Nested try - block :

The concept of defining one try block inside another try block is know as a "Nested try block".

Syntax

```
try:
    ========
   try:
      ========
   except:
      ========
except:
    =========
```

We have 2 types of try-blocks.

1. Outer try block
2. Inner try block
   - A try block which contains another try block is known as ***outer-try*** block.
   - A try block which is defined in another try-block is known as a ***inner-try*** block.
   - If exception is occured in outer try block then control will goto outer except block.
   - If outer try except block is not handled that exception then program will be terminated abnormally.
   - If exception is occured in the inner try block then control will goto inner try related except block.
   - If inner try related except block is not handled that exception then controll will goto the outer try related except block.
   - If outer try related except block is also not handled that exception then program will be terminated abnormally.

Example 1: If no exception is raised then executes try block and finally block statements.

```
try:
   print('in try1')
   try:
      print('in try2')
      try:
         print('in try3')
      except:
         print('in except3')
      finally:
         print('in finally3')
   except:
      print('in except2')
   finally:
         print('in finally2')
except NameError:
   print('in except1')
finally:
   print('in finally1')
```

Output

```
in try1
in try2
in try3
in finally3
in finally2
in finally1
```

Example 2: If RuntimeError is not handled properly then throws exception

```
try
   print('in try1')
   try:
      print('in try2')
      print(10/0)
      try:
         print('in try3')
      except:
         print('in except3')
      finally:
```

```
        print('in finally3')
    except ValueError:
        print('in except2')
    finally:
        print('in finally2')
except NameError:
    print('in except1')
finally:
    print('in finally1')
```

Output:

```
in try1
in try2
in finally2
in finally1
Traceback (most recent call last):
  File "D:\Python@7.30AM\Exceptions\exception_program.py"
    print(10/0)
ZeroDivisionError: division by zero
```

Note: Here our program excution terminated abnormally because Runtime Exception is not handled properly.

Example 3: What is output Indentify ?

```
try:
    print('try-1')
    a = 10 / 2
    try:
        print('try-2')
        b = 10 / 0
    except TypeError:
        print('except-2')
except:
    print('except-1')
print('end line')
```

Output:

```
try-1
try-2
except-1
end line
```

Example 4: What is output Indentify ?

```
try:
   print('try-1')
   a = 10 / 2
   try:
      print('try-2')
      b = 10 / 0
   except TypeError:
      print('except-2')
   finally:
      print('finally-2')
except NameError:
   print('except-1')
finally:
   print('finally-1')
print('end line')
```

Output:

```
try-1
try-2
finally-2
finally-1
Traceback (most recent call last):
  File "D:\Python@7.30AM\Exceptions\exception_program.py"
    b = 10 / 0
ZeroDivisionError: division by zero
```

Example 5: What is output Indentify ?

```
try:
   print('try-1')
   a = 10 / 2
   try:
      print('try-2')
      b = 10 / 0
```

```
    except TypeError:
        print('except-2')
    finally:
        print('finally-2')
except NameError:
        print('except-1')
except:
    print('default except')
finally:
    print('finally-1')
print('end line')
```

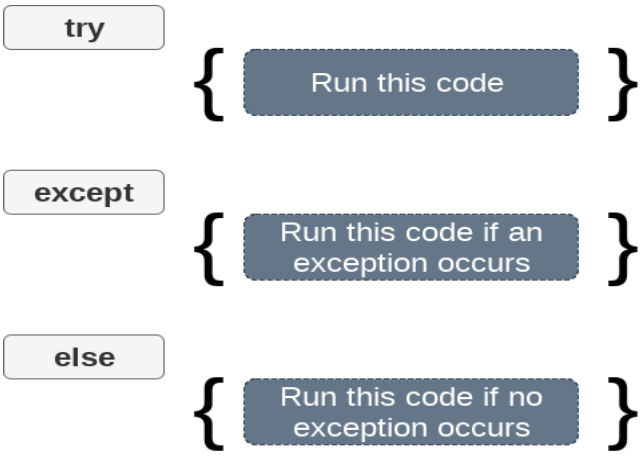
Output:

```
try-1
try-2
finally-2
default except
finally-1
end line
```

else: block

- We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.
- The syntax to use the else statement with the try-except statement is given below.

Syntax:



Example 6: How to read the data from given file using exception handling?

```
fileObject = None
```

```
try:
   fileObject = open('demo.txt' , 'r')
except:
   print('file not found')    # if demo.txt file not available
else:
   data = fileObject.read()   # if demo.txt file available
   print(data)
finally:
   if fileObject:
      fileObject.close()
```

Output 1:

```
python is easy language
python is more powerfull language
python is dynamic
Thank you
```

Output 2:

```
file not found
Thank you
```

The except statement using with "exception variable"

- We can use the exception variable with the except statement. It is used by using the "as" keyword.
- This object will return the cause of the exception.

Syntax:

```
try:
      pass
 except ExceptionName as e:
      pass
```

 # Example 7: return the cause of exception into a exception variable using as keyword

```
try:
   a = int(input("Enter a value :"))
   b = int(input("Enter b value ::"))
   c = a/b
```

```
    print(c)
# Using exception object with the except statement
except Exception  as  e:
    print("can't divide by zero")
    print(e)
else:
     print("Hi I am else block")
```

Output 1:

```
Enter a value :10
Enter b value ::2
5.0
Hi I am else block
```

Output 2:

```
Enter a value :10
Enter b value ::0
can't divide by zero
division by zero
```

TYPES OF EXCEPTIONS :

- Predefined Exceptions
- Userdefined Exceptions

1. Predefined Exceptions :

- The RuntimeError representation classes which are present in python software are known as "predefined execution".
- For example : **ValueError**, **ZeroDivisionError** , **NameError , etc…**
- These are raised automatically when ever corresponding RuntimeError is occured.

2. User defined Exceptions:

- Any user defined class which is extending by any one of the predefined exception class is known as a user defined exception.

Syntax:

```
class   Userdefined_Exception_className(Predefined_exception_class) :
            ======= =======
            ======= =======
```

Example:

```
class   MyClass(ZeroDivisionError):
        pass
```

- User defined exceptions will not "raise" automatiacally. So that we have to write those exceptions explicitly.
- NOTE : Creating the RuntimeError representation class object explicitly is known as a "Raising the Exception"
- By using "raise" keyword, we can raise the userdefined exceptions explicitly.

Syntax: raise userdefined_exception_name

- After raising the exception , we can handle that exception by using "try & except" blocks.

Example 8: How to creating User defined exceptions and raise those exceptions explicitly.

```
class Error(Exception):
   '''base class for other exceptions'''
   pass

class ValueTooLargeError(Error):
   '''raised when input value is too large'''
   pass

class ValueTooSmallError(Error):
   '''raised when input value is too small'''
   pass

number = 10

while True:
   try:
      i_num = eval(input("Enter a Number  : "))

      if  i_num < number:
         raise ValueTooSmallError

      elif i_num  >  number:
         raise ValueTooLargeError
```

```
        else :
            print("Both are Equal numbers.")

        break

    except ValueTooSmallError:
        print("This value is too small, try again")

    except ValueTooLargeError:
        print("This value is Too Large, try again")

print("congrats")
```

Output:

```
Enter a Number  : 7
This value is too small, try again
Enter a Number  : 11
This value is Too Large, try again
Enter a Number  : 10
Both are Equal numbers.
congrats
```

Practice Examples:

Example 9: Find out the given number is within range or not ?

```
try:
    x = int(input('Enter a number upto 100: '))
    if x > 100:
        raise ValueError(x)

except ValueError:
    print(x, "is out of allowed range")

else:
    print(x, "is within the allowed range")
```

Output:

```
Enter a number upto 100: 120
120 is out of allowed range

Enter a number upto 100: 78
78 is within the allowed range
```