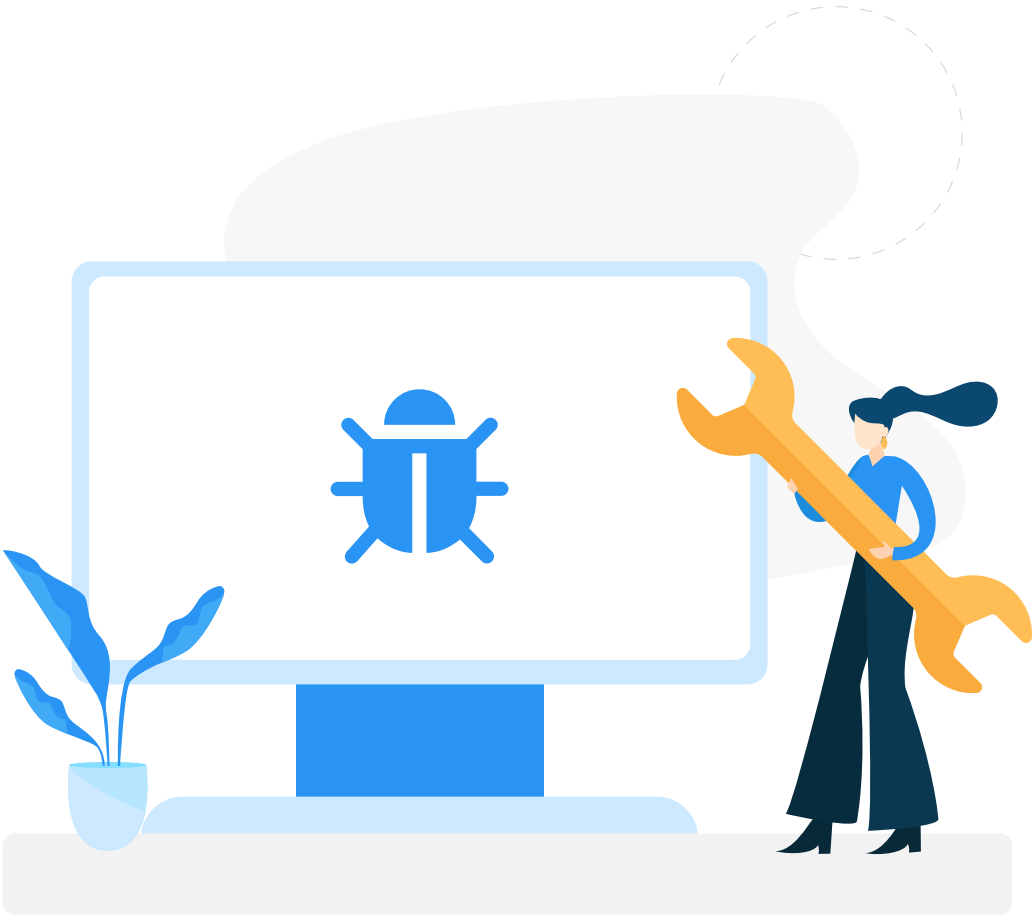
BAB 2

Exception Handling



2.1 Tujuan

1. Mahasiswa dapat memahami tipe error pada kode python
2. Mahasiswa dapat menangani permasalahan error pada kode python

2.2 Pengantar

2.2.1 Error dan Exception

Pada saat kita bekerja dalam pembuatan kode, ada kemungkinan-kemungkinan yang menyebabkan kode kita tersebut mengalami kesalahan sehingga program tidak dapat berjalan. Secara umum kesalahan pada penulisan kode Python terbagi menjadi dua:

1. **Syntax Error**, kesalahan pada penulisan kode. Disebut juga dengan Parsing Errors
2. **Exception**, pengecualian atau disebut dengan Runtime Errors

Penanganan error ini diperlukan pada program karena dapat membantu meningkatkan keandalan dan keamanan program, serta memberikan pengalaman yang lebih baik bagi pengguna. Tanpa adanya penanganan error, sebuah program dapat mengalami crash atau terhenti secara tiba-tiba ketika terjadi kesalahan atau kegagalan dalam proses program. Hal ini dapat mengakibatkan kerugian data atau informasi yang telah diolah oleh program, serta mengakibatkan ketidaknyamanan bagi pengguna.

Dalam pemrograman, penanganan error juga diperlukan untuk menguji kode program dan memastikan bahwa program yang dibuat dapat berjalan dengan baik dalam berbagai kondisi yang mungkin terjadi. Dengan melakukan penanganan error secara baik dan efektif, maka kualitas program yang dibuat dapat ditingkatkan dan meminimalkan risiko kesalahan atau kegagalan dalam program.

2.2.2 Syntax Error

Kesalahan sintaks atau kesalahan pada penulisan kode merupakan hal yang sangat wajar Ketika belajar suatu Bahasa pemrograman. Kesalahan ini bisa terjadi karena kesalahan indentasi, kesalahan penulisan atau kekurangan karakter pada suatu fungsi

```
1. if 1 > 2 :
2. print("1 lebih besar dari 2")
```

kode diatas akan memunculkan pesan error berikut ini

```
File "main.py", line 2
    print("1 lebih besar dari 2")
    ^
IndentationError: expected an indented block
```

Dikarenakan menggunakan blok if maka baris ke dua harus memiliki indentasi, sehingga penulisan kode yang benar adalah sebagai berikut

```
1. if 1 > 2 :
2.     print("1 lebih besar dari 2")
```

2.2.3 Exception

Error yang kemungkinan bisa terjadi walaupun kita sudah memastikan bahwa tidak ada penulisan kode yang salah adalah kesalahan pada saat program dijalankan. Kesalahan ini disebut **runtime errors** atau **exception**. Tipe kesalahan ini harus kita perhatikan dan tangani agar program tidak macet di tengah jalan.

```
1. total = 0
2. def sum(nominal):
3.   global total
4.   total += float(nominal)
5.
6. price = input("Masukkan Harga: ")
7. sum(price)
8. print("Total Harga : {0}".format(total))
```

Kode diatas akan normal Ketika dijalankan, namun Ketika kita menginputkan string ke dalam program tersebut akan muncul error seperti berikut

```
Masukkan Harga: seribu
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    sum(price)
  File "main.py", line 4, in sum
    total += float(nominal)
ValueError: could not convert string to float: 'seribu'
```

Exception yang terjadi adalah **ValueError** karena program tersebut hanya akan memproses inputan angka bukan string. Sehingga python akan memunculkan exception dan program akan berhenti di tengah jalan. Secara default python akan memberitahukan kepada kita nama tipe exception pada pesan error sehingga kita dapat melakukan penanganan error sesuai exception yang terjadi

2.2.4 Exception Handling

Exception handling atau penangan error pada python dapat menggunakan sintaks try dan except sesuai dengan format berikut ini

```
1. try:
2.     # baris kode yang ingin dijalankan
3. except:
4.     # baris kode yang ingin dijalankan
5.     # jika TERJADI Exception
6. else:
7.     # baris kode yang akan dijalankan
8.     # jika TIDAK TERJADI Exception
9. finally:
10.    # baris kode yang akan selalu dijalankan
11.    # di akhir baik ketika TERJADI atau
12.    # TIDAK TERJADI Exception
13.    # baris ini akan selalu di eksekusi
```

Sesuai dengan format diatas maka kita dapat menambahkan try dan except kedalam kode sehingga kode kita menjadi seperti berikut

```
1. total = 0
2. def sum(nominal):
3.   global total
4.   try:
5.     total += float(nominal)
6.   except:
7.     print("Harga harus berupa angka")
8.
9. price = input("Masukkan Harga: ")
10.sum(price)
11.print("Total Harga : {0}".format(total))
```

Perhatikan baris ke 4 dan 6, kita menambahkan sintaks try dan except untuk membungkus baris ke 5. Baris ke 5 inilah program utama kita yaitu untuk menghitung inputan user. Sehingga Ketika terjadi error pada baris ke 5 ini maka program akan memunculkan pesan "`Harga harus berupa angka`".

Syntax except ini akan menghandle semua exception yang terjadi. Kekurangannya adalah semua error akan dianggap sama dan pesan error yang muncul juga akan seragam. Akan lebih baik jika pesan error yang muncul menyesuaikan tipe exception yang terjadi.

Berikut contoh kode untuk menghitung pembagian sederhana dengan tipe exception **ZeroDivisionError** dimana pesan error akan muncul jika user menginputkan pembilang dengan angka 0.

```
1. def division(a, b):
2.   try:
3.     print("{0} / {1} = {2}".format(a, b, a / b))
4.   except ZeroDivisionError:
5.     print("Tidak bisa membagi angka dengan 0")
6.
7. division(10,"2")
```

2.2.5 Tipe Exception Pada Python

Berikut adalah beberapa jenis exception handling pada Python:

1. **Exception** : Exception paling umum yang digunakan untuk menangani kesalahan yang tidak spesifik. Jika kita tidak yakin dengan jenis exception yang akan terjadi, maka kita bisa menggunakan Exception sebagai penanganan kesalahan yang umum.
2. **ValueError** : Exception yang digunakan ketika terjadi kesalahan pada tipe data yang dimasukkan pada program. Misalnya, ketika kita mencoba mengkonversi string menjadi bilangan bulat (integer), tetapi string tersebut tidak valid sebagai bilangan bulat.

3. **TypeError** : Exception yang digunakan ketika kita mencoba menggunakan tipe data yang tidak sesuai untuk melakukan operasi tertentu. Misalnya, ketika kita mencoba menggabungkan (concatenate) string dan integer.
4. **IndexError** : Exception yang digunakan ketika kita mencoba mengakses indeks yang tidak ada pada suatu list atau array.
5. **KeyError** : Exception yang digunakan ketika kita mencoba mengakses kunci yang tidak ada pada suatu dictionary.
6. **FileNotFoundError** : Exception yang digunakan ketika kita mencoba membuka atau mengakses file yang tidak ditemukan atau tidak ada.
7. **ZeroDivisionError** : Exception yang digunakan ketika kita mencoba melakukan pembagian dengan angka nol.
8. **AssertionError** : exception yang digunakan ketika sebuah pernyataan asumsi (assertion) tidak terpenuhi. Misalnya, ketika kita membuat sebuah asumsi bahwa sebuah variabel akan selalu memiliki nilai positif, tetapi ternyata nilainya negatif.
9. **KeyboardInterrupt** : exception yang digunakan ketika kita ingin menghentikan program secara paksa (biasanya dengan menekan tombol Ctrl + C pada terminal).
10. **ImportError** : exception yang digunakan ketika ada kesalahan saat melakukan import module atau package.

2.3 Kegiatan Praktikum

2.3.1 Kegiatan 1 : Exception Handler pada Python

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
1. try:
2.     # blok kode yang mungkin menimbulkan exception
3.     num1 = int(input("Masukkan angka pertama: "))
4.     num2 = int(input("Masukkan angka kedua: "))
5.     hasil = num1 / num2
6.     print(f"Hasil pembagian: {hasil}")
7. except ValueError:
8.     # exception handler untuk nilai yang tidak valid
9.     print("Input harus berupa angka!")
```

```
except ZeroDivisionError:
    # exception handler untuk pembagian dengan nol
    print("Tidak bisa melakukan pembagian dengan nol!")
except Exception as e:
    # exception handler umum
    print(f"Terjadi kesalahan: {e}")
finally:
    # blok kode yang akan selalu dijalankan
    print("Program selesai.")
```

2. Amati Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.3.2 Kegiatan 2 : Exception Handler dengan konsep OOP

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        try:
            if amount <= 0:
                raise ValueError("Jumlah deposit harus lebih dari nol!")
            self.balance += amount
            print(f"Deposit sebesar {amount} berhasil dilakukan. Saldo sekarang: {self.balance}")
        except ValueError as e:
            print(e)

    def withdraw(self, amount):
        try:
            if amount > self.balance:
                raise ValueError("Saldo tidak mencukupi untuk melakukan penarikan!")
            self.balance -= amount
            print(f"Penarikan sebesar {amount} berhasil dilakukan. Saldo sekarang: {self.balance}")
        except ValueError as e:
            print(e)

# membuat objek dari class BankAccount
my_account = BankAccount(1000)

# melakukan deposit sebesar -500 (akan menghasilkan ValueError)
```

```
my_account.deposit(-500)

# melakukan deposit sebesar 500 (berhasil)
my_account.deposit(500)

# melakukan penarikan sebesar 1500 (akan menghasilkan ValueError)
my_account.withdraw(1500)

# melakukan penarikan sebesar 500 (berhasil)
my_account.withdraw(500)
```

2. Amati Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.3.3 Kegiatan 3 : Raise Exception Handler

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
def divide_numbers(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero!")
    else:
        return a/b

try:
    result = divide_numbers(10, 0)
except ValueError as e:
    print(e)
else:
    print("The result is:", result)
```

2. Amati Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

2.4 Tugas

1. Buatlah sebuah program yang meminta input dari user berupa bilangan bulat (integer), lalu program akan mencetak hasil pangkat dua (kuadrat) dari bilangan tersebut. Namun, jika user memasukkan input yang tidak valid (bukan bilangan bulat), maka program akan menampilkan pesan error **"Input yang dimasukkan tidak valid!"** dan meminta user untuk memasukkan input yang valid.