

**MODUL MATA KULIAH**

# **BAHASA PEMROGRAMAN DASAR**

**PG168 – 3 SKS**



**FAKULTAS TEKNOLOGI INFORMASI  
UNIVERSITAS BUDI LUHUR**

**JAKARTA  
SEPTEMBER 2021**

**TIM Penyusun**

Agus Umar Hamdani, M.Kom  
Tri Ika Jaya Kusumawati, M.Kom<sup>1</sup>



# MODUL PERKULIAHAN #11

## EXCEPTION

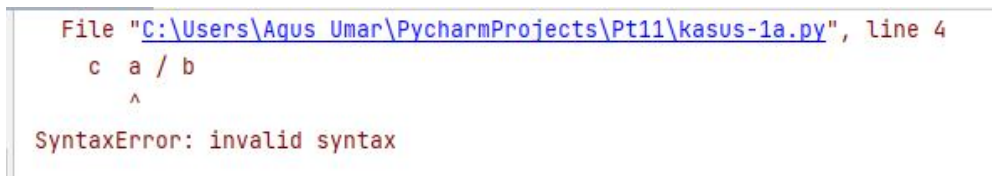
Capaian Pembelajaran	:	<b>Mahasiswa Mampu:</b> <ol style="list-style-type: none"><li>1. Memahami konsep dasar <i>Exception</i> dalam bahasa Python.</li><li>2. Memahami jenis-jenis <i>Exception</i> yang ada pada bahasa Python.</li><li>3. Memahami cara penggunaan <i>Exception</i>.</li></ol>
Sub Pokok Bahasan	:	<ol style="list-style-type: none"><li>1. Konsep Dasar <i>Exception</i></li><li>2. Penanganan <i>Exception</i></li></ol>
Daftar Pustaka	:	<ol style="list-style-type: none"><li>1. Zarman, Wendi dan Wicaksono, Mochamad Fajar. "<i>Implementasi Algoritma dalam bahasa Python</i>". Edisi Pertama. Bandung : Penerbit Informatika. 2020.</li><li>2. Kurniawati, Arik. "<i>Algoritma dan Pemrograman menggunakan Python</i>". Edisi Pertama. Yogyakarta : Depublish. 2016.</li><li>3. Ismah. "<i>Pemrograman Komputer Dasar-dasar Python</i>". Jakarta : Fakultas Ilmu Pendidikan Universitas Muhammadiyah Jakarta. 20110.</li><li>4. Irfani, M. Haviz dan Dafid. "<i>Modul Praktikum Dasar Pemrograman dengan bahasa Python</i>". Palembang : Sekolah Tinggi Manajemen Informatika Global Informatika Multidata Palembang. 2016.</li><li>5. Fikri, Rijalul. "<i>Praktikum Algoritma dan Pemrograman Komputer</i>". Surabaya : Program Studi Teknik Komputer dan Telematika Jurusan Teknik Elektro Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember. 2010.</li><li>6. Wiratmaja, I Gede Harjumawan, et.all. 2021. Program Menghitung Banyak Bata pada Ruangan Menggunakan Bahasa Python. TIERS Information Technology Journal. Vol. 2(1). Undiknas.</li><li>7. Nuraini, Rini. 20110. Desk Check Table Pada Flowchart Operasi Perkalian Matriks. Jurnal Petir. Vol. 10(1). Sekolah Tinggi Teknik – PLN (STT-PLN).</li><li>8. Romzi, Muhammad dan Kurniawan, Budi. 2020. Pembelajaran Pemrograman Python Dengan Pendekatan Logika Algoritma. JTIM : Jurnal Teknik Informatika Mahakarya. Vol. 03(2). Hal. 37-44.</li><li>9. Programiz.com. Python Operators (<a href="https://www.programiz.com/python-programming/operators">https://www.programiz.com/python-programming/operators</a> diakses pada 29 September 2021 pukul 21.32 WIB)</li></ol>

## PRAKTIKUM 11

### EXCEPTION

#### 11.1 Teori Singkat

*Exception* adalah sebuah peristiwa yang terjadi selama eksekusi program yang mengganggu aliran normal instruksi program. Dalam pemrograman, sering kali ditemukan kesalahan (*error*) pada saat mengeksekusi program. Terdapat dua macam kesalahan (*error*) yang biasa terjadi dan perlu penanganan serius, yaitu : ***syntax error*** dan ***Exceptions***. Syntax Error terjadi Ketika kita mengetik kode program yang salah. Dalam beberapa kasus, baris yang keliru diulangi oleh *parser*-nya dengan sebuah panah menunjuk ke lokasi di mana error-nya terdeteksi.



```
File "C:\Users\Agus Umar\PycharmProjects\Pt11\kasus-1a.py", line 4
c a / b
  ^
SyntaxError: invalid syntax
```

Gambar 11.1 Contoh *Error* Pada Kode Program

Sedangkan *Exceptions*, terjadi pada saat mengeksekusi sebuah program dan sesuatu yang tidak diperkirakan terjadi. contohnya, kita meminta pengguna untuk memasukkan sebuah angka untuk melakukan sebuah pembagian. Namun ternyata pengguna memasukkan sebuah string dan program mencoba untuk membagi sebuah angka dengan input yang diberikan, program-nya akan mengeluarkan pesan *TypeError*. Ketika tidak menangani *Exceptions* dengan tepat, maka program akan keluar secara paksa dikarenakan program tidak mengetahui apa yang perlu dilakukan dalam kasus tersebut. Oleh karena itu, setiap kesalahan (*error*) yang terjadi perlu dilakukan penanganan yang serius.

Terdapat 3 (tiga) jenis penanganan *Exception*, yaitu :

a. Blok *Try*

Digunakan untuk menguji sebuah blok kode untuk mengetahui kesalahan (*errors*).

b. Blok *Except*

Digunakan untuk menangani kesalahan (*error*).

c. Blok *Finally*

Digunakan untuk mengeksekusi kode program, terlepas dari hasil blok *Try* dan Blok *Except*.

Beberapa jenis *Exception* standar yang sering digunakan dalam bahasa Python dapat dilihat pada tabel 11.1.

Tabel 11.1 *Standard Exception*

No.	Nama <i>Exception</i>	Penjelasan
1.	<i>EnvironmentError</i>	Kelas dasar untuk semua pengecualian yang terjadi di luar lingkungan Python.
2.	<i>AithmeticError</i>	Kelas dasar untuk semua kesalahan yang terjadi untuk perhitungan Numerik.
3.	<i>AssertionError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan jika ketika kegagalan pernyataan <i>Assert</i> .
4.	<i>AttributeError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan jika terjadi kegagalan referensi atribut atau penugasan.
5.	<i>EOFError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan jika tidak ada input dari fungsi <i>raw_input()</i> atau <i>input()</i> dan akhir file tercapai.
6.	<i>Exception</i>	Kelas dasar untuk semua pengecualian / exception.
7.	<i>FileNotFoundError</i>	Exception ini muncul pada saat berkas atau direktori yang program minta tidak ditemukan atau tidak ada.
8.	<i>FloatingPointError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan pada saat perhitungan Floating Point gagal.
9.	<i>ImportError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan pada saat sebuah pernyataan <i>import</i> gagal.
10.	<i>IOError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan pada saat operasi input / output gagal, seperti pernyataan

		cetak atau fungsi open() pada saat mencoba membuka file yang tidak ada.
11.	<i>IndentationError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan pada saat identasi tidak ditentukan dengan benar.
12.	<i>IndexError</i>	Digunakan pada saat sebuah index tidak ditemukan secara berurutan.
13.	<i>KeyError</i>	Digunakan pada saat kunci yang ditentukan tidak ditemukan dalam kamus.
14.	<i>KeyboardInterrupt</i>	Pengecualian ( <i>Exception</i> ) ini digunakan pada saat pengguna menyela eksekusi program, biasanya dengan menekan CTRL + C.
15.	<i>LookupError</i>	Kelas dasar untuk semua kesalahan pencarian.
16.	<i>NameError</i>	Pengecualian ( <i>Exception</i> ) ini muncul pada saat program tidak dapat menemukan sebuah nama variabel yang bersifat Local atau Global. Nama variabel yang tidak dapat ditemukan dimasukkan ke dalam pesan error.
17.	<i>NotImplementedError</i>	Exception ini muncul pada saat sebuah objek seharusnya mendukung sebuah operasi, namun belum diterapkan. Kita seharusnya tidak menggunakan error ini pada saat fungsi yang diberikan tidak ditujukan untuk mendukung tipe data dari argumen masukan-nya. Pada situasi tersebut, memunculkan TypeError exception jauh lebih tepat.
18.	<i>OSError</i>	Pengecualian ( <i>Exception</i> ) ini dijalankan untuk kesalahan terkait sistem operasi.
19.	<i>OverflowError</i>	Pengecualian ( <i>Exception</i> ) ini digunakan pada saat perhitungan melebihi batas maksimum untuk tipe Numerik.

20.	<i>RuntimeError</i>	Pengecualian ( <i>Exception</i> ) ini terjadi pada saat kesalahan yang dihasilkan tidak termasuk ke dalam kategori apapun.
21.	<i>StandardError</i>	Kelas dasar untuk semua pengecualian Built-In, kecuali <i>StopIteration</i> dan <i>SystemExit</i> .
22.	<i>StopIteration</i>	Pengecualian ( <i>Exception</i> ) ini digunakan ketika metode (iterator) berikutnya dari iterator tidak mengarah ke objek apapun.
23.	<i>SyntaxError</i>	Pengecualian ( <i>Exception</i> ) ini dijalankan pada saat ada kesalahan dengan sintaks Python.
24.	<i>SystemError</i>	Pengecualian ( <i>Exception</i> ) ini dijalankan pada saat Interpreter menemukan masalah internal. Namun bila kesalahan ini ditemui, interpreter Python tidak menampilkan apapun.
25.	<i>SystemExit</i>	Pengecualian ( <i>Exception</i> ) ini digunakan oleh fungsi <code>sys.exit()</code>
26.	<i>TypeError</i>	Pengecualian ini muncul pada saat sebuah fungsi dilewatkan ke sebuah objek dari tipe data yang tidak tepat sebagai sebuah argumen. Detil lengkapnya disediakan melalui pesan error.
27.	<i>UnboundLocalError</i>	Digunakan pada saat mencoba mengakses variabel local dalam suatu fungsi atau method, namun tidak ada nilai yang ditugaskan kepadanya.
28.	<i>ValueError</i>	Pengecualian ini terjadi pada saat sebuah fungsi argumen memiliki tipe data yang benar, namun diberikan nilai yang salah.
29.	<i>ZeroDivisionError</i>	Exception ini muncul pada saat kita menyediakan angka 0 sebagai argumen kedua untuk operasi pembagian atau modulus.

## 11.2 Penanganan Exception Menggunakan Klausula Try

Klausula *Try* dan *Except* digunakan untuk menangani pengecualian (kesalahan yang terdeteksi selama eksekusi program) dalam bahasa Python. Dengan menggunakan *Try* dan *Except*, maka jika kondisi pengecualian (*Exception*) terjadi, maka proses dapat dilanjutkan tanpa penghentian. Kita dapat menggunakan klausula *Else* dan *Finally* untuk mengatur proses akhir.

**Cara 1** : Sintak penulisan untuk penanganan pengecualian (*Exception*) :

```
try:
    #blok program

#blok program yang lain
```

**Cara 2** : Sintak penulisan untuk penanganan pengecualian (*Exception*) :

```
try:
    #blok program

except Exception1:
    #blok program

except Exception2:
    #blok program

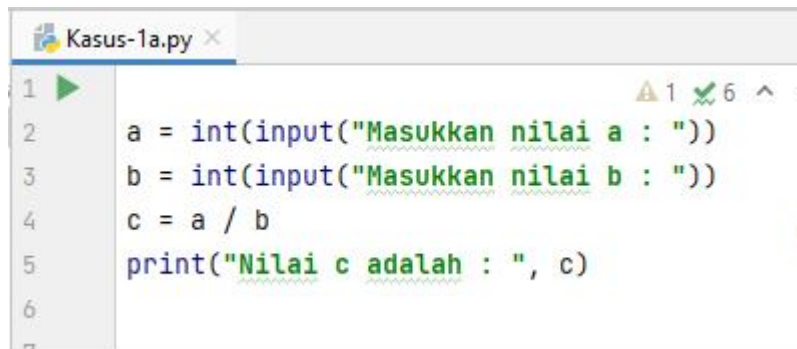
#blok program yang lain
```

Berikut ini adalah Langkah-langkah penanganan *Exception* menggunakan klausula *Try* :

1. Memasukkan kode program yang bisa menimbulkan *Exception* di dalam klausula ***Try***.
2. Menggunakan keyword ***Except*** untuk menangani *Exception* yang terjadi pada kode program diatas.

### Studi Kasus 11.1

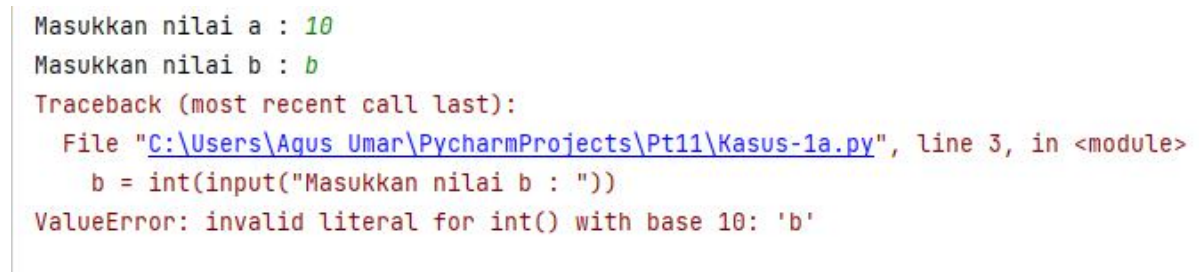
Gambar 11.1 merupakan contoh program tanpa menggunakan *Exception* untuk mencari nilai dari hasil pembagian variabel a dengan variabel b yang disimpan ke variabel c.



```
Kasus-1a.py x
1  ▶ a = int(input("Masukkan nilai a : "))
2    b = int(input("Masukkan nilai b : "))
3    c = a / b
4    print("Nilai c adalah : ", c)
5
6
7
```

Gambar 11.2 Program tanpa *Exception*

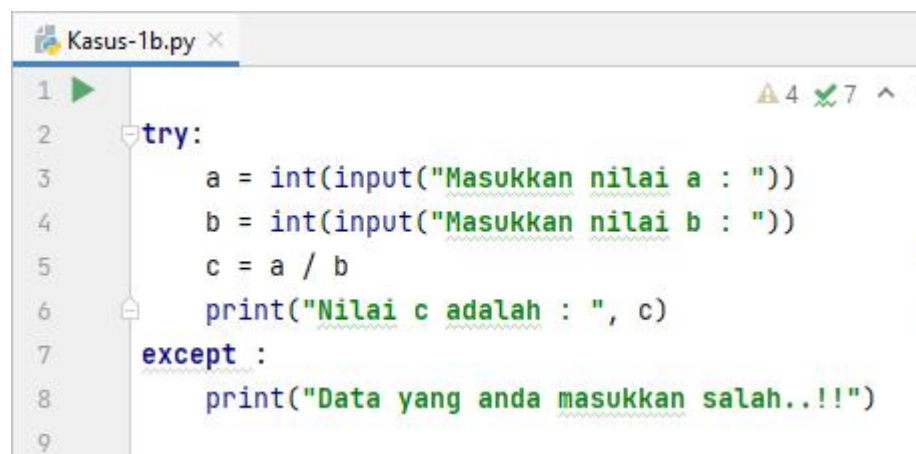
Perhatikan gambar 11.2, kita diminta untuk memasukkan sebuah nilai a yang bertipe Integer dan nilai b bertipe String. Maka program akan mengalami kesalahan (*error*), seperti yang terlihat pada gambar 11.3.



```
Masukkan nilai a : 10
Masukkan nilai b : b
Traceback (most recent call last):
  File "C:\Users\Agus Umar\PycharmProjects\Pt11\Kasus-1a.py", line 3, in <module>
    b = int(input("Masukkan nilai b : "))
ValueError: invalid literal for int() with base 10: 'b'
```

Gambar 11.3 Kesalahan (*Error*) pada Program

Oleh karena itu, program diatas memerlukan penanganan *Exception* dari kondisi yang tidak sesuai. Kita dapat menambahkan klausa Try pada kode program diatas seperti terlihat pada gambar 11.4.



```
Kasus-1b.py x
1  ▶ try:
2    a = int(input("Masukkan nilai a : "))
3    b = int(input("Masukkan nilai b : "))
4    c = a / b
5    print("Nilai c adalah : ", c)
6  except :
7    print("Data yang anda masukan salah..!!")
8
9
```

Gambar 11.4 Menambahkan Klausa *Try* dan *Except*.

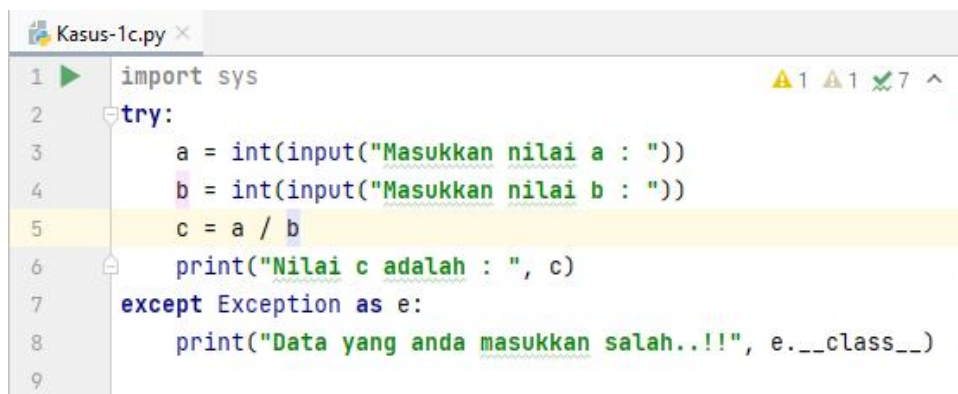


Perhatikan gambar 11.4, kita sudah memberikan kondisi untuk pengecualian (*Exception*), yaitu klausa ***except*** seperti terlihat pada gambar 11.4. Apabila program tersebut dijalankan dan dimasukkan nilai String atau Float, maka contoh keluaran program yang dihasilkan adalah :

```
Masukkan nilai a : 10
Masukkan nilai b : a
Data yang anda masukkan salah..!!
```

Gambar 11.5 Menambahkan Klausa *Try* dan *Except*

Kita dapat mencetak nama pengecualian (*Exception*) menggunakan fungsi ***exc\_info()*** dengan memanggil modul ***sys***. Kita dapat melihat bahwa a menghasilkan kelas *ValueError* dan 0 menyebabkan *ZeroDivisionError*. Kita dapat menambahkan klausa *Try* pada kode program diatas seperti terlihat pada gambar 11.6.



```
Kasus-1c.py x
1 import sys
2 try:
3     a = int(input("Masukkan nilai a : "))
4     b = int(input("Masukkan nilai b : "))
5     c = a / b
6     print("Nilai c adalah : ", c)
7 except Exception as e:
8     print("Data yang anda masukkan salah..!!", e.__class__)
9
```

Gambar 11.6 Menambahkan Klausa *Try* dan *Except*

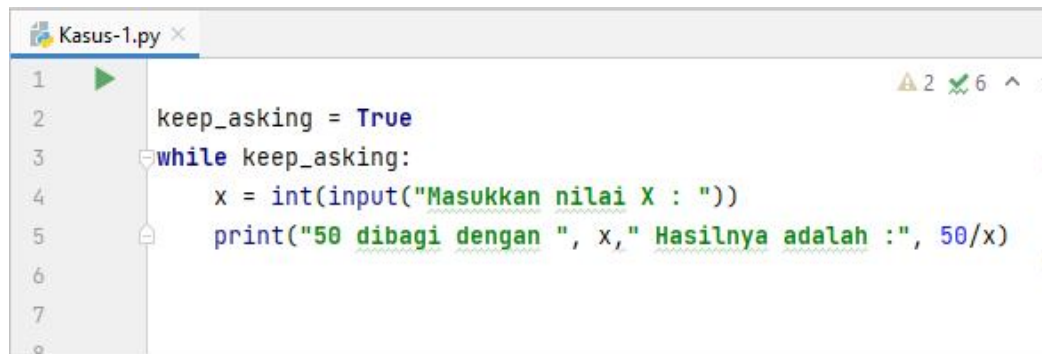
Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.7. Nama kelas *Exception* yang muncul adalah *ValueError*.

```
Masukkan nilai a : 10
Masukkan nilai b : b
Data yang anda masukkan salah..!! <class 'ValueError'>
```

Gambar 11.7 Hasil Keluaran Program

## Studi Kasus 11.2

Gambar 11.8 merupakan contoh program tanpa menggunakan *Exception* pada kondisi perulangan (*loop*). Program tersebut digunakan untuk melakukan pembagian nilai 50 dengan nilai x berulang selama variabel *keep\_asking* bernilai True.

A screenshot of a Python IDE window titled 'Kasus-1.py'. The code is as follows:

```
1 keep_asking = True
2 while keep_asking:
3     x = int(input("Masukkan nilai X : "))
4     print("50 dibagi dengan ", x, " Hasilnya adalah :", 50/x)
```

Gambar 11.8 Program tanpa Exceptions

Perhatikan gambar 11.8, kita diminta untuk memasukkan sebuah nilai x yang bertipe Integer. Kemudian program akan mengulang untuk membagi nilai 50 dengan nilai x selama kondisi *keep\_asking* bernilai True. Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.9.

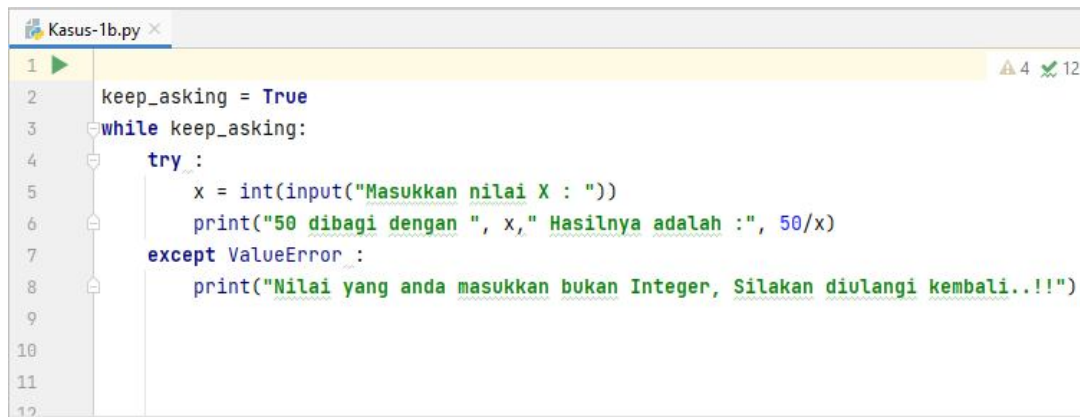
A screenshot of a terminal window showing the execution of the program. The output is as follows:

```
Masukkan nilai X : 5
50 dibagi dengan 5 Hasilnya adalah : 10.0
Masukkan nilai X : a
Traceback (most recent call last):
  File "C:\Users\Agus Umar\PycharmProjects\Pt11\Kasus-1.py", line 3, in <module>
    x = int(input("Masukkan nilai X : "))
ValueError: invalid literal for int() with base 10: 'a'

Process finished with exit code 1
```

Gambar 11.9 *Exception Error*

Perhatikan gambar 11.8, sepanjang kita memasukkan nilai input integral, programnya akan bekerja dengan benar. Namun, kita memasukkan memasukkan sebuah string atau angka desimal, maka kita akan menerima *Exception* berupa *ValueError* seperti terlihat pada gambar 11.10.



Gambar 11.10 Penanganan Exception dengan klausa *Try*

Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.11.

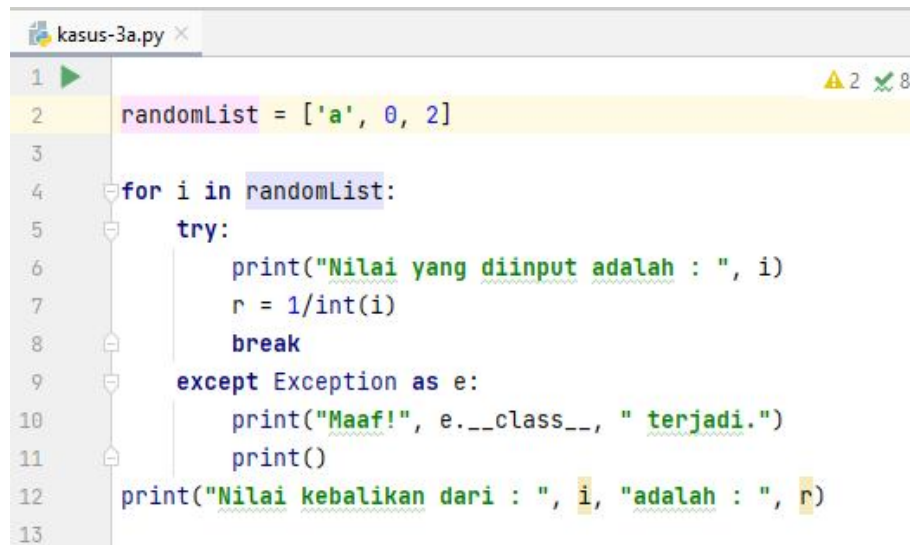
```
Masukkan nilai X : 5
50 dibagi dengan 5 Hasilnya adalah : 10.0
Masukkan nilai X : A
Nilai yang anda masukkan bukan Integer, Silakan diulangi kembali...!!
Masukkan nilai X : 10.5
Nilai yang anda masukkan bukan Integer, Silakan diulangi kembali...!!
Masukkan nilai X :
```

Gambar 11.11 Hasil Eksekusi Program

Pada gambar 11.11 diatas, ketika pengguna memasukkan nilai Integer untuk variabel x, maka program akan dijalankan. Sebaliknya, ketika pengguna tidak memasukkan nilai Integer, maka pesan error akan ditampilkan.

### Studi Kasus 11.3

Gambar 11.12 merupakan contoh program menggunakan Exception pada tipe data List. Program tersebut digunakan untuk melakukan pembagian nilai 50 dengan nilai x berulang selama variabel *keep\_asking* bernilai True.



```
1 2
2  randomList = ['a', 0, 2]
3
4  for i in randomList:
5      try:
6          print("Nilai yang diinput adalah : ", i)
7          r = 1/int(i)
8          break
9      except Exception as e:
10         print("Maaf!", e.__class__, " terjadi.")
11         print()
12     print("Nilai kebalikan dari : ", i, "adalah : ", r)
13
```

Gambar 11.12 Penggunaan *Exception Error* pada tipe List

Perhatikan gambar 11.12, terdapat deklarasi variabel `randomList` yang sudah terisi nilai secara acak. Terdapat variabel `r` yang berisi hasil bagi antara angka 1 dengan nilai dari variabel iterasi `i`. Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.13. Nama kelas *Exception* yang muncul adalah *ValueError*, Ketika dimasukkan nilai `a`. dan ketika variabel `r` berisi hasil pembagian dari 1 dan angka 0 pada `randomList`, maka akan muncul pesan *Exception* dengan nama kelas *ZeroDivisionError*.

```
Nilai yang diinput adalah : a
Maaf! <class 'ValueError'> terjadi.

Nilai yang diinput adalah : 0
Maaf! <class 'ZeroDivisionError'> terjadi.

Nilai yang diinput adalah : 2
Nilai kebalikan dari : 2 adalah : 0.5
```

Gambar 11.13 Hasil Keluaran Program

### 11.3 Penanganan Exception Menggunakan Klausula *Else*

Kita bisa menggunakan klausula *Else* di dalam pernyataan *Try .. Except*. Klausula *Else* digunakan untuk kode program yang dieksekusi, namun klausula *Try* tidak memunculkan *Exception* apapun. Jika kita menggunakan klausula *Else*, maka kita harus memasukkannya setelah semua klausula *Except*, dan sebelum klausula *Finally*.

**Cara 1** : Sintak penulisan untuk penanganan pengecualian (*Exception*) menggunakan ELSE :

```
try:
    #blok program

except Exception1:
    #blok program

Else :
    #kode disini dieksekusi, jika tidak ada blok Except yang dieksekusi
```

**Cara 2** : Sintak penulisan untuk penanganan banyak pengecualian (*Multiple Exception*) :

```
try:
    #blok program

except (<Exception1>, <Exception2>, ...<Exception-n>):
    #blok program

Else :
    #blok program
```

#### Studi Kasus 11.4

Gambar 11.14 merupakan contoh program menggunakan klausula *Else* untuk menangani pengecualian (*Exception*).

```

1  keep_asking = True
2
3  while keep_asking:
4      try:
5          x = int(input("Masukkan nilai X : "))
6      except ValueError:
7          print("Data yang diinput bukan Integer, silakan coba kembali ..")
8      else:
9          print("50 dibagi dengan ", x, " Hasilnya adalah :", 50/x)
10
11

```

Gambar 11.14 Penggunaan Klausula *Else*

Perhatikan gambar 11.14, terdapat deklarasi variabel *x* untuk merekam nilai *x* yang diinput dari keyboard dengan menggunakan klausa *Try*. Terdapat klausa *Except* untuk menangani kondisi tipe data dari nilai yang diinput adalah salah / tidak sesuai dan terdapat klausa *Else* untuk menangani kondisi di luar dari kondisi yang ada di blok *Try* dan blok *Except*. Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.15.

```

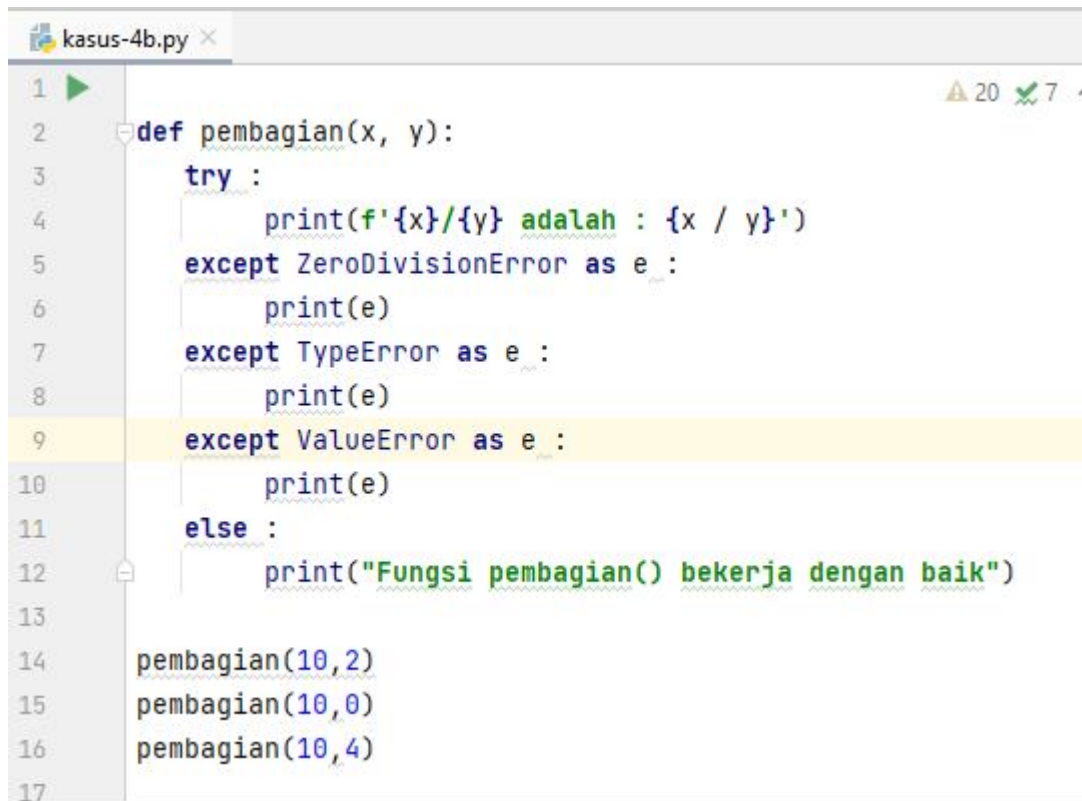
Masukkan nilai X : 2
50 dibagi dengan 2 Hasilnya adalah : 25.0
Masukkan nilai X : b
Data yang diinput bukan Integer, silakan coba kembali ..
Masukkan nilai X :

```

Gambar 11.15 Hasil Keluaran Program

### Studi Kasus 11.5

Gambar 11.16 merupakan contoh program menggunakan klausa *Else* untuk menangani pengecualian (*Exception*) pada sebuah fungsi.



```
1  def pembagian(x, y):
2      try :
3          print(f'{x}/{y} adalah : {x / y}')
4      except ZeroDivisionError as e :
5          print(e)
6      except TypeError as e :
7          print(e)
8      except ValueError as e :
9          print(e)
10     else :
11         print("Fungsi pembagian() bekerja dengan baik")
12
13
14 pembagian(10,2)
15 pembagian(10,0)
16 pembagian(10,4)
17
```

Gambar 11.16 Penggunaan Klausula *Else* pada Fungsi

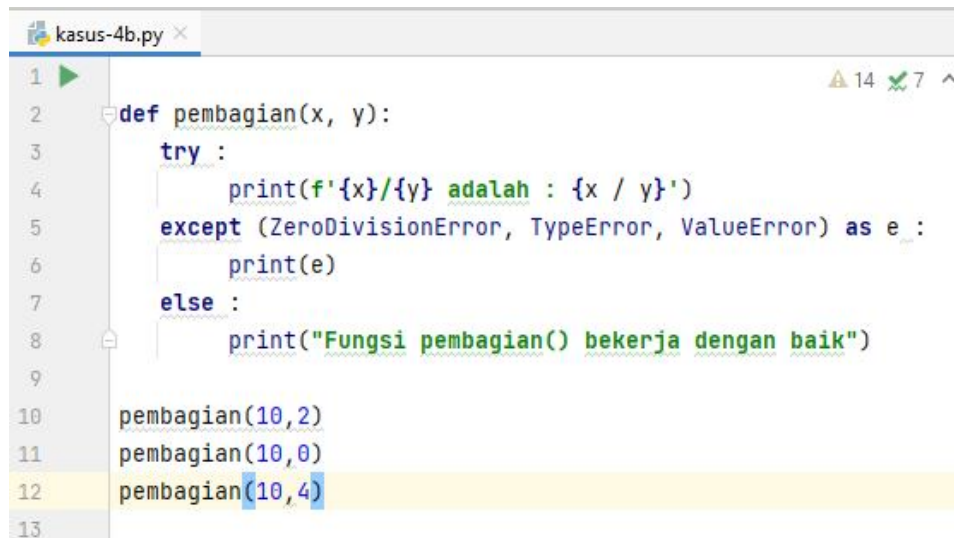
Perhatikan gambar 11.16, terdapat deklarasi fungsi pembagian dengan parameter berupa variabel x dan y. Terdapat klausa Try yang berisi perintah untuk membagi dan mencetak nilai x setelah dibagi dengan nilai y. Terdapat klausa Else yang berisi perintah untuk mencetak isi dari nilai e. Sedangkan kondisi else digunakan untuk mencetak kalimat "Fungsi pembagian() bekerja dengan baik". Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.17.

```
10/2 adalah : 5.0
Fungsi pembagian() bekerja dengan baik
division by zero
10/4 adalah : 2.5
Fungsi pembagian() bekerja dengan baik
```

Gambar 11.17 Hasil Keluaran Program

Penulisan Exception pada gambar 11.16 dapat ditulis juga ke dalam sintak berikut :





```
1 def pembagian(x, y):
2     try :
3         print(f'{x}/{y} adalah : {x / y}')
4     except (ZeroDivisionError, TypeError, ValueError) as e :
5         print(e)
6     else :
7         print("Fungsi pembagian() bekerja dengan baik")
8
9
10 pembagian(10,2)
11 pembagian(10,0)
12 pembagian(10,4)
13
```

Gambar 11.18 Penggunaan Multiple Except Dalam Satu Blok

## 11.4 Penanganan Exception Menggunakan Klausula Finally

Python menyediakan klausa *Finally* yang selalu dieksekusi setelah blok *Try* dan blok *Except*. Blok *Finally* selalu dijalankan setelah penghentian blok *Try* atau setelah blok *Try* berakhir, karena adanya beberapa pengecualian (*Exception*).

**Cara 1** : Sintak penulisan untuk penanganan pengecualian (*Exception*) :

```
try:
    #blok program

finally:
    #blok program Except yang akan dieksekusi
```

**Cara 2** : Sintak penulisan untuk penanganan pengecualian (*Exception*) :

```
try:
    #blok program

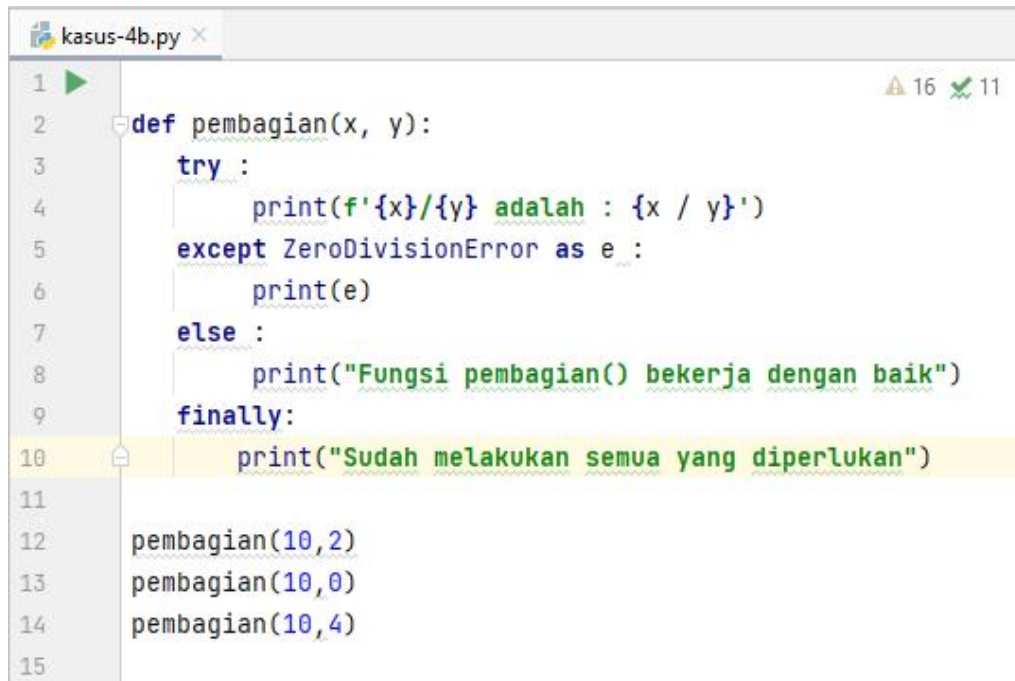
Except Exception1:
    #blok program

finally:
    #blok program Except yang akan dieksekusi
```



### Studi Kasus 11.4

Gambar 11.19 merupakan contoh program menggunakan klausa *Finally* untuk menangani pengecualian (*Exception*).



```
1  def pembagian(x, y):
2      try :
3          print(f'{x}/{y} adalah : {x / y}')
4      except ZeroDivisionError as e :
5          print(e)
6      else :
7          print("Fungsi pembagian() bekerja dengan baik")
8      finally:
9          print("Sudah melakukan semua yang diperlukan")
10
11
12  pembagian(10,2)
13  pembagian(10,0)
14  pembagian(10,4)
15
```

Gambar 11.19 Penggunaan Klausa *Finally*

Perhatikan gambar 11.19, terdapat deklarasi fungsi pembagian dengan parameter berupa variabel x dan y. Terdapat blok *Try* yang berisi perintah untuk membagi dan mencetak nilai x setelah dibagi dengan nilai y. Terdapat blok *else* yang berisi perintah untuk mencetak isi dari nilai e. Terdapat blok *else* digunakan untuk mencetak kalimat "Fungsi pembagian() bekerja dengan baik". Terdapat blok *Finally* yang digunakan untuk mencetak kalimat "Sudah melakukan semua yang diperlukan". Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.20.

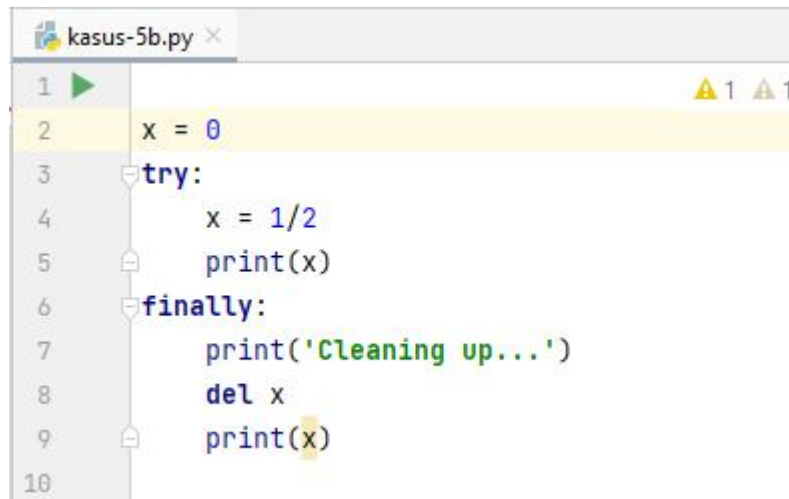
```
10/2 adalah : 5.0
Fungsi pembagian() bekerja dengan baik
Sudah melakukan semua yang diperlukan
division by zero
Sudah melakukan semua yang diperlukan
10/4 adalah : 2.5
Fungsi pembagian() bekerja dengan baik
Sudah melakukan semua yang diperlukan
```

Gambar 11.20 Hasil Keluaran Program

Klausula *Finally* dapat digunakan untuk melakukan pembersihan setelah pengecualian (*Exception*), dimana klausula *Finally* dieksekusi tanpa melihat pengecualian apa yang terjadi pada klausula *Try*, yaitu untuk menutup file atau soket jaringan.

### Studi Kasus 11.5

Gambar 11.21 merupakan contoh program menggunakan klausula *Finally* untuk menangani pengecualian (*Exception*).



```
kasus-5b.py x
1  ▶ 1 1
2  x = 0
3  try:
4      x = 1/2
5      print(x)
6  finally:
7      print('Cleaning up...')
8      del x
9      print(x)
10
```

Gambar 11.21 Penggunaan Klausula *Finally*

Perhatikan gambar 11.21, terdapat deklarasi variabel *x* yang diisi dengan angka 0. Kemudian blok *Try* yang berisi perintah untuk membagi angka 1 dengan angka 2 yang disimpan ke variabel *x* dan blok *Finally* yang berisi perintah untuk mencetak kalimat "Cleaning up ....", perintah menghapus nama variabel dan isi variabel *x* (*del x*) dan perintah untuk mencetak isi dari variabel *x*. Setelah program

dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.22.

```
0.5
Cleaning up...
Traceback (most recent call last):
  File "C:\Users\Agus Umar\PycharmProjects\Pt11\kasus-5b.py", line 9, in <module>
    print(x)
NameError: name 'x' is not defined
```

Gambar 11.22 Hasil Keluaran Program

## 11.5 Penanganan Exception Menggunakan Klausula Raise

Bahasa Python juga menyediakan keyword *raise* untuk digunakan dalam penanganan Exception yang dihasilkan secara eksplisit. Kesalahan Built-in dimunculkan secara implisit. Namun demikian, pengecualian Built-In atau khusus dapat dipaksakan selama eksekusi program.

Kita juga bisa menggunakan klausa *Raise* untuk menangani pengecualian (*Exception*) dengan sintak penulisan :

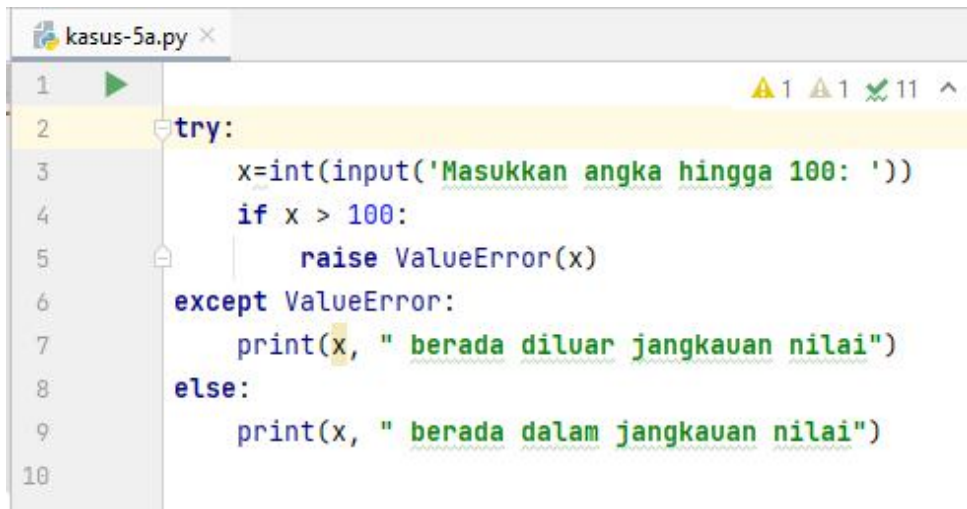
```
raise Exception_class, <value>
```

Hal-hal yang perlu diperhatikan saat menggunakan klausa *raise* :

- Untuk memunculkan *Exception*, kita dapat menggunakan klausa *raise*. Nama *Exception\_Class* disertakan setelahnya.
- Pengecualian dapat diberikan dengan nilai yang bertanda kurung.

### Studi Kasus 11.5

Gambar 11.23 merupakan contoh program menggunakan keyword *raise* untuk menangani pengecualian (*Exception*).



```
kasus-5a.py x
1
2 try:
3     x=int(input('Masukkan angka hingga 100: '))
4     if x > 100:
5         raise ValueError(x)
6 except ValueError:
7     print(x, " berada diluar jangkauan nilai")
8 else:
9     print(x, " berada dalam jangkauan nilai")
10
```

Gambar 11.23 Penggunaan Klausa *raise*

Perhatikan gambar 11.23, terdapat deklarasi blok *Try* yang berisi perintah untuk menginput nilai Integer ke variabel *x* dan kondisi IF, dimana jika variabel *x* bernilai lebih dari 100, maka akan ditangani ke dalam keyword *raise* dengan jenis *exception* adalah *ValueError*. Blok *Except* berisi exception berupa *ValueError* dan mencetak nilai *x* dan keterangan “berada di luar jangkauan nilai”. Blok *else* berisi perintah untuk mencetak nilai *x* dan keterangan “berada dalam jangkauan nilai”. Setelah program dijalankan, maka ditampilkan keluaran program seperti terlihat pada gambar 11.24.

```
Masukkan angka hingga 100: 50
50 berada dalam jangkauan nilai

Masukkan angka hingga 100: 105
105 berada diluar jangkauan nilai
```

Gambar 11.24 Hasil Keluaran Program

## 11.6 Praktikum

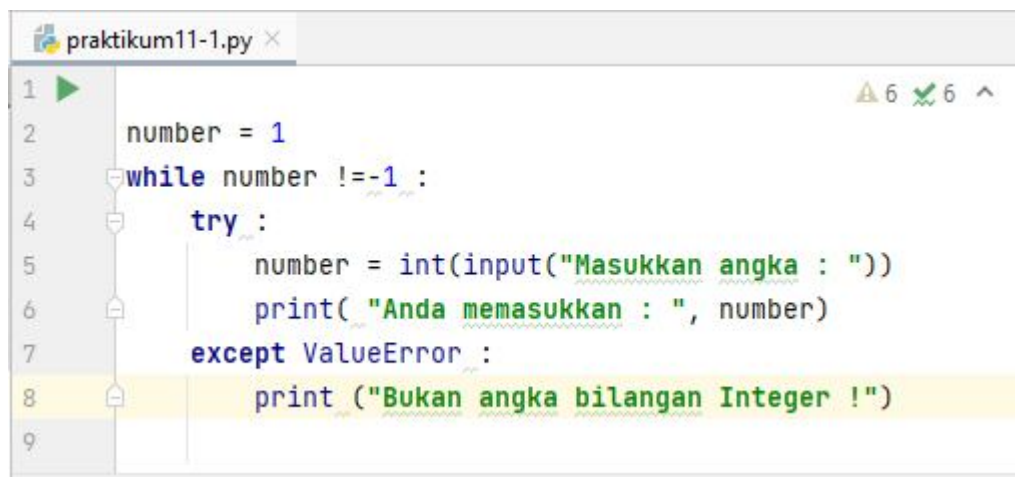
### Langkah-langkah Praktikum

1. Buka Editor Python (IDLE / Pycharm / VSCode).
2. Buatlah file baru dengan membuka menu File > New > Source File atau dengan shortcut Ctrl + N.

3. Tulislah kode program berikut ini :

### Program 11.1 : Praktikum11-1. Py

1. Buatlah modul program yang berisi sebuah exception yang mengharuskan untuk memasukkan nilai minus (-1) dan menampilkan pesan kesalahan (Exceptions), jika yang dimasukkan bukanlah tipe data bilangan Integer atau Float. Ulangi perintah hingga variabel number tidak sama dengan angka -1.



```
1  ▶
2      number = 1
3      while number != -1 :
4          try :
5              number = int(input("Masukkan angka : "))
6              print( "Anda memasukkan : ", number)
7          except ValueError :
8              print ("Bukan angka bilangan Integer !")
9
```

Gambar 11.25 Membuat Modul barang.py

- a. Simpan Program ini dengan nama Praktikum11-1.py
- b. Jalankan program praktikum11-1 di atas, kemudian tuliskan apa yang tercetak di layar pada saat diinput dengan nilai a.

- c. Jalankan program praktikum11-1 di atas, kemudian tuliskan apa yang tercetak di layar pada saat diinput dengan nilai 10.55.

- d. Jalankan program praktikum11-1 di atas, kemudian tuliskan apa yang tercetak di layar pada saat diinput dengan nilai 5.

## 11.7 Rangkuman

- Penanganan pengecualian (*Exception*) dibutuhkan pada saat program mengalami masalah pada saat atau sedang berjalan.
- Masalah yang terjadi pada program biasanya terjadi karena kesalahan input, kesalahan penulisan kode program, kesalahan tipe data dan lain-lain.
- Bahasa Python sudah memiliki daftar penanganan pengecualian (*Exception*) yang dapat digunakan pada saat membuat program.

## 11.8 Latihan

- Buatlah program untuk menangani kondisi pengecualian (*Exception*) dan mencetak informasi nilai mahasiswa menggunakan modul, prosedur dan fungsi sehingga menghasilkan keluaran program sebagai berikut :

```
*****
PROGRAM HITUNG NILAI MAHASISWA
*****

*****
Masukan Nilai UTS : 90
Masukan Nilai UAS : 90
Masukan Nilai QUIZ : 90
Masukan Banyak Tugas : 2
Masukan Nilai Tugas ke-1 : 90
Masukan Nilai Tugas ke-2 : 100
Rata-Rata Nilai Tugas : 95.0
Nilai Akhir : 91.0
Nilai indeks : A
Nilai Predikat : BAIK SEKALI
Nah COBA LAGI(y/t) ?
```

Gambar 11.26 Keluaran program

Jika yang diinput adalah bukan bernilai Integer, maka tampilkan pesan informasi "Data yang anda input salah, silakan dicoba lagi".

## 11.9 Tugas Mandiri

Kerjakan soal berikut :

1. Buatlah modul program menggunakan prosedur atau fungsi dalam bahasa Python untuk menginput data karyawan dengan atribut sebagai berikut :

NIK	: xx
Nama Lengkap	: xx
Umur	: 99
Masa Kerja	: 99
Gaji Pokok	: 99,999,999
Bagian	: xx
Jabatan	: xx

Kemudian, buatlah sebuah exception untuk mengecek nilai yang diinput pada variabel umur, masa kerja dan gaji pokok bernilai Integer atau bukan. Jika bukan bernilai Integer, maka tampilkan pesan informasi "Data yang anda input salah, silakan dicoba lagi".





**FAKULTAS TEKNOLOGI INFORMASI**  
**UNIVERSITAS BUDI LUHUR**

Jl. Raya Ciledug, Petukangan Utara, Pesanggrahan

Jakarta Selatan, 12260

Telp: 021-5853753 Fax : 021-5853752

<http://fti.budiluhur.ac.id>