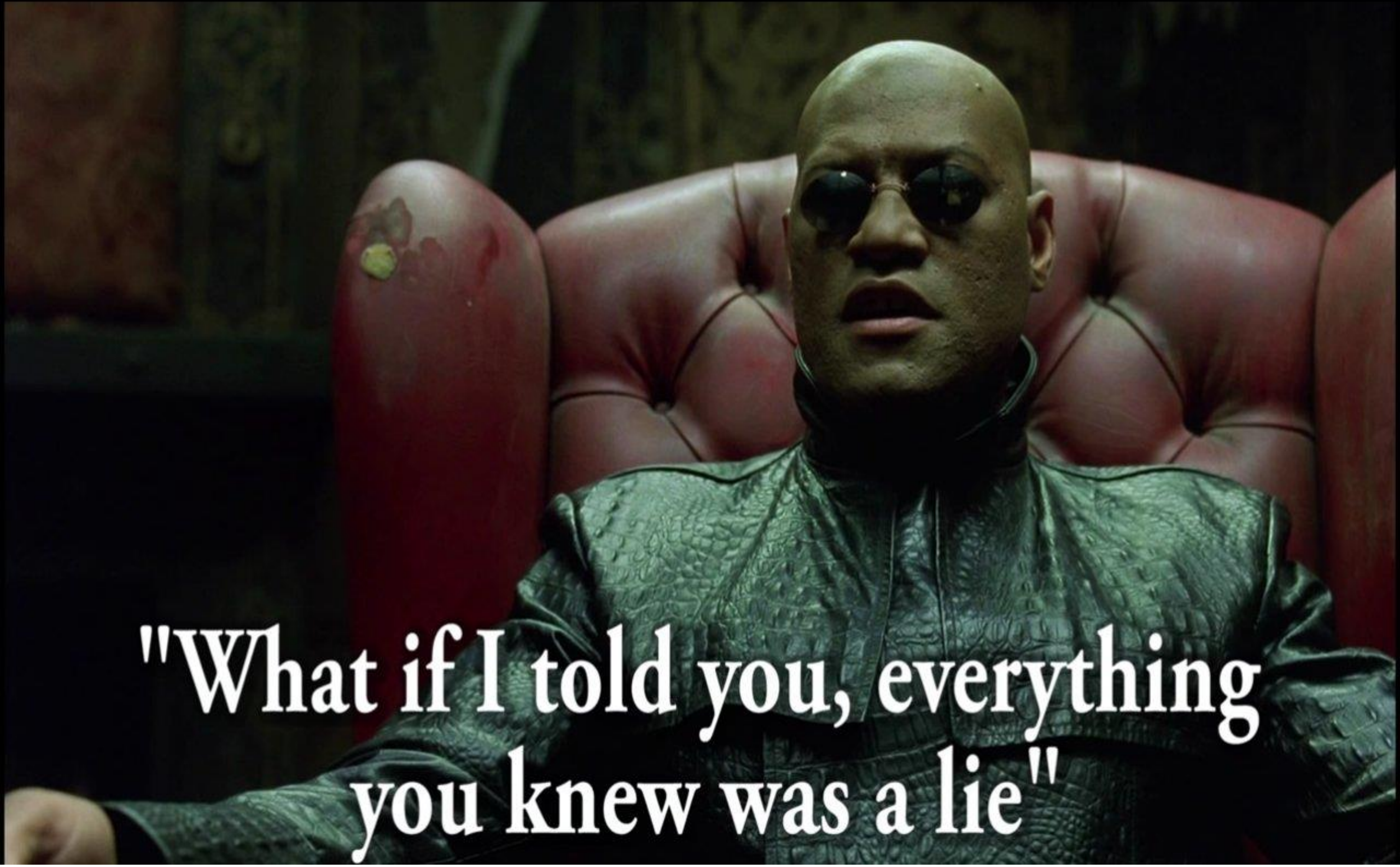
ENGR 102 PROGRAMMING PRACTICE

WEEK 2

İSTANBUL ŞEHİR UNIVERSITY


"What if I told you, everything
you knew was a lie"

Databases (File-based Dictionaries)

- A database is a file that is organized for storing data.
- Some databases are organized like a dictionary
  - Keys → Values
  - The biggest difference:
    - a database is on disk (or other permanent storage), so it persists after the program ends.
- The module **dbm** provides an interface for creating and updating database files.

Databases (File-based Dictionaries)

- Opening a database is similar to opening files:

```
import dbm

db = dbm.open('captions.db', 'c')
```

- Mode 'c':
  - Open for reading and writing,
  - database should be created if it doesn’t exist.
- Returns a database object that can be used (for most operations) like a dictionary.

Databases (File-based Dictionaries)

- If you create a new item, dbm updates the database file.

```
db['cleese.png'] = 'Photo of John Cleese.'
```

- When you access one of the items, dbm reads from the file:

```
print(db['cleese.png'])
```

```
b'Photo of John Cleese.'
```

Databases
(File-based Dictionaries)

- If you make another assignment to an existing key, dbm replaces the old value in the file:

```
db['cleese.png'] = 'Photo of John Cleese eating.'
print(db['cleese.png'])
b'Photo of John Cleese eating.'
```

Databases (File-based Dictionaries)

- Some dictionary methods, like keys() and items(), also work with database objects. You may iterate over keys with a for statement.

```
for key in db:
    print(key)
```

- As with other files, you should close the database when you are done:

```
db.close()
```

Databases

Mode Flags

Value	Meaning
'r'	Open existing database for reading only (default)
'w'	Open existing database for reading and writing
'c'	Open database for reading and writing, creating it if it doesn’t exist
'n'	Always create a new, empty database, open for reading and writing

Pickling

- A **limitation** of dbm is that the **keys and values** have to be **strings**.
  - If you try to use any other type, you get an error.

- **pickle** module may help.
  - pickle.dumps(object) [Object → String]
  - pickle.loads(str) [String → Object]

Pickling

- **pickle.dumps** takes an object as a parameter and returns a string representation
  - (dumps is short for “dump string”).

```
import pickle
t1 = [1, 2, 3]
print(pickle.dumps(t1))
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

Pickling

- Although the new object has the same value as the old, it is not (in general) the same object:

```
str = pickle.dumps(t1)

t2 = pickle.loads(str)

print(t1 == t2)
True

print(t1 is t2)
False
```

Exceptions

The world is not perfect!

```
fin = open('bad_file.txt')
for line in fin:
    print(line)
fin.close()
```

Exceptions

It’s all about errors. What kind?

```
while True print('Hello world')
```

```
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

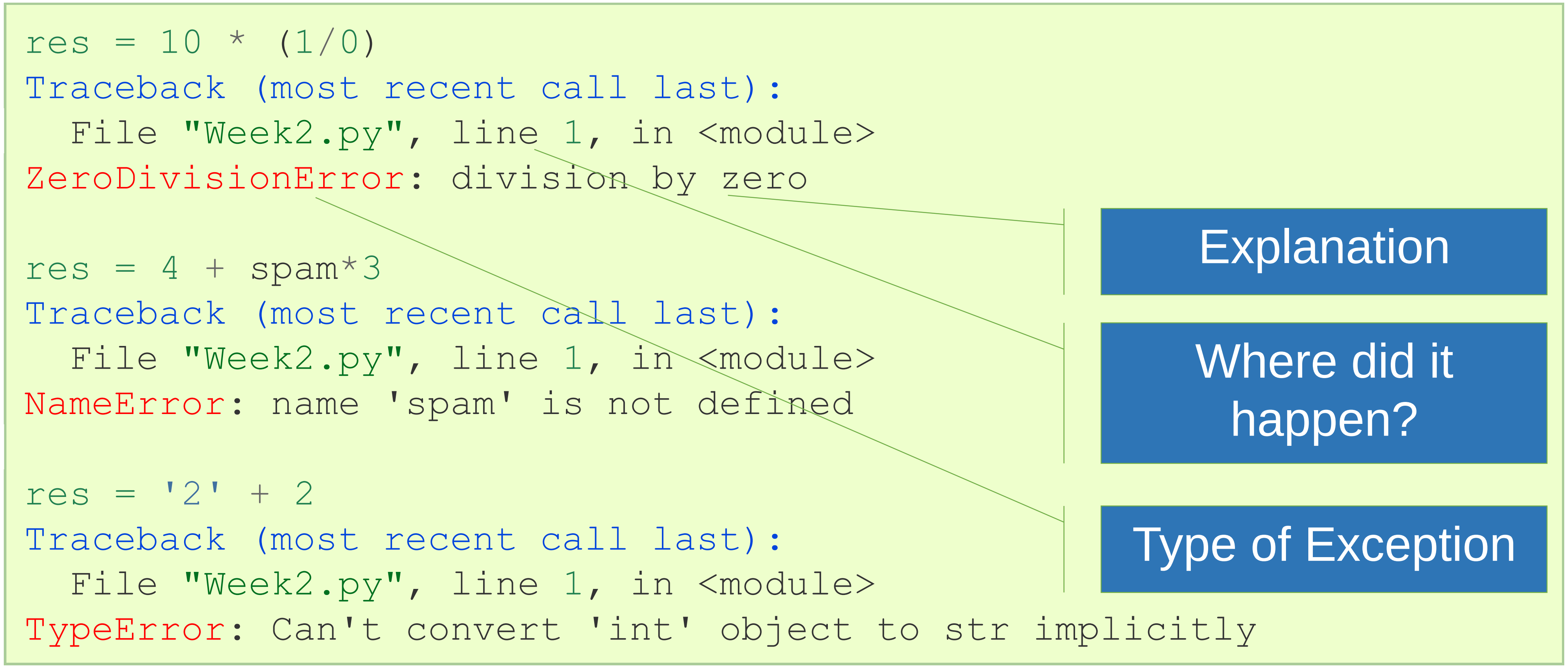
These are parse-time errors

→ detected before running your program.

Exceptions are errors detected during execution!

How do you handle Exceptions?

- Even before that:
  - What happens if you do not handle them?



```
res = 10 * (1/0)
Traceback (most recent call last):
  File "Week2.py", line 1, in <module>
ZeroDivisionError: division by zero

res = 4 + spam*3
Traceback (most recent call last):
  File "Week2.py", line 1, in <module>
NameError: name 'spam' is not defined

res = '2' + 2
Traceback (most recent call last):
  File "Week2.py", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Catching exceptions

- try - except clause

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

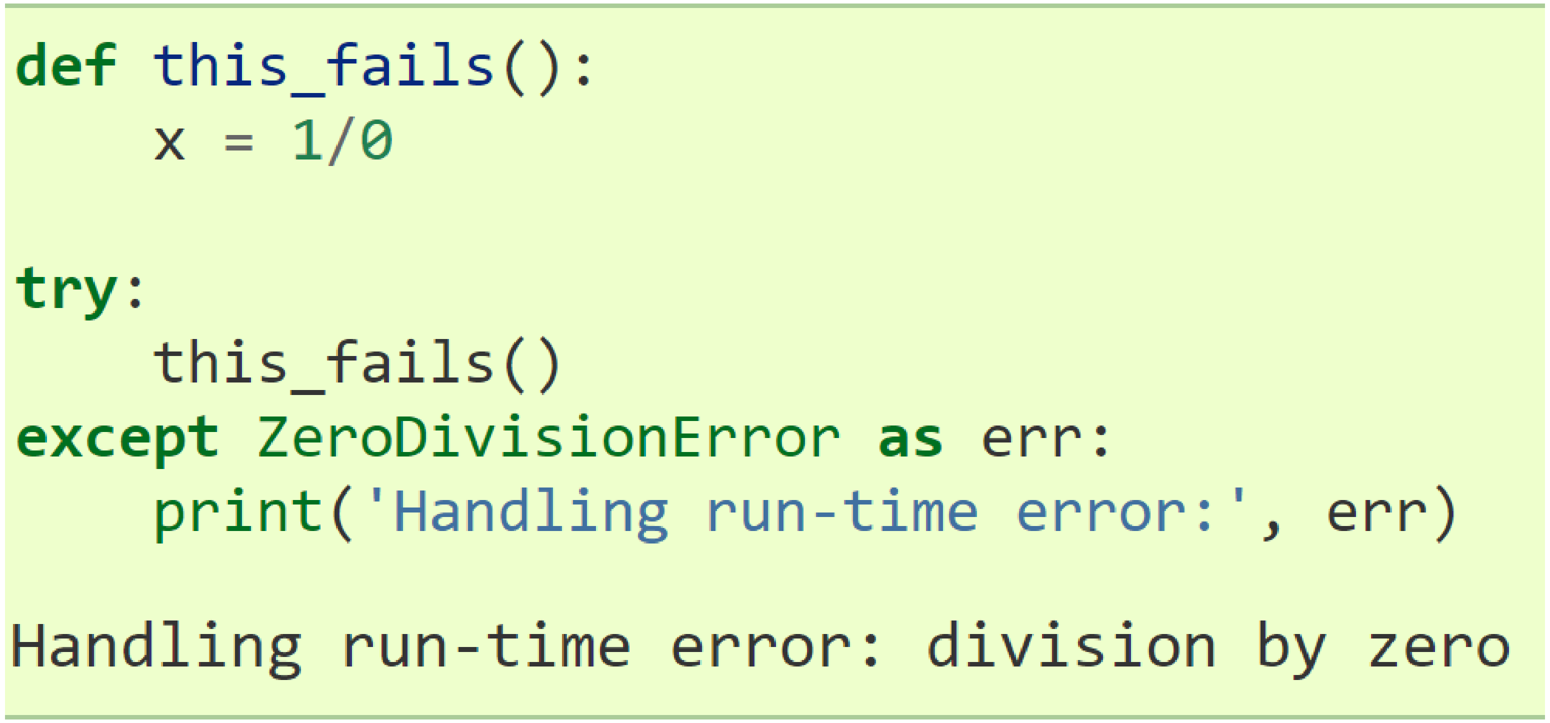
- except clause may have multiple exception types

```
except (RuntimeError, TypeError, NameError):
    pass
```

Multiple exception types

Printing exception details

- use as clause to get and print the exception object



```
def this_fails():
    x = 1/0

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

Handling run-time error: division by zero
```

- This is possible because __str__ method is implemented in Exception class.

Catching exceptions

- multiple except clauses

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error", err)
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error)
    raise
```

Catching exceptions

- optional else clause

```
filenames = ['test.txt', 'program.txt', 'list.txt']
for file in filenames:
    try:
        f = open(file, 'r')
    except OSError:
        print('cannot open', file)
    else:
        print(file, 'has', len(f.readlines()), 'lines')
        f.close()
```

- else block is executed if no exception is thrown.

Raising exceptions

- use raise clause to throw an exception

```
raise NameError('HiThere')
```

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

Raising exceptions

- If you don’t intend to handle an exception, the raise statement with no input allows you to re-raise the exception

```
try:
    raise NameError('HiThere')
except NameError:
    print('An exception flew by!')
    raise

 An exception flew by!
 Traceback (most recent call last):
   File "<stdin>", line 2, in <module>
 NameError: HiThere
```

Defining clean-up actions

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
```

A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.

```
divide(2, 1)
```

```
result is 2.0
executing finally clause
```

```
divide(2, 0)
```

```
division by zero!
executing finally clause
```

```
divide("2", "1")
```

```
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

İSTANBUL ŞEHİR ÜNİVERSİTY