Error Handling

Tim Dosen

Program Studi Sains Data

Fakultas Informatika

Universitas Telkom

Kriteria Program yang Baik

- Performance/Kinerja yang baik
- Simplicity /Sederhana
- Readability/Mudah dibaca & dipahami isinya
- Modularity
- **Robust**
  - **Error Handling**
  - **Testing**

Error

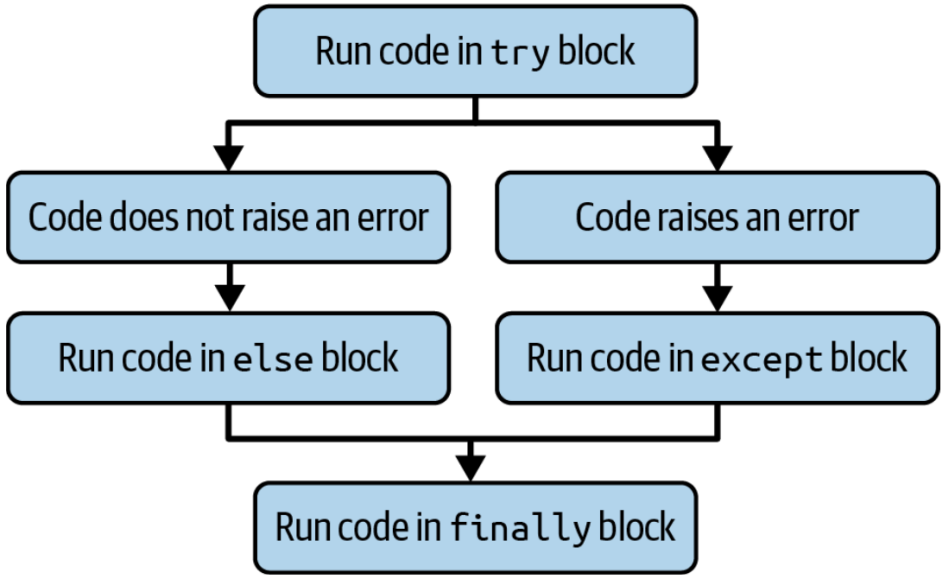
- Sebuah kode program dikatakan error jika ia berhenti tiba-tiba sebelum semua tugas yang harus dikerjakan selesai.
- Dua tipe error pada bahasa Python :
  - Syntax error
  - Exception
- Tips memahami error pada Python : “bacalah mulai dari kalimat terakhir”
- Sumber error :
  - salah ketik syntax (kurang : ,)}, dll)
  - Salah import modul
  - Salah tipe data inputan
  - Salah struktur
  - Salah nama variable
  - dll

Error Handling

“try and except”	Logging	Debugging
Formatting	Linting	Type Checking

Try & Except

```
try:
    # Kode yang mungkin menyebabkan pengecualian
    # Jika tidak ada pengecualian, eksekusi akan dilanjutkan ke blok else
except SomeException:
    # Kode yang dijalankan jika terjadi pengecualian
else:
    # Kode yang dijalankan jika tidak ada pengecualian
finally:
    # Kode yang selalu dijalankan, terlepas dari apakah terjadi pengecualian atau tidak
```



Contoh (1)

```
def bagi(a, b):
    try:
        hasil = a / b
    except ZeroDivisionError:
        print("Error: Pembagian dengan nol tidak diperbolehkan.")
    else:
        print(f"Hasil pembagian {a} / {b} adalah {hasil}")
    finally:
        print("Operasi selesai.")

# Menggunakan fungsi
bagi(10, 2)  # Tidak ada pengecualian
bagi(10, 0)  # Terjadi pengecualian
```

Contoh (2)

```python
from scipy.stats import linregress

def fit_trendline(year_timestamps, data):
    result = linregress(year_timestamps, data)
    slope = round(result.slope, 3)
    r_squared = round(result.rvalue**2, 3)
    return slope, r_squared
```

```python
def fit_trendline(year_timestamps, data):
    try:
        result = linregress(year_timestamps, data)
    except TypeError:
        print(f"Both lists must contain only float or integers,
        got {data.dtype} and {year_timestamps.dtype}")
    else:
        slope = round(result.slope, 3)
        r_squared = round(result.rvalue**2, 3)
        return slope, r_squared
```

```python
timestamps = ["2000", "2001", "2002"]
data = [18.36, 18.36, 17.91]

fit_trendline(timestamps, data)
```

Raising error

- Rutin yang sengaja ”memicu pengecualian” dalam program
- Syntax yang digunakan “raise”

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[27], line 1
----> 1 fit_trendline([], [18.36, 18.36, 17.91])

Cell In[26], line 3, in fit_trendline(year_timestamps, data)
      1 def fit_trendline(year_timestamps, data):
      2     if not year_timestamps or data:
----> 3         raise ValueError('Timestamps and data cannot be empty lists')
      4     result = linregress(year_timestamps, data)
      5     slope = round(result.slope, 3)
```

```
fit_trendline([], [18.36, 18.36, 17.91])
```

Penggunaan “raise”

- **Validasi Input**: untuk memvalidasi input yang diberikan ke fungsi atau metode. Jika input tidak memenuhi kriteria tertentu, Anda dapat menimbulkan pengecualian untuk memberi tahu pengguna bahwa input tersebut tidak valid.
- **Menangani Situasi Tak Terduga**: Dalam beberapa kasus, Anda mungkin ingin menimbulkan pengecualian ketika kondisi tertentu tidak terpenuhi, seperti ketika tidak dapat menemukan file yang diperlukan atau ketika suatu operasi tidak dapat diselesaikan.
- **Membuat Pengecualian Kustom**: Anda dapat membuat pengecualian kustom dengan mendefinisikan kelas pengecualian baru dan kemudian menggunakan raise untuk menimbulkan pengecualian tersebut.

Latihan

Modifikasi program pembagian pada contoh 1 dengan menggunakan fungsi “raise” sehingga jika user langsung dapat peringatan jika memasukkan angka 0 sebagai pembagi.

Buatlah fungsi yang meminta pengguna untuk memasukkan angka dan mengembalikan angka tersebut. Gunakan try dan except untuk menangani pengecualian jika pengguna memasukkan nilai yang bukan angka.

Buatlah fungsi untuk membaca file dan mengembalikan isi file tersebut.
Gunakan try dan except untuk menangani pengecualian jika file tidak ditemukan.

Logging

- Logging adalah metoda untuk merekam apa yang sudah dilakuan oleh program pada file terpisah selama program berjalan.
- File log akan membantu untuk memantau apakah program berjalan sesuai dengan tujuan, memudahkan untuk memahami jalannya program (readable) dan memudahkan untuk melihat kesalahan berdasarkan catatan jika terjadi kesalahan. → debugging
- Python memiliki modul ”logging” untuk menampilkan pesan yang anda inginkan terkait jalannya program , misalnya :
  - a long-running task has started or finished
  - Error messages so that you know what has gone wrong in a production system
  - Which functions called some other functions
  - The inputs and outputs of a function
  - The file path where some data has been saved
  - dll

Konfigurasi Logging (1)

- Digunakan untuk menentukan level ketajaman dan kelengkapan catatan yang disimpan selama program berjalan.

Level	When it's used
DEBUG	Detailed information, typically of interest only when diagnosing problems.
INFO	Confirmation that things are working as expected.
WARNING	An indication that something unexpected happened, or indicative of some problem in the near future (for example, "disk space low"). The software is still working as expected.
ERROR	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	A serious error, indicating that the program itself may be unable to continue running.

Konfigurasi Logging (2)

Untuk melakukan logging ada 3 Langkah yang dilakukan :

1. import modul : `import logging`
2. Konfigurasi log , contoh :

```
logging.basicConfig(level=logging.DEBUG)
ogging.basicConfig(filename="filelog.log",
level=logging.DEBUG)
ogging.basicConfig(filename="filelog.log",
filemode='w',level=logging.DEBUG)
```

3. Menggunakannya dalam kode program

Contoh (1)

```
import logging
from scipy.stats import linregress
def fit_trendline(year_timesteamps,data):
        logging.info("Running fit_trendline function")
        result = linregress(year_timestamps,data)
        slope = round(result.slope, 3)
        r_squared = round(result.rvalue**2, 3)
        logging.info(f"Completed analysis. Slope of the trendline is {slope}.")
        return slope, r_squared
#Jalankan perintah
Fit_trendline(timestamps, data)

#Output
```

INFO: root: Running fit_trendline function

INFO: root: Completed anlysis.Slote of the trendline is 0.836

Contoh (2)

```
import logging
from scipy.stats import linregress
def fit_trendline(year_timesteamps,data):
            logging.basicConfig(filename="chapter_5_logs.log", level=logging.DEBUG,
format='%(asctime)s %(message)s')
            result = linregress(year_timestamps,data)
            slope = round(result.slope, 3)
            r_squared = round(result.rvalue**2, 3)
            logging.info(f”Completed analysis. Slope of the trendline is {slope}.”)
            return slope, r_squared
#Jalankan perintah
Fit_trendline(timestamps, data)
#Output
```

2023-05-26 11:37:43,951 Running fit_trendline function

2023-05-26 11:37:43,953 Completed analysis. Slope of the trendline is 0.836.

Debugging

- Adalah kegiatan untuk membersihkan “perintah-perintah” dalam kode program yang mengakibatkan error atau hasil yang tidak diinginkan.
- Strategi debugging :
  - Jalankan eksperiman dan uji → next topic
  - Ask expert. → mahal
  - Use the tools :
    - Gunakan perintah print untuk fungsi-fungsi yang anda ingin periksa
    - Manfaatkan file log
    - Berikan tanda khusus seperti pemanfaatan breakpoint

Formatting

- Kode program yang ditulis dengan format tertentu membuat kode itu mudah dibaca dan dipahami oleh yang membuat maupuan yang membaca. → Reuseable
- Sudah tersedia tools yang membantu untuk formatting kode menggunakan bahasa Python
- Gaya yang umum digunakan : Google’s style guide & PEP8
- Untuk menggunakannya :
  - `$ pip install isort`
  - `pip install black  / pip install ”black[jupyter]”`

Contoh

Before formatting using isort

```
import time
from sklearn.metrics import mean_absolute_error

import sys, os
import numpy as np
from sklearn.model_selection import train_test_split
import pandas as pd

from sklearn.neural_network import MLPRegressor
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import FunctionTransformer, OneHotEncoder
```

$ isort myscript.py

After

```
import os
import sys
import time

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import (FunctionTransformer, OneHotEncoder,
                                   StandardScaler)
```

Linting

- Linting adalah proses analisis kode program untuk menemukan kesalahan pemrograman, potensi bug, dan masalah gaya penulisan kode. Dengan demikian linting dapat membantu pengembang menulis kode yang lebih bersih dan lebih konsisten.
- Beberapa alat linting yang umum digunakan dalam Python adalah:
  - Pylint
  - Flake8
  - Black
  - Pyflakes
  - mypy (untuk type checking)
- Dengan menginstall tools tsb, umumnya saat anda mengetik kode program IDE akan memberikan ”peringatan” otomatis jika ada kesalahan sintaks, identasi atau tanda-tanda lainnya.

Contoh

```
#Tanpa memperhatikan linting
def luas_segitiga(b,h):
return 0.5*b*h

#Memperhatikan linting
def hitung_luas_segitiga(base,
height):
    """Menghitung luas segitiga."""
    return 0.5 * base * height
```

```
#Tanpa memperhatikan linting
for i in range(1,21):
if i%2==0:
print(i)

#Memperhatikan linting
for number in range(1, 21): if number
% 2 == 0: print(number)
```

Latihan

Tulis kode dengan memperhatikan bebas kesalahan format & linting

- Tulis kode yang mencetak tabel perkalian dari 1 sampai 10.
- Tulis fungsi yang mengecek apakah sebuah angka adalah bilangan prima.

Type Checking

- Digunakan untuk memeriksa tipe inputan, sehingga dapat mencegah pengguna memberikan inputan yang tidak diharapkan.

```
import math

my_int = "100"
print(math.sqrt(my_int))
```

This gives the following error: `TypeError: must be real number, not str.`

- Salah satu cara untuk mengatasi hal ini adalah dengan type annotation

Type Anntation

- Format penulisan : my_variable: type
- Jika ingin mendefinisikannya dalam fungsi dapat mengikuti format sbb :

```
---
def my_function(some_argument: type) -> return_type:
---
```

```
from collections import Counter
from typing import List

def mode_using_counter(list_of_numbers: List[float]) -> float:
    c = Counter(list_of_numbers)
    return c.most_common(1)[0][0]
```

Latihan 1

Diberikan kode sbb :

```
def celsius_to_fahrenheit(c):
return (c * 9/5) + 32
```

Apakah kode tersebut sudah memenuhi format yang baik? Jika ya/tidak berikan penjelasan.

Jika tidak tuliskan kode dengan format yang baik.

Latihan 2

Diberikan kode sbb :

```
def tambah(a, b):
    return a + b
```

Apakah kode tersebut sudah memenuhi format yang baik? Jika ya/tidak berikan penjelasan.

Tuliskan kode dalam format yang baik

Latihan 3

Diberikan kode sbb :

```
def luas_persegi_panjang(panjang, lebar):
    return panjang * lebar
```

Apakah kode tersebut sudah memenuhi format yang baik? Jika ya/tidak berikan penjelasan.

Tuliskan kode dalam format yang baik

Terima Kasih

atas perhatian dan semangat belajarnya ...