Wednesday, April 1

Announcements

- Guerrilla Section 5 this weekend on Scheme & functional programming
- Sunday 4/5 12:30pm – 2pm in 271 Soda
- Homework 7 due Wednesday 4/8 @ 11:59pm
- Quiz 2 released Tuesday 4/7 & due Thursday 4/9 @ 11:59pm
- Project 1, 2, & 3 composition revisions due Friday 4/13 @ 11:59pm
- New policy: Go to lab to earn back up to 5 lost midterm 2 points (some conditions apply)

Lecture 25

Today’s Topic: Handling Errors

Sometimes, computer programs behave in non-standard ways

• A function receives an argument value of an improper type
• Some resource (such as a file) is not available
• A network connection is lost in the middle of data transmission

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

Exceptions

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python raises an exception whenever an error occurs.

Unhandled exceptions will cause Python to halt execution and print a stack trace.

Mastering exceptions:

Exceptions are objects! They have classes with constructors.
They enable non-local continuations of control

If f calls g and g calls h, exceptions can shift control from h to f without waiting for g to return.
(>Exception handling tends to be slow.)

Raising Exceptions

Exceptions are raised with a raise statement

raise <expression>

Raisin12 Exceptions

Assert Statements

Assert statements raise an exception of type AssertionError

assert <expression>, <string>

Assertions are designed to be used liberally. They can be ignored to increase efficiency by running Python with the “-O” flag. “O” stands for optimized

python3 -O

Whether assertions are enabled is governed by a bool __debug__

(Demo)

Raise Statements

Exceptions are raised with a raise statement

raise <expression>

<Expression> must evaluate to a subclass of BaseException or an instance of one

Exceptions are constructed like any other object. E.g., TypeError("Bad argument!")

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

(Demo)
Try Statements

Try statements handle exceptions

```python
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...
```

Execution rule:
The `<try suite>` is executed first. If, during the course of executing the `<try suite>`, an exception is raised that is not handled otherwise, and if the class of the exception inherits from `<exception class>`, then the `<except suite>` is executed, with `<name>` bound to the exception.

Handling Exceptions

Exception handling can prevent a program from terminating.

```python
>>> try:
    x = 1/0
    print('handling a', type(e))
except ZeroDivisionError:
    x = 0

handling a <class 'ZeroDivisionError'>
```

Multiple try statements: Control jumps to the except suite of the most recent try statement that handles that type of exception.

```python
>>> invert_safe(1/0)
handled!
```

WWPD: What Would Python Do?

How will the Python interpreter respond?

```python
>>> invert(0)
Never printed if x is 0
```

```python
def invert(x):
    inverse = 1/x
    return inverse
```

```python
def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)
```

>>> invert_safe(0)
handled!

>>> try:
    ... invert_safe(0)
    ... except ZeroDivisionError as e:
    ...     print('Handled!')
    ...
handled!