Python

Introdução à Linguagem

Prof. Luiz Fernando Carvalho

luizfcarvalho@utfpr.edu.br

Sumário

- Visão geral da linguagem Python;
- Características da linguagem;
- Entrada e saída de dados;
- Estruturas de controle de seleção e repetição;
- Operadores aritméticos, lógicos e relacionais;
- Estrutura de dados: lista, tupla e dicionário;
- Definição de funções;
- Tratamento de exceções;
- Manipulação de arquivos;
- Introdução à *list comprehension*;
- Bibliotecas padrão;
- Visão geral das bibliotecas usadas em ciência de dados: numpy, pandas, scikit-lean e matplotlib;
- Exemplos de uso.

Algoritmos Computacionais

Lógica de programação

Programador formula a solução

Linguagem de Programação

Aplicação/Programa

Informações sobre o curso

- O que esse curso é:
  - Conteúdo motivacional para aprender a linguagem;
  - Exposição dos conceitos e funcionalidades básicas da linguagem;
- O que esse curso não é?
  - Uma revisão sistemática de todos os recursos da linguagem;
- Façam perguntas e interajam!
- Duração de aproximadamente 8 horas;

Motivação

- Porque estudar Python?
  - Diversas empresas usam: Google, Facebook, Spotify, Instagram, Netflix, Reddit, Dropbox...
  - Fácil de aprender;
  - Amplamente utilizado e em crescimento
  - Comunidade ativa;
  - Multiplataforma;

Visão geral da linguagem

Variáveis

- Variáveis são criadas no momento em que valores são associados a elas

```
>>> x = 5
>>> y = 2.0
>>> curso = “Python”
```

- Nenhum tipo foi definido na criação das variáveis...
  - Seus tipos foram definidos implicitamente!

```
>>> type(x)
<class ‘int’>

>>> type(y)
<class ‘float’>

>>> type(curso)
<class ‘str’>
```

Variáveis

- Os tipos básicos de variáveis incluem:
  - int: 5, -128, 43
  - float: 3.14, -45.9
  - str: ‘Teste’, ‘Computacao’
  - bool: True, False

E quanto ao tipo de dados caractere?

```
>>> 5 / 2
2.5
>>> 5 // 2
2
>>> 5 % 2
1
>>> 5 ** 2
25
>>> 5.0 // 2
2.0
>>> (5 + 3) * (-4 + 3)
-8
```

Operadores Aritméticos	
( )	Parênteses
**	Exponenciação
*	Multiplicação
/	Divisão
//	Divisão inteira
%	Módulo
+	Adição
-	Subtração

Variáveis

- Variáveis são criadas no momento em que valores são associados a elas

```
>>> x = 5
>>> y = 2.0
>>> curso = “Python”
```

```
>>> x + y
```

```
7.0
```

```
>>> x + curso
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python possui tipagem **FORTE**, portanto, operações envolvendo tipos diferentes não são resolvidas automaticamente

Variáveis

- Portanto, Python é uma linguagem:
  - **Dinâmica:** Variáveis são definidas no momento da execução
  - **Com tipagem forte:** variáveis tem tipos e o programador pode realizar *casting*

```
>>> versao = 3.0
>>> data = ‘ 30/10/2019’
>>> ‘Curso de Python ‘ + str(versao) + data
```

```
‘Curso de Python 3.0 30/10/2019’
```

Variáveis

Pode-se usar aspas simples ou duplas para definir strings.

- Strings
  - Descobrir o tamanho

```
>>> texto = 'Curso de Python!'
>>> len(texto)
16
```

  - Acessar cada caractere

```
>>> texto[0]
'C'
```

  - Modificar um ou mais caracteres da string

```
>>> texto[8] = 'i'
'TypeError: 'str' object does not support item assignment'
```

  - Modificar um ou mais caracteres da *string* (corretamente)

```
>>> texto.replace('Python', 'Strings')
'Curso de Strings!'
```

Variáveis

- Strings
  - Separando strings

```
>>> texto.split(‘ ’)
[‘Curso’, ‘de’, ‘Python!’]
```

  - Acha a posição da primeira ocorrência de um caractere ou substring

```
>>> texto.find(‘o’)
4
```

  - Concatena strings

```
>>> a = ‘Hello ’
>>> b = ‘World’
>>> a + b
‘Hello World’
```

  - Multiplica uma string

```
>>> a * 4
‘Hello Hello Hello Hello ’
```

Variáveis

- Strings (outras funções)
  - string.count(<caractere>)
    - conta a quantidade de vezes que o caractere ou substring apareceram na string
  - string.upper()
    - Deixa todos os caracteres maiúsculo
  - string.lower()
    - Deixa todos os caracteres minúsculo
  - string.isdigit()
    - Retorna True se todos os caracteres forem números, caso contrário, False
  - string.isalpha()
    - Retorna True se todos os caracteres forem letras

O que vimos até aqui...

- Python é uma linguagem dinâmica e fortemente tipada;
- Definição de variáveis e alguns tipos básicos;
- Operações matemáticas básicas;

Conceitos Básicos: Entrada e Saída

- Toda informação recebida pela função input() é do tipo str

```
x = input('Digite um numero inteiro') #O usuário informou 10
'10'
```

- Fica a critério do programador fazer a conversão

```
x = int(input('Digite um numero inteiro')) #O usuário informou 10
10
```

Tomar cuidado com o tipos de dados fornecidos errado. Por exemplo int('Hello World')

Conceitos Básicos: Entrada e Saída

- A função print() é usada para saída de dados

```
print('Curso de Python')
```
```
'Curso de Python'
```

- Imprimindo valores de variáveis

```
x = 2
print(x)
```
```
2
```

- Imprimindo texto e valores de variáveis

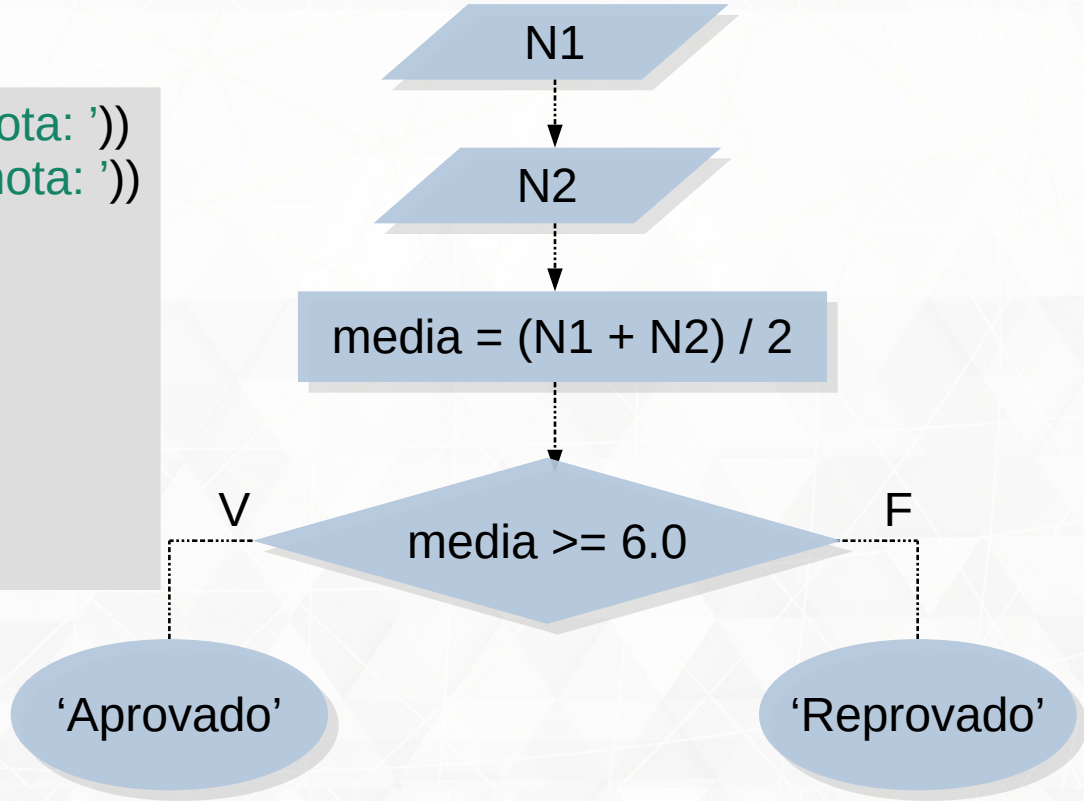
```
r = 2
area = 3.14 * r ** 2
print('A area e', area, 'cm ao quadrado')
```
```
A area e 12.56 cm ao quadrado
```

Alternativa: print(f'A area é {area}cm ao quadrado')

Conceitos Básicos: if-else

- Nem sempre nosso programa vai executar os comandos sequencialmente

```
N1 = float(input('Informe a primeira nota: '))
N2 = float(input('Informe a segunda nota: '))

media = (N1 + N2) / 2

if media >= 6.0:
    print('Aprovado')
else:
    print('Reprovado')
```



Quais as principais diferenças na sintaxe para outras linguagens de programação?

Conceitos Básicos: if-else

- Nem sempre nosso programa vai executar os comandos sequencialmente
  - Python usa espaços e ‘:’ como delimitadores;
  - Indentação é **obrigatória!**

```
N1 = float(input(‘Informe a primeira nota: ’))
N2 = float(input(‘Informe a segunda nota: ’))

media = (N1 + N2) / 2

if media >= 6.0:
    print(‘Aprovado’)
else:
    print(‘Reprovado’)
```

Conceitos Básicos: if-else

- Nem sempre nosso programa vai executar os comandos sequencialmente
  - Python usa espaços e ‘:’ como delimitadores;
  - Indentação é **obrigatória!**

Python

```
if condição:
    pass
elif condição2:
    pass
else:
    pass
```

C

```
if(condição){
    comandos;
}
else if(condição2){
    comandos;
}
else{
    comandos;
}
```

Conceitos Básicos: operadores

Operadores relacionais

```
>
<
>=
<=
==
!=
```

Operadores lógicos

```
and
or
not
```

```
nota = float(input('Informe sua nota'))

if nota < 0 or nota > 10:
    print('Valor invalido para nota')
    print('O programa sera encerrado')
```

Conceitos básicos: while

- Frequentemente temos que repetir um bloco de instrução
  - Python só tem while e for
- O comando `while` é executado enquanto sua condição é verdadeira

```
while condicao:
    pass
```

```
x = 0
while x < 10:
    print(x)
    x = x + 1
```

Conceitos básicos: for

- Frequentemente temos que repetir um bloco de instrução
- O comando `for` é executado enquanto sua condição é verdadeira

```
for valor in variavel:
    pass
```

```
for x in range(0, 10):
    print(x)
```

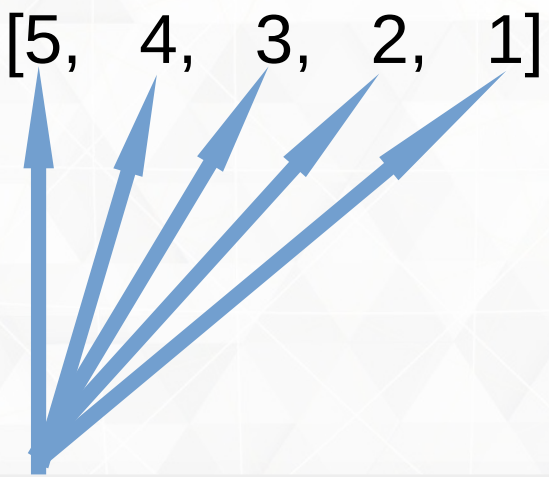
A função range(inicio, fim, [intervalo]) retorna uma série numérica de acordo com os parâmetros passados para ela

A série numérica inicia no valor de início e finaliza em fim-1. O parâmetro intervalo é opcional. Quando omitido, significa +1

Conceitos básicos: for

- Frequentemente temos que repetir um bloco de instrução
- O comando `for` é executado enquanto sua condição é verdadeira

```
range(5, 0, -1)
```




```
for x in range(5, 0, -1):
    print(x)
```

O que vimos até aqui...

- Python é uma linguagem dinâmica e fortemente tipada;
- Definição de variáveis e alguns tipos básicos;
- Operações matemáticas básicas;
- Entrada e saída (input, print)
- Blocos condicionais: if, elif, else
- Laços de repetição: while, for

Estrutura de dados

- Python possui estruturas de dados padrão da linguagem:
  - Lista (*list*);
  - Tupla (*tuple*);
  - Dicionário (*dict*);

Estrutura de dados: Tupla

- Uma sequência de dados **imutável** e com **tamanho fixo**;
  - Aceita qualquer tipo de dados

```
t = (1, 2, 3)
type(t)
print(t)
print(t[1])
```

```
<class 'tuple'>
(1, 2, 3)
2
```

```
curso = (60.0, 'SeComp', 8)
print(curso)
print(curso[1])
```

```
(60.0, 'SeComp', 8)
'SeComp'
```

Os comandos `count()` e `len()` também funcionam com tuplas

Estrutura de dados: Lista

- Listas podem ser vistas como **tuplas mutáveis**
  - Aceita qualquer tipo de dados

```
l = [1, 2, 3]
type(l)
print(l)
l.append('Python Rules!')

print(l[3])
```

```
<class 'list'>
[1, 2, 3]
[1, 2, 3, 'Python Rules']
'Python Rules'
```

casting

```
sequencia = list(range(0,10))
print(sequencia)
print(len(sequencia))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

10
```

Estrutura de dados: Lista

- Alguns métodos utilizados com listas
  - `append`(valor): adiciona o valor no final da lista;
  - `count`(valor): conta o número de ocorrências de um valor;
  - `insert`(valor, índice): insere o valor no índice especificado;
  - Operador `in`: verificar se um valor está na lista

Estrutura de dados: Lista

- Algumas manipulação de listas

Concatenação

```
a = [1, 2, 'Pim']
b = [4, 5, 'Pim']
print(a + b)
```

```
[1, 2, 'Pim', 4, 5, 'Pim']
```

Multiplicação

```
a = [1, 2, 'Pim']
b = 2 * a
print(b)
```

```
[1, 2, 'Pim', 1, 2, 'Pim']
```

Operador in

```
a = [1, 2, 'Pim']
'Pim' in a
8 in a
```

```
True
False
```

Exclusão

```
a = [1, 2, 'Pim']
del a[1]
print(a)
```

```
[1, 'Pim']
```

Ordenação

```
a = [8, 1, 3]
a.sort()
print(a)
```

```
[1, 3, 8]
```

E sobre o comando remove()?

Estrutura de dados: Dicionário

- Coleção de pares chave-valor de temanho flexível
  - Comumente chamado de hash
  - Podemos pensar como se fosse uma sequência de valores que são indexados por uma **chave**, e **não por um índice sequencial numérico**

```
dict = {chave1: valor1, chave2: valor2, …, chaveN, valorN}
```

```
curso = {'linguagem': 'Python', 'duracao': 8, 'evento': 'SeComp'}
print(curso['linguagem'])
print(curso['evento'])
print(curso['duracao'])
```

```
'Python'
'SeComp'
8
```

Estrutura de dados: Dicionário

- Detalhe importante sobre dicionários
  - As chaves de um dicionário devem ser imutáveis;
- Métodos para se usar com dicionários
  - `values()`: retornar os valores presentes no dicionário
  - `keys()`: retorna as chaves do dicionário

Estrutura de dados: iteração

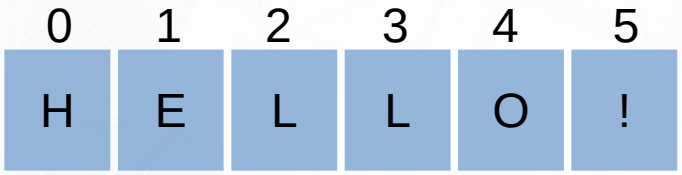
- Como acessar os valores de uma lista ou tupla iterativamente (um por um)?
  - Usando comando for

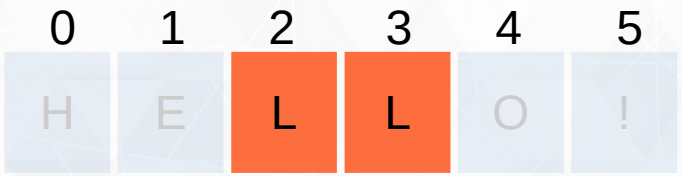
```
l = [1, 2, ‘Pim’, ‘SeComp’]

for i in l:
    print(i)
```

```
1
2
‘Pim’
‘SeComp’
```

Estrutura de dados: *slicing*

- Aplicada em estruturas armazenada em sequência
  - Strings, listas, tuplas...





string[2:4]

string[-2:]

String[:-3]

Estrutura de dados: iteração

- E como iterar em relação aos dados de um dicionário?
  - Usando comando for

```
curso = {'linguagem': 'Python', 'duracao': 8,
'evento': 'SeComp'}

for key in curso:
    print('chave:', key, '| Valor:', curso[key])
```

```
Chave: linguagem | Valor: Python
Chave: duracao | Valor: 8
Chave: evento | Valor: SeComp
```

O que vimos até aqui...

- Python é uma linguagem dinâmica e fortemente tipada;
- Definição de variáveis e alguns tipos básicos;
- Operações matemáticas básicas;
- Entrada e saída (input, print);
- Blocos condicionais: if, elif, eles;
- Laços de repetição: while, for;
- Estruturas de dados: tupla, lista, dicionário;

Conceitos básicos: Funções

- É possível associar um bloco de código a um nome e chamá-lo quando necessário

```
def subtracao(a, b):
    return a - b

resultado1 = subtracao(5, 3)
resultado2 = subtracao(5.8, 2)
print(resultado1)
print(resultado2)
```

```
2
3.8
```

Conceitos básicos: Funções

- Passando listas como parâmetro
  - Quando não sabemos exatamente a quantidade de parâmetros que usuário vai passar

```
def dobra(l):
    for i in range(len(l)):
        l[i] = l[i] * 2
#------------------------
lista = [2, 4, 1, 5]
dobra(lista)
print(lista)
```

Listas são sempre passadas por referência para uma função

Conceitos básicos: Funções

- Desempacotar parâmetros
  - Quando não sabemos exatamente a quantidade de parâmetros que usuário vai passar

```
def soma(*param):
    soma = 0
    for i in param:
        soma = soma + i

    print(soma)

soma(1, 2, 3, 4, 5)
soma(-9, -3, 8)
soma(4)
```

Conceitos básicos: Funções

- Parâmetros opcionais
  - Quando não sabemos exatamente a quantidade de parâmetros que usuário vai passar

```
def elevado(base, exp=2):
    return base ** exp


print(elevado(2, 3))
print(elevado(4))
```

Se nenhum valor for informado para exp, o 2 será assumido como padrão

Conceitos básicos: Funções

- Retornando vários valores

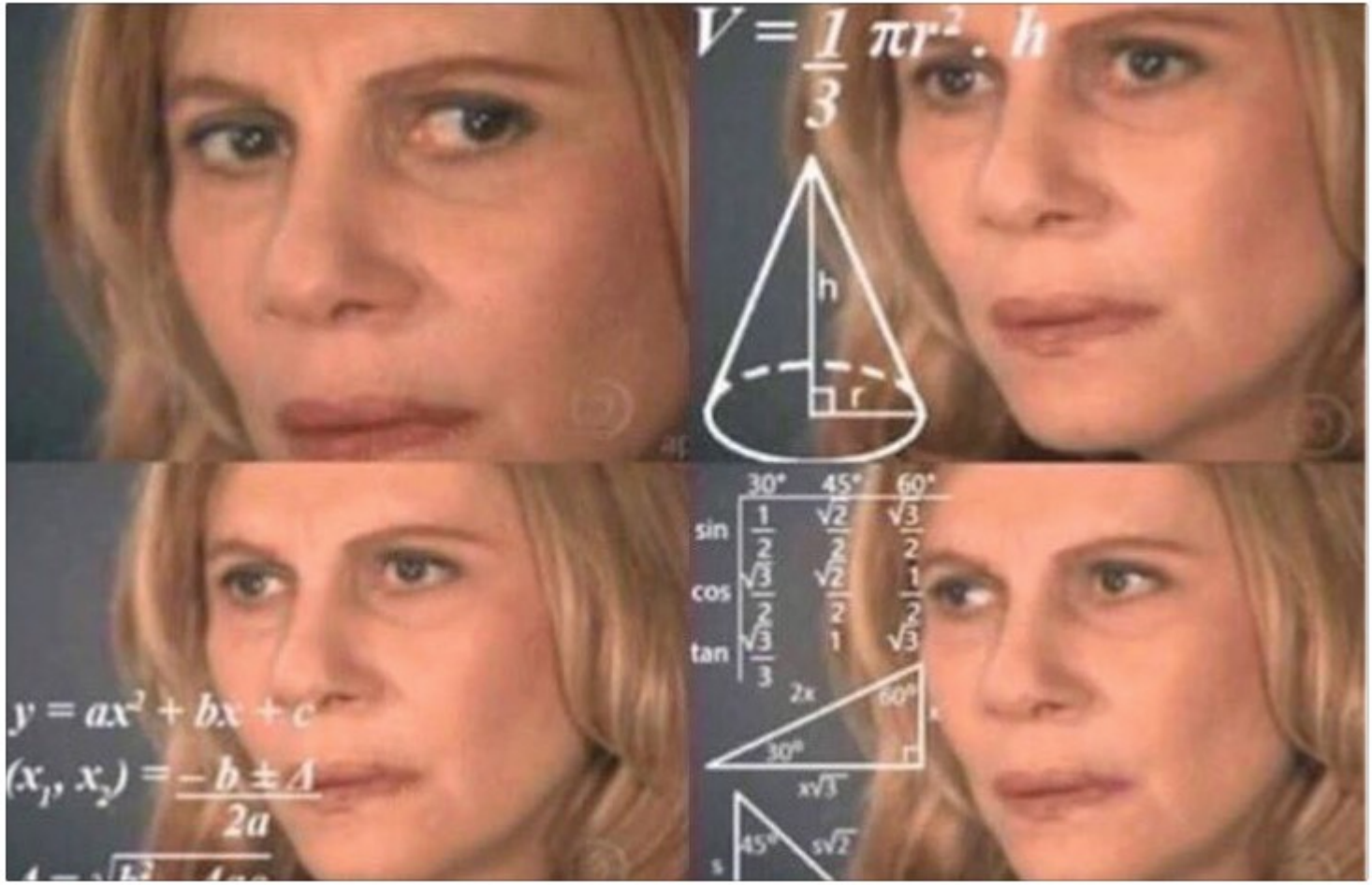
```
def equacao2grau(a, b, c):
    ...
    return x1, x2

a, b = equacao2grau(a, b, c)
print(f‘As raizes são {a} e {b})
```

O que vimos até aqui...

- Python é uma linguagem dinâmica e fortemente tipada;
- Definição de variáveis e alguns tipos básicos;
- Operações matemáticas básicas;
- Entrada e saída (input, print);
- Blocos condicionais: if, elif, eles;
- Laços de repetição: while, for;
- Estruturas de dados: tupla, lista, dicionário;
- Funções

Tratamento de exceções

- Estar certo é diferente de estar livre de erros...



Tratamento de exceções

- O código abaixo está correto?

```
numerador = int(input('Informe um numero inteiro'))
denominador = int(input('Informe um numero inteiro'))
print(numerador / denominador)
```

- Se o usuário fornecer zero para o denominador?

```
ZeroDivisionError: integer division or modulo by zero
```

- Como resolver o problema?
- Como não fazer que o programa pare a execução por conta de erros como esse?

Tratamento de exceções

```
try:
   pass  #comandos que podem gerar problema

except:
   pass  #processa o erro se ele ocorrer

finally:
   pass  #opcional. Sempre será executado
```

```
numerador = int(input('Informe um numero inteiro'))
denominador = int(input('Informe um numero inteiro'))

try:
   print(numerador / denominador)
except Exception as e:
   print('Ocorreu um problema na divisao')
   raise e
```

Leitura e escrita de arquivos

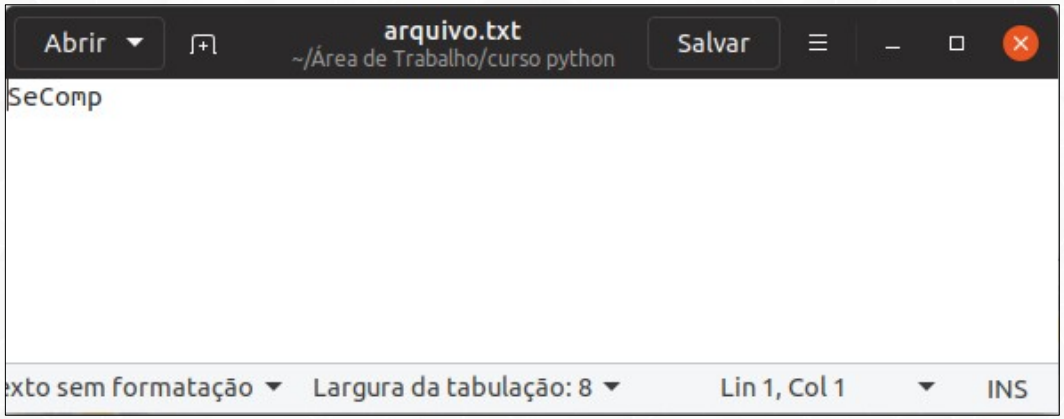
- Usamos o comando `open` em conjunto com `with`

Nome do arquivo

```
with open('arquivo.txt', 'w') as file:
    file.write('SeComp')
```

Tipo de operação

Tipo de Operação	
r (read)	leitura
w (write)	escrita
a (append)	anexo



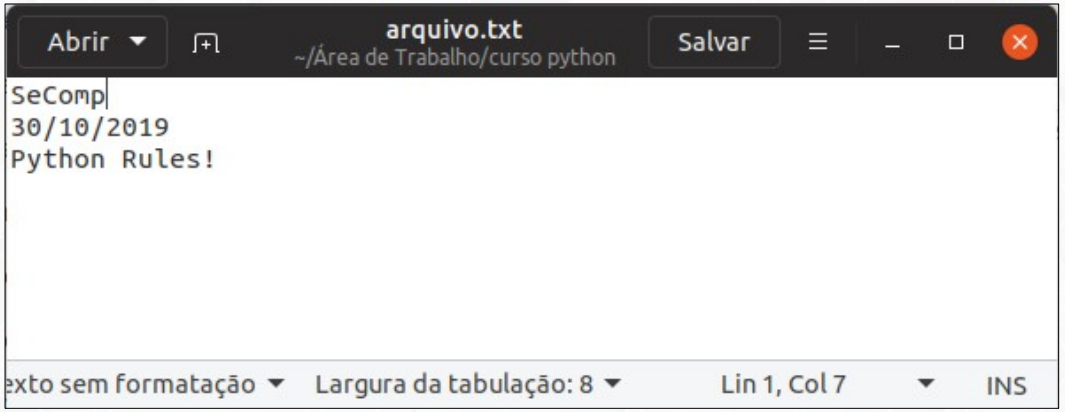
Leitura e escrita de arquivos

- Usamos o comando `open` em conjunto com `with`

```
with open('arquivo.txt', 'a') as file:
    file.write('30/10/2019\nPython Rules!')
```

['SeComp', '30/10/2019', 'Python Rules!']

Tipo de Operação	
r (read)	leitura
w (write)	escrita
a (append)	anexo

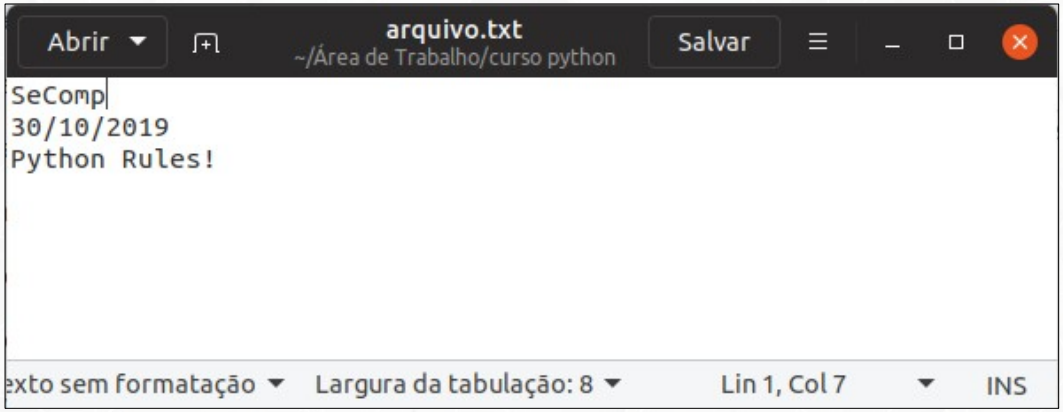


Leitura e escrita de arquivos

- Usamos o comando `open` em conjunto com `with`

```
with open('arquivo.txt', 'r') as file:
    dados = file.readlines()

print(dados)
```

Tipo de Operação	
r (read)	leitura
w (write)	escrita
a (append)	anexo



Conceito avançado: *list comprehension*

- Recurso poderoso para manipulação de listas em Python

```
[exp for item in lista]
```

```
for x in range(10):
    lista.append(x**2)
```

```
lista = [x**2 for x in range(10)]
```

Conceito avançado: *list comprehension*

- Recurso poderoso para manipulação de listas em Python

```
[exp for item in lista if condição]
```

Como criar uma lista de números pares de zero a 20 usando *list comprehension*?

Como encontrar elementos comuns a duas listas A e B usando *list comprehension*?

Características da linguagem

```
import this
```

- O Zen do Python, por Tim Peters
  - Bonito é melhor que feio.
  - Explícito é melhor que implícito.
  - Simples é melhor que complexo.
  - Linear é melhor do que aninhado.
  - Esparso é melhor que denso.
  - Legibilidade conta.
  - Casos especiais não são tão especiais o bastante para quebrar regras.
  - Ainda que a praticidade vença a pureza.
  - Erros nunca devem passar silenciosamente.
  - A menos que sejam explicitamente silenciados.

Características da linguagem

```
import this
```

- O Zen do Python, por Tim Peters
  - Diante de ambiguidade, recuse a tentação de adivinhar.
  - Deveria haver um – e preferencialmente só um – modo óbvio para fazer algo.
  - Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
  - Agora é melhor que nunca.
  - Embora nunca frequentemente seja melhor que já.
  - Se a implementação é difícil de explicar, é uma má ideia.
  - Se a implementação é fácil de explicar, pode ser uma boa ideia.
  - *Namespaces* são uma grande ideia – vamos ter mais dessas!