 1: 12

VALOR 2: 15

VALOR 3: 6

VALOR 4: 5

VALOR 5: 8

VALOR 6: 7

VALOR 7: 15

VALOR 8: 2

VALOR 9: 10

VALOR 10: 3

TAMANHO DA LISTA: 10

[2, 3, 5, 6, 7, 8, 10, 12, 15, 15]

1.22 Correção

```
lista = [0,0,0,0,0,0,0,0,0,0]
lista[0] = int(input("VALOR 1: "))
lista[1] = int(input("VALOR 2: "))
lista[2] = int(input("VALOR 3: "))
lista[3] = int(input("VALOR 4: "))
lista[4] = int(input("VALOR 5: "))
lista[5] = int(input("VALOR 6: "))
lista[6] = int(input("VALOR 7: "))
lista[7] = int(input("VALOR 8: "))
lista[8] = int(input("VALOR 9: "))
lista[9] = int(input("VALOR 10: "))
tamanho = len(lista)
lista.sort()
print("Tamanho da lista: ",tamanho)
print(lista)
```

1.23 Tuplas

Em linhas gerais, uma **Tupla** é uma Lista imutável. O que diferencia a Estrutura de Dados Lista da Estrutura de Dados Tupla é que a primeira pode ter elementos adicionados a qualquer momento, enquanto que a segunda estrutura, após definida, não permite a adição ou remoção de elementos. Uma Lista, tem seus elementos delimitados por colchetes, enquanto que a Tupla, tem seus elementos delimitados por parêntesis.

A ordem dos elementos numa Tupla será a ordem na qual estes foram definidos, ou seja, não é possível ordenar em tempo de execução os elementos.

O primeiro elemento de uma Tupla também possui índice igual a 0 e o último índice igual a -1. Assim o acesso a elementos, bem como o fatiamento funciona da mesma forma como já estudado no fatiamento de Listas.

```
>>> t = ('a', 'b', 'c', 'd', 'e')  
>>> t[0]  
'a'
```

```
>>> t[0] = 'A'  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'tuple' object does not support item assignment
```

```
>>> t = ('A',) + t[1:]  
>>> t  
('A', 'b', 'c', 'd', 'e')
```

1.24 Exercício 5 – Top 10

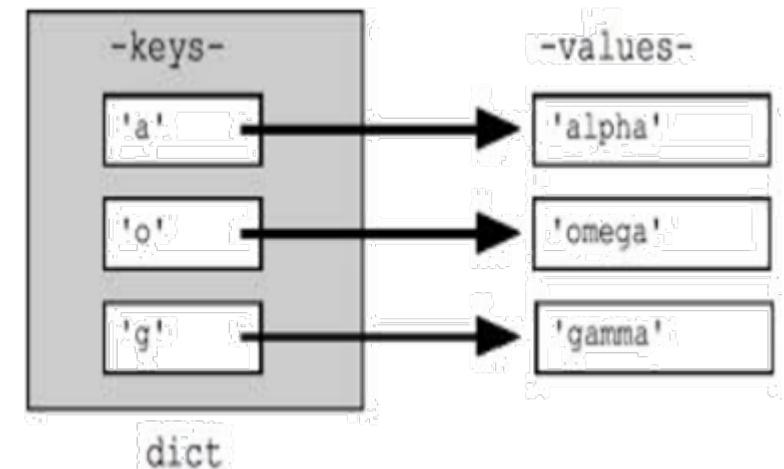
- Crie uma tupla com as 10 pessoas mais ricas do mundo em 2019 e mostre-a ao usuário;
- Mostre os 3 mais ricos;
- Mostre quem é o mais rico.

Output

```
Os 10 mais ricos do mundo são: ('Jeff Bezos', 'Bill Gates', 'Warren Buffet', 'Bernard Arnault', 'Amâncio Ortega', 'Larry Ellison', 'Mark Zuckerberg', 'Miche. Bloomberg', 'Larry Page')  
Os 3 mais ricos do mundo são: ('Jeff Bezos', 'Bill Gates', 'Warren Buffet')  
O mais rico do mundo é: Jeff Bezos
```

1.25 Correção

```
ricos = ('Jeff Bezos', 'Bill Gates', 'Warren Buffet',
         'Bernard Arnault', 'Carlos Slim', 'Amâncio Ortega',
         'Larry Ellison', 'Mark Zuckerberg',
         'Michel Bloomberg', 'Larry Page')
print('Os 10 mais ricos do mundo são: ', ricos)
print('Os 3 mais ricos do mundo são: ', ricos[:3])
print('O mais rico do mundo é: ', ricos[0])
```



1.26 Dicionários

Estrutura de dados que implementa mapeamentos entre uma **chave** (key) e algum **valor** (value).

Mapeamentos também são chamados de pares chave-valor;

A chave funciona como um índice para acessar o conteúdo;

Conteúdo pode ser qualquer tipo de dado, inclusive outro dicionário;

A função **dict** cria um novo dicionário sem itens. Como **dict** é o nome de uma função integrada, você deve evitar usá-lo como nome de variável.

As chaves {} representam um dicionário vazio. Para acrescentar itens ao dicionário, você pode usar colchetes.

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> eng2sp
{'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

```
>>> dicionario = dict()
>>> dicionario
{}
```

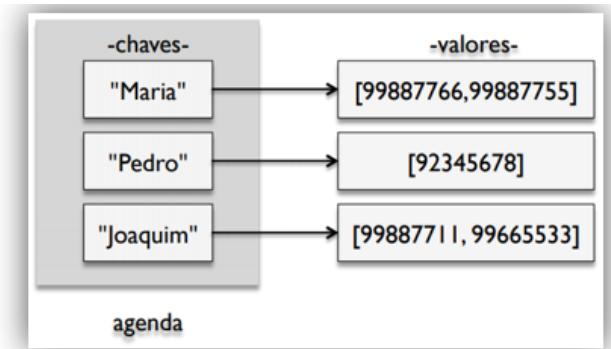
Diferentemente de listas, atribuir a um elemento de um dicionário não requer que a posição exista previamente (isso ocorre porque não se trata de posição, e sim de valor da chave!)

Com dicionários, ocorre o mesmo efeito que ocorre com cópia de listas – o que é copiado é o endereço de memória, e portanto, alterações nas cópias são refletidas umas nas outras.

Resumindo, o dicionário é como uma lista, mas em que os itens não são acessados por sua posição, e sim pelo nome de sua chave.

```
>>> lista = []
>>> lista[10] = 5 # ERRO!
>>> dicionario = {}
>>> dicionario[10] = 5 # OK!
>>> dicionario
{10: 5}
```

```
>>> d1 = {"Catarina":5}
>>> d2 = d1
>>> d1["Jonas"] = 20
>>> d2
{"Catarina": 5, "Jonas": 20}
```



1.27 Exercício 6 - Cardápio

- Crie um dicionário com o cardápio de um restaurante, contendo o nome e o preço de cada item;
- Pergunte ao usuário qual comida e bebida ele deseja;
- Print o valor total da compra.

Output

Menu Fastfood

```
{'Hamburguer': 10, 'Hotdog': 6.5, 'Salada': 4,  
'Suco': 4, 'Refrigerante': 4.5, 'Água': 2}
```

Digite a comida que você deseja: Salada

Digite a bebida que você deseja: Suco

O valor total é de: 8

1.28 Correção

```
print('Menu Fastfood')

menu = {'Hamburguer': 10,
        'Hotdog': 6.5,
        'Salada': 4,
        'Suco': 4,
        'Refrigerante': 4.5,
        'Água': 2
       }
print(menu)
comida = input('Digite a comida que você deseja: ')
bebida = input('Digite a bebida que você deseja: ')
total = menu[comida] + menu[bebida]
print('O valor total é de: ',total)
```

