Python Exceptions Handling

Python menyediakan dua fitur yang sangat penting untuk menangani kesalahan tak terduga dalam program Python Anda dan menambahkan kemampuan debugging di dalamnya -

Exception Handling: Ini akan dibahas dalam tutorial ini. Berikut adalah daftar standar Pengecualian yang tersedia dengan Python: Pengecualian Standar.

Penegasan: Ini akan dibahas dalam Asertions dengan tutorial Python.

Daftar Pengecualian Standar –

EXCEPTION NAME DESCRIPTION
Exception Kelas dasar untuk semua pengecualian

StopIteration Dibesarkan ketika metode (iterator) berikutnya dari iterator tidak mengarah ke objek apa pun.

SystemExit Dibesarkan oleh fungsi sys.exit ()
.
StandardError Kelas dasar untuk semua pengecualian built-in kecuali StopIteration dan SystemExit

ArithmeticError Kelas dasar untuk semua kesalahan yang terjadi untuk perhitungan numerik.

OverflowError Dibesarkan saat perhitungan melebihi batas maksimum untuk tipe numerik.

FloatingPointError Dibesarkan saat perhitungan floating point gagal.

ZeroDivisionError Dibesarkan saat pembagian atau modulo nol dilakukan untuk semua tipe numerik.

AssertionError Dibesarkan jika terjadi kegagalan pernyataan Assert.

AttributeError Dibesarkan jika terjadi kegagalan referensi atribut atau penugasan.

EOFError Dibesarkan bila tidak ada input dari fungsi raw _input () atau input () dan akhir file tercapai.

ImportError Dibesarkan saat sebuah pernyataan impor gagal.

KeyboardInterrupt Dibesarkan saat pengguna menyela eksekusi program, biasanya dengan menekan Ctrl + c.

LookupError Kelas dasar untuk semua kesalahan pencarian.

IndexError

KeyError Dibesarkan saat sebuah indeks tidak ditemukan secara berurutan.
Dibesarkan saat kunci yang ditentukan tidak ditemukan dalam kamus.

NameError Dibesarkan saat pengenal tidak ditemukan di namespace lokal atau global.

UnboundLocalError
EnvironmentError Dibesarkan saat mencoba mengakses variabel lokal dalam suatu fungsi atau metode namun tidak ada nilai yang ditugaskan padanya.
Kelas dasar untuk semua pengecualian yang terjadi di luar lingkungan Python.

IOError
IOError Dibesarkan saat operasi input / output gagal, seperti pernyataan cetak atau fungsi open () saat mencoba membuka file yang tidak ada.
Dibangkitkan untuk kesalahan terkait sistem operasi.

SyntaxError
IndentationError Dibesarkan saat ada kesalahan dengan sintaks Python.
Dibesarkan saat indentasi tidak ditentukan dengan benar.

SystemError Dibesarkan saat penafsir menemukan masalah internal, namun bila kesalahan ini ditemui juru bahasa Python tidak keluar.

SystemExit Dibesarkan saat juru bahasa Python berhenti dengan menggunakan fungsi sys.exit (). Jika tidak ditangani dalam kode, menyebabkan penafsir untuk keluar.

TypeError Dibesarkan saat operasi atau fungsi dicoba yang tidak valid untuk tipe data yang ditentukan.

ValueError Dibesarkan saat fungsi bawaan untuk tipe data memiliki jenis argumen yang valid, namun argumen tersebut memiliki nilai yang tidak valid yang ditentukan.

RuntimeError Dibesarkan saat kesalahan yang dihasilkan tidak termasuk dalam kategori apa pun.

NotImplementedError Dibesarkan ketika metode abstrak yang perlu diimplementasikan di kelas warisan sebenarnya tidak diimplementasikan

Penegasan dengan Python

Penegasan adalah pemeriksaan kewarasan yang dapat Anda aktifkan atau matikan saat Anda selesai dengan pengujian program Anda.

Cara termudah untuk memikirkan sebuah pernyataan adalah menyamakannya dengan pernyataan kenaikan gaji-jika (atau lebih akurat, pernyataan kenaikan-jika-tidak). Sebuah ekspresi diuji, dan jika hasilnya muncul salah, pengecualian akan meningkat.

Penegasan dilakukan dengan pernyataan tegas, kata kunci terbaru untuk Python, diperkenalkan di versi 1.5.

Pemrogram sering menempatkan asersi pada awal fungsi untuk memeriksa masukan yang valid, dan setelah pemanggilan fungsi untuk memeriksa keluaran yang valid.

Pernyataan tegas

Ketika menemukan pernyataan tegas, Python mengevaluasi ekspresi yang menyertainya, yang semoga benar. Jika ungkapannya salah, Python menimbulkan pengecualian AssertionError.

Sintaks untuk menegaskan adalah -

menegaskan Ekspresi [, Argumen]

Jika asersi gagal, Python menggunakan ArgumentExpression sebagai argumen untuk AssertionError. Penegasan Pengecualian pengecualian dapat ditangkap dan ditangani seperti pengecualian lainnya dengan menggunakan perintah try-except, namun jika tidak ditangani, mereka akan menghentikan program dan menghasilkan traceback.

Contoh

Berikut adalah fungsi yang mengubah suhu dari derajat Kelvin sampai derajat Fahrenheit. Karena nol derajat Kelvin sedingin yang didapatnya, fungsi itu mundur jika melihat suhu negatif -

```
#!/usr/bin/python
def KelvinToFahrenheit(Temperature):
  assert (Temperature >= 0),"Colder than absolute zero!"
  return ((Temperature-273)*1.8)+32
print KelvinToFahrenheit(273)
print int(KelvinToFahrenheit(505.78))
print KelvinToFahrenheit(-5)
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut –

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in
print KelvinToFahrenheit(-5)
File "test.py", line 4, in KelvinToFahrenheit
assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

Apa itu Exception?

Pengecualian adalah sebuah peristiwa, yang terjadi selama pelaksanaan program yang mengganggu aliran normal instruksi program. Secara umum, ketika skrip Python menemukan situasi yang tidak dapat diatasi, hal itu menimbulkan pengecualian. Pengecualian adalah objek Python yang mewakili kesalahan.

Ketika skrip Python menimbulkan pengecualian, ia harus menangani pengecualian begitu saja sehingga berhenti dan berhenti.

Menangani pengecualian

Jika Anda memiliki beberapa kode yang mencurigakan yang mungkin menimbulkan pengecualian, Anda dapat mempertahankan program Anda dengan menempatkan kode yang mencurigakan di coba: blokir. Setelah dicoba: blokir, sertakan sebuah pernyataan kecuali:, diikuti oleh blok kode yang menangani masalah ini seaman mungkin.

Sintaksis

Berikut adalah sintaks sederhana coba .... kecuali ... blok lain –

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block
```

Berikut adalah beberapa poin penting tentang sintaks yang disebutkan di atas -

Pernyataan percobaan tunggal dapat memiliki banyak kecuali pernyataan. Ini berguna saat blok coba berisi pernyataan yang mungkin membuang berbagai jenis pengecualian.

Anda juga bisa memberikan klausa umum kecuali klausul, yang menangani pengecualian apapun.

Setelah klausa kecuali, Anda bisa memasukkan klausul lain. Kode di blok yang lain dijalankan jika kode di coba: blok tidak menimbulkan pengecualian.

Blok yang lain adalah tempat yang baik untuk kode yang tidak perlu dicoba: perlindungan blokir.

Contoh

Contoh ini membuka file, menulis konten di file, dan keluar dengan anggun karena tidak ada masalah sama sekali -

```
#!/usr/bin/python

try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print "Error: can \'t find file or read data"
else:
```

```
   print "Written content in the file successfully"
   fh.close()
```

Ini menghasilkan hasil sebagai berikut -

Written content in the file successfully

Klausul kecuali tanpa pengecualian

Anda juga dapat menggunakan pernyataan kecuali tanpa pengecualian yang didefinisikan sebagai berikut -

```
try:
   You do your operations here;
   ......................
except:
   If there is any exception, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Pernyataan try-except semacam ini menangkap semua pengecualian yang terjadi. Dengan menggunakan jenis try-except statement ini tidak dianggap sebagai praktik pemrograman yang bagus, karena menangkap semua pengecualian namun tidak membuat programmer mengenali akar permasalahan yang mungkin terjadi.

Klausul Kecuali dengan Beberapa Pengecualian

Anda juga dapat menggunakan pernyataan kecuali yang sama untuk menangani beberapa pengecualian sebagai berikut -

```
try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

```
#!/usr/bin/python

try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
finally:
   print "Error: can \'t find file or read data"
```

```
#!/usr/bin/python

try:
   fh = open("testfile", "w")
   try:
      fh.write("This is my test file for exception handling!!")
   finally:
      print "Going to close the file"
      fh.close()
except IOError:
   print "Error: can \'t find file or read data"
```

Bila dikecualikan dilempar di blok coba, eksekusi langsung lolos ke blok akhirnya. Setelah semua pernyataan di blok akhirnya dieksekusi, pengecualian dinaikkan lagi dan ditangani dalam pernyataan kecuali jika ada di lapisan yang lebih tinggi dari pernyataan try-except.

Argumen Eksepsi

Pengecualian dapat memiliki argumen, yang merupakan nilai yang memberi informasi tambahan tentang masalah tersebut. Isi argumen bervariasi menurut pengecualian. Anda menangkap argumen pengecualian dengan menyediakan sebuah variabel dalam klausa kecuali sebagai berikut -

```
try:
   You do your operations here;
   ......................
except ExceptionType, Argument:
   You can print value of Argument here...
```

Jika Anda menulis kode untuk menangani satu pengecualian, Anda dapat memiliki variabel mengikuti nama pengecualian dalam pernyataan kecuali. Jika Anda menjebak beberapa pengecualian, Anda dapat memiliki variabel mengikuti tuple pengecualian.

Variabel ini menerima nilai pengecualian yang sebagian besar mengandung penyebab pengecualian. Variabel tersebut dapat menerima satu nilai atau beberapa nilai dalam bentuk tuple. Tuple ini biasanya berisi error string, error number, dan error location.

Pengecualian yang Ditentukan Pengguna

Python juga memungkinkan Anda membuat pengecualian sendiri dengan menurunkan kelas dari pengecualian standar built-in.

Berikut adalah contoh yang berkaitan dengan RuntimeError. Di sini, sebuah kelas dibuat yang dikelompokkan dari RuntimeError. Ini berguna saat Anda perlu menampilkan informasi yang lebih spesifik saat pengecualian tertangkap.

Di blok percobaan, pengecualian yang ditentukan pengguna dinaikkan dan ditangkap di blok kecuali. Variabel e digunakan untuk membuat sebuah instance dari class Networkerror.

```
1 (x,y) = (5,0)
  2 try:
  3  z = x/y
  4 except ZeroDivisionError:
  5  print "divide by zero"
```

Jika Anda ingin memeriksa pengecualian dari kode, Anda bisa memiliki:

```
  1 (x,y) = (5,0)
  2 try:
  3  z = x/y
  4 except ZeroDivisionError as e:
  5  z = e # representation: "<exceptions.ZeroDivisionError instance at 0x817426c>"
  6 print z # output: "integer division or modulo by zero"
```

General Error Catching

Terkadang, Anda ingin menangkap semua kesalahan yang mungkin dihasilkan, tapi biasanya Anda tidak melakukannya. Dalam kebanyakan kasus, Anda ingin menjadi sespesifik mungkin (CatchWhatYouCanHandle). Pada contoh pertama di atas, jika Anda menggunakan klausul pengecualian catch-all dan pengguna menekan Ctrl-C, menghasilkan KeyboardInterrupt, Anda tidak ingin program mencetak "bagi dengan nol".

Namun, ada beberapa situasi di mana yang terbaik untuk menangkap semua kesalahan.

Misalnya, Anda menulis modul ekstensi ke layanan web. Anda ingin informasi kesalahan untuk output output halaman web, dan server untuk terus berjalan, jika mungkin. Tapi Anda tidak tahu kesalahan apa yang mungkin Anda masukkan ke dalam kode Anda.

Dalam situasi seperti ini, Anda mungkin ingin mengode sesuatu seperti ini:

```
1 import sys
  2 try:
  3  untrusted.execute()
  4 except: # catch *all* exceptions
  5  e = sys.exc _info()[0]
  6  write _to _page( "<p>Error: %s</p>" % e )
```

Menemukan Nama Pengecualian Spesifik

Pengecualian standar yang dapat diajukan dijelaskan secara rinci pada:

http://docs.python.org/library/exceptions.html

Lihatlah dokumentasi kelas untuk mengetahui pengecualian apa yang bisa diberikan oleh kelas tertentu.

Lihat juga:

Di wiki ini: WritingExceptionClasses, TracebackModule.

Untuk gagasan umum (non-Python specific) tentang pengecualian, berkonsultasilah dengan ExceptionPatterns.

Untuk menulis tentang ...

Berikan contoh IOError, dan interpretasikan kode IOError.
Berikan contoh beberapa pengecualian. Penanganan beberapa kecuali dalam satu baris.

Pertanyaan

Penanganan Kesalahan Umum

Di bagian "penanganan kesalahan umum" di atas, tertulis untuk menangkap semua pengecualian, Anda menggunakan kode berikut:

import sys
2 try:

3 untrusted.execute()

4 except: # catch *all* exceptions

5 e = sys.exc _info()[0]

6 write _to _page(" Error: %s " % e)

1 try:

2 untrusted.execute()

3 except Exception as e:

4 write _to _page(" Error: %s " % str(e))

Seseorang menunjukkan bahwa "kecuali" menangkap lebih dari sekedar "kecuali Pengecualian sebagai e."

Mengapa demikian? Apa bedanya? - LionKimbro

Untuk saat ini (versi <= 2.4) pengecualian tidak harus diwarisi dari Exception. Jadi polos 'kecuali:' menangkap semua pengecualian, tidak hanya sistem. Pengecualian string adalah salah satu contoh pengecualian yang tidak mewarisi dari Exception. - MikeRovner

Saya percaya bahwa pada 2,7, pengecualian masih tidak harus diwariskan dari Exception atau bahkan BaseException. Namun, seperti Python 3, pengecualian harus subclass BaseException. - gajah jim

Mendapatkan Informasi Berguna dari Pengecualian

Jadi, saya punya sesuatu seperti:

1 (a,b,c) = d

dan Python kembali:

1 ValueError: unpack list of wrong size

... dan begitulah, Anda tentu bertanya-tanya, "Nah, apa yang ada di d?"

Anda tahu - Anda bisa mencetak di sana, dan itu berhasil. Tapi adakah cara yang lebih baik dan lebih menarik untuk mendapatkan informasi yang diketahui orang?

Anda bisa melakukan sesuatu seperti:

```
1 try:
  2  a, b, c = d
  3 except Exception as e:
  4  e.args += (d,)
  5  raise
```

Atribut .args pengecualian adalah tuple dari semua argumen yang dilewatkan (biasanya argumen satu dan satu-satunya adalah pesan kesalahannya). Dengan cara ini Anda dapat mengubah argumen dan menaikkan kembali, dan informasi tambahan akan ditampilkan. Anda juga bisa membuat pernyataan cetak atau login di blok kecuali.

Perhatikan bahwa tidak semua pengecualian subclass Exception (meski hampir semua dilakukan), jadi ini mungkin tidak menangkap beberapa pengecualian; Selain itu, pengecualian tidak diperlukan untuk memiliki atribut .args (meskipun jika pengecualian subclass Exception dan tidak mengesampingkan _ _init _ _ tanpa memanggil superclass-nya), maka kode yang ditulis mungkin gagal Namun dalam prakteknya hampir tidak pernah (dan Jika ya, Anda harus memperbaiki pengecualian yang tidak sesuai!)

Bukankah lebih baik mencegahnya untuk melakukan remediasi?

>

Joel Spolsky mungkin programmer hebat C ++, dan sarannya untuk desain antarmuka pengguna sangat berharga, tapi Python bukan C ++ atau Java, dan argumennya tentang pengecualian tidak berlaku dengan Python.

Joel berpendapat:

"Mereka tidak terlihat dalam kode sumber Melihat kumpulan kode, termasuk fungsi yang mungkin atau mungkin tidak membuang pengecualian, tidak ada cara untuk melihat pengecualian mana yang mungkin dilempar dan dari mana.Ini berarti bahwa pemeriksaan kode yang hati-hati pun tidak. Saya bisa mengungkapkan potensi bug. "

(Perhatikan bahwa ini juga merupakan argumen di balik pengecualian yang diperiksa oleh Java - sekarang eksplisit bahwa pengecualian dapat dilemparkan - kecuali bahwa Run-

timeException masih dapat dibuang ke mana saja. -jJ)

Saya tidak mengerti argumen ini. Dalam kode sumber acak, tidak ada cara untuk mengetahui apakah akan gagal hanya dengan inspeksi. Jika Anda melihat:

```
x = 1
result = myfunction (x)
```

Anda tidak dapat mengetahui apakah fungsi saya gagal pada saat runtime hanya dengan inspeksi, jadi mengapa harus itu penting apakah gagal menabrak pada saat runtime atau gagal dengan meningkatkan pengecualian?

(Crashing itu buruk Dengan secara eksplisit menyatakan pengecualian, Anda memperingatkan orang-orang bahwa mereka mungkin ingin mengatasinya Jawa melakukannya dengan canggung C tidak memiliki cara yang baik untuk melakukannya sama sekali, karena kesalahan kembali masih di band Untuk pengembalian reguler Di python, pengecualian passthrough tidak ditandai, namun kondisi kesalahan menonjol di tempat mereka diciptakan, dan biasanya tidak meniru hasil yang benar. -jJ)

Argumen Joel yang mengemukakan pengecualian hanyalah sebuah goto yang menyamar sebagian benar. Tapi begitu juga untuk loop, sementara loop, fungsi dan metode! Seperti konstruksi lainnya, pengecualian adalah gotos yang dijinakkan dan dipekerjakan untuk Anda, bukan yang liar dan berbahaya. Anda tidak bisa melompat * di mana saja *, hanya tempat yang sangat terbatas.

Joel juga menulis:

"Mereka membuat terlalu banyak titik keluar yang mungkin untuk sebuah fungsi.Untuk menulis kode yang benar, Anda benar-benar harus memikirkan setiap jalur kode yang mungkin melalui fungsi Anda. Setiap kali Anda memanggil fungsi yang dapat meningkatkan pengecualian dan tidak menangkapnya di Spot, Anda menciptakan peluang untuk kejutan bug yang disebabkan oleh fungsi yang dihentikan tiba-tiba, meninggalkan data dalam keadaan tidak konsisten, atau jalur kode lainnya yang tidak Anda pikirkan. "