Pertemuan-5

Exception Handling

A. Exception Handling

Exception handling adalah teknik pemrograman yang digunakan untuk menangani situasi yang tidak diharapkan atau error yang terjadi pada saat program dijalankan. Dalam Python, kita dapat mengatasi error tersebut dengan menggunakan konsep exception handling.

Berikut adalah langkah-langkah penggunaan exception handling pada Python:

1. Gunakan blok try-except:
   Untuk menangkap sebuah error yang mungkin terjadi di dalam blok try.
   Jika terjadi error, maka program akan keluar dari blok try dan masuk ke blok except.
   Contoh penggunaan blok try-except:

```
try:
    my_list = [1, 2, 3]
    my_list.remove(4)
except ValueError:
    print("Error: Elemen tidak ada di dalam list!")
```

2. Gunakan exception specific:
   Kita juga dapat menggunakan exception specific untuk menangani jenis error tertentu.
   Contoh: ValueError, ZeroDivisionError, dan lainnya.

```
try:
    # Buka file dan baca isinya
    with open('file.txt', 'r') as file:
        data = file.read()

    # Ubah teks menjadi bilangan bulat
    num = int(data)

    # Bagi dengan angka yang diinputkan oleh user
    divisor = int(input("Masukkan angka pembagi: "))
    result = num / divisor

    # Tampilkan hasil bagi
    print("Hasil bagi adalah:", result)

# Tangani beberapa jenis exception
except FileNotFoundError:
    print("File tidak ditemukan!")
except ValueError:
    print("Isi file harus berupa bilangan bulat!")
except ZeroDivisionError:
```

```
        print("Angka pembagi tidak boleh nol!")
    except Exception as e:
        print("Terjadi kesalahan:", str(e))
```

3. Gunakan blok else:
   Blok else akan dieksekusi jika tidak ada error yang terjadi pada blok try.

```
try:
    # Buka file dan baca isinya
    with open('file.txt', 'r') as file:
        data = file.read()

    # Ubah teks menjadi bilangan bulat
    num = int(data)

except FileNotFoundError:
    print("File tidak ditemukan!")
except ValueError:
    print("Isi file harus berupa bilangan bulat!")
else:
    # Jika tidak terjadi kesalahan, tampilkan hasil konversi
    print("Isi file berhasil dikonversi menjadi bilangan bulat:", num)
```

4. Gunakan blok finally:
   Blok finally akan dieksekusi selalu, baik terjadi error atau tidak.
   Blok ini biasanya digunakan untuk membersihkan sumber daya atau menutup koneksi.

```
try:
    file = open("file.txt", "r")
    num = int(file.readline())
except ValueError:
    print("Error: Input tidak valid!")
finally:
    file.close()
```

Berikut ini beberapa contoh penggunaan Exception Handling:

Contoh 1: Mengatasi Type Error

```
try:
    x = "Hello"
    y = x + 5
except TypeError:
    print("Terjadi kesalahan tipe data, pastikan variabel yang digunakan sudah benar!")
```

Contoh 2 : Mengatasi ZeroDivisionError

```
try:
    x = 10
    y = x / 0
except ZeroDivisionError:
    print("Terjadi kesalahan pembagian dengan nol!")
```

Contoh 3 : Mengatasi FileNotFoundError

```
try:
    with open("file_yang_tidak_ada.txt") as file:
        data = file.read()
except FileNotFoundError:
    print("File yang diminta tidak ditemukan!")
```

Contoh 4 : Mengatasi KeyError

```
dictionary = {"satu": 1, "dua": 2, "tiga": 3}

try:
    value = dictionary["empat"]
except KeyError:
    print("Key yang diminta tidak ditemukan pada dictionary!")
```

Contoh 5 : Mengatasi IndexError

```
list_angka = [1, 2, 3]

try:
    value = list_angka[4]
except IndexError:
    print("Index yang diminta melebihi jumlah elemen dalam list!")
```

Contoh 6 : Mengatasi AttributeError

```
class Manusia:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

manusia = Manusia("Andi", 20)

try:
    print(manusia.alamat)
```

```
except AttributeError:
    print("Objek tidak memiliki atribut yang diminta!")
```

Contoh 7 : Mengatasi ValueError

```
try:
    angka = int("bukan_angka")
except ValueError:
    print("Terjadi kesalahan konversi nilai ke dalam tipe data yang
diinginkan!")
```

Contoh 8 : Mengatasi NameError

```
try:
    print(nama)
except NameError:
    print("Variabel yang diminta belum didefinisikan!")
```

Contoh 9 : Mengatasi KeyboardInterrupt

```
try:
    while True:
        pass
except KeyboardInterrupt:
    print("Program dihentikan oleh pengguna!")
```

Contoh 10 : Mengatasi MemoryError

```
try:
    data = " " * (10**10)
except MemoryError:
    print("Memori tidak cukup untuk menampung data yang diminta!")
```

Python memiliki banyak exception built-in yang tersedia untuk menangani jenis kesalahan yang berbeda. Berikut adalah daftar semua exception built-in di Python (dalam urutan abjad):

1. AssertionError: Terjadi ketika sebuah pernyataan assert gagal.
2. AttributeError: Terjadi ketika terjadi kesalahan dalam referensi atau pengisian atribut.
3. EOFError: Terjadi ketika fungsi input() atau raw_input() mencapai kondisi akhir berkas tanpa membaca data apapun.
4. FloatingPointError: Terjadi ketika terjadi kesalahan dalam operasi bilangan pecahan.

5. GeneratorExit: Terjadi ketika sebuah generator atau coroutine ditutup.
6. ImportError: Terjadi ketika pernyataan import gagal untuk menemukan definisi modul.
7. IndexError: Terjadi ketika sebuah indeks tidak ditemukan dalam urutan.
8. KeyboardInterrupt: Terjadi ketika pengguna menekan tombol interrupt (biasanya Control-C atau Delete).
9. KeyError: Terjadi ketika sebuah kunci tidak ditemukan dalam kamus.
10. MemoryError: Terjadi ketika sebuah operasi kehabisan memori.
11. NameError: Terjadi ketika sebuah nama tidak ditemukan dalam namespace atau modul saat ini.
12. NotImplementedError: Terjadi oleh metode atau fungsi abstrak yang perlu diimplementasikan dalam subclass atau derived class.
13. OSError: Terjadi ketika fungsi sistem mengembalikan kesalahan.
14. OverflowError: Terjadi ketika perhitungan melebihi batas maksimum untuk tipe numerik.
15. RecursionError: Terjadi ketika kedalaman rekursi maksimum terlampaui.
16. ReferenceError: Terjadi ketika sebuah referensi lemah merujuk pada objek yang telah dikumpulkan oleh garbage collector.
17. RuntimeError: Terjadi ketika terjadi kesalahan yang tidak termasuk dalam kategori apapun.
18. StopAsyncIteration: Terjadi oleh iterator asynchronous untuk menunjukkan bahwa iterator telah selesai.
19. StopIteration: Terjadi oleh fungsi next() untuk menunjukkan bahwa tidak ada lagi item yang akan dikembalikan oleh sebuah iterator.
20. SyntaxError: Terjadi ketika terdapat kesalahan sintaks pada sebuah program Python.
21. SystemError: Terjadi ketika terjadi kesalahan sistem (misalnya ketika interpreter mendeteksi kesalahan internal).
22. SystemExit: Terjadi oleh fungsi sys.exit() untuk menunjukkan bahwa interpreter Python harus keluar.
23. TabError: Terjadi ketika terjadi masalah dengan indentasi pada program Python.
24. TypeError: Terjadi ketika operasi atau fungsi diterapkan pada objek dengan tipe yang tidak tepat.
25. UnboundLocalError: Terjadi ketika referensi dibuat pada variabel lokal dalam sebuah fungsi atau metode, tetapi tidak ada nilai yang telah diberikan padanya.
26. UnicodeError: Terjadi ketika terjadi kesalahan encoding atau decoding yang berkaitan dengan Unicode.
27. UnicodeEncodeError: Terjadi ketika terjadi kesalahan encoding yang berkaitan dengan Unicode.
28. UnicodeDecodeError: Terjadi ketika terjadi kesalahan decoding yang berkaitan dengan Unicode.
29. UnicodeTranslateError: Terjadi ketika terjadi kesalahan terjemahan yang berkaitan dengan Unicode.
30. ValueError: terjadi saat terjadi kesalahan dalam proses konversi tipe data.
31. ZeroDivisionError: Terjadi ketika sebuah bilangan dibagi dengan nol.

Praktikum :

Berikan masing-masing 1 contoh Exception Handling dengan contoh yang berbeda dari contoh yang sudah diberikan.

Jawaban diupload ke github masing-masing di folder: **praktikum5**

Tugas:

Berikan masing-masing 1 contoh untuk Exception Handling yang belum ada contohnya.

Jawaban diupload ke github masing-masing di folder: **tugas5**