Announcements

- Guerrilla Section 5 on Saturday 11/1
- Vanguard section 12-2pm in 271 Soda (max 45 people)
- Main section 2:30-4:30pm in 271 Soda (everyone is welcome)
- Topics: Scheme and Functional Programming
- Homework 7 due Wednesday 11/5 @ 11:59pm
- Project 1 composition revisions due Wednesday 11/5 @ 11:59pm
- Make changes to your project based on the composition feedback you received
- Earn back any points you lost on project 1 composition
- Quiz 2 released Wednesday 11/5 & due Thursday 11/6 @ 11:59pm
  - Open note, open interpreter, closed classmates, closed Internet

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 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

- 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

```
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

(Demo)

---

Handling Exceptions

Exception handling can prevent a program from terminating

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

(Demo)

---

WWPD: What Would Python Do?

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

(Demo)

---

Interpreters

Reading Scheme Lists

A Scheme list is written as elements in parentheses:

```
(element_0) (element_1) ... (element_n)
```

A Scheme list

Each _element_ can be a combination or primitive

```
(+ 3 (+ (2 4) (+ 3 5))) (+ (- 10 7) 6)
```

The task of parsing a language involves coercing a string representation of an expression to the expression itself

Parsers must validate that expressions are well-formed

(Demo)

---

Parsing
A Parser takes text and returns an expression.

Lexical analysis
- Tokens
- Text

Syntactic analysis
- Expression
- A predictive recursive descent parser inspects only k tokens to decide how to proceed, for some fixed k.

Can English be parsed via predictive recursive descent?

The horse raced past the barn fell.

Recursive Syntactic Analysis

Demo

Syntactic analysis identifies the hierarchical structure of an expression, which may be nested.

Each call to scheme_read consumes the input tokens for exactly one expression.

Base case: symbols and numbers

Recursive call: scheme_read sub-expressions and combine them.