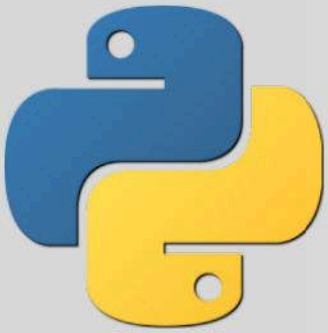
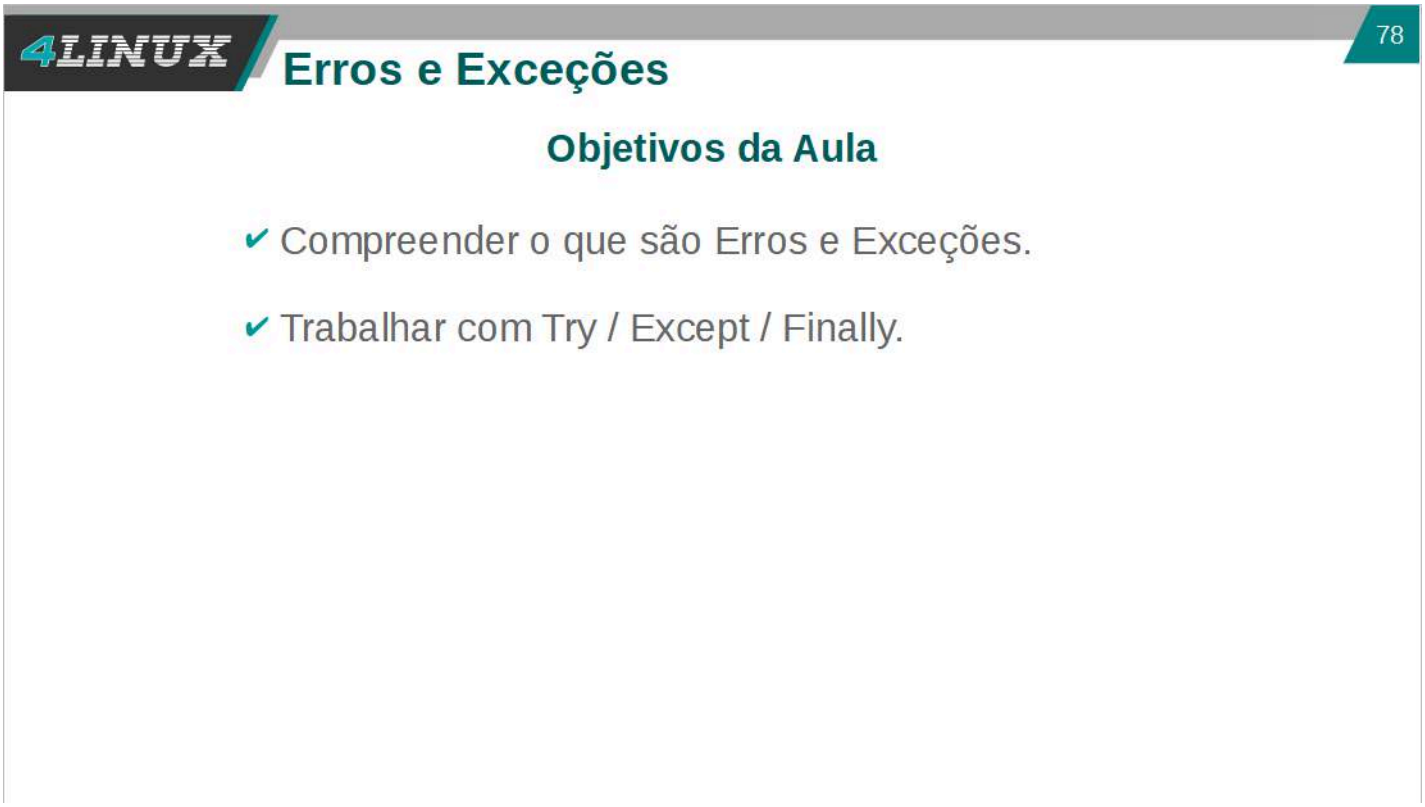
Python Fundamentals



Erros e Exceções

4.1

Anotações



Erros e Exceções

Nesta aula apresentaremos o conceito sobre erros e exceções, veremos quais as suas diferenças, como o Python trabalha com cada tipo e implementaremos o tratamento de erro em nossa aplicação.

Anotações

Erros e Exceções

Em Python os erros são divididos em dois grupos: erros de sintaxe e exceções.
Os erros de sintaxe ou, erros de análise, consistem o tipo mais comum quando estamos aprendendo. Acontecem ao executar o script, o interpretador exibe uma linha indicando “SyntaxError” com uma “seta” apontando onde o erro foi encontrado.

```
    if 'Mariana' == nome
                        ^
SyntaxError: invalid syntax
```

Anotações

Erros e Exceções

Exceções

✔ São geradas pela aplicação, mesmo que a sintaxe esteja correta, indicam que está ocorrendo algum erro ao executá-la.

✔ Exceções podem ser fatais, ou seja, sua aplicação continuará funcionando, mesmo que ocorra alguma, porém sempre devemos tratá-las.

Anotações

Erros e Exceções

✔ Exceções não são tratadas automaticamente, porém exibem mensagens de erro que podem nos ajudar. Exemplo:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>

TypeError: Can't convert 'int' object to str implicitly
```

Exceções

Um exemplo: tentar fazer uma comparação com uma variável, porém sem definir o que será a variável, veja:

```
#!/usr/bin/python3
if 'mariana' == nome:
    print('nome correto')
else:
    print('nome errado')
```

Ao executar o script, a seguinte exceção será gerada:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'nome' is not defined
```

A última mensagem indica o que ocorreu. Mostra o tipo da exceção, seguida de maiores detalhes sobre a ocorrência. O tipo da exceção é “NameError”, sugere que a variável “nome” não foi definida.

Erros e Exceções

Tratamento de exceções

✔ Existem diversos tipos de exceções que conseguimos manipular, evitando, caso ocorram em um trecho de código, que o nosso script seja finalizado.
✔ Para isto, usaremos o try / except:

```
try:
  comandos
except Except_Type:
    comandos
```

```
tente:
  comandos
exceção tipo_da_exceção:
    comandos
```

Como funciona o Try/Except

1 - Os comandos que fazem parte do try são executados.
2 - Caso nenhuma exceção ocorra, o except é ignorado e a execução é concluída.
3 - Ocorrendo alguma exceção durante a execução do try, o resto do código é ignorado, se o Exception Type corresponder ao ocorrido, o except é executado.
4 - Caso ocorra erro e o Exception Type não corresponder, a execução do script será parada.

O principal objetivo ao colocar código dentro de um try / except, é garantir que continue sendo executado caso ocorra algum erro.

Anotações

Erros e Exceções

Utilizando o try/except

```
#!/usr/bin/python3
while True:
    try:
        x = int(input("digite o primeiro numero: "))
        y = int(input("digite o segundo numero: "))
        print (x + y )
    except Exception as e:
        print ("Digite apenas números")
```

Anotações

Erros e Exceções

Finally

- ✔ O try/except possui uma opção destinada a executar independente da ocorrência ou não de exception.
- ✔ Para implementar o finally, adicionamos abaixo da Exception:

```
finally:
    comandos
```

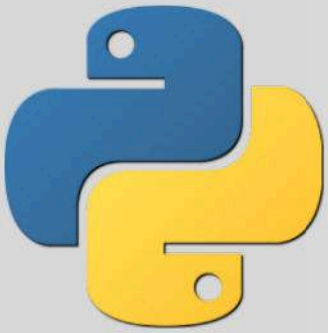
O **"finally"** será executado independente da ocorrência ou não de exceção no seu código. Significa, seja qual for o resultado do código, o que está abaixo dele será executado. Podemos implementar o finally no exemplo anterior da seguinte forma:

```
try:
    x = int(input("digite o primeiro numero: "))
    y = int(input("digite o segundo numero: "))
    print (x + y )
except Exception as e:
    print("Digite apenas números")
Finally:
    print("Saindo do script")
```

Quando implementamos log em nossa aplicação, as linhas de código podem ficar no finally. Independentemente de ocorrer erro ou execução com sucesso, o log será gravado e assim, teremos um controle dos horários da execução ou do erro acontecido.

Anotações

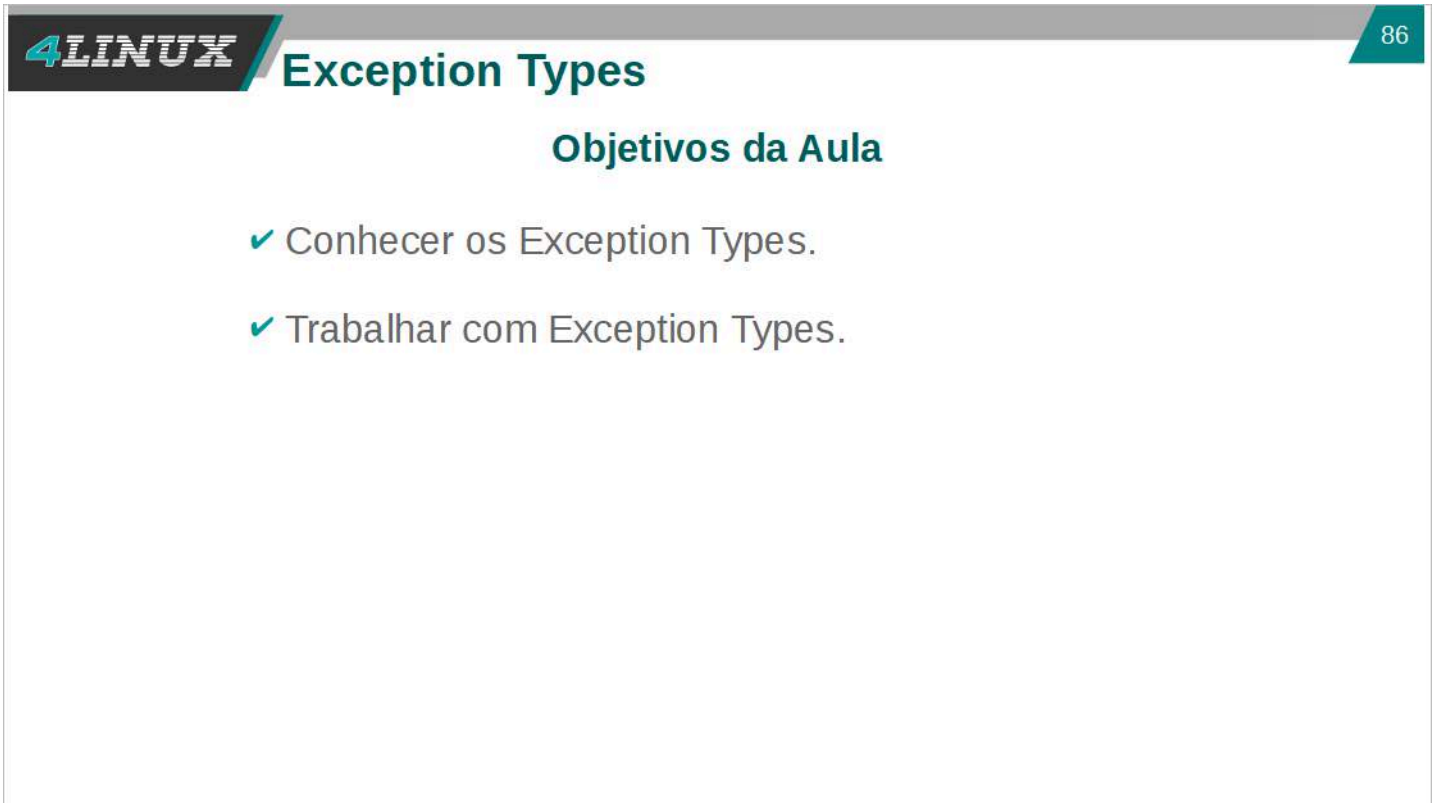
Python Fundamentals



Exception Types

4.2

Anotações



Exception Types

Nesta aula iremos apresentar alguns dos Exception Types existentes em Python e como podemos trabalhar com eles.

Anotações

Exception Types

Exception Types

Quando definimos um bloco de código utilizando o try/except temos duas opções: podemos definir uma **exception genérica**, e qualquer exceção cairá neste bloco ou, podemos definir **exception específica** para determinadas situações.

O melhor, é definirmos as exceptions específicas, assim conseguimos maior controle da nossa aplicação.

Anotações

Exception Types

Trabalhando com Exception Types

Para definir uma exceção genérica:

```
try:
    x = 5
    y = 0
    z = x/y
except Exception as e:
    print ('Erro: {}'.format(e))
```

Anotações

Exception Types

Trabalhando com Exception Types

Podemos utilizar Exception Types especificas, por exemplo:

NameError: quando o nome de uma variável não é encontrado.
TypeError: quando o tipo do objeto é inapropriado.
IndexError: quando o índice de um dicionário não é encontrado.
KeyError: quando a chave de um dicionário não é encontrada.

Anotações

Exception Types

NameError

```
#!/usr/bin/python3

try:
    print ('A Linguagem é: {}'.format(linguagem))

except NameError as e:
    print(e)
```

Anotações

Exception Types

TypeError

```
#!/usr/bin/python3

try:
    numero = 10
    print('O número é: '+
numero)

except TypeError as e:
    print(e)
```

Anotações

Exception Types

IndexError

```
#!/usr/bin/python3

try:
    linguagem = ['python']
    print ('A Linguagem e: {}'.format(linguagem[2]))

except IndexError as e:
    print ('Erro: {}'.format(e))
```

Anotações

Exception Types

KeyError

```
#!/usr/bin/python3

try:
    linguagem = {'curso':'fundamentals'}
    print ('A Linguagem é: {}'.format(linguagem["nome"]))

except KeyError as e:
    print ('Erro: {}'.format(e))
```

Anotações

Exception Types

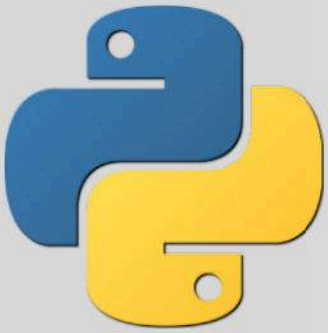
Definindo mais de um Exception Type:

Podemos definir mais de um Exception Type, observe:

```
except NameError as e:
    print ('Erro: {}'.format(e))
except KeyError as e:
    print ('Erro: {}'.format(e))
```

Anotações

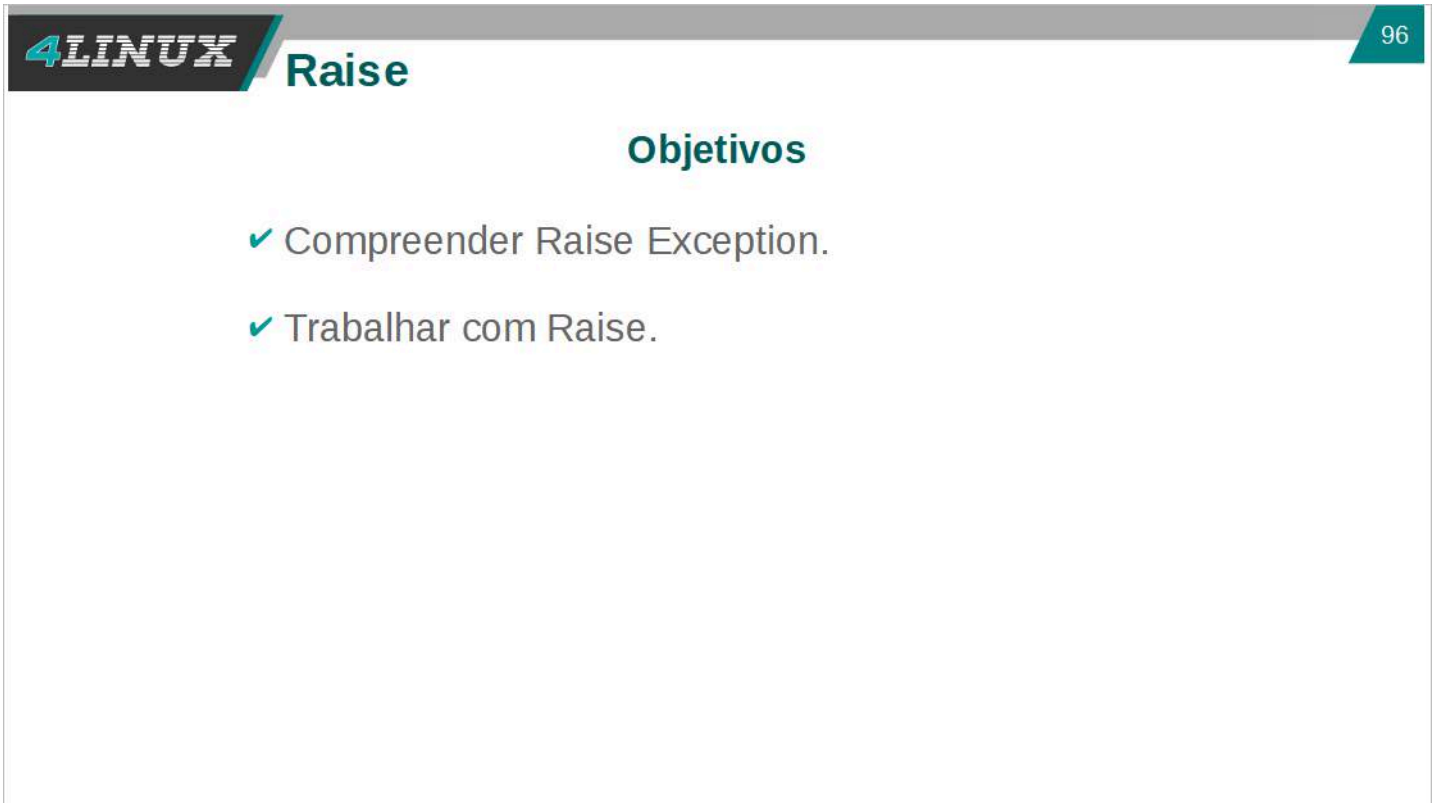
Python Fundamentals



Raise Exception

4.3

Anotações



Raise

Nesta aula iremos entender o que são as Raise Exceptions e como podemos trabalhar com elas.

Anotações

Raise

No Python podemos criar nossas próprias exceções, possibilitam a aplicação ou script tratar determinada situação sem parar a execução.

A declaração raise permite criar estas exceções, ou seja, forço o disparo de uma exceção para o meu sistema.

Por exemplo, podemos criar uma verificação de id de usuário, caso este número retorne vazio, criaremos uma exceção e desta forma a nossa aplicação continuará funcionando.

Anotações

Raise

- ✔ Raise aceita qualquer tipo de Type Error, semelhante ao Except.
- ✔ Para que funcione como desejamos, o mesmo Type Error deverá ser definido em raise e no Except.
- ✔ Normalmente ao utilizar raise, o colocamos em um bloco de código if/else.
- ✔ Caso o retorno não seja o esperado, forçará uma exceção. o script continuará rodando.

Anotações

Raise

Sintaxe

```
try:
  comandos
  raise Except_Type
except Except_Type:
    comandos / raise
```

```
tente:
  comandos
  criando tipo_da_exceção
exceção tipo_da_exceção:
    comandos
```

Anotações

Raise

Trabalhando com raise

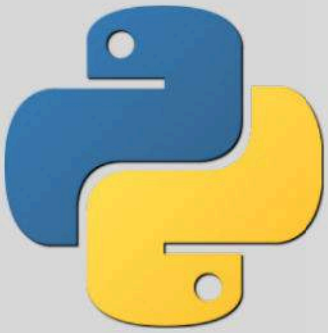
```
nome = 'python'
try:
    if nome == '4linux':
        print ('o nome esta correto')
    else:
        raise ValueError('O campo nome não
                          é correspondente')
except ValueError as e:
    print (e)
```

Raise

Definimos um bloco de condição caso o variável não tenha o valor que esperamos, criamos uma exceção e logo em seguida tratamos a mesma.

Anotações

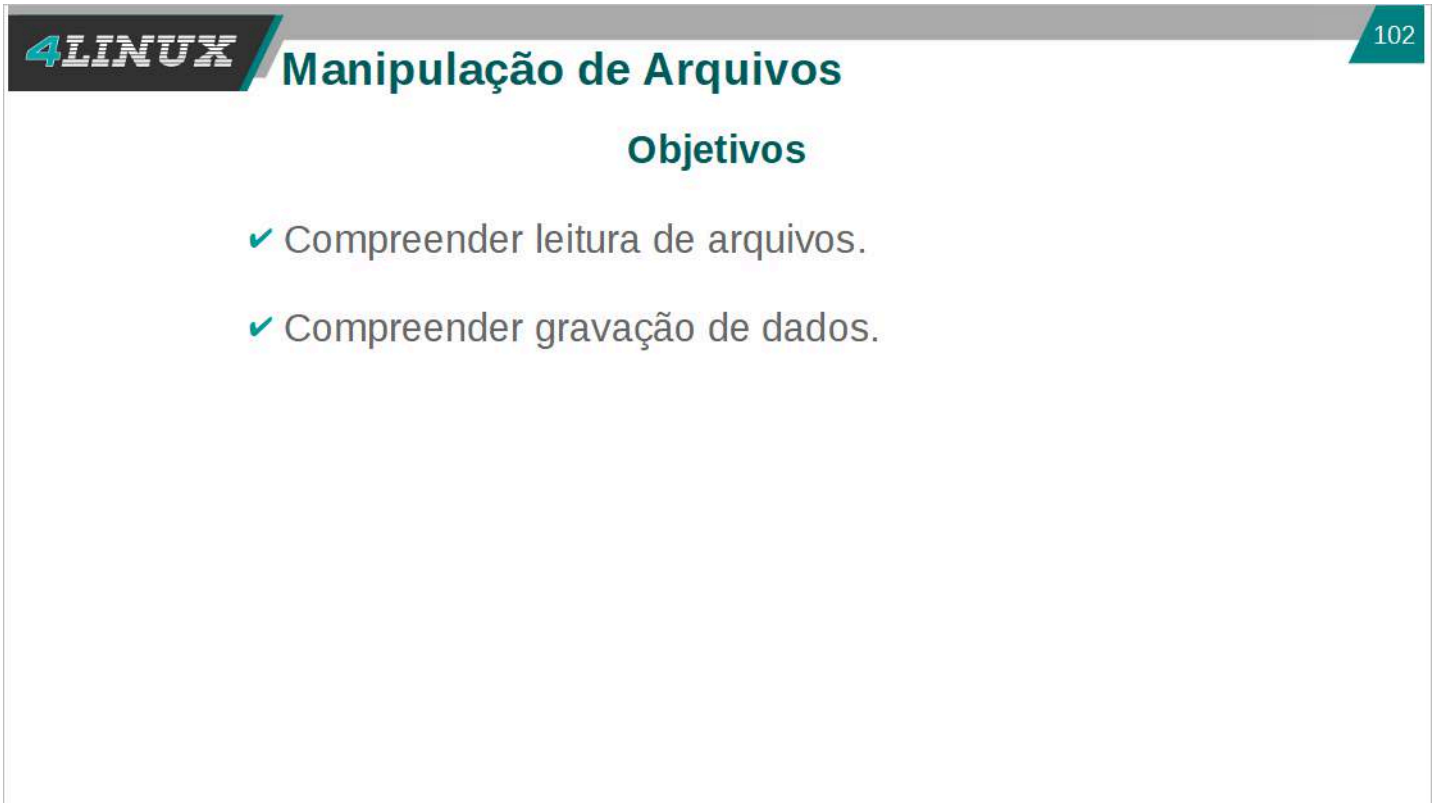
Python Fundamentals



Manipulação de Arquivos

4.4

Anotações



Manipulação de Arquivos

Nesta aula trabalharemos com arquivos, veremos métodos disponíveis para realizar a leitura e escrita em arquivos de texto.

Anotações

Manipulação de Arquivos

✔ Para trabalhar com arquivos, teremos que criar um objeto que nos disponibilizará métodos como read ou write.

✔ O acesso ou leitura ao arquivo dependerá do modo em que o objeto foi criado.

✔ Existem duas categorias de objetos de arquivo: arquivos binários e arquivos de texto.

Manipulação de Arquivos

Arquivos binários: um objeto de arquivo do tipo binário, é capaz de ler e escrever em bytes, exemplo: gzip ou exe. Para abrir este tipo de arquivo, utilizamos: 'rb', 'wb' ou 'rb+'.
Arquivos de texto: este tipo de arquivo é capaz de ler e escrever objetos do tipo **"string"** que podem ser abertos com os seguintes comandos: 'r', 'w' ou 'r+'.

Modos e trabalho com o arquivo.

r – utilizada para leitura de arquivos.
w – usada para a escrita em arquivos.
r+ – empregada para a leitura e escrita em arquivos.

Em arquivos binários colocamos o "b" na frente dos modos, para especificar que o arquivo é deste tipo.

Anotações

Manipulação de Arquivos

- Todos os dados usados até agora foram inseridos pelo usuário ou colocados diretamente no código.
- Ao trabalhar com arquivos, a sintaxe será:

```
open(nome_do_arquivo, 'modo')
```

- **"open()"** retornará um objeto de arquivo, sempre utilizado seguindo o padrão: nome do arquivo e o modo, leitura ou escrita.

Anotações

Manipulação de Arquivos

Modos disponíveis

Os modos disponíveis para trabalhar com arquivos são:

Podemos combinar alguns modos. Exemplo: r+ (abrirá o arquivo, permitindo leitura e escrita).

Modo	Significado
'r'	Abre o arquivo para leitura
'w'	Abre o arquivo para escrita (sobrescreve)
'x'	Abre para criação (falha caso o arquivo exista)
'a'	Abre para escrita (acrescenta no arquivo)
'+'	Abre um arquivo para atualização (leitura e escrita)

Modos de abertura

Podemos utilizar outros modos quando trabalhamos com arquivos. Há um modo específico para trabalhar com arquivos do tipo binário.
A documentação do Python 3 mostra como utilizar este modo:
https://docs.python.org/3/library/functions.html#open

Anotações

Manipulação de Arquivos

Escrevendo em arquivos

```
#/usr/bin/python3
with open('arquivo', 'w') as f:
   f.write('Curso de Python')
```

Desta forma os dados serão sobrescritos. Este modo é indicado para gravar os dados inicialmente.

Trabalhando com arquivos

Quando trabalhamos com arquivos, não é obrigatório o uso do **"with".** Porém, trata-se de uma boa Prática que recomendamos utilizar. Elimina a necessidade de fechar o arquivo após finalizar a execução do necessário, esta é a principal vantagem que nos trás.
Para fazer a gravação sem utilizar o **"with"** podemos fazer da seguinte forma:

```
f  =  open ( 'workfile' ,  'w' )
f.write('Curso de Python')
f.close()
```

Note, é necessário fechar o arquivo após fazer a inserção dos dados.

Anotações

Manipulação de Arquivos

Podemos escrever em um determinado ponto do arquivo:

```
#/usr/bin/python3
with open('arquivo', 'w') as f:
   f.seek(0)
f.write('Curso de Python')
```

"seek" definirá a posição do meu ponteiro, onde 0 (zero) indica a primeira linha.

Anotações

Manipulação de Arquivos

Lendo arquivos

```
#/usr/bin/python3
with open('arquivo', 'r+') as f:
   print(f.read())
```

read permite ler todo o arquivo. Também contamos com **readlines** que trará todo o conteúdo em forma de lista.

Lendo arquivos

Podemos empregar os dois métodos ao utilizar conteúdo a partir de arquivos de texto.
Para obter uma linha específica do arquivo, podermos fazer da seguinte forma:

```
/usr/bin/python3
with open('teste', 'r+') as f:
        b = f.readlines()
        print b[1]
```

Como o retorno de readlines é uma lista, o acesso a um objeto se dá através do índice, a partir de então, consigo ter o retorno de uma só linha.

Anotações