Some more on exception handling

IC152 Feb 2021

More on exceptions

```
9   f = open('exceptData')
10  s = f.readline()
11  i = int(s.strip())
12  print(i)
13  f.close()
```

Without exception handling

```
Traceback (most recent call last):

  File "/home/paddy/pappu/courses/feb2021/ic152/lectures/
lec21_inputOutput/code/exceptDemo2_notry.py", line 11, in <module>
    i = int(s.strip())

ValueError: invalid literal for int() with base 10: 'mars'
```

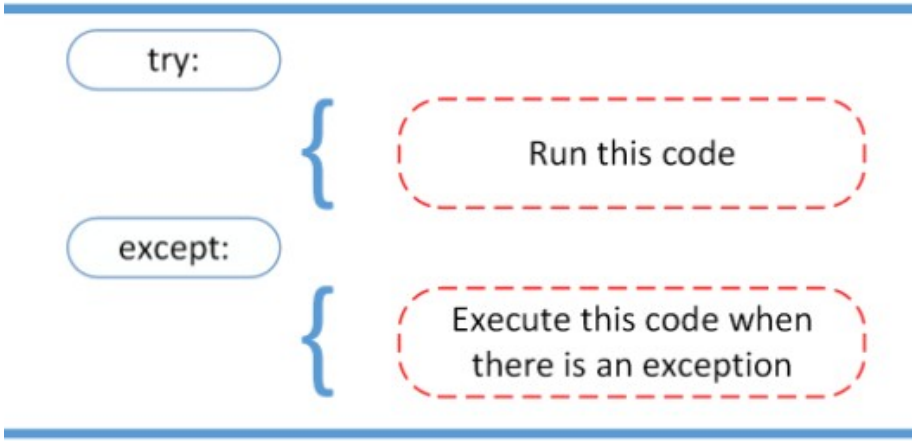
What happens if you are not in a development environment?

Need to handle it and inform the user

We do not want the program to crash

Handling exceptions

```python
while True:
    try:
        x = int(input('Enter an integer: '))
        break
    except ValueError:
        print('Invalid integer. Try again...')
```



- First, the try clause (the statement(s) between the try and except keywords) is executed.
- If no exception occurs, the except clause is skipped and execution of the try statement is finished.
- If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.
- If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements; if no handler is found, it is an unhandled exception and execution stops with a message as shown above.

https://docs.python.org/3/tutorial/errors.html

https://realpython.com/python-exceptions/

Can handle multiple types of exceptions

```
import sys

try:
    f = open('exceptData1')
    s = f.readline()
    i = int(s.strip())
    print(i)
    f.close()
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

Raise the exception for the caller to handle

```python
import traceback

def this_fails(x):
    return x/0

def this_also_fails(x):
    try:
        return x/0
    except ZeroDivisionError as err:
        print('2: Div by zero error inside function', err)
        raise



try:
    x = 5
    this_fails(x)
except ZeroDivisionError as err:
    print('1: Handling run-time error:', err)


try:
    x = 5
    this_also_fails(x)
except ZeroDivisionError as err:
    print('3: Handling run-time error in calling function:', err)
    traceback.print_exc()
```

No exception handler here

Example: raising to the caller

Stack trace can be useful to debug

```
1: Handling run-time error: division by zero
2: Div by zero error inside function division by zero
3: Handling run-time error in calling function: division by zero
Traceback (most recent call last):
  File "/home/paddy/pappu/courses/feb2021/ic152/lectures/
lec21_inputOutput/code/except3.py", line 32, in <module>
    this_also_fails(x)
  File "/home/paddy/pappu/courses/feb2021/ic152/lectures/
lec21_inputOutput/code/except3.py", line 16, in this_also_fails
    return x/0
ZeroDivisionError: division by zero
```

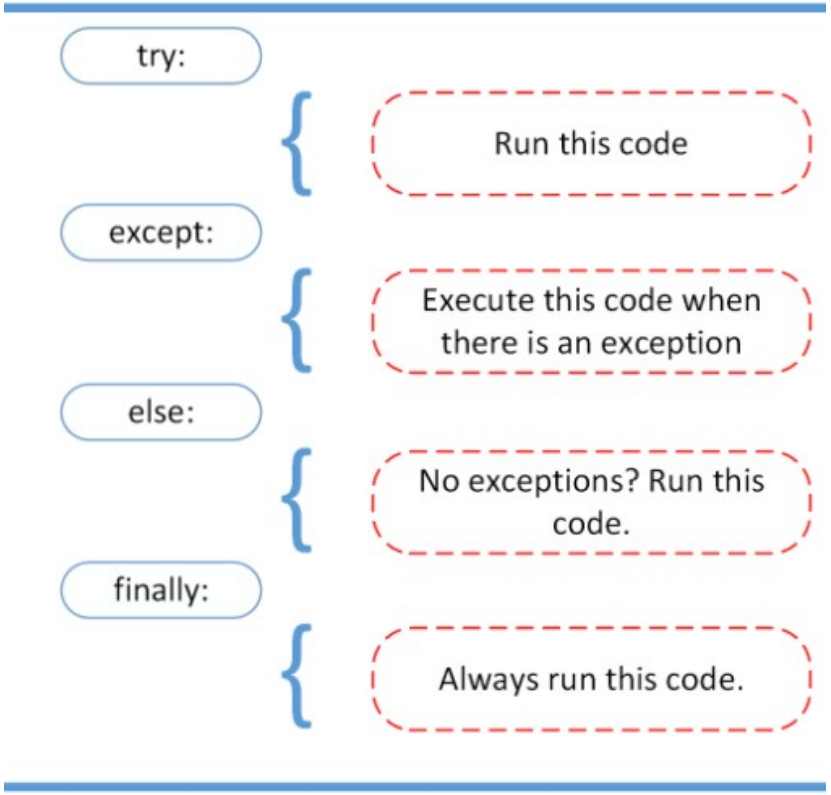
An example of handling an exception

User-defined exception

```
x = int(input('Enter a number less than 5: '))
try:
    if (x>4):
        raise Exception('Input should not exceed 5. You entered {}'.format(x))
except:
    print('Auto-audjusting input value to the max allowed value 4.')
    x = 4

# do something
x = x + 5
print(x)
```

How the whole thing works



https://realpython.com/python-exceptions/

```python

import sys

def linux_interaction():
    assert ('linux' in sys.platform), 'Function can only run on Linux systems.'
    print('Doing something.')


try:
    linux_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
finally:
    print('Cleaning up, irrespective of any exceptions.')

```

Exception hirearchy

https://docs.python.org/3/library/exceptions.html#exception-hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
```