Vitor Amadeu Souza

Introdução a Teoria dos

Grafos

programado em

Python

Agosto de 2024

Direitos reservados por:
Cerne Tecnologia e Treinamento Ltda
Produção: Cerne Tecnologia e Treinamento
E-mail da Empresa: cerne@cerne-tec.com.br
Home Page: www.cerne-tec.com.br.com.br
Atendimento ao Consumidor: sac@cerne-tec.com.br
Contato com o Autor: vitor@cerne-tec.com.br

FEITO NO BRASIL

"Venha também sobre mim a tua benignidade, ó Senhor, e a tua salvação, segundo a tua palavra."

Sl 119:41

Kits Didáticos e Gravadores da Cerne Tecnologia

A Cerne tecnologia têm uma linha completa de aprendizado para os microcontroladores da família PIC, 8051, Holtek, dsPIC, ARM e etc. Veja os detalhes de cada um nas figuras abaixo:

Kit Arduino

- Gravação On-Board
- Comunicação Serial RS232
- Entrada de 12 V

Uma linha completa de componentes para o desenvolvimento de seus projetos eletrônicos como displays, PICs, botões, leds, cristais e etc. Visite a nossa página na Internet, no endereço **www.cerne-tec.com.br** e conheça melhor nossos serviços e produtos.

Cerne
Conhecimento para o Desenvolvimento

www.cerne-

Sumário

Capítulo I
Metodologia de desenvolvimento

1. Introdução

O estudo dos grafos é fundamental em muitas áreas da matemática e da ciência da computação, oferecendo uma base teórica para a análise e a resolução de problemas complexos que envolvem redes e conexões. O conceito de grafo, que pode ser descrito como um conjunto de vértices conectados por arestas, serve como um modelo poderoso para representar relações e estruturas em diversos contextos, desde redes sociais e sistemas de transporte até circuitos eletrônicos e biologia computacional.

O presente documento visa fornecer uma compreensão abrangente sobre os grafos, começando com uma introdução à metodologia de desenvolvimento dos conceitos abordados e seguindo com uma exploração detalhada das definições e características essenciais dos grafos. No Capítulo I, abordaremos a metodologia utilizada para a construção deste trabalho, destacando a importância de cada conceito e a abordagem adotada para a análise e a aplicação dos grafos.

O Capítulo II dedica-se a apresentar os conceitos fundamentais da programação em Python, pois o capítulo seguinte faz uso de tais definições para trabalhar com Grafos em Python.

No Capítulo III, mergulharemos profundamente nas diversas propriedades dos grafos. Iniciaremos com uma visão geral da história e da definição de grafos, passando por suas aplicações práticas e formas de representação. Exploraremos conceitos como a ordem de um grafo, o número de arestas, e as diferenças entre grafos triviais, vazios e multigrafos. Abordaremos também temas importantes como laços, vértices adjacentes, grau ou valência de um vértice, e a classificação de grafos regulares, completos e orientados.

Além disso, o capítulo examina representações de grafos por matrizes de adjacência e incidência, o custo de memória associado, e conceitos fundamentais de passeios, caminhos simples e ciclos. A discussão inclui grafos conexos e totalmente desconexos, além de ciclos e caminhos de Euler, e o famoso problema das Pontes de Königsberg.

A análise se estende a temas avançados como isomorfismo de grafos, teoremas importantes (como os de Dirac e Ore), e problemas

clássicos como o Caixeiro Viajante, com uma ênfase na programação dinâmica e na complexidade computacional associada.

Ao final, o documento também explora a teoria dos grafos planares, o Teorema de Kuratowski, e suas diversas aplicações. A inclusão de algoritmos fundamentais, como o de Kruskal e técnicas de busca em profundidade e largura, garante uma compreensão prática dos grafos e sua aplicabilidade em problemas reais.

Esta introdução fornece um panorama do conteúdo abordado e estabelece o contexto para uma exploração mais detalhada dos grafos e suas propriedades, métodos e aplicações. Ao longo deste documento, o objetivo é oferecer uma compreensão completa e integrada dos grafos, preparando o leitor para enfrentar desafios teóricos e práticos no campo da teoria dos grafos e suas aplicações.

Esta introdução deve ajudar a situar o leitor no contexto do estudo dos grafos, preparando-o para os detalhes e conceitos que serão abordados nos capítulos seguintes.

Capítulo II

Programação em Python

1. Introdução

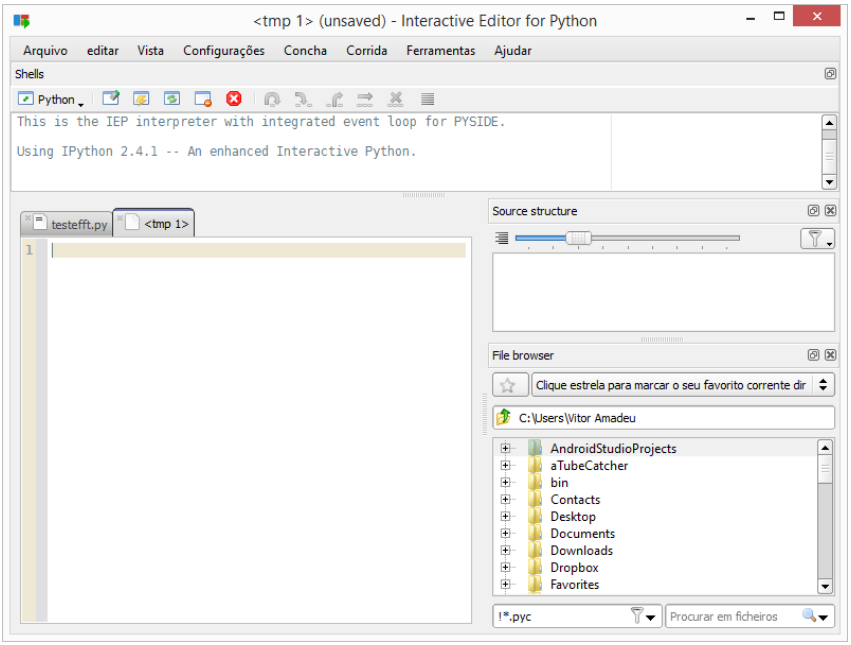
Nesta literatura a distribuição Pyzo foi utilizada, no qual a última versão pode ser baixada através do link abaixo.

http://www.pyzo.org/downloads.html

Baixe e instale também a última versão do Python disponível em:

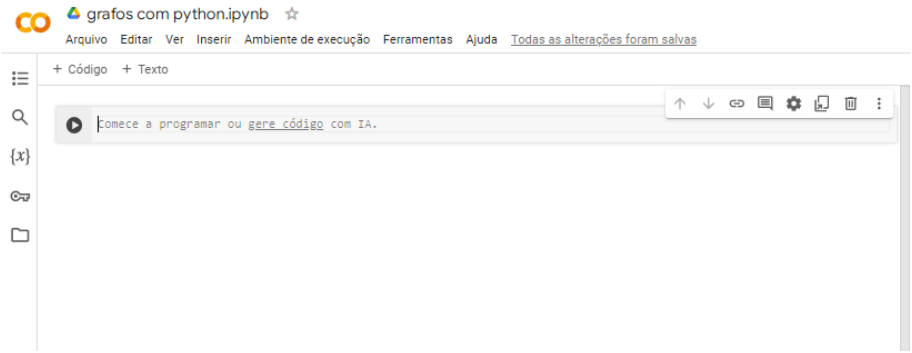
https://www.python.org/

A vantagem desta distribuição é que ela por padrão já vem com as bibliotecas a serem utilizadas ao longo desta obra. Após a instalação inicialize o programa, a tela a seguir será apresentada.



O Python é um software interpretado, ou seja, cada comando digitado no ambiente é logo executado após você pressionar o enter do teclado.

Outra possibilidade é usar o Google Colab, pois esta plataforma permite testar e validar os códigos elaborados em Python. Para acessá-lo, vá até o endereço https://colab.research.google.com/. Para criar um novo notebook, vá ao menu Arquivo → Novo notebook no drive.

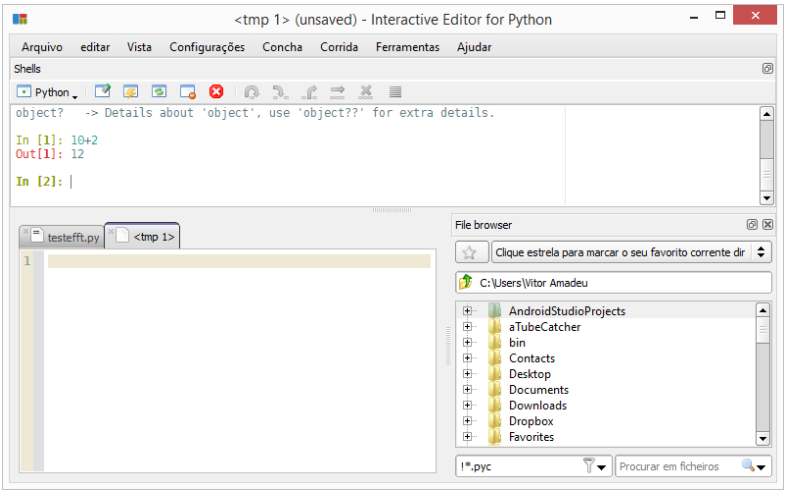


Após escrever o código, basta pressionar o botão Run para testar o código.

Nos próximos tópicos, exercitaremos diversos exemplos no Python, de forma a entender como o mesmo funciona.

2. Operadores aritméticos

Podemos usar o Python como uma calculadora, bastando neste caso digitar diretamente a expressão matemática no mesmo, usando neste caso o prompt. Observe abaixo:



Note que ao digitar a expressão e pressionar o enter, o comando é imediatamente processado, tendo como resultado a soma da operação. Outra forma é escrever o programa como um script, salvá-lo e executá-lo em seguida, indo no menu Corrida-> Corrida arquivo como script.

O Python possui diversos operadores aritméticos, como os citados a seguir.

Operador	Função
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão de inteiro
**	Exponenciação

Observe abaixo outros exemplos no Python, baseado nos comandos apresentados anteriormente:

```
>>> 10 - 6
4
>>>
```

No exemplo acima temos uma operação de subtração sendo executada, onde a temos como resultado 4, que é a subtração de 10 – 6.

```
>>> 2 ** 3
8
>>>
```

Acima temos um exemplo de exponenciação, onde o valor 2 elevado a 3 dá como resultado 8.

```
>>> 100 % 3
1
>>>
```

Acima a operação é diferente, pois temos a apresentação do operador resto da divisão de inteiros (%) onde 100 dividido por 3 dá 33, porém com resto 1 como verificado acima.

3. Operadores lógicos

Faz uma das operações lógicas fornecendo como resultado verdadeiro (True) ou falso (False). Acompanhe um exemplo com os três operadores lógicos.

```
>>> x=1
>>> y=0
>>> x and y
0
>>> x or y
1
>>> not x
False
>>> not y
True
```

4. Operadores de bits (bitwise operators)

Faz a operação lógica bit a bit entre duas variáveis ou constantes. Acompanhe um exemplo.

```
>>> x=0x0F
>>> y=0xF0
```

```
>>> x & y
0
>>> x | y
255
>>> ~x
-16
>>> x ^y
255
>>> x>>1
7
>>> y>>4
15
```

5. Funções de conversão

Utilizadas para converter bases diferentes.

```
>>> x=10
>>> hex(x)
'0xa'
>>> bin(x)
'0b1010'
>>> oct(x)
```

```
'0o12'
```

6. Comentários

Os comentários são uma forma bem interessante de documentar o que o seu programa faz e assim facilitar a sua leitura posteriormente. Todos os comentários no Python são feitos colocando-se o operador # na frente. Observe o exemplo abaixo:

```
Type "copyright", "credits" or "license()" for more information.
>>> 10 % 3 #informa o resto da divisão de 10 por 3 em inteiros
1
>>>
```

Coloque comentários à vontade em seu programa, bastando inserir o caracter # antes de iniciá-lo. Observe que até a cor do texto altera-se para vermelho, para ficar claro que aquele texto é um comentário.

7. Variáveis

As variáveis são um importante meio para se guardar informações no Python. Não é necessário que estas sejam declaradas, bastando neste caso atribuir as variáveis o seu valor de operação. É importante frisar que o Python é case sensitive, ou seja, ele diferencia caracteres maiúsculos e minúsculos. Observe abaixo a

definição de uma variável atribuindo a ela um valor do tipo string (conjunto de caracteres):

```
Type "copyright", "credits" or
>>> info="Treinamento Python"
>>>
```

Se quisermos ver o conteúdo desta variável basta que seja digitado o seu nome e em seguida, pressionado o botão Enter.

Observe outra definição de variável, neste caso uma variável inteira e decimal (float):

```
>>>
>>> var_inteira=10
>>> var_float=20.23
>>>
```

Para ver o conteúdo das variáveis o processo é o mesmo, bastando neste caso digitar o nome da variável e pressionar em seguida o enter. É importante frisar que o Python é case sensitive e neste caso, a declaração da variável *var_float* seguida de outra variável chamada *Var_float* serão diferentes, ou seja, ocuparão diferentes regiões da memória.

Temos três tipos principais de variáveis, que são as variáveis string, float e integer. Na variável do tipo string armazenamos

normalmente nomes, endereços, datas etc, ou seja, caracteres. Na variável do tipo float, armazenamos valores que contenham casas decimais, como por exemplo o valor de uma conta corrente. Na variável do tipo integer, valores inteiros como a idade de uma pessoa. Para sabermos o tipo de uma variável que está declarada no sistema, fazemos uso do comando type(nome_da_variável) como apresentado a seguir:

```
>>> var_inteira=10
>>> var_float=20.23
>>> type(var_float)
<class 'float'>
```

Após digitar type foi informado o nome da variável e em seguida apareceu o tipo que ela faz parte.

As variáveis declaradas não podem usar palavras reservadas da linguagem. O Python possui as seguintes:

```
and       del       from      None      True
as        elif      global    nonlocal  try
assert    else      if        not       while
break     except    import    or        with
class     False     in        pass      yield
continue  finaly    is        raise     def
for       lambda    return
```

8. Operadores Relacionais

Os operadores relacionais são usados para testar o conteúdo de variáveis para saber se, por acaso, alguma delas é igual, menor, maior etc. Veja abaixo a lista de operadores relacionais:

Operador	Função
==	Teste de Igualdade
!=	Diferente
>	Maior
<	Menor
>=	Maior ou Igual
<=	Menor ou Igual

Observe abaixo um exemplo para testarmos os operadores relacionais:

```
>>> a=2
>>> b=3
>>> a==b
False
```

Veja que no exemplo acima foram carregadas duas variáveis, sendo *a* com 2 e *b* com 3 e em seguida, usou-se o operador relacional == para checar se estas variáveis são iguais. O resultado foi False (Falso), como já era esperado. Observe a seguir outros testes condicionais feitos no Python:

```
>>> a=2
>>> b=3
>>> a==b
False
>>> a!=b     #a é diferente de b?
True
```

No exemplo acima é verificado se a é diferente de b, tendo como resultado true (verdadeiro).

```
>>> a=2
>>> b=3
>>> a==b
False
>>> a!=b     #a é diferente de b?
True
>>> a>b      #a é maior que b?
False
```

Já no exemplo anterior é usado o operador maior, obtendo o resultado falso.

```
>>> a<b      #a é menor que b?
True
```

Agora é usado o operador menor, tendo como resultado verdadeiro (true).

```
>>> a>=b
False
```

No exemplo acima foi verificado se a é maior ou igual a b, tendo como resultado falso.

```
>>> a<=b     #a é menor ou igual a b?
True
```

No exemplo acima foi testado se a é menor ou igual a b, tendo neste caso o resultado verdadeiro.

9. Trabalhando com strings

Faça o seguinte teste, atribua a variável *exemplo* o seguinte valor:

```
>>> exemplo="Elétrica"
```

Digite o nome da variável, neste caso *exemplo* e pressione enter, observe o que acontecerá.

```
>>> exemplo
'El\xe9trica'
```

Note que o valor digitado a princípio é diferente do que foi atribuído inicialmente no programa. Isso ocorre por que o Python trabalha com caracteres ASCII e neste conjunto não há caracteres

com acentuação. Para apresentarmos da forma correta a variável, devemos usar o comando print, como ilustrado a seguir:

```
>>> exemplo="Elétrica"
>>> exemplo
'El\xe9trica'
>>> print exemplo
Elétrica
```

Podemos também apresentar individualmente os caracteres presentes na variável exemplo. Por exemplo, temos ao todo 8 caracteres nesta string (Elétrica possui 8 caracteres) e se quisermos mostrar a posição 0 desta variável, teremos o seguinte resultado:

```
>>> exemplo="Elétrica"
>>> exemplo
'El\xe9trica'
>>> print exemplo
Elétrica
>>> print exemplo[0]
E
```

A mesma ideia vale para as outras opções, como no exemplo abaixo:

```
>>> print exemplo[7]
a
>>>
```