From Last Time: Adjoining to a Tree Set

Right!  Left!  Right!  Stop!

```
    8
   / \
  5   9
 / \
3   7
\   \
1   11
```

From the Exam: Pruned Trees

True  True  False

```
(a, b)  (a, c)  (a, d)
```

pruned

```
True  True  False
```

From the Exam: Pruned Trees

pruned(a, c)

implies

pruned(a.right, c.right)

Recursive call: both branches are pruned as well

Base cases: one (or more) of the trees is None

```
def pruned(t1, t2):
    if t2 is None:
        return True
    if t1 is None:
        return False
    return pruned(t1.left, t2.left) and pruned(t1.right, t2.right)
```
Today's Topic: Handling Errors

Sometimes, computers don't do exactly what we expect
- A function receives unexpected argument types
- Some resource (such as a file) does not exist
- Network connections are lost

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

Different Error Handling Policies

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 a crash.
Unhandled exceptions will cause Python to halt execution.

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.
However, exception handling tends to be slow.

Assert Statements

Assert statements raise an exception of type AssertionError

```
assert <expression>, <string>
```

Assertions are designed to be used liberally and then disabled in "production" systems. "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 an exception instance or class.

Exceptions are constructed like any other object; they are just instances of classes that inherit from BaseException.

- 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

Try Statements

Try statements handle exceptions

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

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
except ZeroDivisionError as e:
    print('handling a', type(e))
print(x)
handling a <class 'ZeroDivisionError'>
0
```

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

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

Exception Chaining

The except suite of a try statement can raise another exception that adds additional information.