Computational Structures in Data Science

Lecture:
Exceptions

Berkeley
UNIVERSITY OF CALIFORNIA

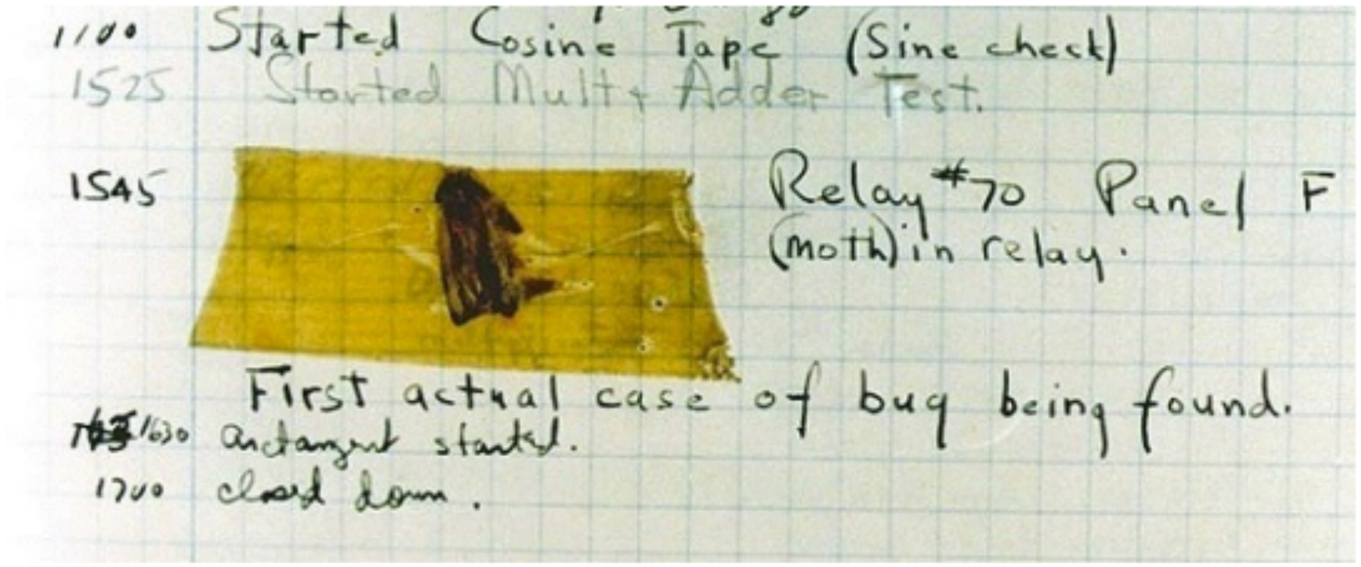
Survey Comments

Learning Objectives

- Exceptions give us a formal way to address error conditions
- "Catch" exceptions in a Python Program
- Define and Raise our own exceptions

Errors Can Occur Just About Anywhere!

- Function receives arguments of improper type?
- Resources (e.g. files or some data) are not available
- Network connection is lost or times out?



1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

Grace Hopper's Notebook, 1947, Moth found in a Mark II Computer

Example exceptions (Docs)

- Unhandled, "thrown" back to the top level interpreter
- **Or halt the program**

```
>>> 3/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> str.lower(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: descriptor 'lower' requires a 'str' object but received a 'int'
>>> ""[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
```

Exceptions mean something bad has happened!

Functions

- Q: What is a function supposed to do?
- A: One thing well
- Q: What should it do when it is passed arguments that don’t make sense?

```
>>> def divides(x, y):
...     return y%x == 0
...
>>> divides(0, 5)
???
>>> def get(data, selector):
...     return data[selector]
...
>>> get({'a': 34, 'cat':'9 lives'}, 'dog’)
????
```

Exceptional exit from functions

- Function doesn’t “return” but instead execution is thrown out of the function

```
>>> def divides(x, y):
...     return y % x == 0
...
>>> divides(0, 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in divides
ZeroDivisionError: integer division or modulo by zero
>>> def get(data, selector):
...     return data[selector]
...
>>> get({'a': 34, 'cat':'9 lives'}, 'dog')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in get
KeyError: 'dog'
>>>
```

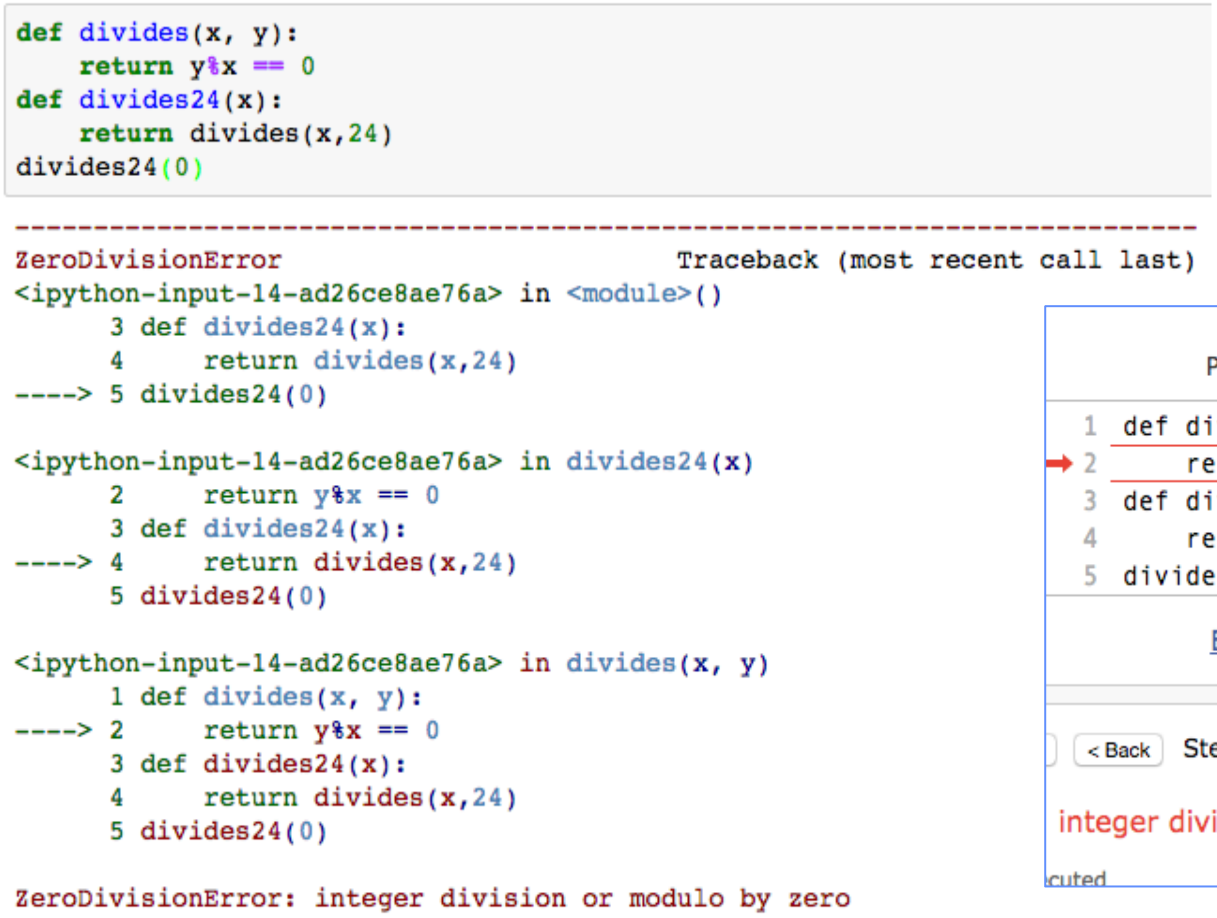
Reading A "Stack Trace" or "Traceback" (Docs)

- All errors in Python *should* return some structured feedback.
- Errors may be dense but contain some really helpful information!

```
👉 python3 -i 18-Exceptions.py
What is your age? 5
Catching CS88Error
Traceback (most recent call last):
  File "…Exceptions.py", line 24, in <module>
    get_age_in_days()
  File "…", line 20, in get_age_in_days
    raise e
  File "…", line 14, in get_age_in_days
    raise CS88Error('You seem young!')
__main__.CS88Error: You seem young!
```

Continue out of multiple calls deep

- Stack “unwinds” until exception is handled or we reach the start of the program



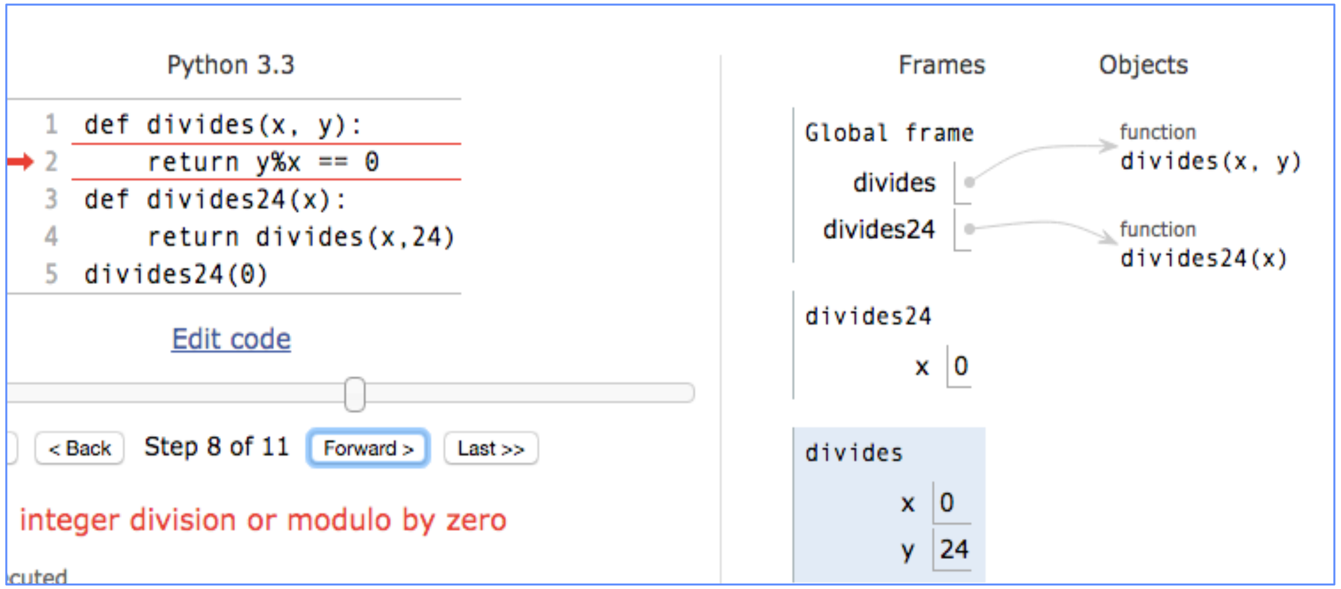
```
def divides(x, y):
    return y%x == 0
def divides24(x):
    return divides(x,24)
divides24(0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-14-ad26ce8ae76a> in <module>()
      3 def divides24(x):
      4     return divides(x,24)
----> 5 divides24(0)

<ipython-input-14-ad26ce8ae76a> in divides24(x)
      2     return y%x == 0
      3 def divides24(x):
----> 4     return divides(x,24)
      5 divides24(0)

<ipython-input-14-ad26ce8ae76a> in divides(x, y)
      1 def divides(x, y):
----> 2     return y%x == 0
      3 def divides24(x):
      4     return divides(x,24)
      5 divides24(0)

ZeroDivisionError: integer division or modulo by zero
```



Python 3.3

```
def divides(x, y):
    return y%x == 0
def divides24(x):
    return divides(x,24)
divides24(0)
```

integer division or modulo by zero

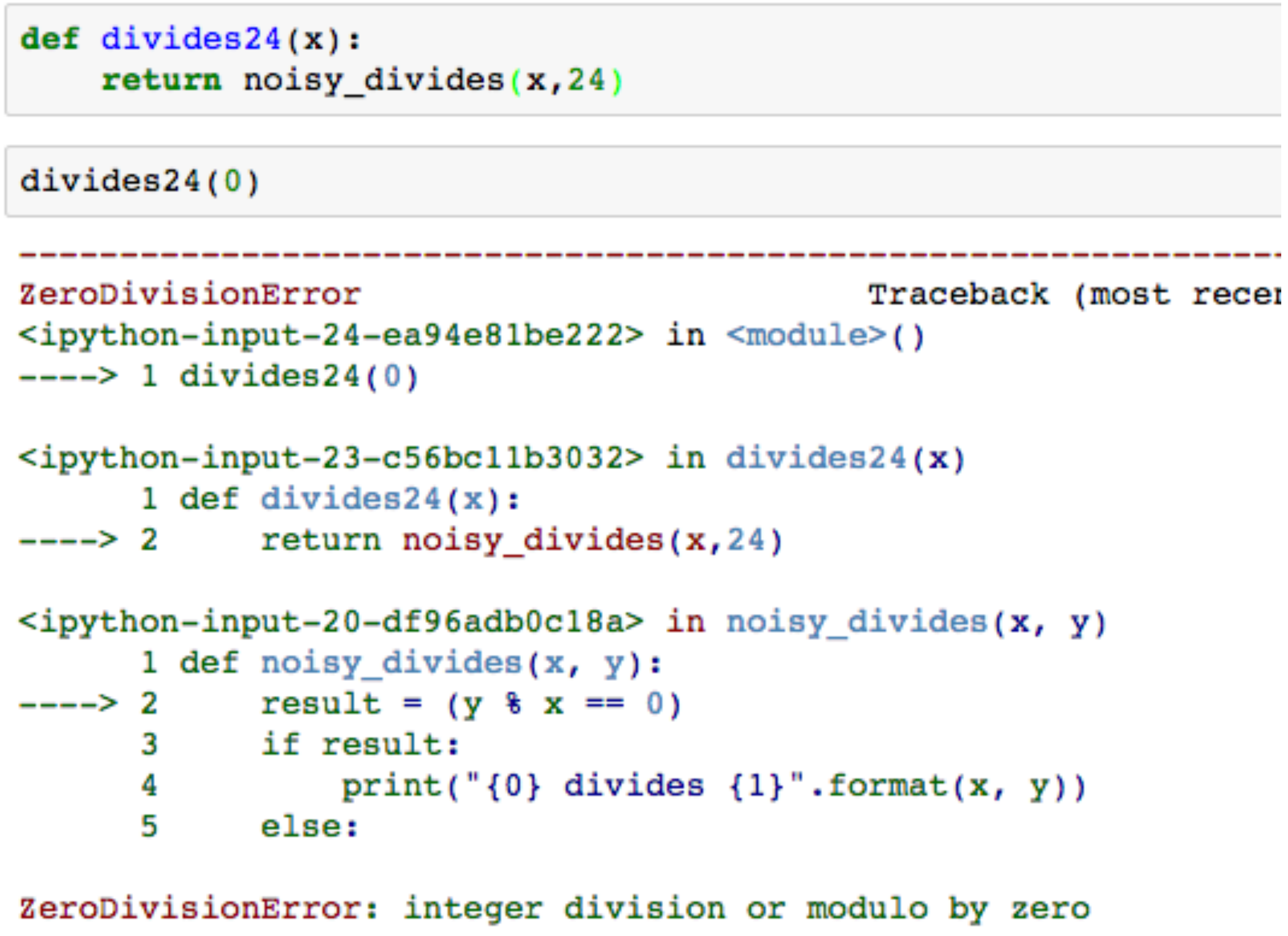
cuted

Types of exceptions

- Exceptions are just classes in Python, with common types for ease of use / clarity.
  - All inherit from BaseException
- AssertionError – The of exception raised by a failing assert statement
- TypeError -- A function was passed the wrong number/type of argument
- NameError -- A name wasn't found
- KeyError -- A key wasn't found in a dictionary
- RuntimeError -- Catch-all for troubles during interpretation
- Your own exceptions!

Flow of control stops at the exception

- And is ‘thrown back’ to wherever it is caught, by default no where.



```
def divides24(x):
    return noisy_divides(x,24)
```

```
divides24(0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recer
<ipython-input-24-ea94e81be222> in <module>()
----> 1 divides24(0)

<ipython-input-23-c56bc11b3032> in divides24(x)
      1 def divides24(x):
----> 2     return noisy_divides(x,24)

<ipython-input-20-df96adb0c18a> in noisy_divides(x, y)
      1 def noisy_divides(x, y):
----> 2     result = (y % x == 0)
      3     if result:
      4         print("{0} divides {1}".format(x, y))
      5     else:

ZeroDivisionError: integer division or modulo by zero
```

Assert Statements

- Allow you to make assertions about assumptions that your code relies on
  - Use them liberally!
  - Incoming data is "dirty" and unsafe till you’ve "cleaned" it

```
assert <assertion expression>, <string for failed>
```

- They "do nothing" if the statement is true.
- Raise an exception of type AssertionError
- You can turn them off:
  - Ignored in optimize flag: python3 –O …
  - Governed by bool __debug__

```
def divides(x, y):
    assert x != 0, ”Denominator must be non-
zero”
    return y % x == 0
```

Demo

- See an exception get raised
- Use an assert statement to validate input
- Use try/catch to recover from an exception

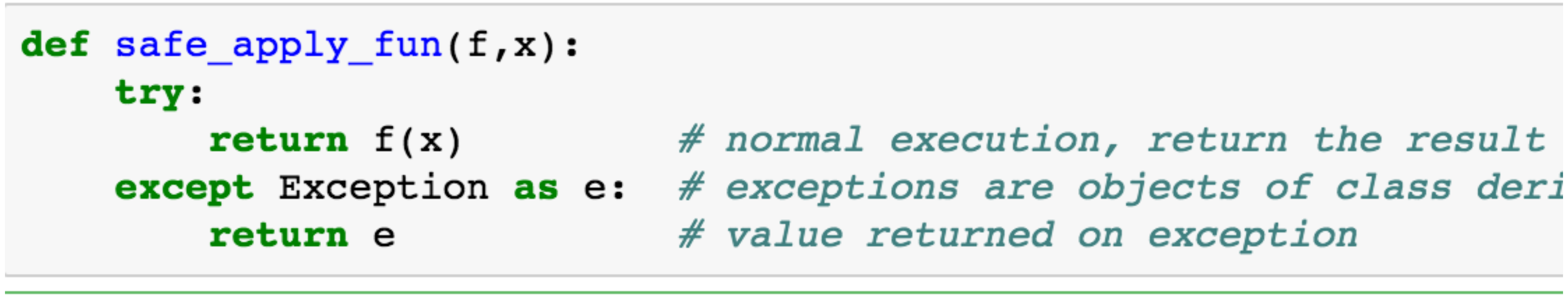
Handling Errors – try / except

- Wrap your code in try – except statements

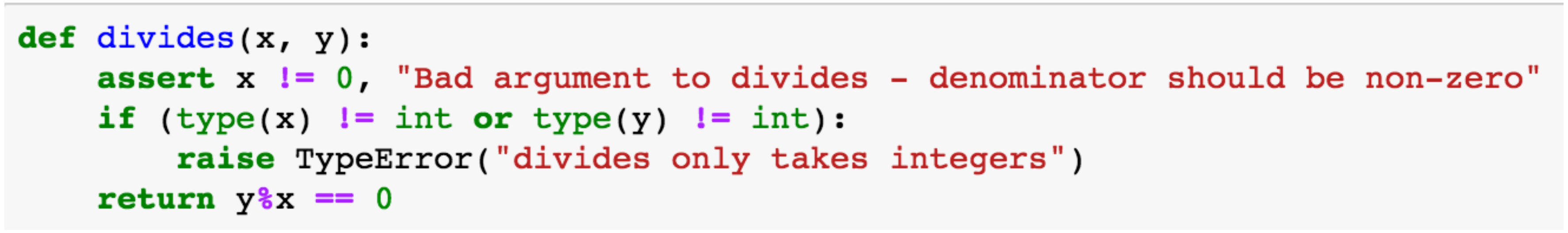
```
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
... # continue here if <try suite> succeeds w/o exception
```

- Execution rule
  - <try suite> is executed first
  - If during this an exception is raised and not handled otherwise
  - And if the exception inherits from <exception class>
  - Then <except suite> is executed with <name> bound to the exception
- Control jumps to the except suite of the most recent try that handles the exception

Demo



```
def safe_apply_fun(f,x):
    try:
        return f(x)          # normal execution, return the result
    except Exception as e:   # exceptions are objects of class deri
        return e             # value returned on exception
```



```
def divides(x, y):
    assert x != 0, "Bad argument to divides - denominator should be non-zero"
    if (type(x) != int or type(y) != int):
        raise TypeError("divides only takes integers")
    return y%x == 0
```

Raise statement

- Exception are raised with a raise statement
  - raise <exception>, e.g.:
  - raise NameError(f"The property {name} does not exist")

- <expression> must evaluate to a subclass of BaseException or an instance of one
- Exceptions are constructed like any other object
  - TypeError(‘Bad argument’)
- **Raise Exceptions for unrecoverable errors!**
  - Something bad has gone on and you cannot continue.

Exceptions are Classes

```
class NoiseyException(Exception):
    def __init__(self, stuff):
        print("Bad stuff happened", stuff)
```

```
class CS88Error(Exception):
    pass # The one time you can skip init. ;)
```

```
try:
    return fun(x)
except:
    raise NoiseyException((fun, x))
```

Demo

Summary

- Approach use of exceptions as a design problem
  - Meaningful behavior => methods [& attributes]
  - ADT methodology: What should a function do?
  - What’s private and hidden? vs What’s public?
- Use it to streamline development

- Anticipate exceptional cases and unforeseen problems
  - try … except
  - raise / assert