SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

III B.Tech.

Computer Science & Engineering

CSE304: PYTHON PROGRAMMING WITH WEB FRAMEWORKS

UNIT-II: Exception Handling

By

Mrs. S. KAMAKSHI, AP-III / CSE

School of Computing

Built-in Exception Classes

- Exceptions
  - SyntaxError
  - AttributeError
  - NameError
  - ValueError
  - LookupError
    - IndexError
    - KeyError
  - TypeError
  - ArithmeticError
    - ZeroDivisionError
    - OverflowError
    - FloatingPointError
  - OSError
    - FileExistsError
    - FileNotFoundError

Default Exception Handling

- Simply prints the standard error message
- Prints the stack trace
- Eg.

```
def fetcher(obj, index):
      return obj[index]
x = 'spam'
fetcher(x, 3)
'm‘
```

 # Default handler - shell interface

```
fetcher(x, 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in fetcher
IndexError: string index out of range
```

 # Default handler - IDLE GUI interface

```
 fetcher(x, 4)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    fetcher(x, 4)
  File "<pyshell#3>", line 2, in fetcher
    return obj[index]
IndexError: string index out of range
```

Exception Handling

- try/except
  - Catch and recover from exceptions raised by Python, or by you.
- try/finally
  - Perform clean-up actions, whether exceptions occur or not.
- raise
  - Trigger an exception manually in your code.
- with/as
  - Implement context managers

Example

```
def catcher():
    try:
        fetcher(x, 4)
    except IndexError:
        print('got exception')
    print('continuing')
catcher()
```

Output:

got exception
continuing

Raising Exceptions

```
try:
    raise IndexError                    # Trigger exception manually
except IndexError:
    print('got exception')
```

Output:

got exception

Interactive shell:

```
raise IndexError
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError
```

Termination Action using finally

- These look like except handlers for exceptions, but the try/finally combination specifies termination actions that always execute “on the way out,” regardless of whether an exception occurs in the try block or not;

```
def after():
    try:
      fetcher(x, 4)
    finally:
      print('after fetch')
    print('after try?')
```

- When an exception does occur in a try block, finally blocks are executed while the program is being unwound:
- control does not resume after the try/finally block when an exception occurs. Instead, Python jumps back to run the finally action, and then propagates the exception up to a prior handler

```
after()
Output:
after fetch
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in after
  File "<stdin>", line 2, in fetcher
IndexError: string index out of range
```

Finally without Exception

- Without Exception:

```
def after():
        try:
            fetcher(x, 3)
        finally:
            print('after fetch')
        print('after try?')
after()
after fetch
after try?
```

- try/except combinations are useful for catching and recovering from exceptions
- try/finally combinations guarantee that termination actions will fire regardless of any exceptions that may occur in the try block's code.
- Eg. Use try/except to catch errors raised by code that are imported from a third-party library,
- Use try/finally to ensure that calls to close files or terminate server connections are always run.

try/except/else

```
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
```

Exception Handling 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.

Catching TypeError

```
def add(x, y):
    print(x + y)                    # Trigger TypeError
try:
    add([0, 1, 2], 'spam')
except TypeError:                   # Catch and recover here
    print('Hello world!')
print('resuming here')              # Continue here if
exception or not
```

Example


```
# File mergedexc.py (Python 3.X + 2.X)
sep = '-' * 45 + '\n'
print(sep + 'EXCEPTION RAISED AND CAUGHT')
try:
   x = 'spam'[99]
except IndexError:
   print('except run')
finally:
   print('finally run')
print('after run')

print(sep + 'NO EXCEPTION RAISED')
try:
   x = 'spam'[3]
except IndexError:
   print('except run')
finally:
   print('finally run')
print('after run')
```

```
print(sep + 'NO EXCEPTION RAISED, WITH ELSE')
try:
   x = 'spam'[3]
except IndexError:
   print('except run')
else:
print('else run')
finally:
   print('finally run')
print('after run')

print(sep + 'EXCEPTION RAISED BUT NOT CAUGHT')
try:
   x = 1 / 0
except IndexError:
   print('except run')
finally:
   print('finally run')
print('after run')
```

Output

```
c:\code> py –3 mergedexc.py
----------------------------------------
EXCEPTION RAISED AND CAUGHT
except run
finally run
after run
----------------------------------------
NO EXCEPTION RAISED
finally run
after run
----------------------------------------
NO EXCEPTION RAISED, WITH ELSE
else run
finally run
after run
----------------------------------------
EXCEPTION RAISED BUT NOT CAUGHT
finally run
Traceback (most recent call last):
 File "mergedexc.py", line 39, in <module>
    x = 1 / 0
ZeroDivisionError:
division by zero
```