# 8. Mengenal Exception

## Jenis – Jenis Exception

*Exception* adalah sebuah cara di Python untuk menjebak *error*, dan menangangi *error* tak terduga pada program Python yang Anda tulis. *Exception* akan tetap menjalankan baris kode program dibawah bagian kode program yang *error*. Hal ini mempermudah proses *debugging*. Lalu apa bedanya jika kita menggunakan kondisional biasa yang menggunakan **if** untuk mencegah *error* ? Pertama Anda harus mencari cara untuk menangkap nilai – nilai yang *error*, misal ketika membuka *file* Anda harus menggunakan *method – method* yang ada pada *file* untuk mengetahui *error* atau tidak. Kedua dengan menggunakan kondisional **if** biasa, program yang Anda tulis akan langsung dihentikan ketika *error* terjadi. Ketiga pengambilan *error* akan otomatis ditangani oleh Python dan *error* tersebut akan ditangani sesuai dengan penanganan yang Anda lakukan, dan baris kode program dibawahnya akan tetap dieksekusi.

Python sendiri sudah menyediakan beberapa *standard error* yang dapat digunakan oleh *programmer* dalam menjaga pengembangan aplikasinya dari *error* yang tak terduga. Anda sendiri dapat membuat *error* menurut definisi Anda. Hal tersebut akan diulas di bagian akhir bab ini. Berikut adalah beberapa *standard error* yangt terdapat di Python :

| No | Nama *Exception* | Keterangan |
|---|---|---|
| 1 | Exception | Menangani semua *exception* |
| 2 | StopIteration | *Exception* ini muncul ketika *method* next() tidak menunjuk ke objek apapun saat iterasi |
| 3 | SystemExit | *Exception* ini muncul ketika sys.exit() dipanggil |
| 4 | StandardError | *Exception* untuk menangani semua *built-in exception* kecuali StopIteration dan SystemExit |
| 5 | ArithmeticError | *Exception* untuk menangani *error* saat perhitungan angka |
| 6 | OverflowError | *Exception* ini muncul ketika perhitungan angka melebihi batas maksimum dari tipe angka yang dihitung |
| 7 | FloatingPointError | *Exception* ini muncul ketika terdapat kegagalan pada perhitungan angka bertipe *float* |
| 8 | ZeroDivisionError | *Exception* ini muncul jika ada pembagian atau modulus oleh 0 terhadap angka tipe apapun |
| 10 | AssertionError | *Exception* ini muncul ketika terjadi kegagalan pada saat perintah **assert** dijalankan |
| 11 | AttributeError | *Exception* ini muncul ketika gagal menunjuk atribut dari suatu objek |
| 12 | EOFError | *Exception* ini muncul ketika tidak ada input saat |

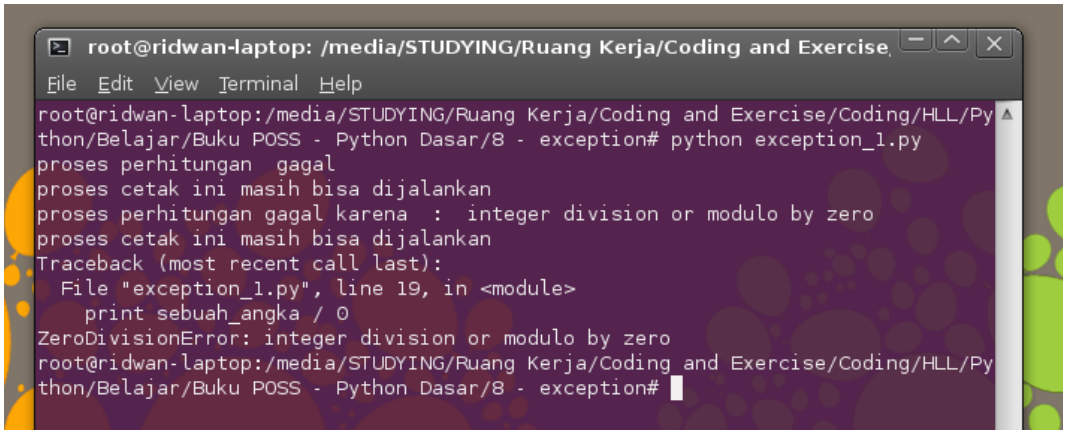
| | | |
|---|---|---|
| | | menggunakan *function* raw_input() atau input dan telah mencapai bagian akhir file saat pembacaan file. |
| 13 | ImportError | *Exception* ini muncul ketika gagal saat menggunakan **import** |
| 14 | KeyboardInterrupt | *Exception* ini muncul ketika user meng-*interrupt* eksekusi program, biasanya ketika menekan kombinasi **ctrl + c** |
| 15 | LookupError | *Exception* muncul ketika gagal pada saat proses *look up* |
| 16 | IndexError | *Exception* ini muncul ketika tidak ada indeks yang dituju pada struktur data seperti **list** atau **tuple** |
| 17 | KeyError | *Exception* ini muncul ketika tidak ada *key* yang dituju pada *dictionary* |
| 18 | NameError | *Exception* ini muncul ketika variabel tidak ditemukan pada lingkup lokal di suatu function dan kondisional atau pada lingkup global |
| 19 | UnboundLocalError | *Exception* ini muncul ketika mencoba mengakses variabel lokal di *function* atau *method* tapi belum ada nilainya |
| 20 | EnvironmentError | *Exception* in muncul ketika terjadi kegagalan diluar lingkup Python |
| 21 | IOError | *Exception* in muncul ketika proses *input/output* gagal, misal saat menggunakan **print** atau saat membuka *file* |
| 22 | OSError | *Exception* ini muncul ketika terjadi kegagalan pada sistem operasi yang digunakan |
| 23 | SyntaxError | *Exception* ini muncul ketika terjadi kesalahan pada penggunaan sintaks Python |
| 24 | IndentationError | *Exception* ini muncul ketika indentasi pada blok kode tidak sesuai penggunaannya. |
| 25 | SystemError | *Exception* ini muncul ketika terdapat masalah internal pada *interpreter*, saat *error* ini muncul interpreter tidak akan keluar |
| 26 | TypeError | *Exception* ini muncul jika ada kesalahan tipe data saat proses perhitungan, misal huruf dibagi angka |
| 27 | ValueError | *Exception* ini muncul ketika argumen yang tidak sesuai diterima oleh *builtin function* |
| 28 | RuntimeError | *Exception* ini muncul ketika terjadi kesalahan yang tidak masuk kategori manapun |
| 29 | NotImplementedEror | *Exception* ini muncul ketika *abstract method* dari suatu *class* tidak diimplementasikan di *class* yang mewarisinya. |

Agar lebih paham dibawah ini ada beberapa contoh kode yang penggunaan *exception*-nya sangat sering digunakan. Sebagai contoh pertama berikut terdapat kode yang berisi pembagian oleh angka nol.

*listing : exception_1.py*

```
sebuah_angka = 29

try:
    print sebuah_angka / 0
except:
    print "proses perhitungan  gagal "

print "proses cetak ini masih dapat dijalankan  "


try:
    print sebuah_angka / 0
except ZeroDivisionError, e:
    print "proses perhitungan gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "


print sebuah_angka / 0

# jika tidak memakai exception maka proses berikutnya tidak akan dijalankan
print "apakah proses cetak ini masih dapat dijalankan ??? "
```

Di dalam try terdapat kode yang kemungkinan akan memunculkan exception. Sedangkan di dalam except adalah kode yang akan dieksekusi jika exception tersebut muncul.  Pada try-except yang pertama, semua error akan ditangani dan Anda tidak akan mengetahui jenis exception apa yang yang ditangani. Pada try-except yang kedua, Anda memprediksi akan menangani error jika terjadi pembagian oleh nol. Manakah yang lebih baik ? Pastikan Anda sudah memiliki perkiraan apa saja error yang akan terjadi sehingga saat debugging nanti akan mempermudah Anda untuk memperbaiki kode program Anda. Pada blok kode try-except sekalipun error kode dibawahnya akan tetap dieksekusi. Pada proses perhitungan di bagian akhir tidak ditangani oleh try-except sehingga kode dibawahnya tidak dieksekusi. Berikut hasil yang diberikan jika kode dieksekusi :



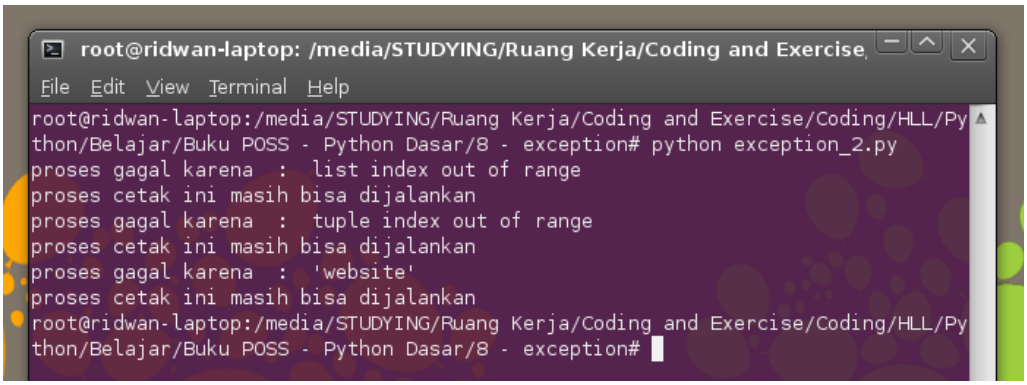

<< gambar 8.1 hasil eksekusi exception_1.py >>

Contoh lain yang umum dipakai adalah IndexError dan KeyError. Kedua *error* ini umum dipakai saat operasi **list**, **tuple**, dan *dictionary*. Berikut terdapat contoh menunjuk indeks dan *key* yang tidak terdapat pada **list**, **tuple**, dan *dictionary* yang didefinisikan dalam kode dibawah ini.

*listing : exception_2.py*

```
sebuah_list = [1, 2, 3, 4, 5]
sebuah_tuple = (1, 2, 3, 4, 5)
sebuah_dictionary = {'nama':'Mangaraja', 'email':'mangaraja@yahoo.com'}

try:
    print sebuah_list[10]
except IndexError, e:
    print "proses gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "

try:
    print sebuah_tuple[10]
except IndexError, e:
    print "proses gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "

try:
    print sebuah_dictionary['website']
except KeyError, e:
    print "proses gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Pada contoh diatas “sebuah_list” dan “sebuah_tuple” ditangani oleh *try-except* yang menangani *exception* IndexError. Pada masing – masing blok, kita ingin mencoba indeks yang tidak ada pada **list** dan **tuple** tersebut. Sedangkan pada blok *try-except* untuk *dictionary*, kita ingin mencoba menunjuk *key* “website” tapi karena *key* tersebut tidak ada, maka akan muncul exception KeyError.




<< gambar 8.2 hasil eksekusi exception_2.py >>

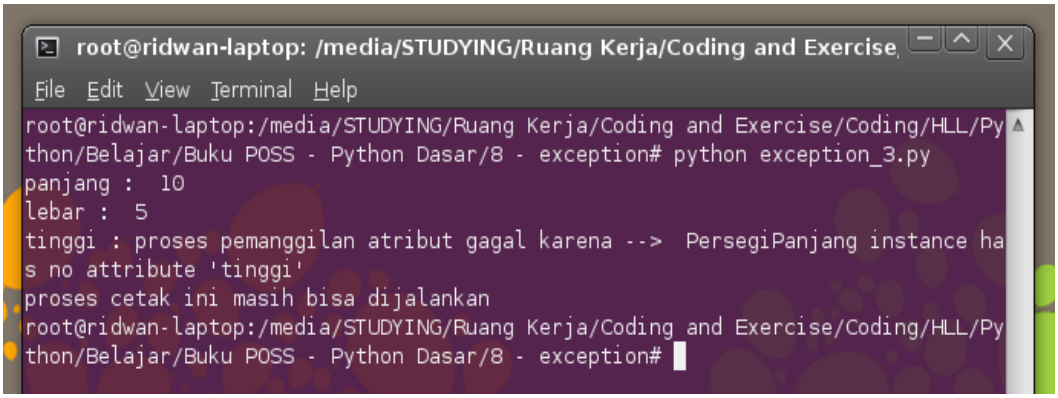
Berikutnya contoh *exception* yang tak kalah populer lainnya adalah AttributeError. *Exception* ini muncul ketika sebuah *class* tidak memiliki atribut (variabel) yang diakses oleh *programmer*. Hal ini sangat penting untuk diperhatikan ketika merancang sebuah aplikasi berbasis objek. Anda harus memeriksa apakah atribut yang Anda akses pada sebuah kelas ada pada saat perancangan atau tidak. Jika tidak yakin gunakanlah *try-except* untuk menjebak AttributeError tersebut.

*listing : exception_3.py*

```
class PersegiPanjang:
    panjang = 0
    lebar = 0
    def __init__(self, p, l):
        self.panjang = p
        self.lebar = l


prsg_pjg = PersegiPanjang(10, 5)

try:
    print "panjang : ", prsg_pjg.panjang
    print "lebar : ", prsg_pjg.lebar
    print "tinggi :", prsg_pjg.tinggi
except AttributeError, e:
    print "proses pemanggilan atribut gagal karena --> ", e
```

```
print "proses cetak ini masih dapat dijalankan"
```

Pada contoh diatas, kita ingin mencoba mengakses atribut tinggi pada objek prsg_pjg. Sebelumnya tahapan yang dilalui adalah proses instansiasi, dimana kita memanggil sebuah *template* objek yang akan dibentuk kedalam sebuah variabel. Kemudian di bagian *try-except* tersebut kita coba akses atribut tinggi. Karena atribut tersebut tidak ada di kelas persegi panjang, maka *exception* AttributeError akan muncul.



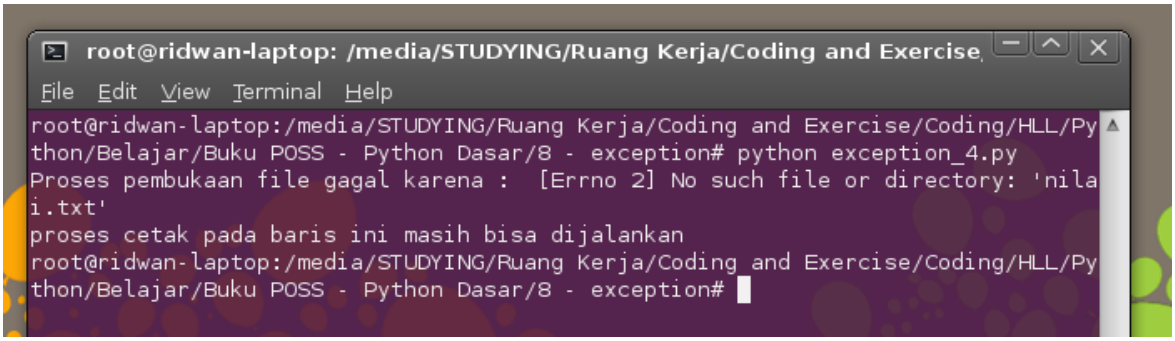

<< gambar 8.3 hasil eksekusi exception_3.py >>

Contoh yang terakhir dari sekian banyak *exception* yang terdapat di Python adalah IOError. *Exception* ini biasa terjadi ketika proses *input* data, saat mencetak data, atau saat operasi *file*. Pada contoh berikut kita akan membuka sebuah *file*, tapi *file* tersebut tidak ada. Secara *default* jika kita membuka *file* tanpa menyertakan mode pembacaan, maka mode tersebut adalah mode 'r' yang artinya *read* atau baca.

*listing : exception_4.py*

```
try :
    f = open('nilai.txt')
except IOError, e:
    print "Proses pembukaan file gagal karena : ", e

print "proses cetak pada baris ini masih dapat dijalankan"
```

Pada contoh diatas kita ingin *file* nilai.txt, tapi karena *file* tersebut belum pernah ditulis sebelumnya maka *exception* akan muncul yaitu IOError. Selain digunakan untuk *file*, IOError dapat terjadi juga saat pembacaan *built-in storage* milik Python seperti saat pembacaan pickle, shelve, dan marshal.




<< gambar 8.4 hasil eksekusi exception_4.py >>

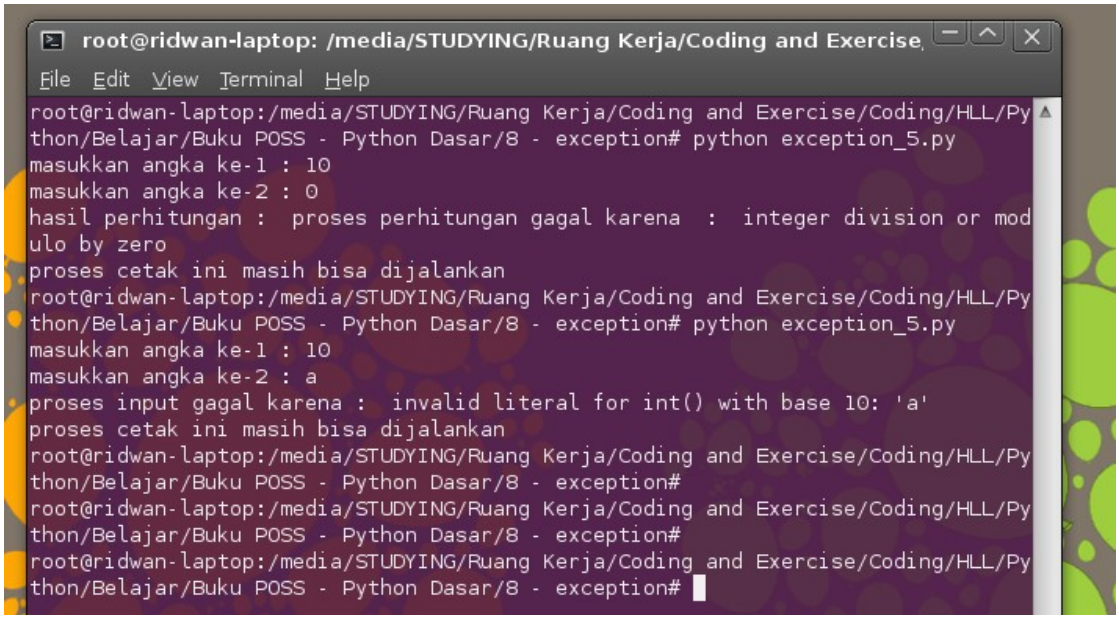
## Menyusun Multiple Except

Apakah kita dapat menangkap *exception* dalam satu blok *try-except* ?. Di Python sendiri terdapat fitur *multiple except*, yaitu kita dapat menangkap *exception* dengan baris **except** yang berbeda. Hal ini dilakukan jika kita ingin memberikan perlakuan berbeda kepada setiap *exception* yang ditangani. Lebih lemgkapnya pantau kode dibawah ini.

*listing : exception_5.py*

```
try:

    angka1 = int(raw_input('masukkan angka ke-1 : '))
    angka2 = int(raw_input('masukkan angka ke-2 : '))

    print 'hasil perhitungan : ', angka1 / angka2

except ZeroDivisionError, e:
    print "proses pembagian gagal karena  : ", e
except ValueError, e:
    print "proses perhitungan gagal karena : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Pada kode diatas kita mencoba menjebak dua buah *exception* dengan dua baris *except* berbeda. Hal tersebut dilakukan agar perlakuan pada penanganan setiap *exception* memiliki penanganan yang berbeda. Misal pada baris **except** pembagian nol ada informasi “proses pembagian gagal karena : “ , sedangkan di baris **except** nilai *error* terdapat informasi “proses perhitungan gagal karena : “. Jadi dengan menggunakan baris **except** yang berbeda Anda dapat menangani *error* yang berbeda sesuai kebutuhan Anda.




<< gambar 8.5 hasil eksekusi exception_5.py >>

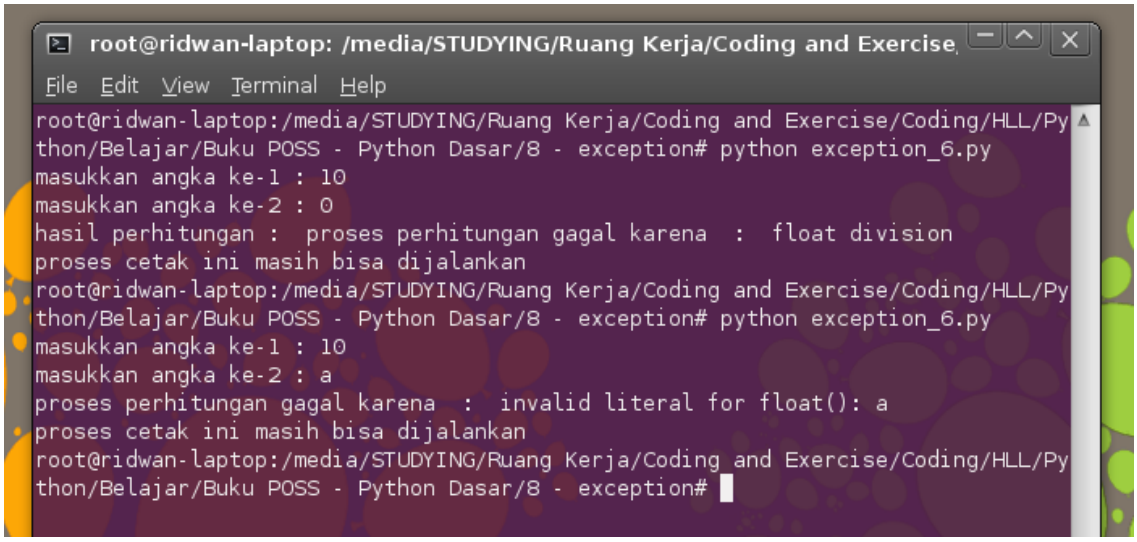
## Menggunakan Multiple Exception

Berbeda sedikit pada contoh sebelumnya, jika pada setiap *exception* ditangani oleh setiap baris **except**. Maka pada kaidah *multiple exception* di satu **except** menangani beberapa *exception*. Bedanya, semua *exception* yang ditangani baris **except** tersebut akan mendapat penanganan yang sama.

*listing : exception_6.py*

```
try:

    angka1 = float(raw_input('masukkan angka ke-1 : '))
    angka2 = float(raw_input('masukkan angka ke-2 : '))

    print 'hasil perhitungan : ', angka1 / angka2

except (ZeroDivisionError, ValueError, TypeError),  e:
    print "proses perhitungan gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Kode diatas jika dieksekusi akan muncul tampilan seperti berikut :




<< gambar 8.6 hasil eksekusi exception_6.py >>

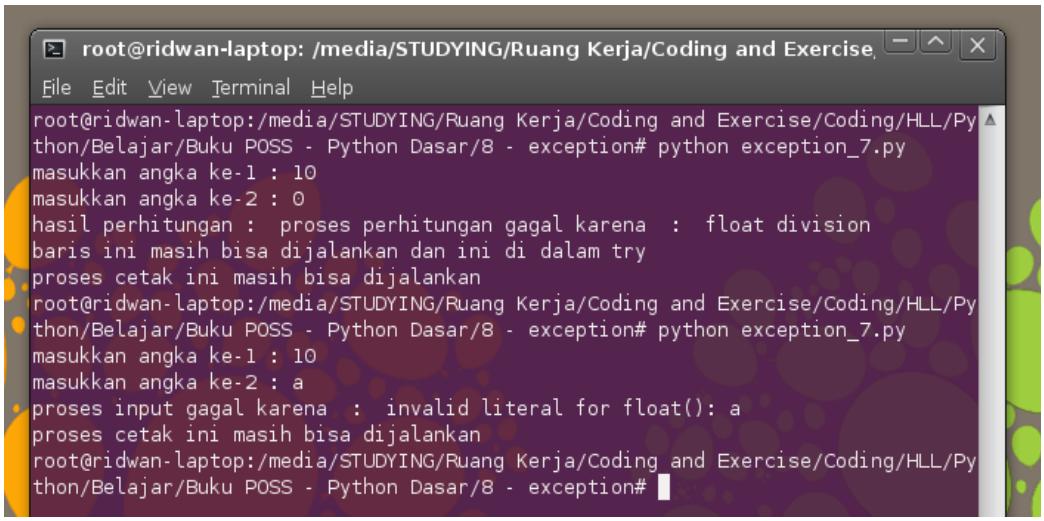
## Try-Except Bersarang

Mirip seperti kondisional atau perulangan yang dapat ditambahkan blok kode kondisional atau perulangan didalamnya. *Try-except* pun mempunyai kaidah yang sama dimana *try-except* dapat disimpan didalam *try-except* yang lainnya. Prioritasnya adalah tangani yang luar terlebih dahulu. Jika terjadi di *try-except* terluar maka blok kode didalamnya yang terdapat *try-except* tidak akan dieksekusi. Jika di blok luar tidak terdapat *error*. Maka penanganan *exception* di *try-except* bagian dalam akan dilakukan.

*listing : exception_7.py*

```
try:

    angka1 = float(raw_input('masukkan angka ke-1 : '))
    angka2 = float(raw_input('masukkan angka ke-2 : '))

    try :
        print 'hasil perhitungan : ', angka1 / angka2
    except ZeroDivisionError, e:
        print "proses perhitungan gagal karena  : ", e

except ValueError,  e:
    print "proses input gagal karena  : ", e

print "proses cetak ini masih dapat dijalankan  "
```

Jika pada contoh exception_5.py baris **except** ZeroDivisionError disimpan di tingkat pertama, sekarang baris tersebut disarangkan di *try-except* yang utama. Dengan demikian Anda dapat menangani *exception* dari dalam secara langsung.




<< gambar 8.7 hasil eksekusi exception_7.py >>

## Membuat Exception Sendiri

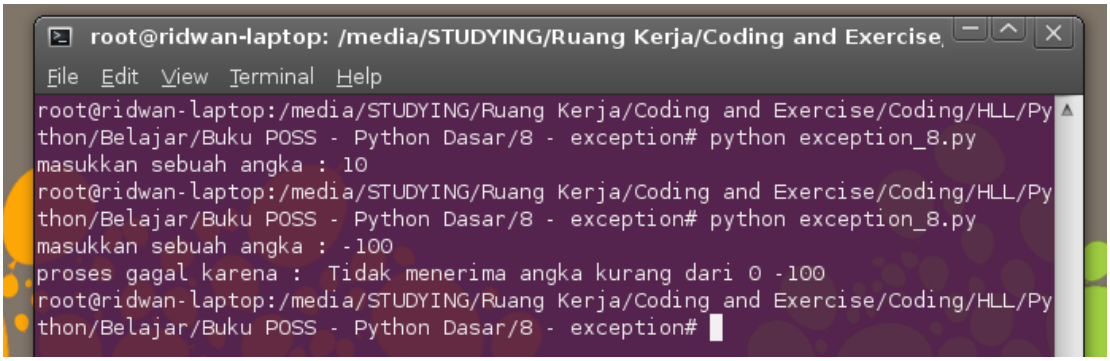
Lalu apakah kita terbatas pada penanganan *standard exception* Python saja ?. Anda dapat membuat *exception* Anda sendiri dengan membuat sebuah kelas yang diturunkan dari kelas Exception. Dengan cara tersebut, Anda dapat membuat *exception* Anda sesuai kebutuhan pada kasus yang akan Anda tangani. Misal kita ingin membuat sebuah *exception* jika angka yang dimasukkan adalah angka negatif. Pertama kita buat dulu **class** nya dengan nama yang diingikan, turunkan dari kelas Exception, dan tentukan penanganan *error* pada *method – method* di kelas tersebut.

*listing : exception_8.py*

```
class NegativeValueError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        s = "Tidak menerima angka kurang dari 0 " + str(self.value)
        return s

def cekAngka(angka):
    if angka < 0:
        raise NegativeValueError(angka)

try:
    sebuah_angka = int(raw_input("masukkan sebuah angka : "))
    cekAngka(sebuah_angka)
except (NegativeValueError, TypeError), e:
```

```
   print "proses gagal karena : ", e
```

Untuk memanggil *exception*-nya kita memerlukan *keyword* **raise** ketika *exception* tersebut dimunculkan maka *exception* akan ditangani **except** dan mengeluarkan pesan *error*-nya. Pesan tersebut berasal dari *function* __str__() yang sebelumnya telah kita definisikan pada kelas NegativeValueError.




<< gambar 8.8 hasil eksekusi exception_8.py >>

## Menggunakan “finally” pada Try-Except

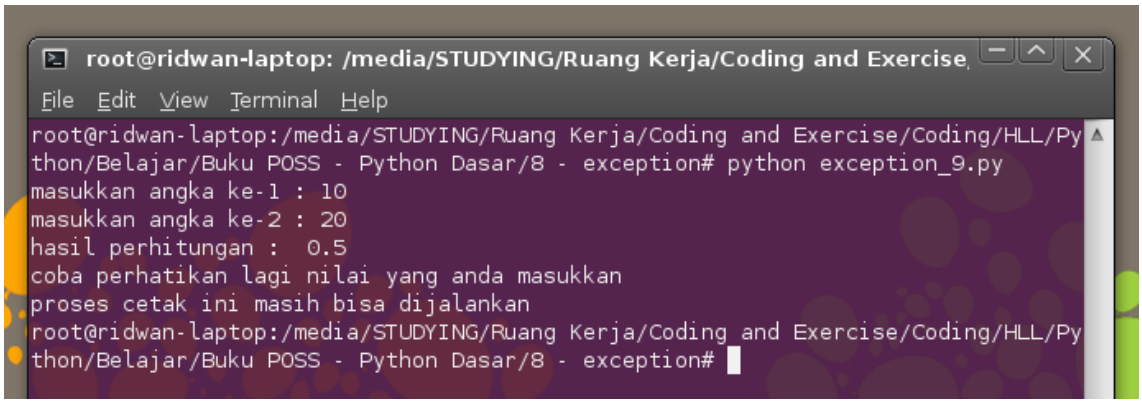
Dan akhirnya sekarang kita membahas **finally**. Keyword ini digunakan untuk menentukan penanganan apa yang harus dilakukan baik ketika *exception* muncul atau tidak. Misal saat mengambil data dari *database*, kita tidak akan tahu ada kegagalan apa yang akan terjadi. Agar program kita tetap aman dan data tidak rusak. Maka baik terjadi kegagalan atau tidak koneksi ke *database* harus ditutup. Hal tersebut juga bisa terjadi saat pembacaan *file*. Untuk mencegah kerusakan *file*, baik akan terjadi *error* atau tidak, *file* harus ditutup. Di blok **finally** ini penanganan yang terjadi ketika *exception* muncul atau tidak disimpan.

*listing : exception_9.py*

```
try:

   angka1 = float(raw_input('masukkan angka ke-1 : '))
   angka2 = float(raw_input('masukkan angka ke-2 : '))

   try :
      print 'hasil perhitungan : ', angka1 / angka2
   except ZeroDivisionError, e:
      print "proses perhitungan gagal karena  : ", e

except ValueError,  e:
   print "proses input gagal karena  : ", e
finally:
```

```
    print "coba perhatikan lagi nilai yang anda masukkan "


print "proses cetak ini masih dapat dijalankan  "
```

Kode diatas jika dieksekusi akan muncul tampilan seperti berikut :




<< gambar 8.9 hasil  eksekusi exception_9.py  >>

# 9. Membuat File

## Pengenalan File

Biasanya jika kita tidak menggunakan *file*, hasil pemrosesan data hanya akan disimpan di *main memory*. Setelah program dihentikan atau tiba – tiba komputer Anda mati, semua data akan hilang. Lalu bagaimana jika ingin menggunakan kembali data yang sudah diproses sebelumnya ?. Untuk menyimpan data agar bisa diproses di kesempatan selanjutnya, misal komputer dimatikan kemudian dinyalakan lagi hari esoknya. Kita butuh sebuah penyimpanan yang bersifat resident dan disimpan di *secondary storage* seperti *harddisk*. Python sendiri menyediakan beberapa media penyimpanan yang bisa digunakan oleh *programmer* Python, ada *file*, **shelve**, **marshal**, **pickle**, dan **sqlite3**.

Pada bab ini kita akan bahas mengenai *file* berupat txt. *File* di Python bisa berupa txt, csv, atau jenis lainnya. Txt merupakan contoh *file* yang sering digunakan. *File* jenis ini berisi *plain text*. *File* jenis ini menyimpan karakter ascii *standard* yang diterima dari *user*.

Pada pembuatan *file* terdapat beberapa mode dalam manipulasi *file*. Berikut daftar mode manipulasi *file* tersebut :

| No | Mode | Keterangan |
|---|---|---|
| 1 | r | Membuka *file* dan hanya untuk pembacaan saja. *Pointer file* akan ditempatkan di awal *file*. Jika pada saat pembukaan *file* tidak disertakan mode manipulasi *file*, maka mode |