TEKNIK
INFORMARTIKA

MODUL PRAKTIKUM

PEMROGRAMAN BERBASIS OBJEK

Pertemuan 4
Error and Exceptions

A. Overview

Error ada 2 jenis:

- Syntax Error
- Exceptions

Error di suatu program adalah suatu masalah pada program yang mana membuat program berhenti dieksekusi atau crash.

Di sisi lain, pengecualian (exception) terjadi ketika beberapa kejadian internal terjadi yang mengubah alur program.

Ketika exception terjadi, program akan berhenti berjalan pada titik tersebut dan alur program akan dialihkan ke blok penanganan exception untuk menangani kesalahan tersebut.

Kejadian internal itu misalnya:

- Kesalahan Logika atau Data Tidak Valid
- Keterbatasan Sumber Daya / OutOfMemoryError

B. Syntax Error

Syntax Error, atau yang juga dikenal sebagai **Parser Error**, terjadi ketika *parser* (bagian yang mengubah kode menjadi serangkaian statement) mendeteksi ada statement yang salah dalam kode Python.

Dalam konteks pemrograman, **statement** merujuk pada perintah atau instruksi yang dieksekusi oleh komputer. Sebuah statement biasanya melakukan suatu tindakan. Seperti perhitungan, pencetakan output, atau kontrol alur program.

Kalian bisa belajar *parser* di https://www.youtube.com/watch?si=dHpO9eMIUGG3__xb&v=eF9qWbuQLuw&feature=youtu.be.

Coba liat contoh dibawah. Kalian akan melihat bahwa *parser* tahu bahwa program kalian kekurangan tanda) pada statement tersebut. Jadi, kalian bisa melihat di mana letak kesalahannya.

C. Exceptions

Walaupun **statement** atau **expression** udah benar secara sintaksis, bisa aja tetep bikin error waktu dieksekusi.

Error yang muncul pas program jalan disebut **exception** dan nggak selalu fatal.

Kebanyakan **exception** nggak ditangani langsung sama program, jadi kalian bakal nemuin error seperti ini

```
print(10 * (1/0))
# ERROR!
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# ZeroDivisionError: division by zero

print(4 + spam**3)
# ERROR!
```

```
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# NameError: name 'spam' is not defined

print('2' + 2)
# ERROR!
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# TypeError: can only concatenate str (not "int") to str
```

Bisa kalian liat, string yang muncul itu disebut **Built-in Exception**. Di bagian error-nya, ada detail soal tipe exception dan kenapa itu bisa terjadi.

D. Beberapa Exception yang Paling Umum

- **TypeError**
  Ini muncul kalau kalian salah gunain tipe data, misalnya nyoba nambahin string sama angka.
- **NameError**
  Error ini muncul kalo kalian nyebut variabel atau nama fungsi yang nggak ada di kode kalian.
- **IndexError**
  Kalo kalian coba akses elemen di luar indeks yang ada di list atau tuple, bakal muncul error ini.
- **KeyError**
  Ini muncul kalo kalian nyari key yang nggak ada di dictionary.
- **ValueError**
  Kalau kalian kasih input yang nggak cocok sama tipe data yang diharapin, misalnya nyoba ubah string ke integer yang nggak valid.
- **AttributeError**
  Kalo kalian coba akses atribut atau method yang nggak ada di objek, bakal muncul error ini.
- **IOError**
  Ini terjadi kalo ada masalah pas kalian baca atau tulis file, misalnya file-nya nggak ada.
- **ZeroDivisionError**
  Error ini muncul kalo kalian coba bagi angka dengan nol.
- **ImportError**
  Ini kejadian kalo kalian gagal nge-import modul yang dicari.

E. Exception Handling

Kalian bisa bikin program yang menangani exception tertentu

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops! That was not a valid number. Try again...")
```

Lihat di program ini. Pertama, bagian **try-clause** (yang ada antara try dan except) dieksekusi.

- Kalau nggak ada error, maka bagian **except** dilewatin dan eksekusi di dalam **try** selesai.
- Kalau terjadi error waktu eksekusi di dalam **try**, sisa dari **except** bakal dilewatin. Kalau jenis error-nya cocok dengan exception yang disebut setelah keyword except, maka bagian **except** dijalankan, dan eksekusi lanjut setelah blok try/except.
- Kalau error-nya nggak cocok sama exception yang ada di **except**, error itu bakal diteruskan ke **try** yang lebih luar; kalau nggak ketemu handler-nya, itu jadi **unhandled exception** dan eksekusi berhenti dengan pesan error.

Kalian juga bisa nangkap beberapa exception dalam satu waktu, misalnya buat case-case yang kalian butuhin.

```
def divide_each(a, b):
    try:
        print(a / b)
    except ZeroDivisionError as e:
        print('catch ZeroDivisionError:', e)
    except TypeError as e:
        print('catch TypeError:', e)

divide_each(1, 0)
# catch ZeroDivisionError: division by zero

divide_each('a', 'b')
# catch TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

F. Raising Exceptions

raise statement memungkinkan programmer untuk memaksa terjadinya exception tertentu. Contohnya:

```
        raise NameError("Hi There")
        # Traceback (most recent call last):
        #   File "<stdin>", line 1, in <module>
# NameError: Hi There
```

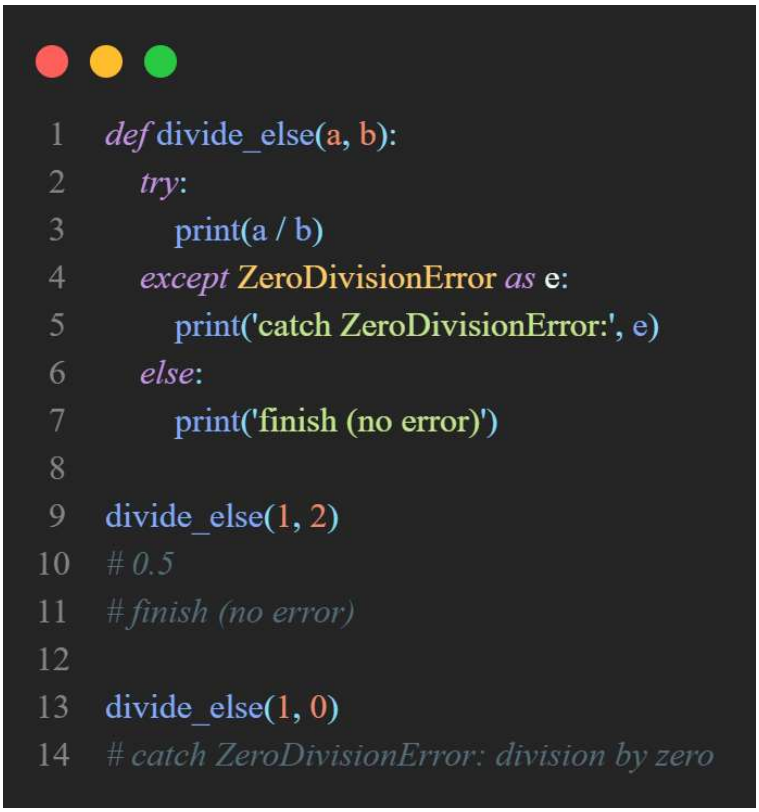
Argumen tunggal untuk **raise** menunjukkan exception yang akan terjadi.

Jadi, **raise** harus menggunakan instance exception atau kelas exception. (Kalian bisa cek di sini saat mau bikin **Raise**).
Kalian juga bisa bikin exception kustom sendiri. Tapi pastikan bahwa exception kustom tersebut diturunkan dari kelas dasar **Exception**.

```
# Definisikan Exception Kustom
class CustomError(Exception):
        pass

# Menghasilkan exception kustom
raise CustomError("Hi mate, this is sort of wrong")
```

G. Execute action if no exception occurs: try ... except ... else ...
Kalian bisa pakai bagian **else** buat menentukan aksi yang dieksekusi jika nggak ada exception yang terjadi. Kalau exception terjadi dan ditangkap oleh **except**, aksi di bagian **else** nggak bakal dieksekusi.
Contohnya:



```
1  def divide_else(a, b):
2      try:
3          print(a / b)
4      except ZeroDivisionError as e:
5          print('catch ZeroDivisionError:', e)
6      else:
7          print('finish (no error)')
8
9  divide_else(1, 2)
10 # 0.5
11 # finish (no error)
12
13 divide_else(1, 0)
14 # catch ZeroDivisionError: division by zero
```

Jadi, bagian **else** hanya dieksekusi kalau nggak ada error yang terjadi di dalam **try**. Kalau ada error, maka **except** yang akan jalan.

H. Clean-up action: try ... except ... finally ...

Di dalam bagian **finally**, kalian bisa tentuin aksi untuk membersihkan yang akan dieksekusi, entah exception terjadi atau nggak.

```python
def divide_finally(a, b):
    try:
        print(a / b)
    except ZeroDivisionError as e:
        print('catch ZeroDivisionError:', e)
    finally:
        print('all finish')

divide_finally(1, 2)
# 0.5
# all finish

divide_finally(1, 0)
# catch ZeroDivisionError: division by zero
# all finish
```

Kalian juga bisa pakai **else** dan **finally** barengan. Kalau nggak ada exception yang terjadi, bagian **else** yang dieksekusi, dan setelah itu baru bagian **finally** yang dijalankan.

```python
def divide_else_finally(a, b):
    try:
        print(a / b)
    except ZeroDivisionError as e:
        print('catch ZeroDivisionError:', e)
    else:
        print('finish (no error)')
    finally:
        print('all finish')

divide_else_finally(1, 2)
# 0.5
# finish (no error)
# all finish

divide_else_finally(1, 0)
# catch ZeroDivisionError: division by zero
# all finish
```

Jadi, **finally** selalu dieksekusi setelah **try/except** apapun yang terjadi, sedangkan **else** hanya jalan kalau nggak ada error.