open()

基本

組み込み open() 関数を使う。

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

file はファイルパスを表す文字列。

mode オプションは文字列として指定：
r: 読み取り専用。オプションが指定されない場合は、r が指定されたとみなされる。
w: 書込み専用。
a: 追加書込み。
r+: 読み取りと書込み。
b: バイナリモード。

b オプションを指定しないとテキストモードでファイルを開く（ファイルのエンコードはデフォルトか、プログラマが encoding パラメータで指定のいずれか)。エンコードが正しくないと UnicodeDecodeError が発生。

b オプションを指定すると、これまでどおりバイナリモードで開く。
読み込まれたデータは bytes として扱う。

close() は

```
f.close()
```

つい close(f) とやってエラーになる。。。

open() 時のファイルエンコード

encoding 引数を指定しないと、OS デフォルトのエンコーディングでファイルが開かれる。
Python 3 からは、指定したエンコーディングと実際のファイルの文字コードが合致していないと、UnicodeDecodeError が吐かれる。

しかし例えば、テキストログ中に非文字コードパターンが含まれているファイルなど、エンコーディングをどうしても指定できないファイルを、その他の部分は通常のテキストファイルとして処理しなければならない場合がある。

そんなときは、errors 引数に "ignore" と指定する。

```
f = open(file_path, errors="ignore")
```

デコードエラーが単に無視される。

"ignore" はデータロスを引き起こすかもしれない、とのこと。

open() が返すオブジェクト

テキストモードで開いた場合、_io.TextIOWrapper クラスのインスタンス（type() で見える）。
これは、イテレータとして扱える。

```
f = open("/var/log/messages")
for l in f:
    ...
```

直感的にテキストストリームが返ってくる。
ラインはそのまま返されてくるので、改行コードは付いたまま返ってくる。

with

with 文を使うと、ファイルオープンの try/except/finally を簡単に書ける。
with 文はブロックを生成し、ブロックを抜けると as のオブジェクトを始末する。

```
with open('C:/targetfile.txt') as f:
    lines = f.readlines()

# with ブロックを抜けると、f は自動的にクローズされている。
```

もし、ファイルをオープンしてさらにファイルハンドルをいろいろ処理するのであれば、try/except/finally で制御した方がいいのだが、この例のように中身を得たらすぐにファイルを閉じるだけ、という場合には、try/except/finally よりも簡潔に書ける。

PEP343 によると、以下が with 文が実現することである、とのこと。

```
mgr = (EXPR)
exit = mgr.__exit__  # Not calling it yet
value = mgr.__enter__()
exc = True
try:
    try:
        VAR = value  # Only if "as VAR" is present
        BLOCK
    except:
        # The exceptional case is handled here
        exc = False
        if not exit(*sys.exc_info()):
            raise
        # The exception is swallowed if exit() returns true
finally:
    # The normal and non-local-goto cases are handled here
    if exc:
        exit(None, None, None)
```

おおー、なるほど！