Chapter – 18 Exception and Error Handling

Contents

18.1 Error Handling 2
18.1.1 Compile-Time Errors 2
18.1.2 Logical Errors 2
18.1.3 Runtime Errors 2
18.2. Exception 3
18.2.1 Exception ဆိုတာ 3
18.2.2 Built-In Exceptions 4
18.3. Error 4
18.3.1 SyntaxError 4
18.3.2 ImportError 4
18.3.3 KeyError 4
18.3.4 TypeError 5
18.3.5 AttributeError 5
18.3.6 IndexError 5
18.3.6 NameError 5
18.3.7 FileNotFoundError 6
18.3.8 Python - Error Types 6
18.4. try…except Block 7
18.4.1 try… except Block အလုပ်လုပ်ပုံ 7
18.5. try…except…else Block 11
18.6. finally Keyword 13
18.6.1. else block မပါသည့် try .. except .. finally 14
18.7. Custom Exceptions 14
18.7.1 Implementing Your Own Exception Class 14
18.7.2 try Statement Clause Forms 15

Chapter – 18 Exception and Error Handling

တစ်ခါတစ်ရံ မိမိစီစဉ်ထားသလို မဖြစ်ဘဲ အဆင်မပြေမှုများနှင့် ကြုံတွေ့ရတတ်သည်။ မျှော်လင့်ထားသည့် ရလဒ်များလွဲသည့်အခါ အမှားများကို ရှာဖွေပြီး နိုင်စွမ်းရှိရမည်။ ပရိုဂရမ် ရေးသားပြီး နေစဉ် မျှော်လင့်မထားသော အခြေအနေများ ပေါ်ပေါက်လာနိုင်သည်။ ကုဒ်သည် ရည်ရွယ်ထားသည်အတိုင်း မလုပ်ဆောင်ဘဲ အမှား (error) တစ်ခု ဖြစ်ပေါ်ပါက program တစ်ခုလုံး ရပ်တန့်သွားလိမ့်မည်။

Python မှာ coding လုပ်နေတဲ့ အချိန် အမှားများ မလွဲမသွေ လုပ်မိနိုင်တယ်။ ဤအခန်းသည် အမှား(error)များ ဖြစ်ပေါ်ရသည့် အကြောင်းအရင်း များကို ပိုမို နားလည်စေရန်နှင့် ဘယ်လို အမှန် ပြင်ရမလဲ ဆိုသည့် နည်းလမ်း များဖြင့် သင့်အား ကူညီပေးရန် ရည်ရွယ်သည်။

18.1 Error Handling

ပရိုဂရမ်ကို run နေသည့်အချိန်(runtime)တွင် Python သည် error တွေ့သည်နှင့် တစ်ပြိုင်နက် exception ထုတ်ပေး(raise လုပ်)သည်။ ထုတ်ပေး exception ကို ဘာမှ မလုပ်ဘဲ လစ်လျူရှုနေနိုင်သည်။ အမှန် ဖြစ်အောင် ပြင်ဆင်ခြင်းဖြင့် တုံ့ပြန်နိုင်သည်။

သင်က ဘာမှ မလုပ်ဘဲ လစ်လျူရှုလိုက်လျှင် Python က သတ်မှတ်ထားသည့်(default) exception-handling တစ်ခုခု လုပ်ပေးလိမ့်မည်။ ပရိုဂရမ်ကို ရပ်ပစ်ပြီး error message ထုတ်ပေးလိမ့်မည်။ ထို default exception-handling ကို အလိုမရှိပါက **try** statement ဖြင့် သင့်စိတ်ကြိုက် တစ်ခုခု လုပ်နိုင်သည်။

အများဆုံးကြုံတွေရလေ့ရှိသည့် error သုံးမျိုးမှာ

(1) Compile-time error the

(2) Logical error နှင့်

(3) Runtime error

18.1.1 Compile-Time Errors

Syntax error သည် compile-time error တစ်ခုဖြစ်သည်။ “ **:** ” ကျန်ခဲ့ခြင်း စာလုံးပေါင်းမှားခြင်း(wrong spelling) စသည်တို့သည် syntactical error ဖြစ်သည်။ Compile လုပ်သည့်အချိန်တွင် ဖြစ်ပေါ်သောကြောင့် compile-time error ဟုခေါ်ဆိုခြင်း ဖြစ်သည်။

18.1.2 Logical Errors

Logical error သည် logic မှားယွင်းမှုကြောင့် ဖြစ်ပေါ်သည့် error ဖြစ်သည်။ ရေးထားသည့်ကုဒ်များ ပုံမှန် အလုပ်လုပ်သည်။ syntax လည်းမှန်ကန်သည်။ compile လုပ်သည့်အချိန်တွင် error တစ်ခုမျှ မဖြစ်ပေါ်ပါ။ output သို့မဟုတ် result လည်း ထုတ်ပေးသည်။ ထွက်လာသည့် ရလဒ် မှားနေခြင်းသည် logical error ဖြစ်သည်။ ထုတ်ပေးသည့် output မှားနေခြင်း(not correct) သည် logical error ဖြစ်သည်။ အခု အခန်းတွင် logical error များနှင့် မသက်ဆိုင်ပါ။

18.1.3 Runtime Errors

ပရိုဂရမ် run နေချိန်တွင် တစ်စုံတစ်ခု မှားယွင်းသွားပါက interpreter က runtime error ထုတ်ပေးသည်။ မည်သည့်နေရာက error ဖြစ်သည်၊ မည်သည့် အလုပ်(function)ကို ဆောင်ရွက်နေစဉ် ဖြစ်ပေါ်သည်ကို runtime

error message တွင် ဖော်ပြပေးသည်။

User က input မှားထည့်မိခြင်းကြောင့် runtime error ဖြစ်နိုင်သည်။ ဥပမာ- user က ကိန်းတစ်ခုကို သုညနှင့် စားရန် input အဖြစ် ထည့်မိခြင်းကြောင့် ZeroDivisionError ပေးလိမ့်မည်။ ZeroDivisionError သည် runtime error တစ်မျိုးဖြစ်သည်။ ZeroDivisionError သည် logical error မဟုတ်သလို compile-time error လည်း မဟုတ်ပေ။

Error (၃)မျိုးတွင် compile-time error သည် ဖြေရှင်းရန် အနည်းငယ် လွယ်ကူသည်ဟု ယူဆနိုင်သည်။ Logical error များကို စမ်းသပ်ပြီး ဖြေရှင်းရန် အဖွဲ့(testing team) ရှိသည်။ Runtime error တွင် ဖြစ်ပေါ်သော တစ်ချို့သော error များသည် user ကြောင့် ဖြစ်သည်။

မည်သည့် error ဖြစ်ပေါ်ပါစေ ပရိုဂရမ် execution လုပ်နေခြင်း ရပ်တန့်သွားလိမ့်မည်။ အလွန်သေးငယ်သည့် runtime error ကြောင့် software တစ်ခုလုံး၊ ပရိုဂရမ် တစ်ခုလုံး ရပ်တန့်သွားခြင်း မဖြစ်ပေါ်အောင် အတတ်နိုင်ဆုံး ကြိုတင် စီမံထားသင့်သည်။ ထို့ကြောင့် မည်သည့် error ဖြစ်ပေါ်ပါစေ execution လုံးဝ ရပ်တန့်သွားခြင်း မဖြစ်အောင် ကာကွယ်နိုင်သည့် နည်းများကို လေ့လာထားသင့်သည်။

18.2. Exception အဓိပ္ပာယ်

Exception အဓိပ္ပာယ်မှာ မင်းလုပ်တာ အားလုံးပြီး သွားပါပြီ နည်းနည်းလွဲနေတာကလွဲလို့။(You're done...except for one small thing)။ ချို့ယွင်းချက်တစ်ခုတစ်လေက ခြွင်းချက်(exception) အဖြစ် ရှိနေသေးတယ် ဟုသည့် အဓိပ္ပာယ်ဖြစ်သည်။

18.2.1 Exception ဆိုတာ

Exception ဆိုသည်မှာ runtime error ဖြစ်သည်။ Python code ကို run နေစဉ် အမှား(error)တစ်ခုခု တွေ့ပါက interpreter က execute လုပ်နေတာကို ရပ်တန့်ပစ်လိုက်ပြီး exception ထုတ်ပေးသည်။ **Exception ဆိုသည်မှာ code ထဲမှာ ဘာလိုအမှားမျိုးတွေ ပါတယ် ဆိုတာကို ဖော်ပြပေးတဲ့ Python object တစ်ခုပါ။** Exception ထဲမှာ ဘယ်နေရာမှာ မှားနေတယ်ဆိုတာကို ဖော်ပြပေးတဲ့ trace back လည်း ဖြစ်ပါတယ်။

- Exception object တစ်ခုမှာ `description` နှင့် trace back တို့ပါရှိသည်။
- အမှား(error)အမျိုးမျိုး ပြဿနာအမျိုးမျိုးကို ဖော်ပြပေးဘို့အတွက် exception အမျိုးအစားများစွာ ရှိသည်။
- Trace back ဆိုတာ runtime error ၏ အသေးစိတ် ဖော်ပြချက်(detail description) ဖြစ်သည်။

Exception များကို အဘယ်ကြောင့်အသုံးပြုရသနည်း

Exception များသည် program ၏ flow ကို ပြောင်းလဲစေနိုင်သည်။ တစ်နည်းအားဖြင့် Python တွင် error များ ဖြစ်ပေါ်လာပါက exception ကို အလိုအလျောက် ထုတ်ပေးပြီး program တစ်ခုလုံး ဆက်မ run တော့ဘဲ ရပ်တန့်သွားနိုင်သည်။

အတိုချုပ်အားဖြင့် မပြောပလောက်တဲ့ အမှားလေးတစ်ခုက exception တစ်ခုထုတ်ပေးပြီး ပရိုဂရမ် တစ်ခုလုံးကို ရပ်တန့်စေသည်။ ပရိုဂရမ် တစ်ခုလုံးကို ရပ်တန့်သွားခြင်းကို ကာကွယ်ရန် မည်သည့်အမှားမျိုး ဖြစ်ပေါ်နေသည်ကို ဖော်ပြရန် exception ကို အသုံးပြုသည်။

18.2.2 Built-In Exceptions

Python ကို install လုပ်ပြီးသည်နှင့်တပြိုင်နက် built-in exception class များ ပါပြီးသား ဖြစ်သည်။ error တော်တော်များများ၏ exception များပါရှိပြီး ဖြစ်ခြင်းကြောင့် သင့်ကိုယ်တိုင် exception class တွေ ရေးစရာ မလိုတော့ပါဘူး။

Base error classes, from which other error classes are defined, and

Concrete error classes, which define the exceptions you are more likely to see from time to time.

18.3. Error

ဤအခန်းတွင် အများဆုံး ကြုံတွေ့နိုင်သည့် များကို လေ့လာကြမည်။ သို့မှသာ ထိုအမှားတွေကို ဘယ်လို ကိုင်တွယ် ဖြေရှင်းရမည်ဆိုသည့်နည်းကို လေ့လာနိုင်လိမ့်မည်။

18.3.1 SyntaxError

SyntaxError ဖြစ်ပေါ်နေသည် သည် လေ့လာခါစသူများ အများကြုံတွေ့ရသည့် Error ဖြစ်သည်။ Python Syntax များကို မှန်ကန်အောင် မရေးခြင်းကြောင့် ဖြစ်ပေါ်သည့် အမှားဖြစ်သည်။ Python interpreter က နားမလည်သောကြောင့် ဖြစ်ပေါ်သည်။

syntax error ဆိုတာ ရေးထားတဲ့ Python code တွေက ချမှတ်ထားတဲ့ စည်းကမ်းတွေကို မလိုက်နာဘဲ ရေးထားခြင်းဖြစ်သည်။ invalid syntax ဆိုပြီး ဖော်ပြပေးပါလိမ့်မည်။ (description ထုတ်ပေးလိမ့်မည်။) Python programming ကို စတင်လေ့လာသူများ ၊ Python code ရေးခါစသူများတွင် အများဆုံး ကြုံတွေ့ ရလေ့ရှိသည်။ Syntax error ဖြစ်နေကြောင်း Python interpreter က ဖော်ပြသည့်အခါ မည်သည့်နေရာတွင် ဖြစ်ပေါ်နေကြောင်းကို တိတိကျကျ(precise location of the syntax error) ဖော်ပြပေးသည့် trace back ပါဝင်သည်။ Syntax error များသည် ပရိုဂရမ်တစ်ခု execution မစတင်မီကတည်း ဖြစ်ပေါ်နေသည်။

18.3.2 ImportError

ImportError သည် module တစ်ခုခု import လုပ်၍ မရသည့်အခါတွင် ဖြစ်ပေါ်သည်။ ဥပမာ- မရှိသည့် (non-existent) module တစ်ခုကို import လုပ်မိသောကြောင့် ဖြစ်နိုင်သလို ၊ module နာမည်ကို စာလုံးပေါင်း မှားရိုက်မိသောကြောင့်လည်း ဖြစ်နိုင်သည်။ ModuleNotFoundError ထုတ်ပေး(raise)လိမ့်မည်။

ModuleNotFoundError သည် ImportError class ၏ subclass ဖြစ်သည်။

```
import nonexistentmodule
```

```
ModuleNotFoundError          Traceback (most recent call last)
----> 1 import nonexistentmodule
ModuleNotFoundError: No module named 'nonexistentmodule'
```

18.3.3 KeyError

Dictionary တစ်ခုကို key ဖြင့် access လုပ်သည့်အခါ dictionary အတွင်း၌ ထို key ကို မတွေ့လျှင် KeyError ဖြစ်ပေါ်လိမ့်မည်။

```
person = {
"name": "Rich Brown", "age": 56
}
```

```
print(person["gender"])
```

```
KeyError                    Traceback (most recent call last)
----> 4 print(person["gender"])
KeyError: 'gender'
```

18.3.4 TypeError

အမျိုးအစားမတူသည့် variable များကို operation အတူတူ လုပ်မိခြင်းကြောင့် TypeError ဖြစ်ပေါ်လိမ့်မည်။ ဥပမာ- အမျိုးအစားမတူသည့် string နှင့် interger ကို ပေါင်းခြင်း

```
"string" + 8
```

```
TypeError                   Traceback (most recent call last)
----> 1 "string" + 8
TypeError: can only concatenate str (not "int") to str
```

18.3.5 AttributeError

Object တွင် မရှိသည့် attribute ကို assigning လုပ်မိခြင်း သို့မဟုတ် referencing လုပ်မိခြင်းတို့ကြောင့် AttributeError ဖြစ်ပေါ်လိမ့်မည်။

```
X = 10
X.append(6) # Raises an AttributeError
```

```
AttributeError              Traceback (most recent call last)
      1 X = 10
----> 2 X.append(6) # Raises an AttributeError

AttributeError: 'int' object has no attribute 'append'
```

18.3.6 IndexError

မရှိသည့် index နံပါတ်ကို access လုပ်မိလျှင် IndexError ဖြစ်ပေါ်လိမ့်မည်။

```
a = [1,2,3]
print(a[3])
```

```
IndexError                  Traceback (most recent call last)
      1 a = [1,2,3]
----> 2 print(a[3])
IndexError: list index out of range
```

List ထဲတွင် indexe နံပါတ် 0, 1, နှင့် 2 သာ ရှိသည်။ a[3] မရှိသောကြောင့် IndexError ပေးသည်။

18.3.6 NameError

ထည့်ပေးလိုက်သည့် နာမည်တစ်ခုကို မည်သည့်နေရာ(locally or globally)တွင်မှ ရှာမတွေ့လျှင် NameError ဖြစ်ပေါ်လိမ့်မည်။ Name သို့မဟုတ် variable ကို define မလုပ်ထားသောကြောင့်သော်လည်းကောင်း မလုပ်ရသေး သောကြောင့်သော်လည်းကောင်း ဖြစ်ပေါ်နိုင်သည်။

```
print(age)
```

```
NameError                   Traceback (most recent call last)
----> 1 print(age)
NameError: name 'age' is not defined
```

18.3.7 FileNotFoundError

File တစ်ခုကို read လုပ်ရန် သို့မဟုတ် write လုပ်ရန် ကြိုးစားသည့်အချိန်တွင် ထို file ကို ရှာမတွေ့ပါက FileNotFoundError ပေးလိမ့်မည်။ အောက်တွင် input.txt file ကို ဖွင့်သည့်အချိန်တွင် ထို file ကို ရှာမတွေ့သောကြောင့် FileNotFoundError ပေးသည်။

```
open('input.txt', 'r')
```

```
FileNotFoundError                         Traceback (most recent)
----> 1 open('input.txt', 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'input.txt'
```

18.3.8 Python - Error Types

အရေးကြီးသည့် built-in exceptions များကို အောက်တွင် ဖော်ပြထားသည်။

Exception	Description
AssertionError	Raised when the assert statement fails.
AttributeError	Raised on the attribute assignment or reference fails.
EOFError	Raised when the input() function hits the end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in the local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when a system operation causes a system-related error.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by the next() function to indicate that there is no further item to be returned by the iterator.
SyntaxError	Raised by the parser when a syntax error is encountered.

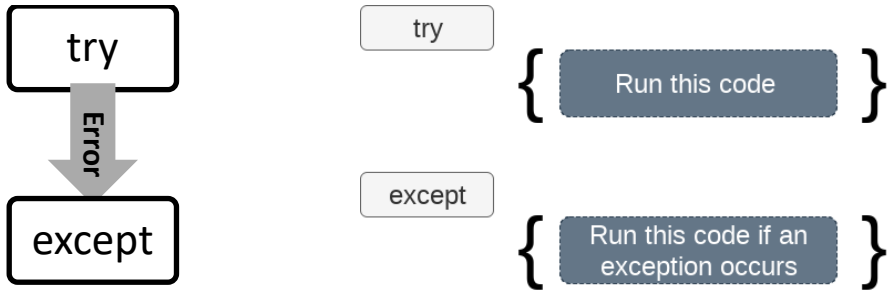
Exception	Description
IndentationError	Raised when there is an incorrect indentation.
TabError	Raised when the indentation consists of inconsistent tabs and spaces.
SystemError	Raised when the interpreter detects internal error.
SystemExit	Raised by the sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of an incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translation.
ValueError	Raised when a function gets an argument of correct type but improper value.
ZeroDivisionError	Raised when the second operand of a division or module operation is zero.

18.4. try…except Block

ပရိုဂရမ် ရေးသားစဉ် အမှား(error)များ မကြာခဏ ဖြစ်ပွားလေ့ရှိသည်။ error ဖြစ်ပေါ်ရသည့် အကြောင်းရင်းများကို သေချာစွာ နားလည်ခြင်းဖြင့် ပရိုဂရမ်တစ်ခုလုံး ရပ်တန့်မသွားအောင် လုပ်နိုင်သည်။ ဖြစ်ပေါ်လာနိုင်မည့် Error များကို ကိုင်တွယ်ဖြေရှင်းနိုင်သည့် အလွယ်ဆုံးနည်းလမ်းတစ်ခုမှာ try…except block ကို အသုံးပြုခြင်း ဖြစ်သည်။ try statement ကို exception handler ဟုလည်းခေါ်သည်။

18.4.1 try… except Block အလုပ်လုပ်ပုံ

try…except block တွင် try block နှင့် except block ဟူ၍ အပိုင်း(၂)ပိုင်း ပါဝင်သည်။ try block ထဲတွင် မိမိကြိုးစားဆောင်ရွက်လိုသည့် အလုပ်တစ်ခုခုကို ထည့်ထားသည်။ try block ထဲရှိ အလုပ်သည် အဆင်ပြေအောင်မြင်သွားလျှင်(no error ဖြစ်လျှင်) except အပိုင်းကို ကျော်၍ အောက်က ကုဒ်များကို ဆက်လက် execute လုပ်မည်။ error ရှိနေလျှင် except block ထဲမှ statement များကို ဆောင်ရွက်(execute လုပ်)မည်။



try အပိုင်းတွင် လုပ်မည် အလုပ် မအောင်မြင်ပါက မျှော်လင့်ထားသည့် error ပေးပါက except အပိုင်းတွင် ရေးထားသည့် ကုဒ်များအတိုင်း execute လုပ်မည်။ Except block တွင် တစ်ခုထက်ပိုများသည့် exception များကို catch လုပ်နိုင်သည်။

```
try:
    action()                    # ကြိုးစားလုပ်လိုသည့် အလုပ်(block of code)

except Exception1:
    ...                         # block of code

except Exception2:
    ...                         #block of code

#other code
```

try...except Block Example - 1

အောက်က ဥပမာတွင် x နှင့် y ကို ပေါင်းမည့် add() function တစ်ခု ရှိသည်။ List တစ်ခုနှင့် string ကို ပေါင်း၍ မရနိုင်ပါ။ ပေါင်းဘို့ ကြိုးစားပါက `TypeError` ပေးလိမ့်မည်။ ထိုအခါ ပရိုဂရမ် execute မလုပ်တော့ရပ်တန့် လိမ့်မည်။ ထိုသို့မဖြစ်စေရန် try...except block ကို အသုံးပြု၍ ကာကွယ်သည်။

```
def add(x, y):
    print(x + y)                    # Trigger TypeError
try:
    add([0, 1, 2], 'spam')      # list တစ်ခုနှင့် string ကို ပေါင်းရန် ကြိုးစားသည်။
except TypeError:                   # Catch and recover here
    print('Catched TypeError, unable to add')
    print('resuming here')          # Continue here if exception or not
```

```
Catched TypeError, unable to add
resuming here
```

try block တွင် မအောင်မြင်ဘဲ မျှော်လင့်ထားသည့် TypeError ဖြစ်ပေါ်လာသောကြောင့် except အပိုင်းရှိ print statement များ အတိုင်း print ထုတ်ပေးပြီး ပရိုဂရမ်ကို ဆက်လက် execute လုပ်သည်။

try...except Block Example - 2

Function ထဲတွင် try...except block ကို ထည့်ရေးသည်။

```
def divbyzero(x,y):
    try:
        return x/y
    except ZeroDivisionError as zde: #zde မဟုတ်ဘဲ ကြိုက်သည့် အက္ခရာထည့်နိုင်သည်။
        print(zde)

divbyzero(6,0)
```

```
division by zero
```

try...except Block Example - 3

Function ထဲရှိ try...except block ထဲတွင် တစ်ခုထက်ပိုများသည့် exception များ ထည့်ရေးနိုင်သည်။

```
def divbyzero(x,y):
    try:
        return x/y
    except ZeroDivisionError as zde: #zde မဟုတ်ဘဲ ကြိုက်သည့် အက္ခရာထည့်နိုင်သည်။
        print(zde)
    except Exception as e:           #e မဟုတ်ဘဲ ကြိုက်သည့် အက္ခရာထည့်နိုင်သည်။
        print(e)

divbyzero(4,0)
```

```
division by zero
```

try...except Block Example - 4

try...except block သဘောတရားကို အောက်တွင် 'my_input.txt' ဖိုင်ကို ဖွင့်သည့် ဥပမာဖြင့် ရှင်းပြထားသည်။ try အပိုင်းတွင် 'my_input.txt' ဖိုင်ကို ဖွင့်ပြီးဖတ်ရန် ကြိုးစားသည်။ ဖိုင်ကို ဖွင့်နိုင်ပါက myinputfile အဖြစ်သတ်မှတ်ပြီး ထိုဖိုင်အတွင်းရှိ လိုင်းများကို print() လုပ်မည်။ အကယ်၍ မအောင်မြင်ဘဲ FileNotFoundError ရရှိပါက “File does not exist” ကို print() လုပ်မည်။ ထို့နောက် ဖိုင်ကို ဖတ်နိုင်သည်ဖြစ်စေ မဖတ်နိုင်သည်ဖြစ်စေ အောက်က ကျန်သည့် ကုဒ်များကို ဆက်လက် execute လုပ်မည်။

ပထမဦးစွာ 'my_input.txt' ဖိုင်ကို same director တွင် မထည့်ထားဘဲ ကြည့်လျှင် အောက်ပါတိုင်း တွေ့ရမည်။ with ... as အကြောင်းကို အောက်တွင် ရှင်းပြထားသည်။

```
try:
    with open('my_input.txt', 'r') as myinputfile:
        for line in myinputfile:
            print(line)

except FileNotFoundError:
    print("Whoops! File does not exist.")

print("Execution will continue to here.")
a = 2+2                    # To show execution continue
a
```

```
Whoops! File does not exist.
Execution will continue to here.
4
```

ထို့နောက် 'my_input.txt' ဖိုင်ကို same director တွင်ထည့်ထားသည်။ 'my_input.txt' ဖိုင်ထဲတွင် “This is my input file.” ဆိုသည့် စာကြောင်း တစ်ကြောင်းသာ ပါသည်။ 'my_input.txt' ဖိုင်ကို ကောင်းစွာ ဖတ်နိုင်သောကြောင့် အောက်က output ကို ထုတ်ပေးသည်။

```
Thi is my input file.
Execution will continue to here.
```

သတိပြုရန်အချက်တစ်ခုမှာ အပေါ်တွင် ရေးထားသည့် try...except ကုဒ်များသည် FileNotFoundError တစ်မျိုးတည်းကို ဖြေရှင်းပေးနိုင်သည်။ FileNotFoundError မှ လွဲ၍ တခြားသော Error များ ဖြစ်ပေါ်ပါက ပရိုဂရမ်သည် ရပ်တန့်သွားလိမ့်မည်။

တခြားသော error များ ဖြစ်ပေါ်လာနိုင်ပါက except block တွင် ထည့်သွင်းနိုင်သည်။ ဥပမာ- ValueError ဖြစ်နိုင်သောကြောင့် except block တွင် ValueError ကို ထည့်သွင်းထားသည်။

```
try:
    with open('input.txt', 'r') as myinputfile:
        for line in myinputfile:
            print(line)

except FileNotFoundError:
    print("Whoops! File does not exist.")

except ValueError:
    print("A value error occurred")
```

ထိုအပြင် Exception အားလုံးကို ခြံငုံပြီး ယေဘူယျသဘော(generic)ဖြင့်လည်း except block တွင် ထည့်သွင်းထားသည်။ ဥပမာ-

```
try:
    with open('my_input.txt', 'r') as myinputfile:
        for line in myinputfile:
            print(line)

except FileNotFoundError:
    print("Whoops! File does not exist.")
except ValueError:
    print("A value error occurred")
except Exception:
    print("Something unforeseen happened")

print("Execution will continue to here.")
```

သို့သော် FileNotFoundError ၊ ValueError စသည် ဖြင့် မရေးဘဲ exception တစ်ခုတည်းသာ ရေးခြင်းသည် မှန်ကန်သည့် ရေးနည်းမျိုး မဟုတ်ပေ။

try...except Block Example - 5

အောက်တွင် input a နှင့် b ကို user က ထည့်ပေးရသည်။ ပိုင်းခြေတွင် သုည(zero)ထည့်၍ ZeroDivisionError ဖြစ်ပေါ်အောင် ပြုလုပ်သည်။ 3 ကို W ဖြင့်စား၍ ValueError ဖြစ်ပေါ်အောင် ပြုလုပ်ပြီး (၂)မျိုး စမ်းပြထားသည်။

```
try:
    a = int(input())
```

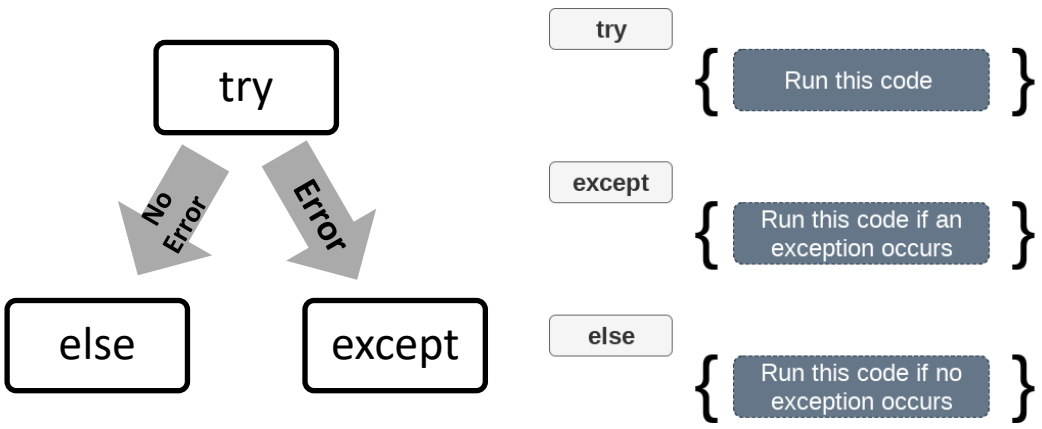
```
    b = int(input())
    print(a/b)

except ZeroDivisionError:
    print('You cannot devide a number by 0')
except ValueError:
    print('invalid input')
except Exception:
    print('error')

print('bye')
```

```
4
0       # input 0 ကြောင့် ZeroDivisionError ဖြစ်ပေါ်သည်။
You cannot devide a number by 0
Bye
```

3 ကို W ဖြင့်စား၍ ValueError ဖြစ်ပေါ်အောင် ပြုလုပ်သည်။

```
3
W    # input W ကြောင့် ValueError ဖြစ်ပေါ်သည်။
invalid input
bye
```

18.5. try...except...else Block



try...except...else block သည် try...except block ကို else statement တစ်ခုထပ်ထည့်ပြီး အနည်းငယ် ပြုပြင်ထားခြင်းသာဖြစ်သည်။ အပိုင်း (၃)ပိုင်း ပါဝင်သည်။ try block ထဲတွင် မိမိဆောင်ရွက်လိုသည့် ကိစ္စကို ထည့်ရသည်။ ပထမဦးစွာ try block ကို စတင် ဆောင်ရွက်သည်။ မအောင်မြင်ဘဲ error ဖြစ်ပေါ်လျှင် except block အတွင်းရှိ statement များကို ဆောင်ရွက်သည်။ error မပေါ်လျှင်(အောင်မြင်လျှင်) else block အတွင်းရှိ statement များကို ဆောင်ရွက်သည်။ မည်သည့် error မှ မဖြစ် ပေါ်ပါက else block အတွင်းရှိ ကုဒ်များကို အမြဲ execute လုပ်သည်။

try...except...else Block Example - 1

```
try:
    with open('my_input.txt', 'r') as myinputfile:
```

```
        for line in myinputfile:
            print(line)

except FileNotFoundError:                    # Catch and recover
    print("Whoops! File does not exist.")
except ValueError:
    print("A value error occurred")
except Exception:
    print("Something unforeseen happened")

else:
    print("No error because file exists")
print("Execution will continue to here.")
```

```
Whoops! File does not exist.
Execution will continue to here.
```

အပေါ်မှ result သည် 'my_input.txt' ကို မတွေ့သောကြောင့် result ထုတ်ပေးသည့် ဖြစ်သည်။ 'my_input.txt' ကို အောင်မြင်စွာ ဖွင့်နိုင်လျှင် အောက် result ကို ရရှိလိမ့်မည်။

```
Thi is my input file.
No error because file exists
Execution will continue to here.
```

try...except...else Block Example – 2

မရှိသည့် file တစ်ခုကို ဖွင့်ရန် ဖတ်ရန် ရေးရန် ကြိုးစားသည်။

```
try:
    f = open('testfile','r')
    f.write('Test write this')

except IOError:
    # This will only check for an IOError exception and then execute th
is print statement
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

```
Error: Could not find file or read data
```

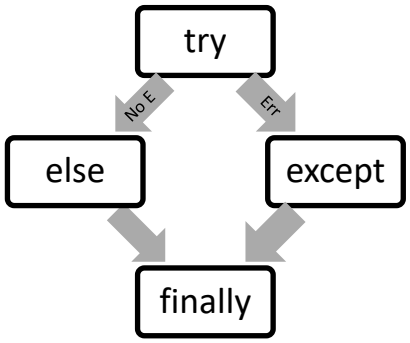
try...except...else Block Format

အပေါ်တွင် ဖော်ပြပြီးခဲ့သည့် ဥပမာများကို စုပေါင်း၍ ပြုစုထားသည့် try...except...else Block Format ဖြစ်သည်။

```
try:
    statements # Run this main action first

except name1:
    statements # Run if name1 is raised during try block
except (name2, name3):
    statements # Run if any of these exceptions occur
```

```
except name4 as var:
    statements # Run if name4 is raised, assign instance raised to var
except:
    statements # Run for all other exceptions raised

else:
    statements # Run if no exception was raised during try block
```

18.6. finally Keyword



တစ်ချို့သော အလုပ်များကို လက်စသတ်ရန် လိုသည်။ ဥပမာ ဖိုင်တစ်ဖိုင်ကို ဖတ်ရန် opening လုပ်ပြီးလျှင် ပြန်ပိတ်ရသည်။ Data base connection ကို ဖွင့်ပြီးလျှင် ပြန်ပိတ်ရသည်။ ယူသုံးထားသည့် system resource များကို release ပြန်ပိတ်ရသည် ထိုသို့ လက်စသတ်ရန် လိုအပ်သော ကိစ္စများအတွက် finally keyword ကို အသုံးပြုသည်။

finally Keyword Example-1

ဖော်ပြခဲ့ပြီးသည့် ဥပမာတွင် finally keyword ကို ထည့်သုံးပြထားသည်။

```
try:
    with open('my_input.txt', 'r') as myinputfile:
        for line in myinputfile:
            print(line)

except FileNotFoundError:
    print("Whoops! File does not exist.")
except ValueError:
    print("A value error occurred")
except Exception:
    print("Something unforeseen happened")

finally:
    print("I will always show up")
print("Execution will continue to here.")
```

```
Whoops! File does not exist.
I will always show up
Execution will continue to here.
```

အပေါ်မှ `result` သည် 'my_input.txt' ကို မတွေ့သောကြောင့် `result` ထုတ်ပေးသည့် ဖြစ်သည်။ 'my_input.txt' ကို အောင်မြင်စွာ ဖွင့်နိုင်လျှင် အောက် `result` ကို ရရှိလိမ့်မည်။

```
Thi is my input file.
I will always show up
Execution will continue to here.
```

18.6.1. else block မပါသည့် try .. except .. finally

```
a = 3
b = 5
try:
    print('resource open')
    print(a/b)
except Exception:
    print('cannot devide a number by 0')

finally:
    print('resource closed')
print('hey')
```

```
resource open
0.6
resource closed
hey
```

18.7. Custom Exceptions

Built-in exception သည် အခြေအနေအများစုအတွင် ရေးသားထားသည့် exception များ ဖြစ်သော်လည် တစ်ခါတစ်ရံ သင့်စိတ်ကြိုက် exception များ လိုချင်သည့်အခါ custom exception များ ရေးနိုင်သည်။ ဥပမာ - ဟင်းချက်နည်း အက်ပလီကေးရှင်းတွင် RecipeNotValidError ကို custom exception အဖြစ် မိမိဖာသာ ထည့်သွင်း နိုင်သည်။ Python ၏ Exception class တွင် custom error ထည့်သွင်းနိုင်သည်။

18.7.1 Implementing Your Own Exception Class

အောက်တွင် RecipeNotValidError ကို custom exception အဖြစ် ထုတ်ပေးနိုင်သည့် ကုဒ်ကို ဖော်ပြထားသည်။

```
class RecipeNotValidError(Exception):
    def __init__(self):
        self.message = "Your recipe is not valid"
try:
    raise RecipeNotValidError
except RecipeNotValidError as e:
    print(e.message)
```

```
Your recipe is not valid
```

အောက်မှာ အများဆုံးကြုံတွေ့ရလေ့ရှိတဲ့ exception များကို ဖော်ပြထားသည်။

try... except

Catch and recover from exceptions raised by Python, or by you.

try... finally

Perform cleanup actions, whether exceptions occur or not.

raise

Trigger an exception manually in your code.

assert

Conditionally trigger an exception in your code.

with… as

with/as statement is an alternative way to ensure that termination actions are carried out for objects that support it.

18.7.2 try Statement Clause Forms

Clause form	Interpretation
`except:`	Catch all (or all other) exception types.
`except name:`	Catch a specific exception only.
`except name as value:`	Catch the listed exception and assign its instance.
`except (name1, name2):`	Catch any of the listed exceptions.
`except (name1, name2) as value:`	Catch any listed exception and assign its instance.
`else:`	Run if no exceptions are raised in the try block.
`finally:`	Always perform this block on exit.

End -