MC-102 — Aula 15
Excessão e Arquivos

Instituto de Computação – Unicamp

26 de Setembro de 2019

Funções - parametros default

- Até agora, voce definia uma função com 3 parametros, na chamada voce precisava passar os 3 parametros

```
def f(a,b,c):
    ...

f(4,lista,"Jose")
```

- É possível definir parametros que tem um valor default que será usado se a chamada da função nao tiver esse parametro.
- parametros com valores default são declarados e usados usando o nome, o = e o valor

```
def f(a, b , c="jose")
    ...
f(4, lista)

f(4, lista , c="antonio")
```

veja a função **print** em
`https://docs.python.org/3/library/functions.html#print`

Excessão

- Vimos várias situações que geram um erro durante a execução de um programa:
- acessar um posição que não existe numa lista
- tentar converter um string p/ inteiro quando ele nao representa um
- dividir por zero
- acessar um dicionario com uma chave que nao existe

```
a=[1,2,3]
b={'b':1,  'c':5}
a[4]
int('  45b   ')
b['casa']
56/0
```

Excessão

- Estes erros são chamados de excessão:
- IndexError
- ValueError
- KeyError
- ZeroDivisionError

Excessão

- O comando **try except** permite tratar destes erros dentro do seu programa

```
try:
    comandos que podem gerar uma excessão
except:
    comandos que sao executados no caso de uma excessão
```

Excessão

```
def soma1(chave,dic):
    if chave in dic:
        dic[chave] = dic[chave]+1
    else:
        dic[chave] = 1


def soma1(chave, dic):
    try:
        dic[chave] = dic[chave]+1
    except:
        dic[chave] = 1
```

Tipos de arquivos

Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:

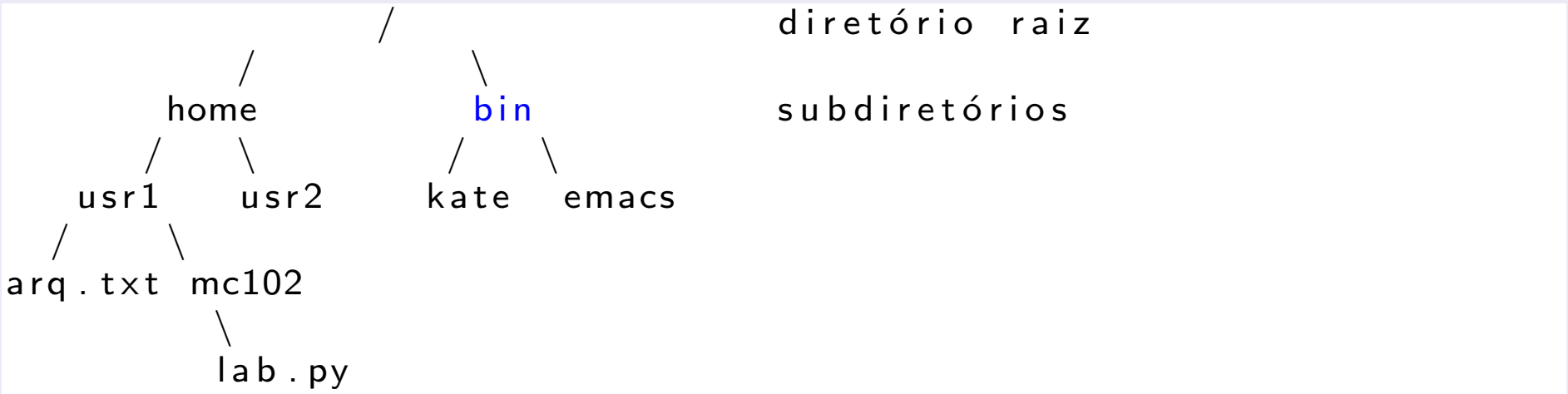
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: programa em Python, documento texto simples, páginas HTML.

Arquivo binário: Seqüência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word, etc

Diretório

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.

Uma hierarquia de diretórios



Caminhos absolutos ou relativos

O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz. Os caminhos podem ser especificados de duas formas:

Caminho absoluto: descrição de um caminho desde o diretório raiz.

```
/bin/emacs
/home/usr1/arq.txt
```

Caminho relativo: descrição de um caminho a partir do diretório corrente.

```
arq.txt
mc102/lab.py
```

Arquivos texto

- Para se trabalhar com arquivos devemos abri-lo e associá-lo com uma variável.
- A variável será um objeto do tipo **file** que contém métodos para ler e escrever no arquivo.
- O primeiro passo então é abrir o arquivo com o comando **open**:

```
var_arquivo = open("nome_do_arquivo", "modo")
```

- O **nome_do_arquivo** pode ser relativo ou absoluto.
- O **modo** pode ser "r" (leitura), "r+" (leitura e escrita), "w" (escrita), "a" (append).

```
arq = open("teste.txt", "r")
```

No exemplo acima, **arq** está associado com o arquivo **teste.txt** que foi aberto apenas para leitura.

Arquivos texto

```
arq = open("teste.txt", "r")
```

- O primeiro parâmetro para **open** é uma string com o nome do arquivo
  - Pode ser absoluto, por exemplo: "/user/eduardo/teste.txt"
  - Pode ser relativo como no exemplo acima: "teste.txt"
- O segundo parâmetro é uma string informando como o arquivo será aberto.
  - Se para leitura ou gravação de dados, ou ambos.
  - Se é texto ou se é binário.
  - No nosso exemplo o **r** significa que abrimos um arquivo texto para leitura.

Abrindo um arquivo texto para leitura

- Se abrirmos um arquivo para leitura e ele não existir, ocorrerá uma excessão FileNotFound:

```
>>> arq = open("blabla.txt", "r")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'naoExiste.tx
```

```
try:
    arq = open("blabla.txt", "r")
except:
    print("arquivo blabla.txt não existe")
```

Lendo dados de um arquivo texto

- Para ler dados do arquivo aberto, usamos o método **read**.
  - **read(num_bytes)**: Retorna uma string contendo os próximos **num_bytes** do arquivo.
  - **read()**: Sem parâmetro é retornado uma string contendo todo o arquivo! Se o arquivo tiver Gigas de tamanho o problema será seu para lidar com isso!

```
arq = open("teste.txt", "r")
conteudo = arq.read()
```

Lendo dados de um arquivo texto

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado para o próximo dado não lido.
- Eventualmente o indicador de posição chega ao fim do arquivo:
  - O método **read** devolve uma string vazia caso o indicador de posição esteja no fim do arquivo.

O exemplo abaixo mostra o conteúdo do arquivo **teste.txt** na tela.

```
arq = open("teste.txt", "r")
while True:
    s = arq.read(1)
    print(s,end="")
    if(s == ""):
        break
arq.close()
```

Lendo dados de um arquivo texto

```
arq = open("teste.txt", "r")
while True:
    s = arq.read(1)
    print(s,end="")
    if(s == ""):
        break
arq.close()
```

- O método **close** deve sempre ser usado para fechar um arquivo que foi aberto.
  - Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.
  - Ele também libera recursos que são alocados para manter a associação da variável com o arquivo.

Lendo dados de um arquivo texto

- O programa anterior pode ser alterado para ler todo o arquivo de uma vez.
  - Mas lembre-se que se o arquivo for muito grande isto pode acarretar em uma sobrecarga da memória do seu computador fazendo com que este fique lento ou mesmo trave.

```
arq = open("teste.txt", "r")
s = arq.read()
print(s,end="")
arq.close()
```

Lendo dados de um arquivo texto

- Uma maneira mais eficiente do que se ler um *byte* por vez e menos arriscada do que se ler todo o arquivo de uma única vez, é ler uma linha por vez.
- Para isso usamos o método **readline()** que devolve uma linha do arquivo em formato string.

```
arq = open("teste.txt", "r")
while True:
    s = arq.readline()
    print(s)
    if(s == ""):
        break
arq.close()
```

Lendo dados de um arquivo texto

- O **for** funciona num arquivo texto e retorna uma linha por vez.

```
arq = open("teste.txt", "r")
for s in arq:
    print(s)
arq.close()
```

Escrevendo dados em um arquivo texto

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada usando o modo **w**, **a** ou **r+**.
- **arq = open("nome_arquivo", "modo")**
  - **w**: se o arquivo existir ele será sobreescrito, ou seja todo o conteúdo anterior será apagado.
  - **a**: o indicador de posição ficará no fim do arquivo, e dados escritos serão adicionados no fim do arquivo.
  - **r+**: o indicador de posição ficará no início do arquivo, e dados serão escritos sobre dados anteriores.

Escrevendo dados em um arquivo texto

- O metodo **write** do arquivo escreve um string no arquivo texto.
- o **write** NAO insere uma mudança de linha ao final. Se voce quer a mudança de linha o string tem que conter esse caracter ('\n')
- nao existe um comando do tipo o **print** (que converte outros tipos para string) para arquivos. VOce tem que fazer a conversão usando o **str**

Escrevendo dados em um arquivo texto

Uma funçao que recebe uma lista de floats e os imprime no arquivo arq

```
def imprimearq(arq, lista):
  saida = ''
  for i in lista:
    x = str(i)
    saida = saida + ' ' + x
  saida = saida + '\n'
  arq.write(saida)
```

Modos do open

```
open(nome_do_arquivo, modo);
```

Modos de abertura de arquivo texto

modo	operações	indicador de posição começa
r	leitura	início do arquivo
r+	leitura e escrita	início do arquivo
w	escrita	início do arquivo
a	(append) escrita	final do arquivo

Lembre-se: open

- Se um arquivo for aberto para leitura (**r**) e ele não existir, **open** gera um erro.
- Se um arquivo for para escrita (**w**) e existir ele é sobrescrito. Se o arquivo não existir um novo arquivo é criado.
- Se um arquivo for aberto para leitura/escrita (**r+**) e existir ele NÃO é apagado. Se o arquivo não existir, **open** gera um erro.

Exemplo

Lei um arquivo com numeros separados por virgulas e retorne uma lista dos numeros

```
def learq(nome):
  out = []
  try:
      arq = open(nome, "r+")
      t = arq.read()
      x = split(',')
      for i in t:
          try:
            n = int(i)
            out.append(n)
          except:
            print("problemas com o dado",i)
      arq.close()
  except:
    print("Problemas com o arquivo", nome)
  return out
```

Exemplo

Programa que altera arquivo texto trocando ocorrências de 'a' por 'A'.

```
try:
    arq = open("teste.txt", "r+")
    t = arq.read()
    t = list(t) #transformamos em lista
    for i in range(len(t)):
        if(t[i] == 'a'):
          t[i] = 'A'
    arq.seek(0,0)  <- vai p/ o inicio
    t = "".join(t)
    arq.write(t)
    arq.close()
except:
    print("Problemas no arquivo teste.txt")
%
```