

# Bab 5. File dan Eksepsi

---

## OBJEKTIF:

1. Mahasiswa mampu memahami *File* pada Python.
  2. Mahasiswa mampu memproses Data Numerik dalam *File Teks* pada Pyhton.
  3. Mahasiswa mampu menggunakan *loop* untuk Memroses *File* pada Python.
  4. Mahasiswa mampu bekerja dengan *List* dan *File* pada Pyhton.
  5. Mahasiswa mampu memroses *Record* pada Python.
  6. Mahasiswa mampu memahami Eksepsi pada Python.
- 

## 5.1 File

Program yang kita tulis sejauh ini mengharuskan pengguna untuk meng-*input* kembali setiap kali program berjalan. Ini karena data yang *di-input* pengguna saat program berjalan, disimpan dalam RAM (*Random Access Memory*) sehingga data tersebut akan hilang ketika program berhenti.

Salah satu cara untuk mempertahankan data meskipun program berhenti adalah dengan menyimpan data tersebut ke dalam sebuah **file**. Kita menyebut proses menyimpan data ke dalam sebuah file sebagai "menulis data ke" file dan kita menyebut proses mengambil data dari sebuah file sebagai "membaca data dari" file.

File biasanya disimpan dalam *hard disk* (sering disebut dengan *disk* saja) yang mempertahankan isi file walaupun komputer dalam keadaan mati. Gambar berikut adalah ilustrasi hard disk.



### 5.1.1 Jenis File dan Path File

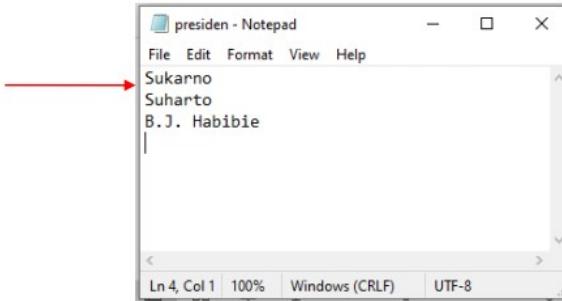
Secara umum, terdapat dua jenis file yaitu teks dan biner. File teks berisi data yang telah *encode* (diterjemahkan) sebagai teks. File teks bisa kita baca menggunakan teks editor. Sedangkan file biner berisi kode-kode biner. File biner hanya bisa dibaca oleh program spesifik. Misalkan, file gambar JPG adalah contoh file biner yang hanya bisa dibaca menggunakan program *image* editor dan tidak bisa dibaca oleh teks editor.

File diidentifikasi dengan nama dan ekstensi. Ekstensi file digunakan untuk mengidentifikasi jenis file: .txt untuk file teks, .jpg untuk file gambar (kompresi JPEG), .mp4 untuk video, dsb.



Pada bab ini, kita hanya akan membahas cara membaca dan menulis ke file teks. Hal penting yang perlu diketahui mengenai *file* teks adalah keseluruhan isi file teks adalah sebuah string. Gambar berikut mengilustrasikan file teks yang dibuka dengan teks editor dan isi file tersebut yang dibaca oleh program.

Isi file jika kita melihatnya dalam aplikasi teks editor.



Isi file sebenarnya yang dibaca oleh program.

Sukarno\nSuharto\nB.J. Habibie\n

Perhatikan pada gambar di atas, escape sequence `\n` dicetak sebagai baris baru pada teks editor.

### 5.1.2 Menggunakan File dalam Program

Untuk menggunakan file dalam program (untuk dibaca maupun ditulis), kita melakukan tiga langkah berikut:

1. Membuka *file* – Membuka sebuah *file* membuat sebuah koneksi antara *file* dan program.
2. Memproses *file* – Dalam langkah ini data ditulisi ke *file* atau data dibaca dari *file*.
3. Menutup *file* – Ketika program selesai menggunakan *file*, *file* tersebut harus ditutup.  
Menutup *file* memutus koneksi antara *file* dan program.

Tabel berikut menunjukkan langkah-langkah menggunakan file dalam program dengan contoh kode untuk menulis ke file teks dan untuk membaca dari file teks.

Menulis ke File Teks	Langkah	Membaca dari File Teks
<code>file = open('data.txt', 'w')</code>	Membuka File	<code>file = open('data.txt', 'r')</code>
<code>file.write('Baris1')</code> <code>file.write('Baris2')</code>	Memproses File	<code>isi_file = file.read()</code>
<code>file.close()</code>	Menutup File	<code>file.close()</code>

### 5.1.3 Membuka sebuah File

Kita menggunakan fungsi *built-in* `open` untuk membuka sebuah file. Fungsi `open` membuat sebuah object file yang memungkinkan kita memproses file tersebut melalui method-method pada object file tersebut.

```
<variabel> = open(<nama_file>, <modus>)
```

Dimana `<modus>` adalah salah satu dari sejumlah string tertentu yang diantaranya dapat dilihat pada tabel berikut:

Modus	Arti	Keterangan
'r'	Read (Baca)	Buka sebuah <i>file</i> untuk baca saja. <i>File</i> tidak dapat diubah atau ditulis
'w'	Write (Tulis)	Buka sebuah <i>file</i> untuk ditulis. Jika <i>file</i> sudah ada, hapus semua isinya. Jika <i>file</i> belum ada, buat <i>file</i> .
'a'	Append (Sambung)	Buka sebuah <i>file</i> untuk ditulis. Semua data yang ditulis akan ditambahkan pada akhir <i>file</i> .

#### Catatan:

Terdapat modus lainnya, namun tiga modus pada tabel ini yang paling mudah dan sering digunakan.

Ketika kita memberikan nama file sebagai argumen pertama dari fungsi `open`, interpreter Python mengasumsikan file tersebut berada di dalam lokasi (*path/directory*) yang sama dengan lokasi program. Jika kita ingin menggunakan file yang tidak berada dalam satu lokasi dengan program kita, kita harus menambahkan path lengkap dari file tersebut. Misalkan, program kita berada di folder dengan path: `c:\users\Budi\Documents\Python`, dan kita ingin membuat sebuah file dengan nama `test.txt` di folder dengan path: `c:\users\Budi\data`. Untuk melakukannya, kita menggunakan fungsi `open` dengan argumen pertama berupa *raw string* dari path lengkap file `test.txt`:

```
file_test = open(r'C:\User\Budi\Data\test.txt', 'w')
```

#### Catatan:

Huruf `r` sebelum nilai *string* menandakan *raw string* (*string* mentah), yang berarti karakter `\` diartikan sebagai nilai sebenarnya dan bukan karakter yang menandakan *escape sequence*.

### 5.1.4 Memproses File Teks

Setelah kita mempunyai object file yang didapatkan dari fungsi `open`, kita dapat menggunakan method-method dari object file untuk memproses file tersebut.

Terdapat tiga method yang umum digunakan untuk membaca isi file teks:

1. `<file>.read()`: membaca keseluruhan isi teks dalam file dan mengembalikan sebuah string yang berisi keseluruhan isi teks tersebut.
2. `<file>.readline()`: membaca baris per baris dan mengembalikan isi satu baris dalam bentuk string.

3. `<file>.readlines()`: membaca keseluruhan isi teks dan mengembalikan sebuah `list` yang masing-masing elemennya berupa teks per baris dari file.

Terdapat dua method yang umum digunakan untuk menulis ke file:

1. `<file>.write(<string>)`: menulis sebuah string ke file.
2. `<file>.writelines(<list_string>)`: menulis sebuah `list` dengan elemen-elemen bertipe string ke file.

## Menulis ke File Teks

Berikut adalah contoh program yang menulis ke sebuah *file*:

```
# tulis_file.py
# Program ini mendemonstrasikan menulis teks
# ke sebuah file teks

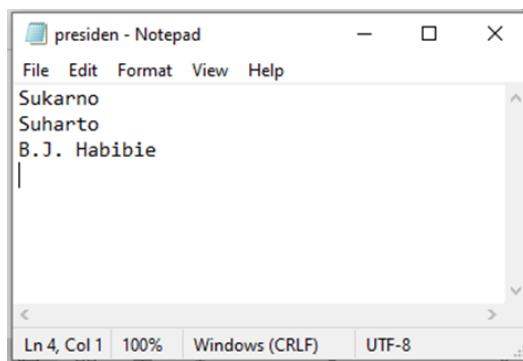
def main():
    # Buat object file dengan fungsi open
    outfile = open('presiden.txt', 'w')

    # Tuliskan tiga nama presiden pertama Indonesia
    # ke file
    outfile.write('Sukarno\n')
    outfile.write('Suharto\n')
    outfile.write('B.J. Habibie\n')

    # Tutup file
    outfile.close()

# Panggil fungsi main
main()
```

Program di atas akan membuat sebuah file yang jika dibuka dalam teks editor berisi seperti gambar berikut:



Penjelasan program di atas dapat dilihat pada gambar berikut:

Argumen pertama dari fungsi open adalah string nama file lengkap beserta ekstensi (dan juga path jika tidak berada dalam satu folder).

```
def main():
    # Buat object file dengan fungsi open
    outfile = open('presiden.txt', 'w')

    # Tuliskan tiga nama presiden pertama Indonesia
    # ke file
    outfile.write('Sukarno\n')
    outfile.write('Suharto\n')
    outfile.write('B.J. Habibie\n')

    # Tutup file
    outfile.close()

# Panggil fungsi main
```

Argumen kedua adalah modus buka file. Modus 'w' berarti write yang digunakan untuk menulis ke file. Dalam modus 'w', jika sudah terdapat file dengan nama seperti pada argumen pertama (**presiden.txt**), maka keseluruhan isi file tersebut dihapus. Dan jika tidak ada, maka file baru dengan nama tersebut akan dibuat.

#### Langkah 1. Membuka File(untuk ditulis)

Kita membuat sebuah object file dengan menggunakan fungsi built-in open dan menugaskan object tersebut ke variabel outfile.

#### Langkah 2. Memroses File(menuulis)

Kita menulis ke file menggunakan method write dengan argumen berupa string.

#### Langkah 3. Menutup File

Setelah selesai memroses file, kita menutup file untuk memutus koneksi antara file dengan program.

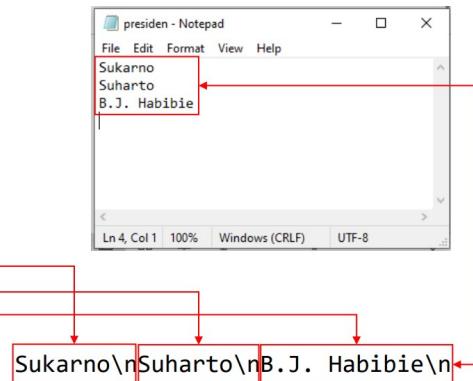
Perhatikan pemanggilan method `write` pada program di atas. Kita memanggil method `write` dengan argumen berupa string yang dua karakter terakhirnya adalah escape sequence dari baris baru ('`\n`'). Ini diperlukan untuk menuliskan teks dalam baris tersendiri. Gambar berikut menjelaskan hal ini:

```
# tulis_file.py
# Program ini mendemonstrasikan menulis teks
# ke sebuah file teks

def main():
    # Buat object file dengan fungsi open
    Kita menambahkan escape sequence \n yang
    berarti baris baru untuk memisahkan string-string
    yang kita tulis dengan baris baru.
    # Tuliskan tiga nama presiden pertama Indonesia
    # ke file
    outfile.write('Sukarno\n')
    outfile.write('Suharto\n')
    outfile.write('B.J. Habibie\n')

    # Tutup file
    outfile.close()

# Panggil fungsi main
main()
```



## Membaca dari File Teks

Program berikut mencontohkan cara membaca file `presiden.txt` yang isinya ditulis pada contoh program sebelumnya:

```
# baca_file.py
# Program ini mendemonstrasikan membaca sebuah file teks

def main():
    # Buat sebuah object file dengan fungsi open
    file = open('presiden.txt', 'r')

    # Baca semua isi teks dengan method read dari object file
    isi_file = file.read()

    # Tampilkan isi file
    print(isi_file)
```

```

# Tutup file
file.close()

# Panggil fungsi main
main()

```

Output:

```

Sukarno
Suharto
B.J. Habibie

```

Gambar berikut menjelaskan kode program di atas:

```

def main():
    # Buat sebuah object file dengan fungsi open
    file = open('presiden.txt', 'r')

    # Baca semua isi teks dengan method read dari object file
    isi_file = file.read()

    # Tampilkan isi file
    print(isi_file)

    # Tutup file
    file.close()

# Panggil fungsi main
main()

```

Argumen pertama dari fungsi `open` adalah string nama file lengkap beserta ekstensi (dan juga path jika tidak berada dalam satu folder).

Argumen kedua adalah string dari modus buka file. Modus '`'r'`' berarti read yang digunakan untuk membaca file.

**Langkah 1. Membuka File (untuk dibaca)**  
Kita membuat sebuah object file dengan menggunakan fungsi built-in `open` dan menugaskan object tersebut ke suatu variabel.

**Langkah 2. Memroses File (membaca)**  
Kita menggunakan method `read` dari object file untuk membaca keseluruhan isi file.

**Langkah 3.**  
Setelah selesai membaca isi file, kita menutup file dengan menggunakan method `close`.

## Menulis Baris Data ke File

Pada contoh sebelumnya, kita menulis ke file dengan memanggil method `write` dengan argumen berupa nilai string langsung yang ditambahkan escape sequence baris baru (`'\n'`):

```

file.write('Baris1\n')

```

Kita mengakhiri string yang ditulis ke file dengan escape sequence baris baru (`'\n'`) untuk menuliskan string baris per baris.

Dalam banyak kasus, kita tidak menuliskan langsung nilai string sebagai argumen dari method `write` namun umumnya kita menuliskan variabel. Misalkan kita menuliskan input dari pengguna ke sebuah file, kita dapat menuliskan kode seperti berikut:

```

nama1 = input('Masukkan nama1: ')
nama2 = input('Masukkan nama2: ')

myfile = open('nama.txt', 'w')

myfile.write(nama1 + '\n')
myfile.write(nama2 + '\n')

```

Perhatikan pada kode di atas, untuk menulis nilai setiap variabel dalam sebuah baris, pada argumen method `write`, kita mengkonkatenasi string dengan escape sequence `\n`:

```

nama1 = input('Masukkan nama 1: ')
nama2 = input('Masukkan nama 2: ')

myfile = open('nama.txt', 'w')

myfile.write(nama1 + '\n')
myfile.write(nama2 + '\n')
myfile.close()

```

Kita mengkonkatenasi variabel dengan escape sequence baris baru ('\n') untuk menuliskan nilai variabel baris per baris.

Contoh program yang menulis baris data:

```

# tulis_file_baris.py
# Program ini menuliskan input dari pengguna ke file

def main():
    # Ambil nama-nama dari pengguna
    nama1 = input('Masukkan nama 1: ')
    nama2 = input('Masukkan nama 2: ')
    nama3 = input('Masukkan nama 3: ')

    outfile = open('nama.txt', 'w')
    # Tulis nama-nama yang di-input pengguna
    # ke file
    outfile.write(nama1 + '\n')
    outfile.write(nama2 + '\n')
    outfile.write(nama3 + '\n')
    # Tutup file
    outfile.close()

    # Tampilkan nama file
    print('Data ditulis ke file nama.txt.')
    print()

# Panggil fungsi main
main()

```

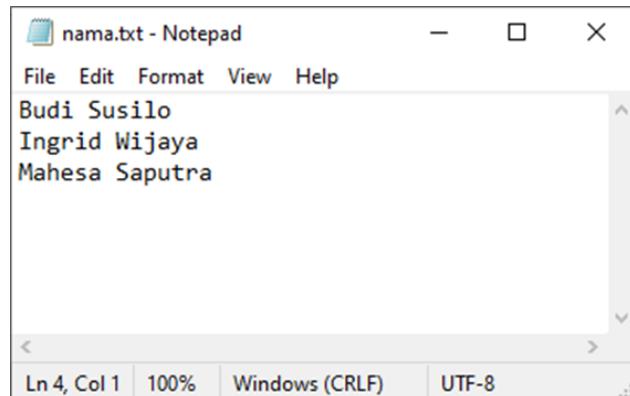
Output dari program di atas:

```

Masukkan nama 1: Budi Susilo
Masukkan nama 2: Ingrid Wijaya
Masukkan nama 3: Mahesa Saputra
Data ditulis ke file nama.txt.

```

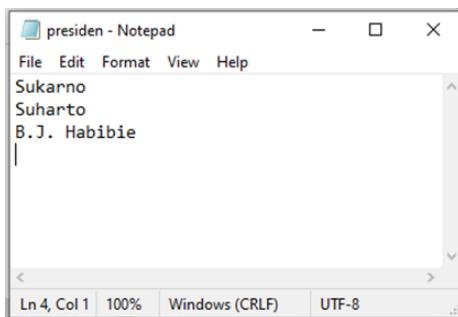
Setelah program di atas selesai dieksekusi, sebuah file bernama `nama.txt` akan terbentuk dan mempunyai isi seperti pada gambar berikut:



## 5.1.5 Menulis ke File Teks dengan Modus 'a'

Modus '`w`' pada fungsi `open` membuka *file* untuk ditulis dan menghapus isi *file* sebelumnya. Untuk mempertahankan isi *file* dan menambahkan data baru ke dalamnya, kita menggunakan modus '`a`' (yang berarti *append* atau sambung) pada fungsi `open`.

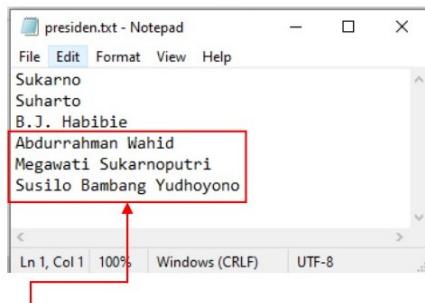
Misalkan kita mempunyai file teks bernama `presiden.txt` dengan isi seperti berikut:



Kode berikut menulis ke file `presiden.txt` dengan modus *append*:

```
outfile = open('presiden.txt', 'a')
outfile.write('Abdurrahman Wahid\n')
outfile.write('Megawati Sukarnoputri\n')
outfile.write('Susilo Bambang Yudhoyono\n')
outfile.close()
```

Isi file setelah kode di atas dijalankan:



Data baru yang dituliskan di-*append* (disambung) ke akhir file.

Berikut adalah contoh program lengkap yang menulis ke file dengan modus 'a':

```
# tulis_file_append.py
# Program ini mendemonstrasikan menulis teks
# ke sebuah file teks dengan modus 'a'.

def main():
    # Buat object file dengan fungsi open
    outfile = open('presiden.txt', 'a')

    # Tuliskan tiga nama presiden pertama Indonesia
    # ke file
    outfile.write('Abdurrahman Wahid\n')
    outfile.write('Megawati Sukarnoputri\n')
    outfile.write('Susilo Bambang Yudhoyono\n')

    # Tutup file
    outfile.close()

# Panggil fungsi main
main()
```

## 5.1.6 Statement with

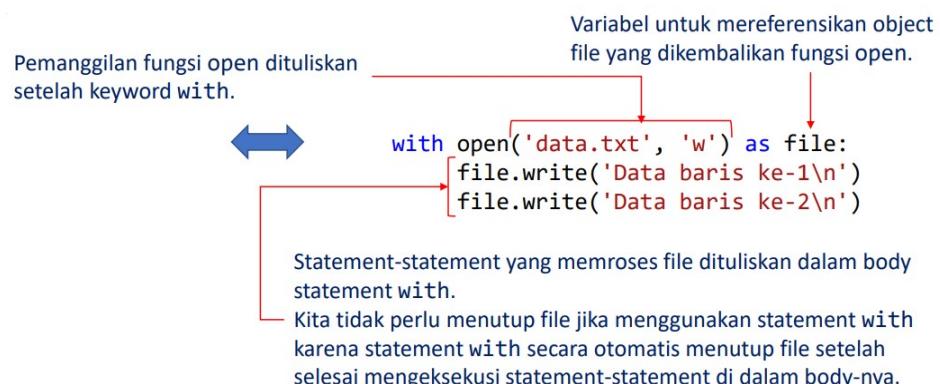
Python menyediakan statement `with` yang dapat digunakan sebagai alternatif untuk membuka dan menutup file. Berikut adalah contoh kode dengan statement `with` untuk menulis ke sebuah file:

```
with open('data.txt', 'w') as file:
    file.write('Data baris ke-1\n')
    file.write('Data baris ke-2\n')
```

Kode di atas ekuivalen dengan kode berikut:

```
file = open('data.txt', 'w')
file.write('Data baris ke-1\n')
file.write('Data baris ke-2\n')
file.close()
```

Gambar berikut menjelaskan kode dengan statement `with` di atas:



Menggunakan statement `with` untuk membuka file menjadikan kode kita lebih ringkas, sehingga cara ini yang lebih dipilih untuk bekerja dengan file.

Program `tulis_file.py` dapat kita tulis ulang menggunakan statement `with` seperti berikut:

```
# tulis_file_with.py
# Program ini mendemonstrasikan menulis teks
# ke sebuah file teks

def main():

    # Buat object file dengan fungsi open
    with open('presiden.txt', 'w') as outfile:

        # Tuliskan tiga nama presiden pertama Indonesia
        # ke file
        outfile.write('Sukarno\n')
        outfile.write('Suharto\n')
        outfile.write('B.J. Habibie\n')

    # Panggil fungsi main
    main()
```

### 5.1.7 Membaca dari File Teks Baris per Baris

Method `read` mengambil keseluruhan isi file dan mengembalikannya dalam sebuah string. Seringkali, kita memerlukan untuk membaca isi file baris per baris.

Kita dapat membaca isi *file* teks baris per baris menggunakan method `readline()`:

```
# baca_baris_file.py
# Program ini membaca isi dari file presiden.txt
# baris per baris

def main():
    # Buka file dengan nama presiden.txt
    with open('presiden.txt', 'r') as infile:
        # Baca baris per baris isi dari file
        baris1 = infile.readline()
        baris2 = infile.readline()
        baris3 = infile.readline()

    # Tampilkan isi baris per baris
    print(baris1)
    print(baris2)
    print(baris3)

    # Panggil fungsi main
    main()
```

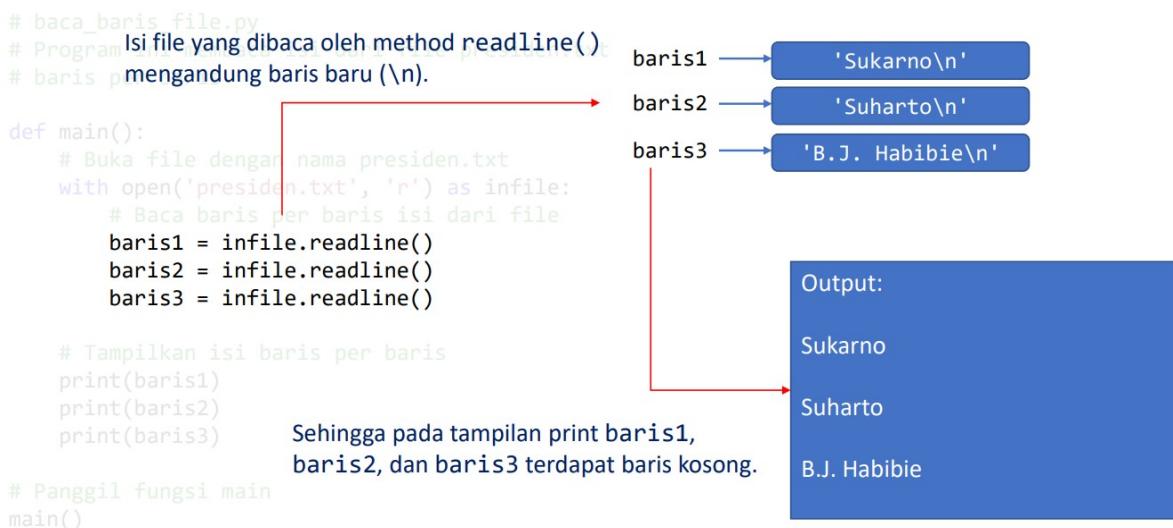
*Output:*

Sukarno

Suharto

B.J. Habibie

Perhatikan pada output dari program, terdapat baris kosong pada setiap baris teks yang dibaca. Ini karena, method `readline()` membaca semua teks dalam suatu baris termasuk dengan escape sequence dari baris baru (`'\n'`). Pada program di atas, variabel `baris1` menyimpan string `Sukarno\n`, `baris2` menyimpan string `Suharto\n`, dan `baris3` menyimpan string `B.J. Habibie\n`. Dan karena, fungsi `print()` juga mencetak baris baru, maka program di atas akan mencetak baris-baris dari isi file dengan tambahan baris kosong diantaranya. Gambar berikut mengilustrasikan ini:



Method `readline()` mengembalikan string isi file per baris namun masih mengandung escape sequence baris baru (`'\n'`). Bagaimana jika kita ingin menampilkan output yang sama persis dengan isi file yang ditampilkan teks editor? Kita harus menghilangkan karakter `'\n'` pada setiap baris yang dibaca. Untuk menghilangkan karakter baris baru kita dapat menggunakan method string `rstrip()`.

Contoh program mendapatkan isi file per baris sesuai dengan apa yang ditampilkan oleh teks editor:

```
# baca_baris_file2.py
# Program ini membaca isi dari file presiden.txt
# baris per baris tanpa karakter baris baru

def main():
    # Buka file dengan nama presiden.txt
    with open('presiden.txt', 'r') as infile:
        # Baca baris per baris isi dari file
        baris1 = infile.readline()
        baris2 = infile.readline()
        baris3 = infile.readline()

    # Hilangkan karakter baris baru ('\n')
    baris1 = baris1.rstrip('\n')
```

```
baris2 = baris2.rstrip('\n')
baris3 = baris3.rstrip('\n')
# Tampilkan isi baris per baris

print(baris1)
print(baris2)
print(baris3)
# Panggil fungsi main
main()
```

Output dari program di atas:

```
Sukarno
Suharto
B.J. Habibie
```

## 5.2 Memroses Data Numerik dalam File Teks

### 5.2.1 Menulis Data Numerik

String dapat ditulis langsung ke file dengan method `write`, namun data numerik harus dikonversi ke string sebelum dapat ditulis ke file. Kita mengonversi data numerik ke string dengan fungsi built-in `str()`. Sesi interaktif berikut mencontohkan penggunaan fungsi `str()` untuk mengonversi string ke data numerik:

```
>>> str_angka1 = str(99)
>>> print(str_angka1)
99
>>> str_angka2 = str(7.4)
>>> print(str_angka2)
7.4
```

Program berikut mencontohkan cara menulis data numerik ke sebuah file:

```
# tulis_angka.py
# Program ini mendemonstrasikan bagaimana angka
# harus dikonversi ke string sebelum ditulis ke sebuah file teks

def main():
    # Buka sebuah file untuk ditulis
    outfile = open('angka.txt', 'w')

    # Minta tiga angka ke user
    num1 = int(input('Masukkan sebuah angka: '))
    num2 = int(input('Masukkan sebuah angka lain: '))
    num3 = int(input('Masukkan sebuah angka lain: '))

    # Tulis angka-angka ke file
    outfile.write(str(num1) + '\n')
    outfile.write(str(num2) + '\n')
    outfile.write(str(num3) + '\n')

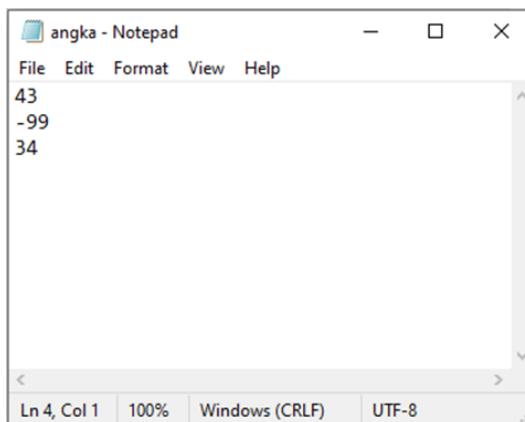
    # Tutup file
    outfile.close()
    print('Data ditulis ke file angka.txt')
```

```
# Panggil fungsi main
main()
```

Berikut adalah contoh output dari program:

```
Masukkan sebuah angka: 43
Masukkan sebuah angka lain: -99
Masukkan sebuah angka lain: 34
Data ditulis ke file angka.txt
```

Setelah program selesai dijalankan, akan terdapat sebuah file bernama `angka.txt` dengan isi seperti terlihat pada gambar berikut.



## 5.2.2 Membaca Data Numerik

Method-method yang digunakan untuk membaca isi file (`read`, `readline`, `readlines`) kesemuanya mengembalikan string meskipun isi file berupa angka-angka. Kita harus mengonversi data yang dibaca ke dalam tipe numerik (dengan fungsi `int` atau fungsi `float`) sebelum kita dapat melakukan operasi-operasi aritmatika terhadapnya.

Berikut adalah contoh program untuk membaca data numerik dari file teks:

```
# baca_angka.py
# Program ini mendemonstrasikan bagaimana angka yang dibaca
# dari sebuah file harus dikonversi dari string ke tipe numerik
# sebelum digunakan dalam operasi matematika

def main():
    # Buka sebuah file untuk dibaca
    infile = open('angka.txt', 'r')

    # Baca ketiga angka dari file
    num1 = int(infile.readline())
    num2 = int(infile.readline())
    num3 = int(infile.readline())

    # Tutup file
    infile.close()

    # Tambahkan ketiga angka
    total = num1 + num2 + num3
```

```
# Tampilkan angka-angka dan totalnya
print('Angka-angka dalam file: ', num1, num2, num3)
print('Total ketiganya: ', total)

# Panggil fungsi main
main()
```

Output dari program di atas:

```
Angka-angka dalam file: 43 -99 34
Total ketiganya: -22
```

Perhatikan pada kode program di atas, kita tidak menggunakan method `rstrip` untuk menghilangkan escape sequence baris baru pada setiap baris yang dibaca. Ini karena fungsi `int` dan `float` secara otomatis menghilangkan escape sequence baris baru.

## 5.3 Menggunakan Loop untuk Memroses File

File biasanya digunakan untuk menyimpan data berjumlah banyak. Untuk membuat program yang memproses file dengan data berjumlah banyak kita umumnya menggunakan sebuah loop.

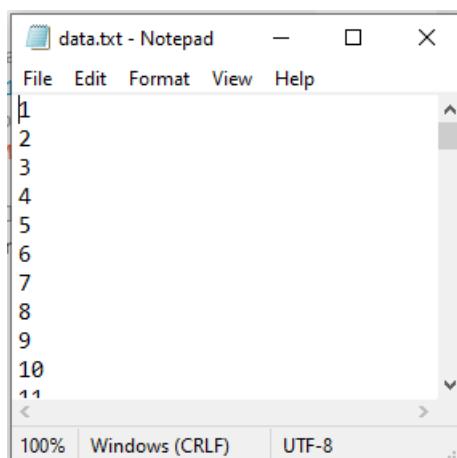
Kode berikut mencontohkan penggunaan sebuah loop `for` untuk menulis angka 1 sampai dengan 100 ke dalam sebuah file bernama `data.txt`:

```
# Buka file baru bernama sales.txt
outfile = open('data.txt', 'w')

# Menulis ke file menggunakan loop
for i in range(1, 101):
    outfile.write(str(i) + '\n')

# Tutup file
outfile.close()
```

Isi file `data.txt` setelah kode di atas selesai dieksekusi:



### 5.3.1 Menulis ke File Menggunakan Loop

Program berikut meminta pengguna untuk memasukkan data penjualan per hari dan menyimpan data tersebut ke dalam sebuah file:

```
# menulis_penjualan.py
# Program ini meng-prompt pengguna untuk jumlah penjualan
# dan menulis jumlah tersebut ke file sales.txt

def main():
    # Minta banyaknya hari penjualan yang ingin diinput
    num_hari = int(input('Berapa hari penjualan yang ingin Anda masukkan: '))

    # Buka file baru bernama sales.txt
    file_penjualan = open('sales.txt', 'w')

    # Ambil jumlah penjualan setiap hari dan tulis ke file
    for count in range(1, num_hari + 1):
        # Ambil jumlah penjualan per hari
        penjualan = float(input('Masukkan penjualan untuk hari ke ' + str(count)
+ ': '))

        # Tuliskan jumlah penjualan ke file
        file_penjualan.write(str(penjualan) + '\n')

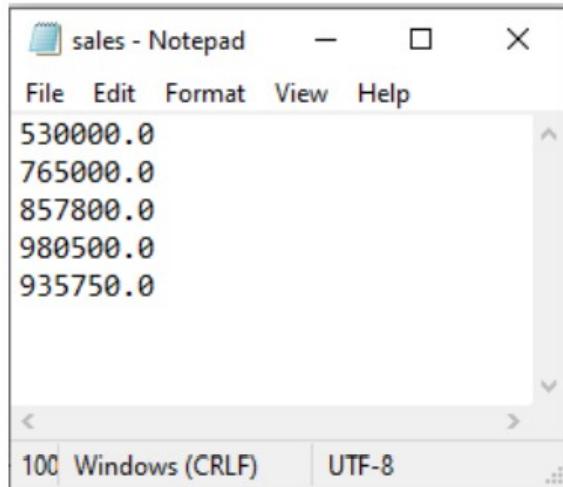
    # Tutup file
    file_penjualan.close()
    print('Data ditulis ke sales.txt')

# Panggil fungsi main
main()
```

Contoh output dari program di atas:

```
Berapa hari penjualan yang ingin Anda masukkan: 5
Masukkan penjualan untuk hari ke 1:
530000
Masukkan penjualan untuk hari ke 2:
765000
Masukkan penjualan untuk hari ke 3:
857800
Masukkan penjualan untuk hari ke 4:
980500
Masukkan penjualan untuk hari ke 5:
935750
Data ditulis ke sales.txt
```

Isi file sales.txt setelah program di atas selesai dieksekusi:



## 5.3.2 Membaca dari File Menggunakan Loop

### Membaca dari File Menggunakan Loop `while`

Seringkali kita memerlukan sebuah program yang membaca file tanpa mengetahui berapa banyak baris data yang tersimpan dalam file tersebut. Misalkan, pada contoh sebelumnya sales.txt dapat mempunyai berapapun banyak baris data karena program meminta pengguna untuk menentukan banyak hari penjualan yang ingin diinput. Untuk membaca file tanpa mengetahui berapa banyak baris data di dalam file tersebut kita dapat menggunakan sebuah loop. Namun kita memerlukan suatu cara untuk mengetahui akhir dari file untuk mengakhiri loop. Method `readline` mengembalikan string kosong ('') ketika mencoba membaca melebihi akhir dari file. Ini berarti kita dapat menggunakan loop `while` yang menguji kondisi string kosong untuk membaca file yang berapa banyaknya baris data dalam file tersebut tidak kita ketahui.

Program berikut mencontohkan penggunaan loop `while` untuk membaca isi sebuah file file:

```
# membaca_penjualan.py
# Program ini membaca semua baris data dalam file sales.txt

def main():
    # Buka file sales.txt untuk dibaca
    file_sales = open('sales.txt', 'r')

    # Baca baris pertama dari file.
    baris = file_sales.readline()

    # Selama baris tidak berisi string kosong lanjutkan ke baris berikutnya.
    while baris != '':
        # Konversi baris ke float
        penjualan = float(baris)

        # Tampilkan penjualan dengan format dua angka desimal
        print(f'{penjualan:.2f}')

        # Baca baris selanjutnya
        baris = file_sales.readline()

    # Tutup file
    file_sales.close()
```

```
# Panggil fungsi main
main()
```

Output dari program di atas:

```
530000.00
765000.00
857800.00
980500.00
935750.00
```

## Membaca dari File Menggunakan Loop `for`

Teknik yang kita gunakan sebelumnya untuk membaca file dengan loop `while` dan kondisi yang menguji akhir dari file merupakan teknik membaca file yang umum dan dapat digunakan pada bahasa-bahasa pemrograman lain. Python menyediakan teknik yang lebih mudah untuk membaca file menggunakan loop yaitu dengan menggunakan loop `for`. Dengan menggunakan loop `for` kita tidak perlu menguji kondisi yang menandakan akhir dari file, karena Python secara otomatis mengakhiri loop ketika mencapai akhir dari file. Bentuk umum loop `for` untuk membaca file:

```
for <variabel> in <object_file>:
    <statement>
    <statement>
    ...
    <statement>
```

Program berikut mencontohkan cara membaca sebuah file menggunakan loop `for`:

```
# membaca_penjualan2.py
# Program ini menggunakan loop for untuk
# membaca semua data dalam file sales.txt

def main():
    # Buka file sales.txt untuk dibaca
    file_sales = open('sales.txt', 'r')

    # Loop for untuk membaca tiap baris file
    for baris in file_sales:
        # Konversi string ke float
        penjualan = float(baris)

        # Tampilkan penjualan dengan format dua angka desimal
        print(f'{penjualan:.2f}')

    # Tutup file
    file_sales.close()

# Panggil fungsi main
main()
```

Output dari program di atas:

```
530000.00
765000.00
857800.00
980500.00
935750.00
```

## 5.4 Bekerja dengan List dan File

Ketika kita membuat program yang melakukan pengolahan data yang banyak, kita umumnya memroses file dengan menggunakan list. Untuk memroses file dengan menggunakan list, kita menggunakan dua method berikut:

1. Method `writelines` untuk menulis sebuah list langsung ke sebuah file.
2. Method `readlines` untuk membaca isi file ke sebuah list.

### 5.4.1 Method `writelines`

Method `writelines` menerima sebuah argumen berupa list dan menulis semua elemen list ke file. Masing-masing elemen list ditulis tidak dipisahkan baris baru. Program berikut mencontohkan penggunaan method `writelines`:

```
# tulisbaris.py
# Program ini menggunakan method writelines untuk menulis
# list ke sebuah file

def main():
    # Buat sebuah list dari string
    kota = ['Jakarta', 'Bogor', 'Depok', 'Tangerang']

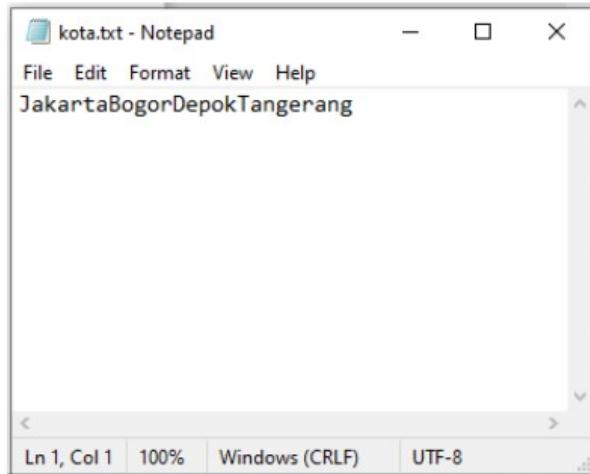
    # Buka sebuah file untuk ditulis
    outfile = open('kota.txt', 'w')

    # Tulis list ke file
    outfile.writelines(kota)

    # Tutup file
    outfile.close()

# Panggil fungsi main
main()
```

Program di atas akan menghasilkan file bernama `kota.txt` dengan isi seperti gambar berikut:



## 5.4.2 Alternatif Penulisan List ke File

Method `writelines` memiliki kekurangan yaitu:

1. Tidak secara otomatis menuliskan masing-masing elemen dalam baris baru.
2. Hanya bisa menuliskan sebuah list yang semua elemennya bertipe string.

Terdapat cara lain untuk menuliskan list ke file yaitu dengan menggunakan loop `for`:

```
# tulis_list.py
# Program ini menulis sebuah list dari string ke sebuah file

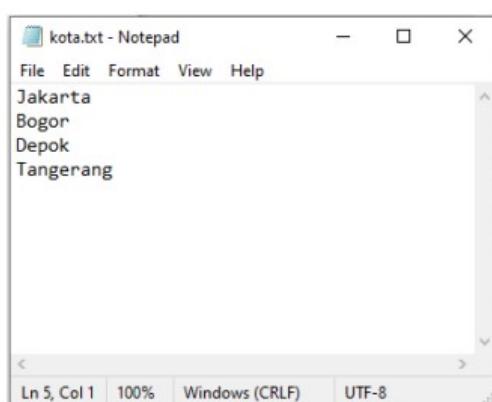
def main():
    # Buat sebuah list dari string
    kota = ['Jakarta', 'Bogor', 'Depok', 'Tangerang']

    # Buat sebuah file untuk ditulis
    outfile = open('kota.txt', 'w')

    # Tulis elemen list dengan pemisah baris baru
    for elm in kota:
        outfile.write(elm + '\n')

# Panggil fungsi main
main()
```

Program di atas akan menghasilkan file bernama `kota.txt` dengan isi seperti gambar berikut:



### 5.4.3 Membaca File ke List

Untuk membaca isi file dan menyimpannya ke sebuah list, kita menggunakan method `readlines`. Program berikut mencontohkan penggunaan method `readlines`:

```
# baca_list.py
# Program ini membaca isi sebuah file dan
# menyimpannya dalam sebuah list

def main():
    # Buka file dan proses
    with open('kota.txt', 'r') as infile:
        # Baca isi file dan simpan dalam sebuah list
        kota = infile.readlines()

    # Strip karakter baris baru
    index = 0
    while index < len(kota):
        kota[index] = kota[index].rstrip('\n')
        index += 1

    # Tampilkan isi dari list
    print(kota)

# Panggil fungsi main
main()
```

Output dari program dia tas:

```
['Jakarta', 'Bogor', 'Depok', 'Tangerang']
```

Gambar berikut menjelaskan program `baca_list.py` di atas:

```
# baca_list.py
# Program ini membaca isi sebuah file dan
# menyimpannya dalam sebuah list

def main():
    # Buka file dan proses
    with open('kota.txt', 'r') as infile:
        # Baca isi file dan simpan dalam sebuah list
        kota = infile.readlines() ← Setelah statement ini dieksekusi, variabel kota akan
                                mereferensikan list berikut:
                                ['Jakarta\n', 'Bogor\n', '\Depok\n', 'Tangerang\n'].  

                                Elemen pertama adalah isi file baris 1, elemen kedua  

                                adalah isi file baris 2, dan seterusnya. Perhatikan, setiap  

                                elemen masih mengandung karakter baris baru, dan  

                                umumnya kita menghilangkan karakter baris baru ini  

                                dengan method string rstrip.
```

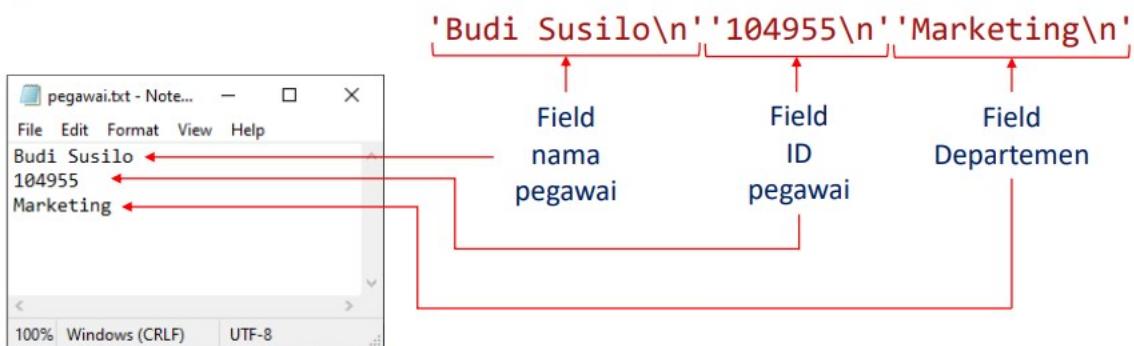
Kita mengiterasi setiap elemen dalam list dan  
menggunakan method `rstrip` untuk menghilangkan  
karakter baris baru pada setiap elemen.

```
    # Strip karakter baris baru
    index = 0
    while index < len(kota):
        kota[index] = kota[index].rstrip('\n')
        index += 1

    # Tampilkan isi dari list
    print(kota)
main()
```

### 5.5 Memroses Record

Data yang disimpan dalam sebuah file seringkali diorganisasi dalam bentuk **record**. Record adalah kumpulan data utuh yang menjelaskan satu hal dan terdiri dari field-field. Misalkan record dari data pegawai terdiri dari field nama pegawai, field nomor id, dan field departemen. Gambar berikut mencontohkan sebuah file teks yang berisi satu record data pegawai:



Program berikut menulis record data pegawai sesuai dengan input pengguna:

```
# simpan_record_pegawai.py
# Program ini meminta data pegawai dari pengguna
# dan menyimpannya ke file pegawai.txt

def main():
    # Ambil banyak data pegawai yang ingin di-input
    num_pegawai = int(input('Berapa banyak record pegawai yang ingin di-input?'))
    )

    # Buka file untuk di-append
    with open('pegawai.txt', 'a') as file:
        # Ambil data pegawai dan tulis ke file
        for count in range(1, num_pegawai + 1):
            # Ambil data untuk satu pegawai
            print(f'Masukkan data untuk pegawai #{count}')
            nama = input('Nama: ')
            id_pegawai = input('Nomor ID: ')
            dept = input('Departemen: ')

            # Tulis data sebagai record ke file
            file.write(nama + '\n')
            file.write(id_pegawai + '\n')
            file.write(dept + '\n')

            # Tampilkan baris baru
            print()

    # Tampilkan nama file
    print('Record pegawai ditulis ke file pegawai.txt.')

# Panggil fungsi main
main()
```

Contoh output dari program di atas:

```
Berapa banyak record pegawai yang ingin di-input? 3
```

```
Masukkan data untuk pegawai #1
```

```
Nama: Budi Susilo
```

```
Nomor ID: 104955
```

```
Departemen: Marketing
```

```
Masukkan data untuk pegawai #2
```

```
Nama: Ingrid Wijaya
```

```
Nomor ID: 324023
```

```
Departemen: Accounting
```

```
Masukkan data untuk pegawai #3
```

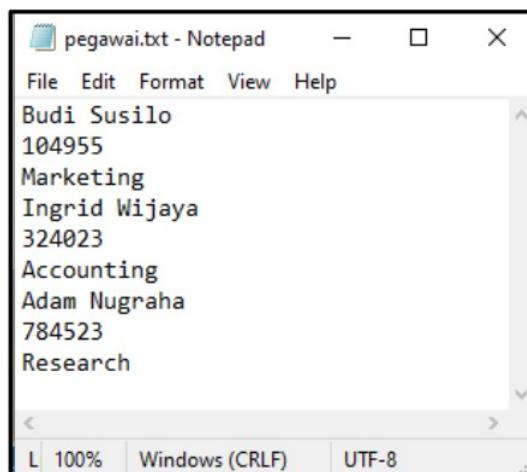
```
Nama: Adam Nugraha
```

```
Nomor ID: 784523
```

```
Departemen: Research
```

```
Record pegawai ditulis ke file pegawai.txt.
```

Setelah program selesai dieksekusi akan terdapat sebuah file `pegawai.txt` dengan isi seperti berikut:



Program berikut membaca file yang berisi record pegawai:

```
# baca_record_pegawai.py
# Program ini menampilkan record dalam
# file pegawai.txt

def main():

    # Buka file pegawai.txt
    with open('pegawai.txt', 'r') as file_pegawai:
        # Baca baris pertama file (field nama dari record pertama)
        nama = file_pegawai.readline()
        # Jika nama tidak kosong lanjutkan membaca field lain
        while nama != '':
            id_pegawai = file_pegawai.readline()
            dept = file_pegawai.readline()

            # Strip karakter baris baru dari field
            nama = nama.rstrip('\n')
            id_pegawai = id_pegawai.rstrip('\n')
            dept = dept.rstrip('\n')
```

```

# Tampilkan record
print(f'Nama: {nama}')
print(f'ID: {id_pegawai}')
print(f'Dept: {dept}')
print()

# Baca nama field ke record berikutnya
nama = file_pegawai.readline()

# Panggil fungsi main
main()

```

Output dari program `baca_record_pegawai.py` di atas:

```

Nama: Budi Susilo
ID: 104955
Dept: Marketing

Nama: Ingrid Wijaya
ID: 324023
Dept: Accounting

Nama: Adam Nugraha
ID: 784523
Dept: Research

```

## 5.6 Eksepsi

Eksepsi adalah error yang terjadi saat program berjalan. Eksepsi biasanya menyebabkan program berhenti secara tiba-tiba.

Perhatikan contoh program berikut:

```

# pembagian.py
# Program ini membagi sebuah angka dengan angka lain

def main():
    # Ambil dua angka
    num1 = int(input('Masukkan sebuah angka: '))
    num2 = int(input('Masukkan sebuah angka lain: '))

    # Bagi num1 dengan num2 dan tampilkan hasil
    hasil = num1 / num2
    print(num1, 'dibagi dengan', num2, 'sama dengan', hasil)

# Panggil fungsi main
main()

```

Program di atas meminta pengguna memasukkan dua buah angka dan melakukan pembagian angka pertama dengan angka kedua. Program ini akan menghasilkan eksepsi jika pengguna memasukkan angka kedua dengan nilai 0. Ini karena kita tidak dapat melakukan pembagian dengan nilai 0. Gambar berikut menjelaskan ini:

```

# pembagian.py
# Program ini membagi sebuah angka dengan angka lain

def main():
    # Ambil dua angka
    num1 = int(input('Masukkan sebuah angka: '))
    num2 = int(input('Masukkan sebuah angka lain: '))

    # Bagi num1 dengan num2 dan tampilkan hasil
    hasil = num1 / num2
    print(num1, 'dibagi dengan', num2, 'sama dengan', hasil)

# Panggil fungsi main
main()

```

Ketika terjadi eksepsi, interpreter memberikan pesan yang disebut traceback yang menjelaskan jenis eksepsi dan letak statement yang menyebabkan eksepsi.

Ketika pengguna meng-input num2 dengan nilai 0, program melakukan pembagian dengan 0 yang menyebabkan terjadinya eksepsi.

Output:

```

Masukkan sebuah angka: 10
Masukkan sebuah angka lain: 0
Traceback (most recent call last):
  File ".\pembagian.py", line 14, in <module>
    main()
  File ".\pembagian.py", line 10, in main
    hasil = num1 / num2
ZeroDivisionError: division by zero

```

Ketika terjadi eksepsi (sering disebut eksepsi di-raise atau diangkat), interpreter memberikan pesan error yang disebut dengan **traceback**. Pada contoh sebelumnya, ketika kita meng dengan nilai 0, terjadi eksepsi dan interpreter memberikan traceback yang menginformasikan letak baris dalam kode program yang menyebabkan terjadinya eksepsi:

```

Traceback (most recent call last):
  File ".\pembagian.py", line 14, in <module>
    main()
  File ".\pembagian.py", line 10, in main
    hasil = num1 / num2
→ ZeroDivisionError: division by zero

```

Baris terakhir dari traceback menunjukkan nama dari eksepsi: ZeroDivisionError (yang berarti Error Pembagian Nol) yang diikuti dengan deskripsi singkat dari error yang menyebabkan terjadinya eksepsi: division by zero (pembagian dengan nol)

Di dalam fungsi main baris 10 adalah statement yang menyebabkan terjadinya eksepsi: hasil = num1 / num2

Eksepsi terjadi saat pemanggilan fungsi main di baris 14

Kita dapat mencegah terjadinya eksepsi pembagian dengan nol pada contoh program sebelumnya dengan menambahkan kondisional yang menguji nilai  seperti berikut:

```

# pembagian2.py
# Program ini membagi sebuah angka dengan angka lain

def main():
    # Ambil dua angka
    num1 = int(input('Masukkan sebuah angka: '))
    num2 = int(input('Masukkan sebuah angka lain: '))

    # Jika num2 tidak 0, lakukan pembagian num1 / num2
    # dan tampilkan hasilnya.
    if num2 != 0:
        hasil = num1 / num2
        print(num1, 'dibagi dengan', num2, 'sama dengan', hasil)
    else:
        print('Tidak dapat membagi dengan nol.')

# Panggil fungsi main
main()

```

Output dari program di atas:

```
Masukkan sebuah angka: 10
Masukkan sebuah angka lain: 0
Tidak dapat membagi dengan nol.
```

Perhatikan pada program di atas kita menambahkan statement `if-else` dengan kondisi angka pembagi `num2` tidak sama dengan 0. Jika ya, operasi pembagian dilakukan tetapi jika tidak program menampilkan pesan tidak dapat melakukan pembagian.

Contoh di atas merupakan contoh cara untuk mencegah terjadinya eksepsi pada program. Namun, beberapa eksepsi tidak dapat kita cegah. Misalkan, perhatikan program berikut:

```
# upah_kotor.py
# Program ini menghitung upah kotor

def main():
    # Ambil banyak jam bekerja
    jam_kerja = int(input('Berapa jam Anda bekerja? '))

    # Ambil upah per jam
    upah_jam = float(input('Masukkan upah per jam: '))

    # Hitung upah kotor
    upah_kotor = jam_kerja * upah_jam

    # Tampilkan upah kotor
    print(f'Upah kotor: Rp.{upah_kotor:.2f}')

# Panggil fungsi main
main()
```

Misalkan pengguna memasukkan `'empat puluh'` pada prompt yang meminta jam kerja, maka eksekusi fungsi `int()` akan mengangkat sebuah eksepsi `ValueError`:

```
Berapa jam Anda bekerja? empat puluh
Traceback (most recent call last):
  File ".\upah_kotor.py", line 18, in
    main()
  File ".\upah_kotor.py", line 6, in main
    jam_kerja = int(input('Berapa jam Anda bekerja? '))
ValueError: invalid literal for int() with base 10: 'empat puluh'
```

Pada program di atas, kita tidak dapat mencegah terjadinya eksepsi menggunakan statement `if` yang menguji tipe data yang diinput pengguna, karena semua input adalah string. Untuk menangani eksepsi seperti ini, kita menggunakan statement `try/except`.

## 5.6.1 Menangani Eksepsi

Python menyediakan sebuah cara untuk meng-*handle* (menangani) eksepsi dengan menuliskan *exception handler* menggunakan statement `try/except`. Berikut adalah syntax statement `try/except`:

```
try:  
    <statement>  
    ...  
    <statement>  
except <nama_eksepsi>:  
    <statement>  
    ...  
    <statement>
```

Statement `try/except` terdiri dari dua klausa: klausa `try` dan klausa `except`. Pada klausa `try`:

1. Header dituliskan dengan keyword `try` yang diikuti titik dua.
2. Body berisi rangkaian statement-statement yang jika saat pengeksekusian salah satu statementnya terjadi eksepsi ingin kita *handle* (tangani).

Pada klausa `except`:

1. Klausa `try` selalu diikuti klausa `except`.
2. Header klausa `except` dituliskan dengan keyword `except` yang diikuti nama eksepsi (seperti `ZeroDivisionError`) lalu titik dua.
3. Body klausa `except` berisi rangkaian statement-statement yang dieksekusi ketika terjadi eksepsi dengan nama yang dituliskan pada header. Jika tidak terjadi eksepsi, statement-statement pada body ini dilewati.
4. Biasanya statement pada klausa `except` berisi tampilan teks yang memberitahukan telah terjadi error pada program.

Pada program `upah_kotor.py`, eksepsi `ValueError` dapat terjadi ketika pengguna meng karakter-karakter bukan angka. Kita dapat menambahkan *exception handler* untuk menangani eksepsi `ValueError` pada program `upah_kotor.py` seperti pada program berikut:

```
# upah_kotor.py  
# Program ini menghitung upah kotor  
  
def main():  
  
    try:  
        # Ambil banyak jam bekerja  
        jam_kerja = int(input('Berapa jam Anda bekerja? '))  
  
        # Ambil upah per jam  
        upah_jam = float(input('Masukkan upah per jam: '))  
  
        # Hitung upah kotor  
        upah_kotor = jam_kerja * upah_jam  
  
        # Tampilkan upah kotor  
        print(f'Upah kotor: Rp.{upah_kotor:.2f}')  
  
    except ValueError:  
        print('ERROR: Jam kerja dan upah per jam harus berupa angka.')  
  
    # Panggil fungsi main  
main()
```

Berikut adalah output program di atas jika tidak terjadi eksepsi:

```
Berapa jam Anda bekerja? 40
Masukkan upah per jam: 200000
Upah kotor: Rp.8,000,000.00
```

Berikut adalah output program di atas jika terjadi eksepsi:

```
Berapa jam Anda bekerja? empat puluh
ERROR: Jam kerja dan upah per jam harus berupa angka.
```

## 5.6.2 Menangani Eksepsi Saat Bekerja dengan File

Ketika kita bekerja dengan file, eksepsi yang mungkin terjadi adalah saat kita membuka file dengan nama tertentu untuk dibaca (modus `'r'`) namun tidak ditemukan file dengan nama tersebut. Berikut adalah contoh program yang dapat menghasilkan eksepsi jika file tidak ditemukan:

```
# tampilan_file.py
# Program ini menampilkan isi dari file

def main():
    # Minta nama file
    nama_file = input('Masukkan nama file: ')

    # Buka file
    infile = open(nama_file, 'r')

    # Baca isi dari file
    isi_file = infile.read()

    # Print isi file
    print(isi_file)

    # Tutup file
    infile.close()

# Panggil fungsi main
main()
```

Pada program di atas, ketika fungsi `open` mencoba membuka file dengan nama yang diinput pengguna dan tidak menemukan file tersebut, maka statement ini meng-*raise* sebuah eksepsi. Berikut adalah contoh output yang menghasilkan eksepsi:

```
Masukkan nama file: namafile.txt
Traceback (most recent call last):
  File ".\tampilan_file.py", line 18, in
    main()
    File ".\tampilan_file.py", line 9, in main
      infile = open(nama_file, 'r')
FileNotFoundError: [Errno 2] No such file or directory:
'namafile.txt'
```

Perhatikan pada contoh output di atas, sebuah eksepsi `FileNotFoundException` di-*raise* ketika program tidak dapat menemukan file dengan nama yang di-input pengguna. Kita dapat menangani eksepsi `FileNotFoundException` dengan memodifikasi program di atas menjadi seperti berikut:

```

# tampilkan_file2.py
# Program ini menampilkan isi dari file

def main():

    try:
        # Minta nama file
        nama_file = input('Masukkan nama file: ')
        # Buka file
        infile = open(nama_file, 'r')
        # Baca isi dari file
        isi_file = infile.read()
        # Print isi file
        print(isi_file)
        # Tutup file
        infile.close()
    except FileNotFoundError:
        print('File tidak ditemukan.')

# Panggil fungsi main
main()

```

Output dari program di atas jika file tidak ditemukan:

```

Masukkan nama file: namafайл.txt
File tidak ditemukan.

```

Perhatikan pada program `tampilkan_file2.py` di atas. Kita menuliskan statement-statement yang jika terjadi eksepsi pada pengeksekusianya ingin kita *handle* (tangani) pada body klausa `try`. Kemudian, kita menuliskan header klausa `except` dengan nama eksepsi `FileNotFoundException` dan di dalam body `except` tersebut kita menuliskan statement-statement yang dieksekusi ketika terjadi eksepsi `FileNotFoundException`.

Kita dapat juga menggunakan statement `with` di dalam *exception handler*. Program sebelumnya jika dituliskan menggunakan statement `with`:

```

# tampilkan_file3.py
# Program ini menampilkan isi dari file

def main():

    try:
        # Minta nama file
        nama_file = input('Masukkan nama file: ')

        # Buka file
        with open(nama_file, 'r') as infile:
            # Baca isi dari file
            isi_file = infile.read()
            # Print isi file
            print(isi_file)

    except FileNotFoundError:
        print('File tidak ditemukan.')

```

```
# Panggil fungsi main
main()
```

### 5.6.3 Menggunakan Satu Klaus `except` untuk Semua Eksepsi

Kita dapat menangani semua eksepsi yang mungkin terjadi pada pengeksekusian statement-statement dalam klaus `try` dengan menuliskan header klaus `except` tanpa nama eksepsi. Berikut adalah contoh program dengan klaus `except` tanpa nama eksepsi:

```
# laporan_penjualan1.py
# Program ini menampilkan total jumlah penjualan
# dalam file penjualan.txt

def main():
    # Inisialisasi akumulator
    total = 0.0

    try:
        # Proses file penjualan.txt
        with open('penjualan.txt', 'r') as infile:
            # Baca nilai-nilai dalam file dan akumulasikan
            for line in infile:
                jumlah = float(line)
                total += jumlah
        # Tampilkan total
        print(f'Total penjualan: {total:.2f}')

    except:
        print('Telah terjadi kesalahan.')

# Panggil fungsi main.
main()
```

Output program `laporan_penjualan1.py` di atas jika tidak terjadi eksepsi:

```
Total penjualan: 41,840,603.94
```

Output program `laporan_penjualan1.py` di atas jika program tidak dapat menemukan file bernama `penjualan.txt` yang menyebabkan terjadinya eksepsi:

```
Telah terjadi kesalahan.
```

Klaus `except` tanpa nama eksepsi berarti jika terjadi eksepsi apapun saat pengeksekusian statement-statement dalam body `try` maka statement-statement di dalam body `except` akan dieksekusi.

### 5.6.4 Menangani Multiple Eksepsi

Menggunakan sebuah klaus `except` tanpa nama eksepsi untuk menangani semua kemungkinan eksepsi yang terjadi membuat program hanya dapat memberikan pesan umum untuk semua eksepsi. Hal ini akan membuat pengguna tidak dapat mengetahui apa sebenarnya yang menyebabkan program berhenti.

Kita dapat menuliskan klausa `except` lebih dari satu untuk menangani lebih dari satu eksepsi. Sebagai contoh, dalam membuat sebuah program yang melakukan pengolahan data-data numerik dalam sebuah file, eksepsi-eksepsi yang mungkin terjadi antara lain:

1. Eksepsi `FileNotFoundException`: ketika file tidak ditemukan.
2. Eksepsi `ValueError`: ketika data pada file tidak dapat dikonversi ke data numerik.

Misalkan kita membuat program yang membaca file bernama `data_penjualan.txt` yang berisi data-data penjualan dalam satu bulan seperti berikut:

```
7584859.26
6544896.34
4568932.45
8493021.90
6859903.23
7788990.76
```

Kita dapat menuliskan program yang menghitung jumlah penjualan dari file `data_penjualan.txt` menggunakan exception handler seperti berikut:

```
# laporan_penjualan2.py
# Program ini menampilkan total jumlah penjualan
# dalam file penjualan.txt

def main():
    # Inisialisasi akumulator
    total = 0.0

    try:
        # Proses file_penjualan.txt
        with open('penjualan.txt', 'r') as infile:
            # Baca nilai-nilai dalam file dan akumulasikan
            for line in infile:
                jumlah = float(line)
                total += jumlah
            # Tampilkan total
            print(f'Total penjualan: {total:.2f}')

    except FileNotFoundError:
        print('File tidak ditemukan.')

    except ValueError:
        print('Terdapat data non-numerik dalam file.')

    except:
        print('Kesalahan yang tidak diketahui telah terjadi.')

# Panggil fungsi main.
main()
```

Output program `laporan_penjualan2.py` di atas jika tidak terjadi eksepsi:

```
Total penjualan: 41,840,603.94
```

Output program `laporan_penjualan2.py` di atas jika file `penjualan.txt` tidak ditemukan:

```
File tidak ditemukan.
```

Output program `laporan_penjualan2.py` di atas jika isi file `penjualan.txt` mengandung karakter bukan numerik:

```
Terdapat data non-numerik dalam file.
```

Kasus di atas, ketika program menemukan data non numerik dalam file `penjualan.txt` dapat terjadi jika file tersebut berisi seperti berikut:

```
7584859.26  
6544896.34  
4568932.45  
8493021.90  
6859903.23  
7788990.76d
```

## 5.6.5 Klausula `else`

Statement `try/except` dapat ditambahkan klausula `else` setelah klausula `except` menjadi seperti syntax berikut:

```
try:  
    <statement>  
    ...  
    <statement>  
except <nama_eksepsi>:  
    <statement>  
    ...  
    <statement>  
else:  
    <statement>  
    ...  
    <statement>
```

Jika statement `try/except` dituliskan dengan klausula `else` maka statement-statement dalam body klausula `else` dieksekusi hanya jika tidak terjadi eksepsi.

Berikut adalah contoh program yang memodifikasi program `laporan_penjualan.py` sebelumnya dengan menambahkan klausula `else` pada statement `try/except`:

```
# laporan_penjualan3.py  
# Program ini menampilkan total jumlah penjualan  
# dalam file penjualan.txt  
  
def main():  
    # Inisialisasi akumulator  
    total = 0.0  
  
    try:
```

```

# Proses file_penjualan.txt
with open('penjualan.txt', 'r') as infile:
    # Baca nilai-nilai dalam file dan akumulasikan
    for line in infile:
        jumlah = float(line)
        total += jumlah
    # Tampilkan total
    print(f'Total penjualan: {total:.2f}')

except:
    print('Telah terjadi kesalahan.')

else:
    print('File telah diproses tanpa masalah.')

# Panggil fungsi main.
main()

```

Output program `laporan_penjualan3.py` di atas jika tidak terjadi eksepsi:

```

Total penjualan: 41,840,603.94
File telah diproses tanpa masalah.

```

Output program `laporan_penjualan3.py` di atas jika file `penjualan.txt` tidak ditemukan:

```
Telah terjadi kesalahan.
```

## 5.6.6 Klausula `finally`

Statement `try/except` dapat juga ditambahkan klausula `finally` setelah klausula `except` sehingga mempunyai syntax seperti berikut:

```

try:
    <statement>
    ...
    <statement>
except <nama_eksepsi>:
    <statement>
    ...
    <statement>
else:
    <statement>
    ...
    <statement>
finally:
    <statement>
    ...
    <statement>

```

Jika statement `try/except` dituliskan dengan klausula `finally`, maka statement-statement dalam body klausula `finally` akan selalu dieksekusi terjadi atau tidak terjadinya eksepsi.

Berikut adalah contoh program yang memodifikasi program `laporan_penjualan3.py` sebelumnya dengan menambahkan klausula `finally` pada statement `try/except`:

```
# laporan_penjualan4.py
# Program ini menampilkan total jumlah penjualan
# dalam file penjualan.txt

def main():
    # Inisialisasi akumulator
    total = 0.0

    try:
        # Proses file_penjualan.txt
        with open('penjualan.txt', 'r') as infile:
            # Baca nilai-nilai dalam file dan akumulasikan
            for line in infile:
                jumlah = float(line)
                total += jumlah
            # Tampilkan total
            print(f'Total penjualan: {total:.2f}')

    except:
        print('Telah terjadi kesalahan.')

    else:
        print('File telah diproses tanpa masalah.')

    finally:
        print('Terima kasih.')

# Panggil fungsi main.
main()
```

Output program `laporan_penjualan4.py` di atas jika tidak terjadi eksepsi:

```
Total penjualan: 41,840,603.94
File telah diproses tanpa masalah.
Terima kasih.
```

Output program `laporan_penjualan4.py` di atas jika terjadi eksepsi:

```
Telah terjadi kesalahan.
Terima kasih.
```

**REFERENSI:**

- [1] Gaddis, Tony. 2012. *Starting Out With Python Second Edition*. United States of America: Addison-Wesley.