

# Live de python #34

Trabalhando com arquivos de texto

# Roteiro

- A função `open()`
  - Modos de criação
  - Lendo um arquivo
  - Escrevendo um arquivo
- Entendendo o mundo existente em `open()`
- IO
  - Hierarquia de classes
  - `IOBase`
  - `TextIOBase`
  - `TextIOWrapper`

# Função open()

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

```
In [1]: arquivo = open('teste.txt')  
  
In [2]: arquivo.read()  
Out[2]: 'kkkkkkk\nzzzzzzzz\nyyyyyyyy\n'
```

# Função open()

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

```
In [1]: arquivo = open('teste.txt')  
  
In [2]: arquivo.read()  
Out[2]: 'kkkkkkk\nzzzzzzzz\nyyyyyyyy\n'
```

Duas linhas dizem  
mais que 1000  
palavras

# Função open()

Abre a o arquivo e devolve um objeto correspondente. Caso o arquivo não puder ser aberto retornará um OSError

- Python 2 X Python 3
  - No python2 será criado um objeto 'file' (isso é passado, vamos migrar?)
  - No python3 um objeto io

# Função open(mode=?) - Modos

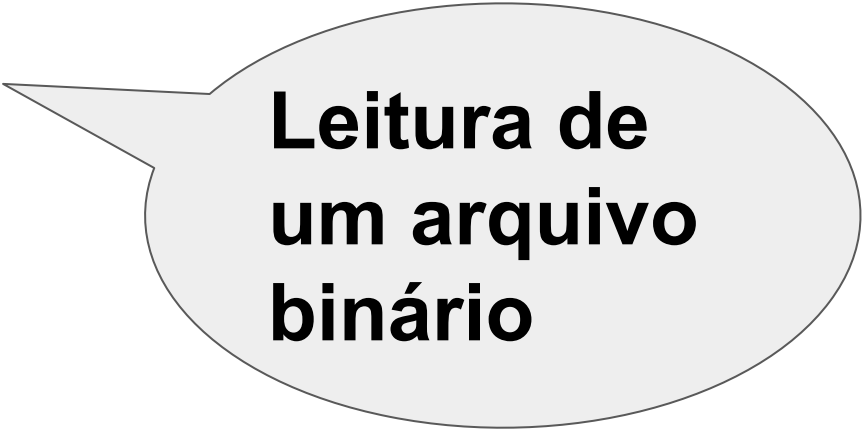
Modo	Significado
<b>'r'</b>	<b>Abre o arquivo em modo de leitura (default)</b>
<b>'w'</b>	<b>Abre o arquivo para escrita, “inicia um novo arquivo, caso já exista”</b>
<b>'a'</b>	<b>Cria um arquivo, retorna erro, caso já exista</b>
<b>'t'</b>	<b>modo texto (default)</b>
<b>'+'</b>	<b>Deixa o arquivo aberto, para leitura e escrita</b>
<b>'b'</b>	<b>Especifica que o arquivo é binário</b>
<b>'U'</b>	<b>Não vou explicar, vai sair na próxima release</b>

## Função open(mode=?) - Modos

`open(<file>, 'rb')`

`open(<file>, 'w')`

`open(<file>, 'a')`



**Leitura de  
um arquivo  
binário**

## Função open(mode=?) - Modos

`open(<file>, 'rb')`

`open(<file>, 'w')`

`open(<file>, 'a')`



**Escrita de  
um arquivo  
de texto**



## Função open(mode=?) - Modos

`open(<file>, 'rb')`

`open(<file>, 'w')`

`open(<file>, 'a')`



**“continuar”  
escrevendo  
no arquivo**

CODE !! [parte 1]

# Explicando parâmetros - Buffering

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

**buffering**: tamanho que será ocupado em memória durante a leitura. Neste caso o tamanho do blocos (chunks) que serão “armazenados”.

**Default**: -1 (nesse caso o python vai escolher o tamanho dos blocos por você)

Você pode optar por 0 (sem buffer) em alguns casos (só funciona com leituras binárias)

# Explicando parâmetros - Buffering

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

**Como o python escolhe por mim?**

O python vai chamar uma função chamada `fstat` e passar seu arquivo como parâmetro. Caso não consiga, vai usar o padrão de IO.

# Explicando parâmetros - Buffering

```
In [2]: import os
```

```
In [3]: os.stat('teste.txt').st_blksize
```

```
Out[3]: 4096
```

```
In [4]: import io
```

```
In [5]: io.DEFAULT_BUFFER_SIZE
```

```
Out[5]: 8192
```

# Explicando parâmetros - Encoding

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

**Encoding:** Encode do arquivo que vai ser lido. O formatos suportados [são esses](#)

**Default:** None, caso você não diga o encode do seu arquivo, o python vai descobrir ele usando  
``locale.getpreferredencoding()``

## Explicando parâmetros - Encoding

```
In [1]: import locale
```

```
In [2]: locale.getpreferredencoding()
```

```
Out[2]: 'UTF-8'
```

# Explicando parâmetros - Errors

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

**Errors:** Uma string opcional que especifica como os erros de encoding deve ser tratados (não funciona com binários)

**Default:** None, não fará nada. Quando um erro for encontrado, retornará **ValueError** porém existem uma gama de manipuladores disponíveis. Você também pode criar os seu próprio manipulador, mas terá que registrar o mesmo em ``codecs.register_error()``



# Explicando parâmetros - Errors

Modo	Significado
<b>'strict'</b>	<b>(Rigoroso) Abrirá um Raise com ValueError caso não consiga parsear</b>
<b>'ignore'</b>	<b>Ignora os erros de codificação, caso encontre, deixa de lado (perde dados)</b>
<b>'replace'</b>	<b>Troca os caracteres com erro por '?'</b>
<b>'surrogateescape'</b>	<b>(escape de substituição) Substitui os erros por escape os troca pelos bytes</b>
<b>'backslashreplace'</b>	<b>Substitui os dados mal formatados por sequências de backslash (\xc3\xa3o)</b>
<b>...</b>	<b>...</b>

# Explicando parâmetros - Newline

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

**Newline:** Controla como o `universal newlines` (PEP278) vai funcionar

**Default:** None. Nesse caso será interpretado usando universal newlines e todas as quebras (`\n`, `\r`, `\r\n`) serão traduzidas para `\n`, o mesmo vale para `"""` que lerá qualquer quebra, mas retornará o valor original.

# Explicando parâmetros - Newline

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

Caso você escolha o interpretador errado, o texto será parseado, porém não haverá quebra de linha

CODE !! [parte 2]

# Explicando parâmetros - Closefd

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

**Closefd:** Determina se você vai trabalhar com a descrição de um arquivo (arquivos em memória por exemplo)

**Default:** True

(precisamos de outra live só pra isso)

`/proc/1/fdinfo/0` -> é um exemplo de arquivo que só existe em runtime

# Explicando parâmetros - Opener

```
open(file,  
      mode='r',  
      buffering=-1,  
      encoding=None,  
      errors=None,  
      newline=None,  
      closefd=True,  
      opener=None)
```

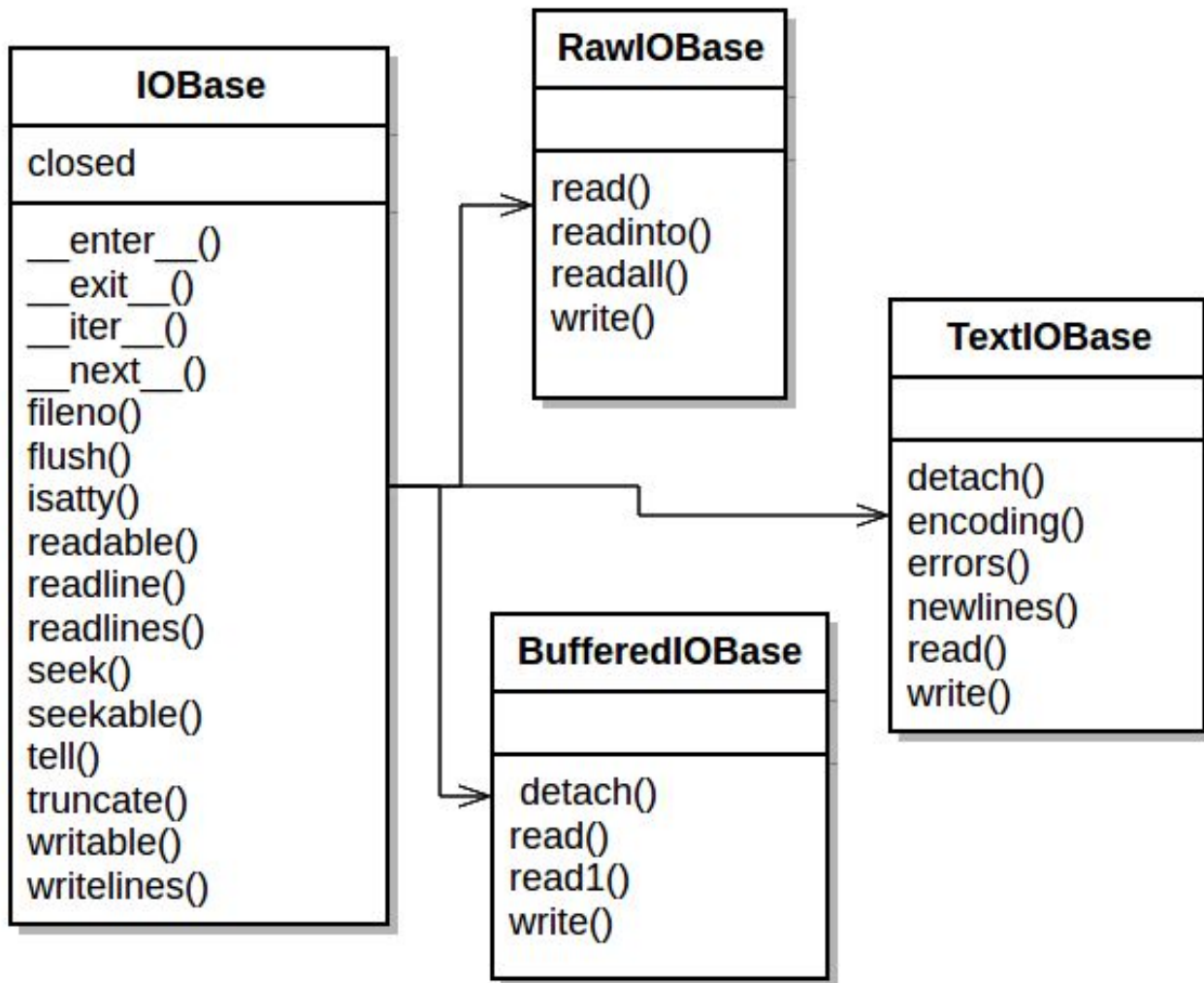
**Opener:** Diz a função open que outra maneira de abrir um arquivo vai ser usada

**Default:** None

Você pode construir um `open` e dizer ao python que abra o arquivo usando ele

(Assunto de uma outra live super avançada)

IO (PEP 3116)





**Criam o  
gerenciador  
de contexto**

IOBase
closed
<b><code>__enter__()</code></b>
<b><code>__exit__()</code></b>
<code>__iter__()</code>
<code>__next__()</code>
<code>fileno()</code>
<code>flush()</code>
<code>isatty()</code>
<code>readable()</code>
<code>readline()</code>
<code>readlines()</code>
<code>seek()</code>
<code>seekable()</code>
<code>tell()</code>
<code>truncate()</code>
<code>writable()</code>
<code>writelines()</code>

TextIOBase
<code>detach()</code>
<code>encoding()</code>
<code>errors()</code>
<code>newlines()</code>
<code>read()</code>
<code>write()</code>

**Criam o  
gerenciador  
de contexto**

IOBase
closed
<u>__enter__()</u>
<u>__exit__()</u>
__iter__()
__next__()
fileno()
flush()
isatty()
readable()
readline()
readlines()
seek()
seekable()
tell()
truncate()
writable()
writelines()

```
In [1]: with open('teste.txt') as file:  
...:     print(file.read())  
...:  
xxxxx  
yyyyy
```

TextIOBase
detach() encoding() errors() newlines() read() write()

**Criam o  
gerenciador  
de contexto**

IOBase
closed
<b>__enter__()</b>
<b>__exit__()</b>
__iter__()
__next__()
fileno()
flush()
isatty()
readable()
readline()
readlines()
seek()
seekable()
tell()
truncate()
writable()
writelines()

```
In [1]: with open('teste.txt') as file:  
...:     print(file.read())  
...:  
xxxxx  
yyyyy
```

TextIOBase
detach() encoding() errors() newlines() read() write()

**Transforma  
o objeto em  
um iterável**

IOBase
closed
<code>__enter__()</code>
<code>__exit__()</code>
<code>__iter__()</code>
<code>__next__()</code>
<code>fileno()</code>
<code>flush()</code>
<code>isatty()</code>
<code>readable()</code>
<code>readline()</code>
<code>readlines()</code>
<code>seek()</code>
<code>seekable()</code>
<code>tell()</code>
<code>truncate()</code>
<code>writable()</code>
<code>writelines()</code>

TextIOBase
<code>detach()</code>
<code>encoding()</code>
<code>errors()</code>
<code>newlines()</code>
<code>read()</code>
<code>write()</code>

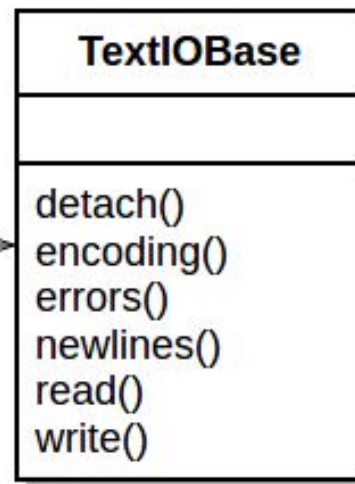
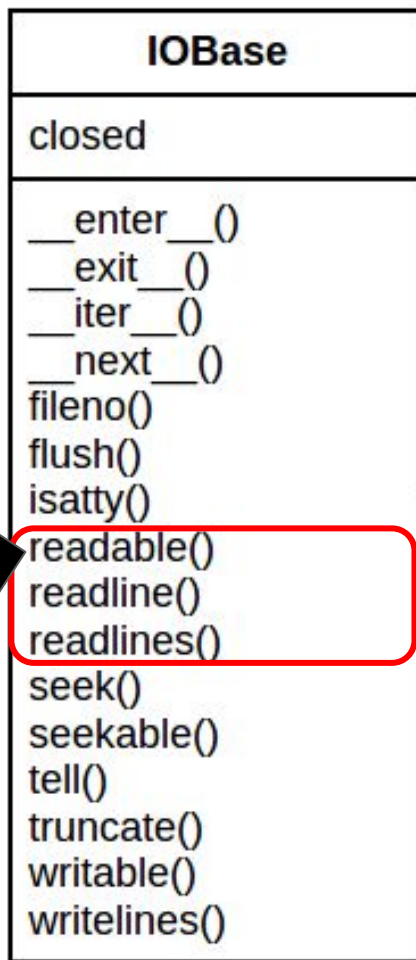
**Transforma  
o objeto em  
um iterável**

IOBase
closed
<code>__enter__()</code>
<code>__exit__()</code>
<code>__iter__()</code>
<code>__next__()</code>
<code>fileno()</code>
<code>flush()</code>
<code>isatty()</code>
<code>readable()</code>
<code>readline()</code>
<code>readlines()</code>
<code>seek()</code>
<code>seekable()</code>
<code>tell()</code>
<code>truncate()</code>
<code>writable()</code>
<code>writelines()</code>

```
In [2]: with open('teste.txt') as file:
...:     for x in file:
...:         print(x)
...:
xxxxx
yyyyy
```

`detach()`  
`encoding()`  
`errors()`  
`newlines()`  
`read()`  
`write()`

Diz se o  
arquivo está  
aberto para  
leitura



leem linhas  
do arquivo

IOBase
closed
<code>__enter__()</code>
<code>__exit__()</code>
<code>__iter__()</code>
<code>__next__()</code>
<code>fileno()</code>
<code>flush()</code>
<code>isatty()</code>
<code>readable()</code>
<code>readline()</code>
<code>readlines()</code>
<code>seek()</code>
<code>seekable()</code>
<code>tell()</code>
<code>truncate()</code>
<code>writable()</code>
<code>writelines()</code>

```
In [7]: with open('teste.txt') as file:
...:     print(file.readline())
...:
XXXXX
```

TextIOBase
<code>detach()</code>
<code>encoding()</code>
<code>errors()</code>
<code>newlines()</code>
<code>read()</code>
<code>write()</code>

```
In [8]: with open('teste.txt') as file:
...:     print(file.readlines())
...:
['xxxxx\n', 'yyyyy\n']
```

IOBase
closed
<code>__enter__()</code> <code>__exit__()</code> <code>__iter__()</code> <code>__next__()</code> <code>fileno()</code> <code>flush()</code> <code>isatty()</code> <code>readable()</code> <code>readline()</code> <code>readlines()</code> <code>seek()</code> <code>seekable()</code> <code>tell()</code> <code>truncate()</code> <code>writable()</code> <code>writelines()</code>

TextIOBase
<code>detach()</code> <code>encoding()</code> <code>errors()</code> <code>newlines()</code> <code>read()</code> <code>write()</code>

**lê/escreve o  
arquivo todo  
de uma  
única vez**



