Aula 7

Docupedia Export

Author:Sobolevski Nycollas (CtP/ETS)
Date:26-Feb-2024 15:48

Table of Contents

1 Introdução à Programação em Python

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

2 Tratamento de exceções

- Mesmo que um comando ou expressão estejam sintaticamente corretos, talvez ocorra um erro na hora de sua execução. Erros detectados durante a execução são chamados **exceções** e não são necessariamente fatais: logo veremos como tratá-las em programas Python. A maioria das exceções não são tratadas pelos programas e acabam resultando em mensagens de erro:

- É possível escrever programas que tratam exceções específicas. Observe o exemplo seguinte, que pede dados ao usuário até que um inteiro válido seja fornecido:

```
while True:
    try:
        x = int(input("Digite um número: "))
        break
    except ValueError:
        print("Valor inválido! Tente novamente...")
```

- O bloco **try** testa um trecho de código a procura de erros;
- O bloco **except** define como vai ser tratado o erro;
- **ValueError** foi a exceção prevista para esse input, quando esse erro acontece o bloco **except** é executado. Para consultar os tipos de erro: https://docs.python.org/2/library/exceptions.html

- A função a seguir analisa se existe erro de atributo. A função sorte funciona para listas, mas para tuplas e strings não, pois são imutáveis.

```
def imprima_ordenado(colecao):
    try:
        colecao.sort()
    except AttributeError:
        print('Imutável')
        pass
    print(colecao,'\n')

imprima_ordenado([3,2,1])
imprima_ordenado((3,2,1))
imprima_ordenado('321')
```

```
[1, 2, 3]

Imutável
(3, 2, 1)

Imutável
321
```

- O bloco **finally** executa o trecho ocorrendo a exceção ou não, vejamos o exemplo a seguir:

```
while True:
    try:
        arquivo = open('exemplo.txt','w')
        x = int(input("Digite um número: "))
        arquivo.write(str(x))
    except ValueError:
        print("Valor inválido")
    finally:
        arquivo.close()
        print("Até mais!")
        break
```

- O arquivo txt deve ser fechado mesmo que ocorra o erro de input e nada seja escrito. Por isso é útil o **finally**, neste caso, para fechar o arquivo em qualquer ocasião.

- Podemos usar a keyword **raise** para gerar uma exceção, ela pode ser útil para levantar flags no programa.

```
def exemplo(x):
    if x < 0:
        raise StopIteration
    x = x - 1
    return x

x = 5

while True:
    try:
        x = exemplo(x)
        print(x)
    except StopIteration:
        break
```

- No exemplo acima o programa gera uma exceção para parar a iteração quando x é menor que 0.
- Também é possível criar novas exceções, assim não precisamos usar um erro padrão para levantar flags.

```
class NovoErro(Exception):
    pass
```

```
def exemplo(x):
    if x < 0:
        raise NovoErro
    x = x - 1
    return x

x = 5

while True:
    try:
        x = exemplo(x)
        print(x)
    except NovoErro:
        break
```

3 Exercício 7.1 – Calculadora a prova de erros

- Edite seu programa da calculadora para ser a prova de erros. (**ValueError, ZeroDivisionError**)
- Utilize o método **try-except**.
- TEMPO ESTIMADO: 30min

4 Correção – Calculadora a prova de erros

```
print("------------Calculadora---------------")

while True:
    while True:
        try:
            valor1 = int(input("Digite o primeiro valor: "))
            valor2 = int(input("Digite o segundo valor: "))
            break
        except ValueError:
            print("Valor inválido. Digite um número!!!")
    operacao = input("Digite a operação(+,-,*,/): ")
    if operacao == '+':
        resultado = valor1 + valor2
    elif operacao == '-':
        resultado = valor1 - valor2
    elif operacao == '*':
        resultado = valor1 * valor2
    elif operacao == '/':
        try:
            resultado = valor1 / valor2
        except ZeroDivisionError:
            print("Operação inválida. Divisão por zero!")
    else:
        print("Operação inválida!")
    try:
        print('{} {} {} = {}'.format(valor1,operacao,valor2,resultado))
    except NameError:
        pass
    repet = input("Para sair digite 0: ")
    if repet == '0':
        break
```

5 Manipulação de arquivos

- Em Python é possível editar arquivos de texto escrevendo qualquer coisa a partir do seu programa, também é possível importar informações de arquivos de texto para dentro do programa;
- Para isso aprenderemos as funções de leitura e escrita de arquivos;
- Para abrir um arquivo, utilizamos a função **open**.
- Ele tem dois parâmetros: o nome do arquivo, e o modo que vamos trabalhar, podendo ser ‘w’, ‘r’ ou ‘a’, para escrita ou leitura, o ‘a’ vem de **append** e escreve sem substituir o conteúdo.

```
arquivo = open('arquivo.txt','r')
```

```
arquivo = open('arquivo.txt','w')
```

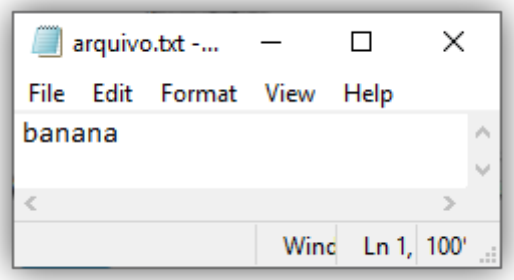
- Com o arquivo aberto no modo de escrita, podemos escrever usando a função **write**.

```
arquivo.write('banana')
```

- Se você checar o arquivo que foi editado, verá que não mudou nada, por que?
- Sempre devemos fechar o arquivo que abrimos, para ele atualizar seu conteúdo, para fechar utilizamos a função **close**.

```
arquivo.close()
```

- Agora se checarmos o conteúdo do arquivo:



- Abrindo o arquivo com o modo de leitura ‘r’, utilizamos a função **read** para ler o conteúdo do arquivo.

```
>>> arquivo = open('arquivo.txt','r')
>>> print(arquivo.read())
banana
```

- Se você tentar abrir novamente o arquivo no mesmo script, não será lido nada, pois o cursor de leitura ficou no fim do arquivo.
- Para ler o mesmo arquivo de novo, você precisa fechar e abrir de novo o arquivo.
- A função **strip** tira os caracteres especiais, como **\n**.

- Se quisermos ler linha por linha do arquivo podemos criar um **loop for** para fazer a leitura de cada linha.

```
for linha in arquivo:
    print(linha)
```

- Podemos simplificar tudo que fizemos usando o método **with**, ele é responsável por abrir, manipular, e fechar o arquivo, não precisando utilizar a função **close**.

```
with open('arquivo.txt') as arquivo:
    for linha in arquivo:
        print(linha)
```

6 Exercício 7.2 – Manipulando textos

- Crie um programa em que o usuário escolhe uma palavra e o programa diz quantas vezes essa palavra aparece em algum arquivo.
- DICA: Utilize as funções **split** e **strip**.
- TEMPO ESTIMADO: 30min

7 Correção – Manipulando textos

```
palavra_usuario = input("Digite a palavra: ")
arquivo = open('arquivo.txt','r')

contagem = 0
for palavra in arquivo:
    print(palavra)
    palavra = palavra.strip()
    palavra=palavra.split(' ')
    for i in palavra:
        if i == palavra_usuario:
            contagem+=1
print("A palavra \'{}\' aparece {} vezes".format(palavra_usuario,contagem))
```

8 Exercício 7.3 – Manipulando textos

1. Crie um arquivo de texto chamado `texto_exemplo.txt` com várias frases e palavras diferentes.
2. Escreva um programa em Python que realiza as seguintes tarefas:
    - Abre o arquivo `texto_exemplo.txt` em modo de leitura.
    - Lê o conteúdo do arquivo.
    - Conta o número de palavras distintas no arquivo.
    - Exibe a contagem de palavras distintas.
    - Coloque em outro arquivo a todas as palavras distintas com quebra de linha a cada uma delas.

9 Argumentos na linha de comando

- Quando queremos acessar o nosso programa a partir da linha de comando, não é estratégico ter a entrada de dados utilizando **input**, para fazer integrações entre programas e até mesmo para um acesso mais rápido é muito útil entrar com os argumentos diretamente na linha de comando.
- Por exemplo, suponha que criamos um programa que soma duas variáveis. É muito mais fácil executar o programa assim:

```
Desktop>python soma.py --var1 10 --var2 2
12
```

- Do que ter que dar aos poucos as informações ao programa, assim:

```
Desktop>python soma_2.py
Primeiro número: 10
Segundo número: 2
12
```

- Para fazer isso utilizaremos a biblioteca **argparse**.
- Com ele o nosso único trabalho é informar as variáveis esperadas e ele faz todo o trabalho de verificação e atribuição de entradas a variáveis internas.
- A primeira coisa a se fazer é importar o módulo **argparse** e criar um objeto **ArgumentParser**, que é o responsável por fazer a análise das entradas.

```
import argparse
parser = argparse.ArgumentParser(description = 'Programa exemplo')
```

- Depois de criado o parser, basta informar quais são as variáveis aceitas.

```
parser.add_argument('--var1', action='store', dest='var1',
                    required=True,help='Primeira variável criada')
```

- Agora o programa sabe que a entrada *--var1* é um argumento esperado, a ação a ser tomada é armazenar o valor desta entrada à variável **var1**, ela é uma entrada obrigatória, e se o usuário pedir uma mensagem de ajuda o programa irá explicar o que ela faz.
- Se não for uma entrada obrigatória, devemos atribuir um valor **default** para o argumento.
- Depois de declarar todos os argumentos, o parser deve fazer a verificação.

```
arguments = parser.parse_args()
```

- Assim, todos os argumentos dados serão armazenados no objeto **arguments** do tipo Namespace.
- Então para acessar internamente os valores, basta fazer:

```
var1 = arguments.var1
```

- Nosso programa está pronto! Agora é só utilizar as variáveis do jeito que quisermos.
- Devemos acessar o programa da seguinte forma:

```
python soma.py --var1 10 --var2 5
```

- Sempre colocar dois traços antes do nome da variável, seguido de um espaço e o seu respectivo valor.
- Para ver as informações do programa e das variáveis, chame o programa assim:

```
python soma.py -h
```

- Serão mostradas as informações dadas no **description** do parser e no parâmetro **help** dos argumentos.

10 Exercício 7.3 – Manipulando textos 2.0

- Repita o último exercício mas agora utilizando argumentos da linha de comando.
- O usuário deve acessar o programa com dois argumentos:
- O nome do arquivo a ser lido;
- A palavra a ser procurada.
- TEMPO ESTIMADO: 1 hora

11 Correção – Manipulando textos 2.0

```
import argparse

parser = argparse.ArgumentParser(description = 'Localizador de palavras')
parser.add_argument('--file',action='store',dest='arquivo',
                    required=True,help='Arquivo a ser lido')
parser.add_argument('--word',action='store',dest='palavra',
                    required=True,help='Palavra a ser procurada')
arguments = parser.parse_args()
palavra_usuario = arguments.palavra
arq = arguments.arquivo
arquivo = open(arq,'r')
contagem = 0
for palavra in arquivo:
    print(palavra)
    palavra = palavra.strip()
    palavra=palavra.split(' ')
    for i in palavra:
        if i == palavra_usuario:
            contagem+=1
print("A palavra \'{}\' aparece {} vezes".format(palavra_usuario,contagem))
```