SICP

God's Programming Book

Lecture-22 Exceptions

Exceptions

Slides Adapted from cs61a of UC Berkeley

Exceptions

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
- Exceptions can be handled by the program, preventing the interpreter from halting
- 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 continuation 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

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__

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

RecursionError -- Too many recursive calls

Try Statements

Try Statements

Try statements handle exceptions

```
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

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

handling a <class 'ZeroDivisionError'>
>>> x
0
```

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

WWPD: What Would Python Display?

How will the Python interpreter respond?

```
def invert(x):
    inverse = 1/x  # Raises a ZeroDivisionError if x is 0
    print('Never printed if x is 0')
    return inverse

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

```
>>> invert_safe(1/0)
>>> try:
...     invert_safe(0)
... except ZeroDivisionError as e:
...     print('Hello!')
>>> inverrrrt_safe(1/0)
```

Example: Reduce

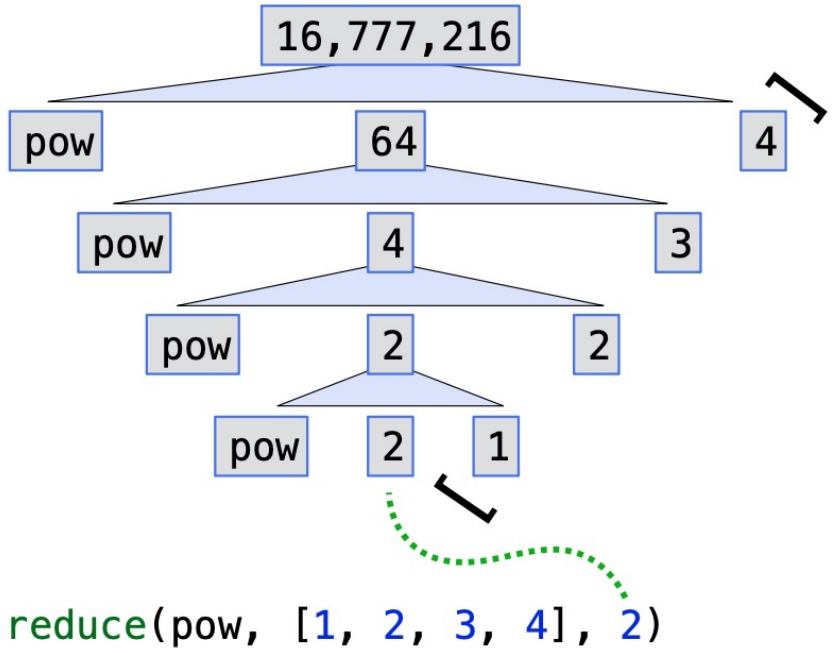
Reducing a Sequence to a Value

```
def reduce(f, s, initial):
    """Combine elements of s pairwise using f, starting with initial.

    E.g., reduce(mul, [2, 4, 8], 1) is equivalent to mul(mul(mul(1, 2), 4), 8).

    >>> reduce(mul, [2, 4, 8], 1)
    64
    """
```

f is ...
  a two-argument function
s is ...
  a sequence of values that can be the second argument
initial is ...
  a value that can be the first argument



Thanks for Listening