Exceptions

- Exceptions are events that can modify the flow or control through a program.
- They are automatically triggered on errors.

- try/except : catch and recover from raised by you or Python exceptions
- 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

1

Exception Roles

- Error handling
  - Wherever Python detects an error it raises exceptions
  - Default behavior: stops program.
  - Otherwise, code try to catch and recover from the exception (try handler)
- Event notification
  - Can signal a valid condition (for example, in search)
- Special-case handling
  - Handles unusual situations
- Termination actions
  - Guarantees the required closing-time operators (try/finally)
- Unusual control-flows
  - A sort of high-level “goto”

2

Example

```
>>> def fetcher(obj, index):
        return obj[index]

>>> x = 'spam'
>>> fetcher(x, 3)                          # Like x[3]
'm'

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


Catching Exceptions

- If you don't want the default exception behavior, wrap the call in a try statement to catch exceptions yourself:

```
>>> try:
...     fetcher(x, 4)
... except IndexError:                      # Catch and recover
...     print('got exception')
...
got exception
>>>
```

- Python jumps to your *handler*—the block under the except clause that names the exception raised—automatically when an exception is triggered while the try block is running

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

>>> catcher()
got exception
continuing
```

Raising Exceptions

- To trigger an exception manually, simply run a raise statement

```
>>> try:
...     raise IndexError
... except IndexError:
...     print('got exception')
...
got exception
```

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

5

User-Defined Exceptions

- User-defined exceptions are coded with *classes*, which inherit from a built-in exception class: usually the class named Exception

```
>>> class AlreadyGotOne(Exception): pass      # User-defined exception

>>> def grail():
        raise AlreadyGotOne()                 # Raise an instance

>>> try:
...     grail()
... except AlreadyGotOne:                     # Catch class name
...     print('got exception')
...
got exception
>>>
```

```
>>> class Career(Exception):
        def __str__(self): return 'So I became a waiter...'

>>> raise Career()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.Career: So I became a waiter...
>>>
```

6

Termination Actions

- 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

```
>>> try:
...     fetcher(x, 3)
... finally:                                  # Termination actions
...     print('after fetch')
...
'm'
after fetch
>>>
```

7

Termination Actions

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

>>> after()
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
>>>
```

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

>>> after()
after fetch
after try?
>>>
```

8