



# Introdução à Linguagem Python

Paradigmas de Linguagens de Programação

**Rômulo Souza Fernandes**  
**Ausberto S. Castro Vera**

2 de novembro de 2022



Copyright © 2022 Rômulo Souza Fernandes e Ausberto S. Castro Vera

UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO

CCT - CENTRO DE CIÊNCIA E TECNOLOGIA

LCMAT - LABORATÓRIO DE MATEMÁTICAS

CC - CURSO DE CIÊNCIA DA COMPUTAÇÃO

*Primeira edição, Setembro 2022*

# Sumário

<b>1</b>	<b>Introdução .....</b>	<b>5</b>
1.1	História da linguagem Python	5
1.2	Áreas de Aplicação da Linguagem	6
1.2.1	Big Data .....	6
1.2.2	Orientação a objetos .....	7
1.2.3	Pentest .....	7
<b>2</b>	<b>Conceitos básicos da Linguagem Python .....</b>	<b>9</b>
2.1	Variáveis e constantes	9
2.2	Tipos de Dados Básicos	10
2.2.1	Inteiro .....	10
2.2.2	Ponto Flutuante .....	10
2.2.3	Booleano .....	10
2.2.4	String .....	11
2.2.5	Lista .....	12
2.3	Tipos de Dados de Coleção	13
2.3.1	Tipos Sequenciais .....	13
2.3.2	Tipos Conjunto .....	14
2.3.3	Tipos Mapeamento .....	14
2.4	Estrutura de Controle e Funções	14
2.4.1	O comando IF .....	14
2.4.2	Laço FOR .....	15
2.4.3	Laço WHILE .....	16
2.5	Módulos e pacotes	16
2.5.1	Módulos .....	16
2.5.2	Pacotes .....	17

<b>3</b>	<b>Programação Orientada a Objetos com Python .....</b>	<b>19</b>
<b>3.1</b>	<b>Classes e Objetos</b>	<b>19</b>
<b>3.2</b>	<b>Operadores ou Métodos</b>	<b>20</b>
<b>3.3</b>	<b>Herança</b>	<b>20</b>
<b>4</b>	<b>Aplicações da Linguagem Python .....</b>	<b>23</b>
<b>4.1</b>	<b>Operações básicas</b>	<b>23</b>
<b>4.2</b>	<b>Programas gráficos</b>	<b>26</b>
<b>4.3</b>	<b>Programas com Objetos</b>	<b>28</b>
<b>4.4</b>	<b>O algoritmo Quicksort</b>	<b>29</b>
<b>4.5</b>	<b>Mobile</b>	<b>30</b>
	<b>Bibliografia .....</b>	<b>33</b>
	<b>Index .....</b>	<b>35</b>



# 1. Introdução

Segundo [Bor14], o Python é uma linguagem orientada a objetos de alto nível, que possui uma sintaxe simples e objetiva, assim colaborando para a fácil compreensão do código-fonte e permitindo que a linguagem seja produtiva. O Python contém várias estruturas de alto nível, como hora, data, dicionários, listas, complexos, entre outras estruturas. Contém um amplo conjunto de módulos disponíveis para utilização, frameworks que podem ser acrescentados, ferramentas de outras linguagens atuais, como persistência, unidades de teste, geradores, introspecção e metaclasses, além de ter disponíveis diversas bibliotecas, como IPython, Matplotlib, mIPy, NumPy, Pandas, SciPy, Scrapy, entre outras bibliotecas conhecidas.

O Python é uma linguagem multiparadigma, suportando a programação orientada a objetos, modular e funcional. A linguagem Python foi criada na Holanda, no ano de 1990, por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação. [Bor14]

A linguagem Python é de código aberto, porém o criador Guido van Rossum possui a função central de decidir a evolução da linguagem. O Python se popularizou e se tornou a linguagem de desenvolvimento de aplicações mais indicada para iniciantes, assim sendo aconselhada como primeira linguagem de programação. [Per16]

## 1.1 História da linguagem Python

O intuito de Guido van Rossum era criar uma linguagem que pudesse suprir suas exigências, assim criando o Python, com base na linguagem ABC, mas solucionando os problemas encontrados por ele na linguagem. O Python tinha como usuários principais os engenheiros e físicos.

A seguir um pouco da história da linguagem Python, baseados em [Per16] e [Bor14] :

- O Holandês Guido van Rossum foi o autor principal da linguagem Python. O autor trabalhava no CWI (Centrum Wiskunde & Informatica), localizada em Amsterdã na Holanda.
- O nome Python não veio da espécie de serpente e sim do seriado de comédia preferido do autor da linguagem, chamado Monty Python's Flying Circus.
- A versão 0.9.0 do Python foi lançado em 1991, incluindo manipulação de exceções, classes, listas e strings. Incluía também alguns aspectos de programação funcional como lambda, maps, filter e reduce.

- No ano de 1995, o autor da linguagem continuou seu trabalho sobre Python na Corporation for National Research Initiatives (CNRI) em Reston, Virginia, USA.
- Em Maio de 2000, Guido van Rossum e o grupo de desenvolvimento do Python se mudaram para BeOpen.com, assim formando a equipe BeOpen PythonLabs.
- A versão 1.6 do Python foi lançada em 5 de setembro de 2000.
- A versão 2.0 do Python foi lançada em 16 de outubro de 2000.
- A versão 3.0 do Python foi lançada em 3 dezembro de 2008.

## 1.2 Áreas de Aplicação da Linguagem

O Python está entre as linguagens de programação mais utilizadas no mundo, é muito utilizado por usuários individuais, mas sua aplicação se estende para empresas reais. A natureza do Python é de propósito geral, assim tornando a linguagem aplicável em quase todas as áreas. Como a IBM, Seagate e Hewlett-Packard, que utilizam o Python para testes de hardware, o Yahoo! e Google usam a linguagem em serviços de Internet, já a empresa Industrial Light and Magic e outras empresas de filmes utilizam o Python na produção de animações. Entre todas as aplicações para o Python atualmente, o ponto em comum é que a linguagem é usada em todo o espectro, em questão de domínios de aplicação. [Lut07]

### 1.2.1 Big Data

Com base em [McK19], atualmente o Python é uma das melhores linguagens de programação para trabalhar com Big Data. Um dos motivos dessa preferência de uso é o suporte avançado de inúmeras bibliotecas e frameworks, muitas das bibliotecas são voltadas para lidar com Big Data, dando suporte e auxiliando na implementação de algoritmos de Machine Learning e Data Analytics. Abaixo algumas das bibliotecas de software livre:

- SciPy: Utilizada para computação técnica e computação científica, possibilita a interpolação, otimização, integração e modificação de dados utilizando funções especiais, álgebra linear, etc.
- NumPy: Utilizada para computação numérica para dados com formas de grandes matrizes multidimensionais e Arrays. A biblioteca também oferece diversas funções matemáticas de alto nível, para manipular os dados com transformada de Fourier, álgebra linear, processamento de números aleatórios, etc.
- Scikit-learn: Utilizada para Machine Learning, relacionada a vários algoritmos de regressão, clustering e classificação. Pode ser utilizada também em conjunto com outras bibliotecas, como a NumPy e SciPy.
- Pandas: Utilizada para análise e manipulação de dados, oferece diversas estruturas de dados e operações para manipulação de dados, no formato de séries temporais e tabelas numéricas. A biblioteca também dispõe de diferentes ferramentas para gravar e ler dados, entre estruturas de dados na memória e diferentes formatos de arquivo.

O Python possui uma sintaxe simples, possibilitando uma fácil leitura do código, assim tanto os iniciantes quanto os desenvolvedores experientes, podem se concentrar melhor no objetivo, ao invés de se desgastar se concentrando nas nuances técnicas da linguagem que está utilizando. Sendo assim, o Python é a linguagem preferida dos Cientistas de dados e Engenheiros de Big Data.

A linguagem Python é extremamente flexível, permitindo finalizar mais trabalhos com menor número de linhas de código. O Python também é escalável na manipulação de dados em grandes quantidades, sendo um ponto muito importante quando se trata de Big Data. Comparando o Python com outras linguagens de programação utilizadas em Big Data Analytics, como R e Java, elas não são tão escaláveis e flexíveis como o Python, onde havendo um aumento no volume de dados, o Python sem dificuldades pode aumentar a velocidade de processamento dos dados, sendo uma

tarefa complicada para fazer em R ou Java. [McK19]

### 1.2.2 Orientação a objetos

Segundo [Lut07], o Python é uma linguagem multiparadigma, suportando diferentes abordagens de programação, um jeito de solucionar problemas de programação é criar objetos, esse método é conhecido como Programação Orientada a Objetos(POO) e a orientação a objetos é um dos paradigmas da linguagem Python, com isso, a criação de objetos e classes é mais simples.

No Python os dados são guardados em objetos, em outras linguagens, determinados tipos são guardados na memória, não em entidades abstratas como objetos. A programação orientada a objetos é um importante método de organizar e desenvolver códigos, focando em criar códigos reutilizáveis. [Lut07]

No Python, a programação orientada a objetos possui alguns pontos importantes, baseados no mesmo autor citado no parágrafo anterior:

- Encapsulamento: Restrição ao acesso de métodos e atributos de uma classe, evitando que os dados sejam alterados diretamente, na linguagem Python existe apenas o `private` e `public`.
- Métodos: São funções definidas no corpo da classe. Utilizados para definir o comportamento do objeto.
- Objeto: É a instância de uma classe. Somente a definição do objeto é definida no momento em que a classe é estabelecida. Consequentemente nenhum espaço de memória é alocado.
- Classe: Juntam as funcionalidades de um certo objeto, servindo como um modelo.
- Polimorfismo: É a capacidade de utilizar uma interface para vários tipos de dados. Possibilitando que o objeto possua o poder de assumir diversas formas.
- Herança: Cria uma classe nova que será descendente e irá utilizar particularidades de outra classe já existente, sem realizar modificações na mesma.

### 1.2.3 Pentest

O Python está entre as melhores linguagens para pentest e segurança da informação, isso é devido a grande quantidade de bibliotecas, ferramentas, frameworks, por ter versatilidade e ser multi-plataforma, entre outros pontos que tornam o Python uma das melhores linguagens para essa aplicação, como ter um código de fácil leitura e sintaxe simples. A linguagem é usada para diversas finalidades dentro do processo, possibilitando que as soluções sejam vinculadas e automatizadas sem dificuldades, como decodificar e enviar pacotes, varrer redes e portas, analisar malwares, acessar servidores, etc, com base em [Mor18] e [Sei15].

Como citado, a linguagem Python possui uma grande diversidade de ferramentas, de acordo com os mesmos autores citados no parágrafo anterior, essas são algumas das ferramentas disponíveis:

- Análise de malware:
  - Exefilter: Utilizada para filtrar tipos de arquivos para páginas web e e-mails, podendo detectar diversos tipos de arquivos e apagar o conteúdo ativo.
  - PyClamAV: Acrescenta a detecção de vírus maliciosos nas ferramentas do Python.
  - Pyew: Utilizada geralmente para analisar malwares, o pyew desmonta e edita hexadecimais de linha de comando.
- Utilitários de rede:
  - Dpkt: Utilizado para gerar e analisar pacotes de dados utilizando definições do protocolo TCP/IP.
  - Spoodle: Usado para verificar subdomínios e vulnerabilidade.
  - Knock Subdomain Scan: utilizado para retornar a lista de subdomínios do domínio de destino através da técnica de lista de palavras.
- Explorar bibliotecas:
  - Scapy: É uma biblioteca e ferramenta de processamento de pacotes, podendo decodificar

diversos protocolos, enviar pela rede, capturar e combinar respostas e solicitações. Oferece funções como as oferecidas pelo Tcpdump, Wireshark e Nmap, também oferece acesso programático.

- Python Nmap: Usado para analisar os resultados da varredura do Nmap e lançar ataques particularizados contra hosts específicos.
- Monda: O Monda é um depurador de imunidade, que auxilia no desenvolvimento de programas de exploração.
- Forense
  - Rekall: O Rekall é um framework criado pelo Google para realizar varreduras e análises de memória.
  - LibForensics: É uma biblioteca para desenvolvimento de aplicativos Forenses digitais
  - Aft: É um pacote de ferramentas forenses voltado para Android.





## 2. Conceitos básicos da Linguagem Python

Neste capítulo é apresentado alguns conceitos básicos da linguagem de programação Python, como Variáveis, constantes e tipos de dados básicos aceitos pela linguagem, que são inteiro, ponto flutuante, booleano, string e lista. Alguns dos livros indicados para iniciar o estudo sobre a linguagem de programação Python são: [Lut07], [Per16], [Sev16], [Bor14], que foram os mesmos livros usados como base para escrever grande parte desse e outros capítulos.

### 2.1 Variáveis e constantes

De acordo com [Sev16] o Python possui um ótimo recurso, a manipulação de variáveis, essas variáveis são nomes atribuídos a valores, que possuem como propósito armazenar valores de forma que possam ser recuperados mais tarde. Para criar variáveis é necessário fazer uma declaração por atribuição e assim atribuir valores a essas novas variáveis.

```
>>> texto = 'Hello, world!'
>>> pi = 3,14159265
>>> numero = 1500
```

No exemplo acima podemos observar a declaração por atribuição de 3 variáveis diferentes. A primeira está atribuindo uma string para uma variável, chamada texto, a segunda variável atribui o valor 3,14 e a variável possui o nome pi. Já a terceira variável, chamada numero está recebendo o valor 1500, também por atribuição. Os usuários tem uma grande liberdade na hora de escolher os nomes das variáveis, apenas não sendo possível usar palavras reservadas da linguagem como nome de uma variável.

As constantes na linguagem Python, diferente de outras linguagens de programação, não podem ser criadas de forma que seu valor não seja alterado. Na documentação existem algumas orientações caso o usuário queira criar uma constante com sintaxe de variável, uma delas é que todas as letras da variável que será utilizada como constante, deverão ser maiúsculas, em casos do nome desejado possuir espaço, deverá ser utilizado underline.

```
>>> PRECO = 5
>>> PRECO_PRODUTO = 2
```

O exemplo acima é o padrão recomendado pela documentação do Python.

## 2.2 Tipos de Dados Básicos

A linguagem de programação Python é de tipagem dinâmica, com isso não é preciso declarar o tipo de variável, o tipo será definido através do valor que a variável receber, isso possibilita que o tipo mude no decorrer da execução do programa. Vamos falar sobre esses tipos de dados a seguir, com base nos autores [Lut07], [Per16], [Sev16].

### 2.2.1 Inteiro

De acordo com o autor [Per16], o tipo inteiro ou *int* representa caracteres numéricos inteiros positivos e negativos. Como já informado no texto anteriormente, na linguagem Python não é necessário informar o tipo da variável na sua declaração, o exemplo abaixo deixará isso mais claro.

```
>>> n1 = 5
>>> n2 = 10
>>> soma = n1 + n2
>>> print(soma)
>>> print(type(soma))
15
<class 'int'>
```

### 2.2.2 Ponto Flutuante

Com base no [Per16], o *float* ou ponto flutuante é um tipo de dado usado para os números reais, pois eles possuem casas decimais, explicando melhor, são números com vírgula, como por exemplo um peso ou altura. Em uma expressão numérica, se um dos valores for um *float*, o resultado da expressão será um *float*, o exemplo abaixo irá demonstrar.

```
>>> #exemplo utilizando ponto flutuante
>>> peso = 1.77
>>> altura = 62.50
>>> print(type(peso))
>>> print(type(altura))
<class 'float'>
<class 'float'>

>>> #exemplo utilizando 1 ponto flutuante e 1 inteiro
>>> a = 5
>>> b = 2.5
>>> soma = 5 + 2.5
>>> print(soma)
>>> print(type(resultado))
7.5
<class 'float'>
```

### 2.2.3 Booleano

Segundo [Per16], o booleano é um tipo de dado lógico, utilizado para armazenar valores lógicos, que no Python pode ser representado pelo valor True ou False. O True significando que o valor é verdadeiro e False significa que seu valor é falso. Seguimos com alguns exemplos do tipo de dado lógico booleano.

```
>>> x = false
>>> print(type(x))
<class 'bool'>

>>> #operacoes logicas
>>> a = 10 < 5
>>> print(a)
False

>>> b = c = 5
>>> print(b <= c and c <= b)
True
```

### 2.2.4 String

Com base no autor [Sev16], uma string é uma sequência de caracteres imutável, não permitindo alterar uma string que já existe. É classificado como um item de dado simples. Para o Python, uma string é um array de caracteres ou qualquer grupo de caracteres escritos entre aspas duplas ou aspas simples, por exemplo:

```
>>> #Aspas simples
>>> mensagem1 = 'Hello, World'
>>> print(mensagem1)
Hello, World

>>> #Aspas duplas
>>> mensagem2 = "0 dia esta chuvoso"
>>> print(mensagem2)
0 dia esta chuvoso
```

- *Concatenação de strings*

A união de strings é chamado de concatenação, isso pode ser feito utilizando o operador +, que possui essa função de concatenar quando usado com operandos do tipo string. O comprimento de uma string pode ser calculado utilizando o operador len(string).

```
>>> # concatenando 2 strings
>>> moto = "Titan " + "150 ESD"
>>> print(moto)
Titan 150 ESD

>>> print(len(moto))
13
```

- *Operador de indexação*

Utilizando o operador de indexação, é possível acessar os caracteres um por um, utilizando o operador colchetes, o número dentro do colchetes é denominado index, usado para indicar a posição do caractere da variável e atribuir esse caractere à uma variável. Existem duas formas de indexar os caracteres de um string em Python:

**Index com inteiros positivos** indexando a partir da esquerda, começando com 0, sendo o 0 o index do primeiro caractere da sequência.

**Index com inteiros negativos** indexando a partir da direita, começando com -1, sendo -1 o último elemento da sequência, -2 sendo o penúltimo elemento da sequência, e assim sucessivamente.

```
>>> #Inteiro positivo
>>> moto = 'titan'
>>> letra = moto[0]
>>> print(letra)
t

>>>#inteiro negativo
>>>moto = 'titan'
>>>letra = moto[-1]
>>>print(letra)
n
```

- *Operador de Fatias*

O operador de acesso a itens de forma individual, também pode ser usado como operador de fatias, podendo assim extrair uma fatia inteira (subsequência) de caracteres de uma string. O operador de Fatias dispõe de três sintaxes:

```
sequencia[ inicio ]
sequencia[ inicio : fim ]
sequencia[ inicio : fim : step ]
```

onde início, fim e step são números inteiros.

```
>>> sequencia = 'Linguagem Python'
>>> print(sequencia[0:9])
Linguagem

>>> print(sequencia[0:9:8])
Lm
```

### 2.2.5 Lista

Segundo o autor [Sev16], lista é uma sequência de objetos, diferente de uma string onde os valores são caracteres, na lista esses valores podem ser de qualquer tipo, até mesmo outras listas. Outra característica da lista que difere de uma string, as listas são mutáveis, assim permitindo que o seu conteúdo seja modificado em qualquer momento, esse conteúdo das listas é chamado de elementos ou itens. Para criar uma lista é necessário colocar seus elementos apenas entre colchetes ou entre aspas dentro dos colchetes caso os elementos sejam caracteres.

```
['verde ', 'amarelo', 'azul']
[1, 2, 3, 4, 5]
```

A primeira lista do exemplo é formada por três strings, enquanto a segunda é formada por cinco números inteiros. Como já citado, é possível criar uma lista com elementos de tipos diferentes, como no exemplo a seguir, onde a lista contém uma lista, uma string, um float e um inteiro..

```
[[1, 2], 'amarelo', 3.14, 100]
```

Também é possível atribuir valores de uma lista a variáveis.

```
>>> numeros = [1, 2, 3, 4, 5]
>>> cores = ['verde ', 'amarelo', 'azul']
>>> lista_vazia = []
```

```
>>> print(numeros, cores, lista_vazia)
[1, 2, 3, 4, 5] ['verde ', 'amarelo', 'azul'] []
```

O acesso de elementos de uma lista é da mesma forma que o acesso de caracteres de uma string, o exemplo a seguir demonstra o funcionamento, o valor dentro dos [] é o índice.

```
>>> print(cores[1])
amarelo
```

Como as listas são mutáveis, é possível alterar seu conteúdo, atribuindo novos valores a itens ou mudando a ordem desses itens. A seguir temos um exemplo de como realizar essa alteração, observe que os [] à esquerda na atribuição significa o elemento que será alterado.

```
>>> numeros = [1, 2, 3, 4, 5]
>>> numeros[0] = 6
>>> print(numeros)
[6, 2, 3, 4, 5]
```

- Operações com Listas

Novamente como nas strings, o + é o operador utilizado para concatenar listas e o \* repete a lista determinada a quantidade de vezes desejada.

```
>>> #concatenacao
>>> numeros = [1, 2, 3]
>>> cores = ['verde ', 'amarelo', 'azul']
>>> concatena = numeros + cores
>>> print(concatena)
[1, 2, 3, 'verde ', 'amarelo', 'azul']

>>> #repeticao de lista
>>> print([1,2,3] * 3)
[1, 2, 3, 1, 2, 3, 1, 2, 3]

>>> print(['vermelho'] * 3)
['vermelho', 'vermelho', 'vermelho']
```

## 2.3 Tipos de Dados de Coleção

### 2.3.1 Tipos Sequenciais

Com base em [Sev16], as tuplas são sequências de valores parecidas com a lista, com a diferença que as tuplas são imutáveis, porém é possível criar uma tupla que contenha objetos mutáveis, um exemplo seria uma lista. Os valores para se armazenar podem ser de todos os tipos, são indexados usando números inteiros. A tupla é uma sequência de valores separados por vírgulas.

```
>>> A = 123, 456, 'ola mundo'
>>> A[1]
456
```

Para aninhar tuplas é necessário envolver por parênteses, permitindo que possam ser lidas corretamente. O uso de parênteses não é obrigatório para a criação de tuplas, apenas para tuplas dentro de expressões. A seguir temos uma demonstração.

```
>>> B = A, (1, 2, 3)
>>> B
((123, 456, 'ola mundo'), (1, 2, 3))
```

### 2.3.2 Tipos Conjunto

Segundo o autor [Per16], no Python um conjunto ou *set* é uma coleção de itens fora de ordem, não podendo incluir itens duplicados. Desde que sejam imutáveis, as chaves podem ser de qualquer tipo. A linguagem oferece diferentes formas eficazes para criação e manipulação desses *sets*, assim admitindo operadores para interseção, união, inclusão em conjunto, diferença simétrica, entre outros operadores disponíveis. Os *sets* são definidos utilizando a mesma sintaxe utilizada para os conjuntos matemáticos, sendo itens separados por vírgula e em sequência, sendo delimitados por chaves.

```
>>> #criando e printando um conjunto
>>> anotacao = {'123-456-789', '987-654-321',}
>>> print(anotacao)
{'987-654-321', '123-456-789'}

>>> #verificando o tipo
>>> type(anotacao)
<class 'set'>
```

### 2.3.3 Tipos Mapeamento

Com base no autor [Sev16], na linguagem Python o dicionário é o único tipo de mapeamento nativo, seu funcionamento é como de uma lista, porém é mais geral. Explicando melhor, nas listas o índice necessariamente é um inteiro, já no dicionário o índice pode ser quase todos os tipos de dados. O dicionário é um mapeamento entre um conjunto de valores e índices, assim a chave que é o índice, é usado para localizar um valor. O dicionário é indicado por *dict*.

```
>>> #criando e printando um dicionario
>>> dic = {'gasolina': 5.30, 'alcool': 4.30, 'GNV': 4.50}
>>> print(dic)
{'gasolina': 5.3, 'alcool': 4.3, 'GNV': 4.5}

>>> #item que localiza pela chave 'GNV'
>>> print(dic['GNV'])
4.5
```

Como mostrado no exemplo acima, a chave 'GNV' sempre irá localizar o valor 4.5, assim mostrando também que a ordem dos itens não interfere.

## 2.4 Estrutura de Controle e Funções

### 2.4.1 O comando IF

Com base nos autores [Per16] e [Sev16], o *if* é uma estrutura de controle, sendo de decisão fundamental, permitindo a execução de blocos de códigos alternativos, se baseando em condições.

```
>>> if A > 0
>>>     print('A e um numero positivo')
```

Depois da declaração `textit` temos uma expressão booleana, essa expressão é a condição. Como vemos no exemplo acima, a declaração termina com o símbolo de dois pontos (`:`) e a linha após o `textit` devem ser indentadas. A condição lógica sendo verdadeira, logo a declaração é executada, caso seja falsa, a condição é ignorada.

As declarações que se alongam por mais de uma linha e consistem em uma linha de cabeçalho e um bloco indentado, são chamadas de `textit`declarações compostas. O número de declarações possíveis não tem um limite, porém é obrigatório que tenha no mínimo uma.

- *Execução alternativa*

A *Execução alternativa* é a segunda forma de declarar um `if`, nessa declaração existem duas possibilidades e a sua condição, que irá determinar qual delas será executada. Abaixo temos uma demonstração se o número é par ou ímpar.

```
>>> if A%2 == 0
>>>     print('A e Par')
>>> else:
>>>     print('A e Impar')
```

- *Condição encadeada*

A condição encadeada é um jeito de expressar uma lógica computacional, utilizada quando existe mais de uma possibilidade e necessitamos de mais de duas ramificações.

```
>>> if n1 < n2:
>>>     print('n1 e menor que n2')
>>> elif n1 > n2:
>>>     print('n1 e maior que n2')
>>> else:
>>>     print('n1 e n2 sao iguais')
```

- *Condição aninhadas* Também podemos escrever com 3 ramos, assim a condição passará a ser aninhada, vamos exemplificar na demonstração abaixo.

```
>>> if n1 == n2:
>>>     print('n1 e n2 sao iguais')
>>> else:
>>>     if p1 < p2:
>>>         print('n1 e menor que n2')
>>>     else:
>>>         print('n1 e maior que n2')
```

### 2.4.2 Laço FOR

Segundo [Sev16], podemos criar um laço definido utilizando uma declaração `for`, quando temos um conjunto de itens para iterar. É chamado de laço definido, pois irá iterar sobre uma lista de itens até que o número de iterações seja o mesmo de itens da lista. Abaixo temos um exemplo de declaração `for`, onde o `for` e `in` são palavras-chave reservadas do Python, `colegas` e `colega` são variáveis, `colega` em específico é uma *variável de iteração* do laço `for`, que percorre o conteúdo da variável `amigos`, mudando a cada iteração, assim controla quando o laço deve finalizar.

```
>>> colegas = ['Marta', 'Lucas', 'Diego']
>>> for colega in colegas:
>>>     print('Bom dia', colega)
Bom dia Marta
Bom dia Lucas
Bom dia Diego
```

Para o Python, a variável `colegas` é considerada uma lista contendo 3 cadeias de caracteres e um laço *for*, que percorrerá a lista e executará o corpo 1 vez para cada palavra da lista, por isso a saída mostra a mensagem para todo os colegas da lista.

### 2.4.3 Laço WHILE

De acordo com [Sev16], a linguagem Python oferece vários recursos para tornar a automatização de tarefas repetitivas mais fácil. No Python, o *while* é uma forma de iteração. A seguir temos uma demonstração de uso do *while* e também podemos perceber novamente a clareza da sintaxe do Python, a declaração pode ser lida como uma frase usual.

```
>>> x = 2
>>> while x > 0:
>>>     print(x)
>>>     x = x-1
>>> print('Go!!!')
```

O fluxo de execução é, analisar a condição, assim retornando o valor *True* ou *False*, caso a condição seja falsa, sai da declaração e segue para as próximas declarações, já caso seja verdadeira, executa o bloco *while* e retorna para o primeiro passo, que é analisar a condição. O bloco de instruções também pode ser chamado de *laço*, visto que o terceiro e último passo retorna para o primeiro. A cada vez que as instruções internas de um laço são executadas, chamamos de *iteração*. A *variável de iteração* é a variável que tem seu valor modificado a cada execução do laço, assim controlando quando o laço deve acabar. Podemos chamar uma declaração *while* de laço indefinido, pois a declaração continua iterando até que uma das condições seja falsa. [Sev16]

## 2.5 Módulos e pacotes

### 2.5.1 Módulos

Baseado em [Per16], os módulos são arquivos que contém definições e instruções, são utilizados em uma execução ou script interativa do interpretador. Essas definições podem ser importadas para o módulo principal ou outro módulo. O arquivo tem o mesmo nome do módulo, porém com o acréscimo do *.py*. O próprio Python possui uma variedade de módulos na sua biblioteca padrão. Após a importação do módulo podemos usar as instruções e definições que estão contidas nele. Abaixo um exemplo prático.

```
>>> # Modulo de numeros Fibonacci
>>> def fib(n):
>>>     x, y = 0, 1
>>>     while x < n:
>>>         print(x, end=' ')
>>>         x, y = y, x+y
>>>     print()

>>> def fib2(n):
>>>     resultado = []
>>>     x, y = 0, 1
>>>     while x < n:
>>>         resultado.append(x)
>>>         x, y = y, x+y
>>>     return resultado
```



Esse deverá ser o conteúdo do arquivo *fibonacci.py*, após, é necessário importar esse módulo pelo interpretador, para isso basta inserir o comando:

```
>>> import fibonacci
```

Com isso os nomes das funções definidas não serão adicionados, apenas o nome do módulo é acrescentado. Utilizando o nome é possível acessar as funções.

```
>>> fibonacci.fibonacci(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibonacci.fibonacci2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

>>> fibonacci.__name__
'fibonacci'
```

Como dito, o Python oferece uma diversidade de módulos na biblioteca padrão. No próprio interpretador existem alguns desses módulos, permitindo o acesso a operações de fora do núcleo da linguagem. O *winreg* e *sys* são um exemplos de módulos padrão. [Per16]

### 2.5.2 Pacotes

Segundo [Bor14], uma coleção de módulos é considerado um pacote, são utilizados quando os módulos tomam grandes proporções de tamanho, podendo assim dividir esses módulos em pacotes. Para evitar que, os autores de módulos se preocupem com colisão entre os nomes de variáveis globais dos seus módulos e de outros autores, é necessário usar os pacotes.

Enquanto os módulos são estruturados em arquivos, os pacotes são estruturados em pastas. A ferramenta utilizada para gerenciar esses pacotes no Python é o PIP, que significa em português, Índice de Pacotes Python. O PIP permite que através dele, os pacotes sejam instalados, atualizados e removidos em projetos. Seu uso é simples, basta utilizar o comando *pip install* e o nome do pacote desejado. Exemplo:

```
>>> pip install NumPy
```

Em seguida o gerenciador fará o download e a instalação do pacote. Alguns dos pacotes mais utilizados no Python são: NumPy, Pillow, Matplotlib, OpenCV, Delorean, Scipy, Pandas, Requests, Fire, etc.





## 3. Programação Orientada a Objetos com Python

Como sabemos, a Programação Orientada a Objetos (POO) é um dos paradigmas da linguagem Python, com base no conceito de classes e objetos. Com isso trás ótimas e poderosas ferramentas que auxiliam no desenvolvimento de softwares seguros e confiáveis, uma dessas ferramentas que auxiliam é a herança, que evita trabalhos repetidos, permite também que o desenvolvedor codifique com maior velocidade, poupando tempo. Esse paradigma é voltado para os objetos desejados pelos desenvolvedores, ao contrário da lógica essencial para manipular. A seguir vamos explicar e demonstrar cada conceito da Programação Orientada a Objetos.

### 3.1 Classes e Objetos

No Python uma entidade é representada por uma abstração computacional chamada objeto, que possui os atributos, são as qualidades e os métodos que são as ações que a entidade pode fazer. Na orientação a objetos, classe é a estrutura básica, simbolizando o tipo de um objeto, assim definindo o que o objeto pode realizar e suas características. A seguir temos um exemplo simples de como definir uma classe:

```
class NomeDaClasse:
    <statement - 1>
    .
    .
    .
    <statement - N>
```

Para que o objeto permaneça na memória é preciso de no mínimo uma referência, pois o interpretador da linguagem Python tem uma ferramenta de limpeza que exclui todos os objetos que não possuem referência, essa ferramenta é chamada de coletor de lixo ou *Garbage Collector*, assim que os objetos sem referência são excluídos o `__done__()`, que é um método especial, é executado.

Há diversos motivos para o Python aceitar que classes novas sejam definidas por desenvolvedores, um dos motivos é que o programa de aplicação será mais simples de desenvolver, ler, depurar e intuitivo, através das classes projetadas unicamente para essa aplicação. Junto com

a possibilidade de criar classes novas é permitido um novo jeito de estruturar um programa de aplicação. O comportamento de uma função é exposto ao usuário, mas sua implementação é encapsulada(ocultada) [Bor14].

A seguir temos uma demonstração de uso do objeto no Python:

```
stuff = list()
stuff.append('python')
stuff.append('chuck')
stuff.sort()
print (stuff[0])
print (stuff.__getitem__(0))
print (list.__getitem__(stuff,0))
```

Na primeira linha um objeto do tipo *list* é construído, na segunda e terceira linhas do código o método *append()* é chamado, na quarta linha o método *sort()* é chamado, na quinta linha o item da posição 0 é recuperado.

Na sexta linha o método *\_\_getitem\_\_()* com 0 como parâmetro, é chamado na lista *stuff*.

```
print (stuff.__getitem__(0))
```

Na sétima linha mostra como recuperar o primeiro item da lista de uma forma mais detalhada.

```
print (list.__getitem__(stuff,0))
```

Neste código o método *\_\_getitem\_\_()* é chamado na classe *list* e também é passado o item que deve ser recuperado e a lista como parâmetro. As 3 últimas linhas do código percebemos que são semelhantes, porém fazer uso da sintaxe com os colchetes ([ ]) é mais adequado para visualizar um item em uma posição específica da lista [Sev16].

### 3.2 Operadores ou Métodos

De acordo com o autor [Bor14], um método é uma função utilizada para detalhar o comportamento do objeto. Os métodos que conseguem se aplicar aos objetos da classe são expostos ao usuário, mas o modo como essas informações inclusas nos objetos são salvas, é encapsulado, além de encapsular o modo como os métodos das classes estão sendo implementados.

Um operador é uma chamada para os métodos especiais. Os métodos especiais são caracterizados pelos nomes que possuem um padrão, utilizando 2 sublinhados no começo e 2 no final de cada nome, definindo de que forma os objetos que são derivados da classe devem funcionar em casos específicos, conforme na sobrecarga de operadores.

Na linguagem de programação Python um objeto é criado com base na classe, usando a atribuição. O construtor dessas classes que é um método especial, entra em execução quando novos objetos são criados. Esse construtor é chamado de *\_\_new\_\_()*, depois de chamar o construtor para inicializar uma nova instância é chamado o método *\_\_init\_\_()*. Uma forma da classe definir um método especial *\_\_init\_\_()* é:

```
def __init__(self):
    self.data = []
```

### 3.3 Herança

A herança é uma grande ferramenta do Python, devido a programação orientada a objetos. Essa ferramenta possui objetivo de facilitar o reaproveitamento de códigos. Organizando as classes definidas pelo usuário é possível reutilizá-las em outros códigos, de forma semelhante como é

possível utilizar uma função no desenvolvimento de outra função. Assim novos atributos e métodos podem ser implementados por uma nova classe e ao mesmo tempo a classe pode herdar atributos e métodos de uma classe antiga.

Existem 2 tipos de heranças, simples e múltipla. Na herança simples a classe pode derivar apenas de uma classe que já existe, diferentemente na herança múltipla a classe deriva de diversas classes que já existem, outra diferença é na ordem de resolução dos métodos, seguindo o algoritmo diamante [Bor14].

A seguir uma demonstração de herança simples e herança múltipla:

```
# Heranca simples
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>

# Heranca multipla
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```





## 4. Aplicações da Linguagem Python

Neste capítulo será apresentada 5 aplicações completas na linguagem de programação Python, com base nos autores [Per16], [Bor14], [Sev16] e [Lut07]. Cada caso contém:

- Uma breve descrição da aplicação
- O código completo da aplicação,
- Imagens do código fonte no compilador-interpretador,
- Imagens dos resultados após a compilação-interpretação do código fonte
- Links e referencias bibliográficas de onde foi obtido a aplicação

### 4.1 Operações básicas

O código a seguir apresenta algumas operações básicas da matemática que podem ser feitas na linguagem Python, como adição, subtração, multiplicação e divisão. Essas operações serão escolhidas em um menu de opções.

```
# Autor: Romulo Souza Fernandes
# E-mail: 00119110559@pq.uenf.br
# Data de criacao: 28/10/22
# Ciencia da Computacao - UENF
# Disciplina: PLP

continuar_usando = "SIM"

while continuar_usando == "SIM":
    # Criando um menu de opcoes
    print("SELECIONE A OPERACAO DESEJADA")
    print("+ para Adicao")
    print("- para Subtracao")
    print("* para Multiplicacao")
```

```
print("/ para Divisao")

# Interacao com o usuario
operacao = input("\nQual operacao voce deseja realizar? ")

# Criando as operacoes e as apresentacoes de respostas

# Adicao
if operacao == "+":
    a1 = float(input("\nDigite o primeiro valor: "))
    a2 = float(input("Digite o segundo valor: "))
    adicao = a1 + a2
    print("\nA soma entre", a1, "e", a2, "e:", adicao, "\n")
    print("*****33, \n")
    continuar_usando = input("Gostaria de fazer outra operacao? ").upper()
    print("*****33, \n")

# Subtracao
if operacao == "-":
    b1 = float(input("\nDigite o primeiro valor: "))
    b2 = float(input("Digite o segundo valor: "))
    subtracao = b1 - b2
    print("\nA subtracao entre", b1, "e", b2, "e:", subtracao, "\n")
    print("*****33, \n")
    continuar_usando = input("Gostaria de fazer outra operacao? ").upper()
    print("*****33, \n")

# Multiplicacao
if operacao == "*":
    c1 = float(input("\nDigite o primeiro valor: "))
    c2 = float(input("Digite o segundo valor: "))
    multiplicacao = c1 * c2
    print("\nA multiplicacao entre", c1, "e", c2, "e:", multiplicacao, "\n")
    print("*****33, \n")
    continuar_usando = input("Gostaria de fazer outra operacao? ").upper()
    print("*****33, \n")

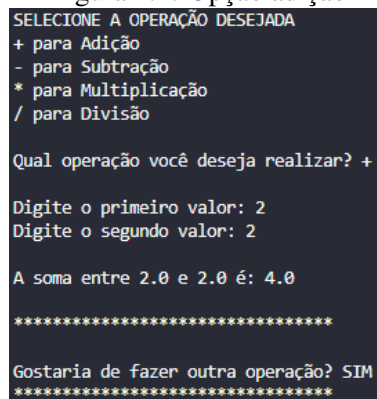
# Divisao
if operacao == "/":
    d1 = float(input("\nDigite o primeiro valor: "))
    d2 = float(input("Digite o segundo valor: "))
    while d2 == 0: # Garantindo que d2 nao seja zero!!
        print("O segundo valor nao pode ser zero!")
    d2 = float(input("\nDigite o segundo valor (diferente
```



```
de zero): "))
divisao = d1 / d2
print("\nA divisao entre", d1, "e", d2, "e:", divisao, "\n")
print("*****33, "\n")
continuar_usando = input("Gostaria de fazer outra operacao?
").upper()
print("*****33, "\n")
```

Como vemos no código, inicialmente uma variável chamada "continuar\_usando", é criada e definida como "SIM". Na estrutura de repetição While, enquanto a variável "continuar\_usando", for igual a "SIM", o laço continuará. Um menu de interação é criado dentro desse While, o menu oferece as seguintes opções de escolha, soma, subtração, multiplicação e divisão. A variável chamada "operação" tem a função de receber e guardar a opção desejada pelo usuário. Caso a opção escolhida seja "+", a operação de soma será realizada, apresentará o resultado e também irá perguntar se o usuário deseja continuar usando, caso a resposta não seja "SIM", o programa irá finalizar, funcionando da mesma forma para as outras operações, caso forem escolhidas. A seguir temos algumas imagens demonstrando o código de operações básicas rodando no Visual Studio Code.

Figura 4.1: Opção adição



```
SELECIONE A OPERAÇÃO DESEJADA
+ para Adição
- para Subtração
* para Multiplicação
/ para Divisão

Qual operação você deseja realizar? +

Digite o primeiro valor: 2
Digite o segundo valor: 2

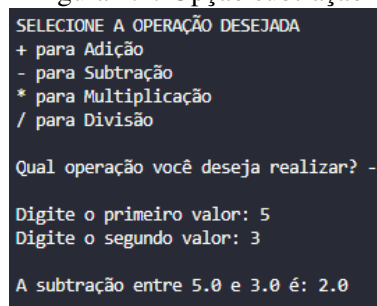
A soma entre 2.0 e 2.0 é: 4.0

*****

Gostaria de fazer outra operação? SIM
*****
```

Fonte: Autor

Figura 4.2: Opção subtração



```
SELECIONE A OPERAÇÃO DESEJADA
+ para Adição
- para Subtração
* para Multiplicação
/ para Divisão

Qual operação você deseja realizar? -

Digite o primeiro valor: 5
Digite o segundo valor: 3

A subtração entre 5.0 e 3.0 é: 2.0
```

Fonte: Autor

Figura 4.3: Opção multiplicação

```
Gostaria de fazer outra operação? SIM
*****

SELECIONE A OPERAÇÃO DESEJADA
+ para Adição
- para Subtração
* para Multiplicação
/ para Divisão

Qual operação você deseja realizar? *

Digite o primeiro valor: 2
Digite o segundo valor: 3

A multiplicação entre 2.0 e 3.0 é: 6.0

*****

Gostaria de fazer outra operação? sim
*****
```

Fonte: Autor

Figura 4.4: Opção divisão

```
SELECIONE A OPERAÇÃO DESEJADA
+ para Adição
- para Subtração
* para Multiplicação
/ para Divisão

Qual operação você deseja realizar? /

Digite o primeiro valor: 10
Digite o segundo valor: 2

A divisão entre 10.0 e 2.0 é: 5.0

*****

Gostaria de fazer outra operação? não
*****

PS C:\PLP\Trabalho-PLP\Arquivos\CodigosAplicacao>
```

Fonte: Autor

## 4.2 Programas gráficos

A seguir temos um exemplo de programa gráfico que é possível desenvolver utilizando a linguagem de programação Python e um framework chamado Tkinter. O Tkinter é uma biblioteca da linguagem Python, o mesmo vem instalado no pacote padrão de instalação do Python. Permitindo que qualquer computador que possua o interpretador Python instalado consiga desenvolver interfaces gráficas.

```
# Autor: Romulo Souza Fernandes
# E-mail: 00119110559@pq.uenf.br
# Data de criacao: 28/10/22
# Ciencia da Computacao - UENF
# Disciplina: PLP

# Importando todo conteudo do Tkinter
from tkinter import *

# Classe que exhibe os controles na tela
```

```
class Application:
    def __init__(self, master=None):
        # Criacao do primeiro container, chamado widget1
        self.widget1 = Frame(master)
        # Informando o gerenciador de geometria pack
        self.widget1.pack()
        # Utilizando o widget label para imprimir na tela
        self.msg = Label(self.widget1, text="Romulo
        Fernandes - UENF")
        self.msg["font"] = ("Verdana", "10", "italic",
            "bold")
        self.msg.pack()
        self.sair = Button(self.widget1)
        self.sair["text"] = "Sair"
        self.sair["font"] = ("Calibri", "10")
        self.sair["width"] = 5
        self.sair["command"] = self.widget1.quit
        self.sair.pack()

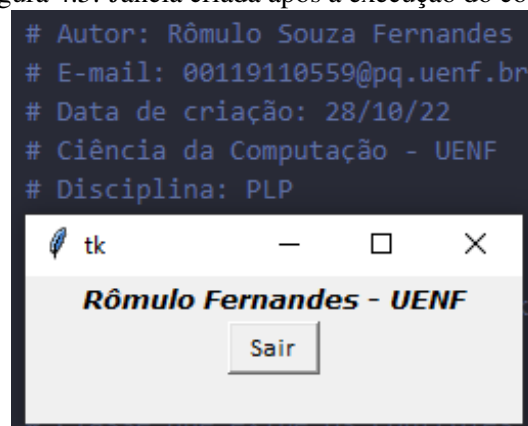
# Instanciando a classe Tk()
# Ela permite que os widgets sejam utilizados na aplicacao
root = Tk()

# Passando a variavel root como parametro do metodo
# construtor da classe Application
Application(root)

# Chamada do metodo para exibir na tela
root.mainloop()
```

A seguir, uma imagem demonstrando o programa gráfico rodando no Visual Studio Code.

Figura 4.5: Janela criada após a execução do código



Fonte: Autor

### 4.3 Programas com Objetos

Como sabemos, a Programação Orientada a Objetos (POO) é um dos paradigmas da linguagem Python, com base no conceito de classes e objetos. Abaixo temos um exemplo de código usando objetos na linguagem de programação Python.

```
# Autor: Romulo Souza Fernandes
# E-mail: 00119110559@pq.uenf.br
# Data de criacao: 28/10/22
# Ciencia da Computacao - UENF
# Disciplina: PLP

# Definindo a Classe pessoa
class Pessoa:
# Construtor
def __init__(self, nome: str, idade: int, altura: float):
self.nome = nome
self.idade = idade
self.altura = altura

# Definindo o metodo dizer_ola()
def dizer_ola(self):
print(f'Ola, meu nome e {self.nome}. Tenho {self.idade} ',
f'anos e minha altura e {self.altura}m.')

# Definindo o metodo cozinhar()
def cozinhar(self, receita: str):
print(f'Estou cozinhando um(a): {receita}')
```

```
# Definindo o metodo andar()
def andar(self, distancia: float):
print(f'Sai para andar. Volto quando completar {distancia} metros')
```

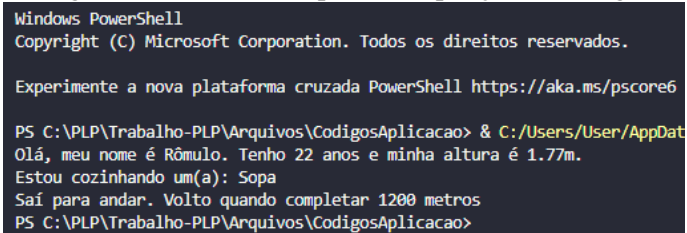
```
# Instancia um objeto da Classe "Pessoa"
pessoa = Pessoa(nome='Romulo', idade=22, altura=1.77)

# Chama os metodos de "Pessoa"
pessoa.dizer_ola()
pessoa.cozinhar('Sopa')
pessoa.andar(1200)
```

Iniciamos definindo classe pessoa, informando ao Python que definições para a nova classe serão criadas. Logo temos o `__init__`, é um método construtor, sendo chamado ao instanciar objetos, é nesse método que os atributos do objeto são setados. Após vem a definição dos métodos `dizer_ola()`, `cozinhar` e `andar()`. No método `dizer_ola()` é referenciado os atributos do próprio objeto. O método construtor `__init__`, é chamado quando `"pessoa = Pessoa()"`, passando nome, idade e altura como parâmetro.

A seguir temos uma imagem demonstrando o código utilizando objetos no Python, rodando no Visual Studio Code.

Figura 4.6: Resultado após a compilação do código



```
Windows PowerShell
Copyright (c) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\PLP\Trabalho-PLP\Arquivos\CodigosAplicacao> & C:/Users/User/AppDat
Olá, meu nome é Rômulo. Tenho 22 anos e minha altura é 1.77m.
Estou cozinhando um(a): Sopa
Saí para andar. Volto quando completar 1200 metros
PS C:\PLP\Trabalho-PLP\Arquivos\CodigosAplicacao>
```

Fonte: Autor

## 4.4 O algoritmo Quicksort

O código a seguir apresenta o algoritmo Quicksort no Python.

```
# Autor: Romulo Souza Fernandes
# E-mail: 00119110559@pq.uenf.br
# Data de criacao: 28/10/22
# Ciencia da Computacao - UENF
# Disciplina: PLP

# Python program for implementation of Quicksort Sort

# This implementation utilizes pivot as the last element in the nums list
# It has a pointer to keep track of the elements smaller than the pivot
# At the very end of partition() function, the pointer is swapped with the pivot
# to come up with a "sorted" nums relative to the pivot

# Function to find the partition position
def partition(array, low, high):

    # choose the rightmost element as pivot
    pivot = array[high]

    # pointer for greater element
    i = low - 1

    # traverse through all elements
    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:

            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

    # Swapping element at i with element at j
    (array[i], array[j]) = (array[j], array[i])
```

```
# Swap the pivot element with the greater element specified by i
(array[i + 1], array[high]) = (array[high], array[i + 1])

# Return the position from where partition is done
return i + 1

# function to perform quicksort

def quickSort(array, low, high):
    if low < high:

        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quickSort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quickSort(array, pi + 1, high)

data = [1, 7, 4, 1, 10, 9, -2]
print("Unsorted Array")
print(data)

size = len(data)

quickSort(data, 0, size - 1)

print('Sorted Array in Ascending Order:')
print(data)
```

A seguir temos algumas imagens demonstrando o código Quicksort rodando no Visual Studio Code.

## 4.5 Mobile

O código a seguir apresenta

```
# Autor: Romulo Souza Fernandes
# E-mail: 00119110559@pq.uenf.br
# Data de criacao: 28/10/22
# Ciencia da Computacao - UENF
# Disciplina: PLP

import kivy
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout
```

```
from kivy.uix.button import Button

kivy.require('1.9.1')

var = 0

def soma_um(instance):
    global var
    var += 1
    instance.text = str(var)

class MeuApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical',
            padding=[40, 20, 40, 20])

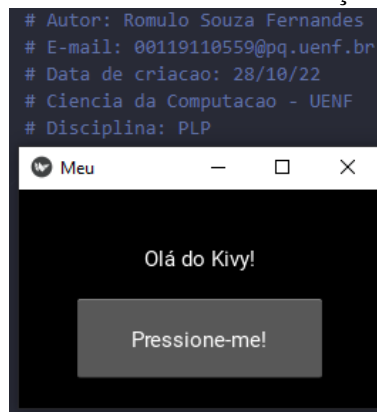
        layout.add_widget(Label(text='Ola do Kivy!'))
        btn = Button(text='Pressione-me!', size=(100, 50))

        btn.bind(on_press=soma_um)
        layout.add_widget(btn)
        return layout

if __name__ == '__main__':
    MeuApp().run()
```

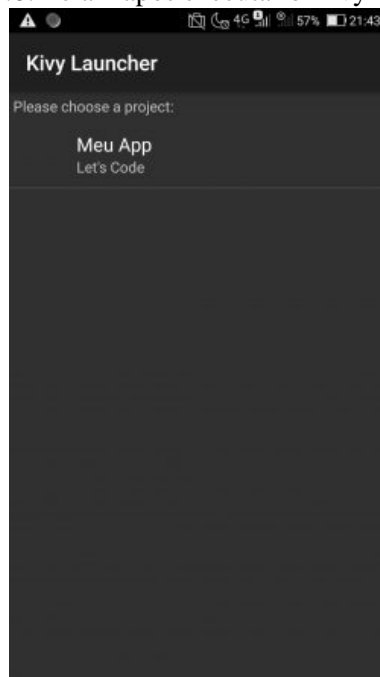
A seguir temos algumas imagens demonstrando o código da aplicação mobile rodando no Visual Studio Code.

Figura 4.7: Janela criada com a execução do código



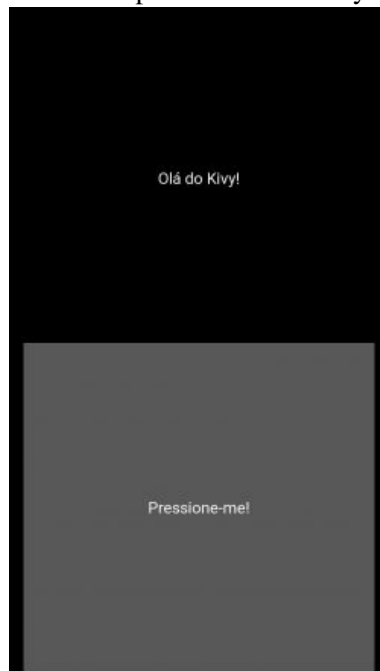
Fonte: Autor

Figura 4.8: Tela 1 após executar o Kivy no celular



Fonte: Autor

Figura 4.9: Tela 2 após executar o Kivy no celular



Fonte: Autor





## Referências Bibliográficas

- [Bor14] Luiz Eduardo Borges. *Python para desenvolvedores: aborda Python 3.3*. Novatec Editora, Sao Paulo, SP, 2014. Citado 6 vezes nas páginas 5, 9, 17, 20, 21 e 23.
- [Lut07] M. Lutz. *Aprendendo Python*. Bookman, Porto Alegre, RS, 2007. Citado 5 vezes nas páginas 6, 7, 9, 10 e 23.
- [McK19] Wes McKinney. *Python para analise de dados*. Novatec Editora, Sao Paulo, SP, May 2019. Citado 2 vezes nas páginas 6 e 7.
- [Mor18] Daniel Moreno. *Python para pentest*. Novatec Editora, Sao Paulo, SP, 2018. Citado na página 7.
- [Per16] Ljubomir Perkovic. *Introducao a computacao usando Python: um foco no desenvolvimento de aplicacoes*. Rio de Janeiro, RJ, 2016. Citado 7 vezes nas páginas 5, 9, 10, 14, 16, 17 e 23.
- [Sei15] Justin Seitz. *Black Hat Python: Python Programming for Hackers and Pentesters*. No Starch Press, Sao Paulo, SP, 2015. Citado na página 7.
- [Sev16] Dr. Charles Russell Severance. *Python for Everybody: Exploring Data in Python 3*. CreateSpace Independent Publishing Platform, Ann Arbor, MI, 1 edition, 2016. Citado 10 vezes nas páginas 9, 10, 11, 12, 13, 14, 15, 16, 20 e 23.



**Disciplina:** *Paradigmas de Linguagens de Programação 1970*

**Linguagem:** *Linguagem Python*

**Aluno:** *Rômulo Souza Fernandes*

### Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	Pontos
<b>Introdução (Máximo: 01 pontos)</b> <ul style="list-style-type: none"> <li>• Aspectos históricos</li> <li>• Áreas de Aplicação da linguagem</li> </ul>	
<b>Elementos básicos da linguagem (Máximo: 01 pontos)</b> <ul style="list-style-type: none"> <li>• Sintaxe (variáveis, constantes, comandos, operações, etc.)</li> <li>• Cada elemento com exemplos (código e execução)</li> </ul>	
<b>Aspectos Avançados da linguagem (Máximo: 2,0 pontos)</b> <ul style="list-style-type: none"> <li>• Sintaxe (variáveis, constantes, comandos, operações, etc.)</li> <li>• Cada elemento com exemplos (código e execução)</li> <li>• Exemplos com fonte diferenciada (listing)</li> </ul>	
<b>Mínimo 5 Aplicações completas - Aplicações (Máximo : 2,0 pontos)</b> <ul style="list-style-type: none"> <li>• Uso de rotinas-funções-procedimentos, E/S formatadas</li> <li>• Uma Calculadora</li> <li>• Gráficos</li> <li>• Algoritmo QuickSort</li> <li>• Outra aplicação</li> <li>• Outras aplicações ...</li> </ul>	
<b>Ferramentas (compiladores, interpretadores, etc.) (Máximo : 1,0 pontos)</b> <ul style="list-style-type: none"> <li>• Ferramentas utilizadas nos exemplos: pelo menos DUAS</li> <li>• Descrição de Ferramentas existentes: máximo 5</li> <li>• Mostrar as telas dos exemplos junto ao compilador-interpretador</li> <li>• Mostrar as telas dos resultados com o uso das ferramentas</li> <li>• Descrição das ferramentas (autor, versão, homepage, tipo, etc.)</li> </ul>	
<b>Organização do trabalho (Máximo: 01 ponto)</b> <ul style="list-style-type: none"> <li>• Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia</li> <li>• Cada elemento com exemplos (código e execução, ferramenta, nome do aluno)</li> </ul>	
<b>Uso de Bibliografia (Máximo: 01 ponto)</b> <ul style="list-style-type: none"> <li>• Livros: pelo menos 3</li> <li>• Artigos científicos: pelo menos 3 (IEEE Xplore, ACM Library)</li> <li>• Todas as Referências dentro do texto, tipo [ABC 04]</li> <li>• Evite Referências da Internet</li> </ul>	
<b>Conceito do Professor (Opcional: 01 ponto)</b>	
<p style="text-align: right;"><b>Nota Final do trabalho:</b></p>	

*Observação:* Requisitos mínimos significa a metade dos pontos