MODUL PRAKTIKUM

PROGAMA KOMPUTER

PROGRAM STUDI TEKNIK INDUSTRI
UNIVERSITAS SEBELAS MARET

TIM ASISTEN LABORATORIUM PERANCANGAN DAN OPTIMASI SISTEM INDUSTRI 2020

MODUL 9
PENANGANAN EKSEPSI

A. Tujuan

Berikut merupakan tujuan Praktikum Programa Komputer Modul IX.

1. Dapat menangani kesalahan dalam penulisan bahasa pemrograman Python.
2. Praktikan dapat menangani jenis *error* dengan menggunakan penanganan eksepsi.

B. Eksepsi

Eksepsi atau yang disebut juga dengan *Runtime Errors* merupakan suatu kesalahan (*error*) yang terjadi saat proses eksekusi program sedang berjalan, kesalahan ini akan menyebabkan program berakhir dengan tidak normal. Dalam python, kesalahan-kesalahan tersebut diidentifikasi dengan nama-nama tertentu dan direpresentasikan sebagai objek. Sederhananya, eksepsi adalah suatu objek yang merepresentasikan kesalahan. Berikut merupakan contoh eksepsi :

```
a = 1
b = 'x'
print(a+b)
```

Gambar 1 Contoh eksepsi

Program ini mencoba untuk menjumlahkan variabel a+b namun variabel b memiliki tipe data string, sehingga hal ini akan memicu bangkitnya eksepsi karena akan terjadi *error*.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 3
      1 a = 1
      2 b = 'x'
----> 3 print(a+b)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Gambar 2 Hasil contoh eksepsi

Dalam kasus ini eksepsi yang dibangkitkan adalah **TypeError** dimana operan tidak didukung penjumlahan antara tipe *integer* dan *string*.

C. Penggunaan Eksepsi

Eksepsi (*exception*) merupakan suatu kesalahan (*error*) yang terjadi saat proses eksekusi program sedang berjalan, kesalahan ini akan menyebabkan program berakhir dengan tidak normal. Dalam python, kesalahan-kesalahan tersebut diidentifikasi dengan nama-nama tertentu dan direpresentasikan sebagai objek. Sederhananya, eksepsi adalah

suatu objek yang merepresentasikan kesalahan. Berikut adalah contoh penggunaan eksepsi :

1. Membangkitkan Eksepsi dengan **raise**

Di dalam Python ada banyak tipe eksepsi di dalam python, yang semuanya bisa dibangkitkan baik pada saat penanganan kesalahan (*error*) atau bisa juga kita bangkitkan secara paksa dengan menggunakan perintah **raise**.

Perintah **raise** digunakan untuk membangkitkan atau melempar eksepsi berdasarkan kondisi-kondisi tertentu. Sebagai contoh, anggap kita akan melakukan operasi x = 0. Selanjutnya, kita dapat membangkitkan eksepsi dengan tipe **ValueError** ketika pengguna program mengisi nilai 0 ke dalam variabel x.

```
x = 0

if x == 0:
    raise ValueError("Nilai Error")
```

Gambar 3 Contoh penggunaan *raise*

Pada contoh program di atas akan menghasilkan outputnya ialah

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[15], line 4
      1 x = 0
      3 if x == 0:
----> 4         raise ValueError("Nilai Error")

ValueError: Nilai Error
```

Gambar 4 Hasil contoh penggunaan *raise*

2. Penanganan Eksepsi Menggunakan **Blok try ... except**

```
import sys
try:
    a = int(input("Masukan nilai a : "))
    b = int(input("Masukan nilai b : "))
except ValueError:
    print("Nilai harus bertipe numerik")
    sys.exit()
hasil=a+b
print("Hasil Penjumlahan :",hasil)
```

Gambar 5 Contoh eksepsi menggunakan Blok try - except

Pada program tersebut akan menjumlahkan nilai a dan b yang akan di masukan oleh pengguna. Berhubung karena inputan dari pengguna memungkinkan adanya kesalahan dalam memasukan angka numerik, maka perintah tersebut kita tempatkan di

dalam *blok* **try**, apabila terjadi eksepsi dengan tipe **ValueError** maka kode di dalam blok **except** akan di eksekusi.

Tipe eksepsi **ValueError** adalah eksepsi untuk menangani kesalahan konversi tipe data. ketika pengguna memasukan nilai selain integer maka eksepsi ini akan dibangkitkan. Selain itu, digunakan perintah **exit()** di dalam modul **sys** untuk menghentikan eksekusi program yang sedang berjalan saat terjadi eksepsi.

Apabila tidak terjadi eksepsi maka kode di dalam blok **except** tidak akan di eksekusi oleh program dan langsung melompat ke perintah program selanjutnya. Berikut adalah hasil jika nilai a dan b tidak numerik.

```
Nilai harus bertipe numerik
```

Gambar 6 Hasil contoh eksepsi menggunakan Blok try - except

3. Menambahkan Klausa **else** dalam blok **try ... except**

Pada contoh kedua, program menambahkan klausa **else** dimana kode-kode yang berada pada bagian **else** hanya akan dieksekusi ketika tidak terjadi kesalahan saat mengeksekusi kode di bagian **try**. Hal ini berarti jika terjadi eksepsi kode pada bagian **else** tidak akan di eksekusi.

```
try:
    a = int(input("Masukan nilai a : "))
    b = int(input("Masukan nilai b : "))
except ValueError:
    print("Nilai harus bertipe numerik")
else:
    hasil=a+b
    print("Hasil Penjumlahan :",hasil)
```

Gambar 7 Hasil contoh eksepsi menambahkan klausa *else*

```
Nilai harus bertipe numerik
```

Gambar 8 Hasil contoh eksepsi menambahkan klausa *else*

Perbedaan program sebelumnya dan sekarang selain ada penambahan klausa **else**, pada program kedua yaitu sudah tidak penggunaan fungsi **exit()** dari modul **sys** karena program akan melompat ke bagian **else** saat tidak terjadi eksepsi.

4. Menggunakan Kata Kunci **finally**

Python dapat diberikan **finally**, yang selalu dieksekusi setelah blok **try** dan **except**. Blok **finally** selalu dieksekusi tidak peduli apakah ada eksepsi atau tidak. Blok **finally** adalah opsional. Dan, untuk setiap blok **try**, hanya ada satu blok **finally**.

```
try:
    dibagi = 10
    pembagi = 0

    hasil = dibagi/pembagi

    print(hasil)
except:
    print("Error: Pembagi tidak boleh 0.")

finally:
    print("Ini adalah final blok")
```

Gambar 9 Contoh eksepsi menggunakan kata kunci *finally*

```
Error: Pembagi tidak boleh 0.
Ini adalah final blok
```

Gambar 10 Contoh *output* menggunakan kata kunci *finally*

Pada contoh di atas, kita membagi angka dengan 0 di dalam blok **try**. Di sini, kode ini menghasilkan eksepsi. Eksepsi ditangkap oleh blok **except**. Dan, kemudian blok **finally** dijalankan.

5. Menggunakan Perintah **assert**

Perintah assert merupakan bentuk sederhana untuk memeriksa suatu ekspresi bertipe boolean apakah bernilai benar (*True*) atau salah (*False*). Perintah assert biasanya digunakan untuk melacak kesalahan (*debugging*) di dalam kode program. Berikut ini adalah bentuk umumnya:

```
#Masukan nilai a = 0

a = int(input("Masukan nilai a : "))
assert a > 0, "Nilai harus lebih dari 0"
```

Gambar 11 Contoh eksepsi menggunakan perintah assert

Jika ekspresi bernilai salah, maka eksepsi akan di bangkitkan dengan tipe **AssertionError**.

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[4], line 2
      1 a = int(input("Masukan nilai a : "))
----> 2 assert a > 0, "Nilai harus lebih dari 0"

AssertionError: Nilai harus lebih dari 0
```

Gambar 12 Contoh eksepsi menggunakan perintah assert

D. Daftar Tipe Eksepsi dalam Python

Berikut merupakan tabel dari daftar tipe eksepsi dalam phyton, yaitu :

Tabel 1 Daftar Tipe Eksepsi dalam Python

NO	Nama	Penjelasan
1	ArithmeticError	Kelas dasar untuk semua kesalahan yang terjadi untuk perhitungan numerik.
2	AssertionError	Muncul jika terjadi kegagalan pernyataan Assert.
3	AttributeError	Muncul jika terjadi kegagalan referensi atribut atau penugasan.
4	EnvironmentError	Kelas dasar untuk semua pengecualian yang terjadi di luar lingkungan Python.
5	EOFError	Muncul apabila tidak ada input dari fungsi raw_input () atau input () dan akhir file tercapai.
6	Exception	Kelas dasar untuk semua pengecualian / exception
7	FloatingPointError	Muncul saat perhitungan *floating point* gagal.
8	GeneratorExit	Muncul saat metode close() generator dipanggil
9	ImportError	Muncul saat sebuah pernyataan impor gagal.
10	IndentationError	Muncul saat indentasi tidak ditentukan dengan benar.
11	IndexError	Muncul saat sebuah indeks tidak ditemukan secara berurutan.
12	IOError	Muncul saat operasi *input* / *output* gagal, seperti pernyataan cetak atau fungsi open () saat mencoba membuka file yang tidak ada.
13	KeyboardInterrupt	Muncul saat pengguna menyela eksekusi program, biasanya dengan menekan Ctrl + c.
14	KeyError	Muncul saat kunci yang ditentukan tidak ditemukan dalam kamus.
15	LookupError	Kelas dasar untuk semua kesalahan pencarian.
16	MemoryError	Muncul saat operasi kehabisan memori
17	NameError	Muncul saat pengenal tidak ditemukan di *namespace* lokal atau global.
18	NotImplementedError	Muncul ketika metode abstrak yang perlu diimplementasikan di kelas warisan sebenarnya tidak dilaksanakan.
19	OSError	Dibangkitkan untuk kesalahan terkait sistem operasi.

20	OverflowError	Muncul saat perhitungan melebihi batas maksimum untuk tipe numerik.
21	ReferenceError	Muncul saat weak reference digunakan untuk mengakses referensi sampah program
22	RuntimeError	Muncul saat kesalahan yang dihasilkan tidak termasuk dalam kategori apa pun.
23	StandardError	Kelas dasar untuk semua pengecualian built-in kecuali StopIteration dan SystemExit.
24	StopIteration	Muncul ketika metode (iterator) berikutnya dari iterator tidak mengarah ke objek apa pun.
25	SyntaxError	Muncul saat ada kesalahan dengan sintaks Python.
26	SystemError	Muncul saat penafsir menemukan masalah internal, namun bila kesalahan ini ditemui juru bahasa Python tidak keluar.
27	SystemExit	Muncul oleh fungsi sys.exit ().
28	SystemExit	Muncul saat juru bahasa Python berhenti dengan menggunakan fungsi sys.exit (). Jika tidak ditangani dalam kode, menyebabkan penafsir untuk keluar.
29	TabError	Muncul saat indentasi memiliki jumlah spasi atau tab yang tidak konsisten
30	TypeError	Muncul saat operasi atau fungsi dicoba yang tidak valid untuk tipe data yang ditentukan.
31	UnboundLocalError	Muncul saat mencoba mengakses variabel lokal dalam suatu fungsi atau metode namun tidak ada nilai yang ditugaskan padanya.
32	UnicodeError	Muncul saat terjadi kesalahan berkenaan dengan encoding dan decoding unicode
33	UnicodeEncodeError	Muncul saat terjadi kesalahan pada proses encoding
34	UnicodeDecodeError	Muncul saat terjadi kesalahan pada proses decoding
35	UnicodeTranslateError	Muncul saat terjadi kesalahan berkenaan dengan penerjemahan unicode
36	ValueError	Muncul ketika fungsi bawaan untuk tipe data memiliki jenis argumen yang valid, namun argumen tersebut memiliki nilai yang tidak valid yang ditentukan.
37	ZeroDivisonError	Muncul saat pembagian atau modulo nol dilakukan untuk semua tipe numerik.