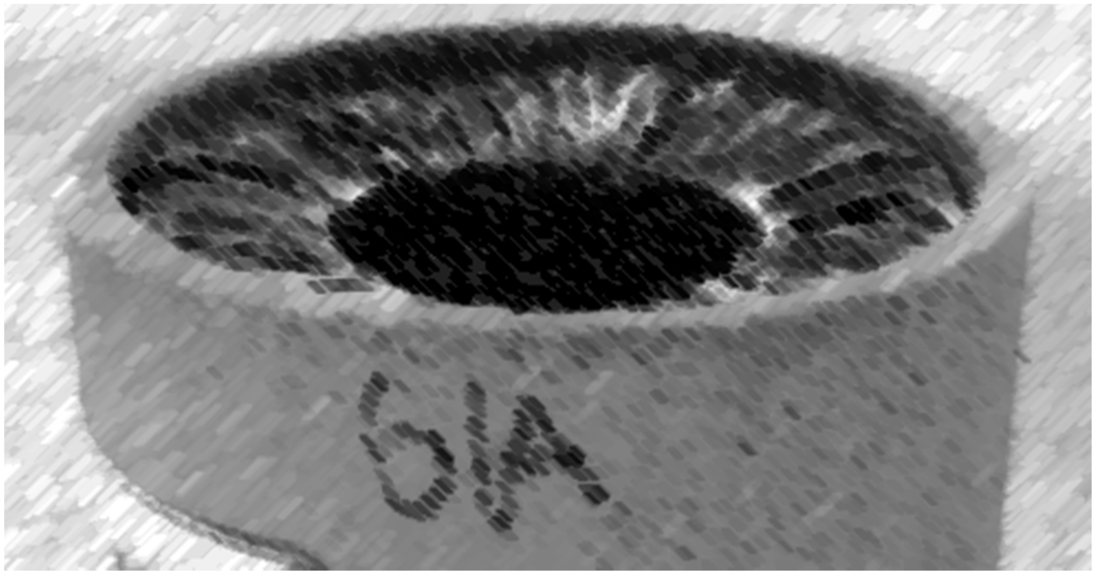


CS61A Lecture 32

Amir Kamil

UC Berkeley

April 5, 2013

Announcements

- Hog revisions due Monday
- HW10 due Wednesday
- Make sure to fill out survey on Piazza
  - We need to schedule alternate final exam times for those who have a conflict, so if you do, let us know on the survey when you are available

The Begin Special Form

Begin expressions allow sequencing

The Begin Special Form

Begin expressions allow sequencing

```
(begin <exp1> <exp2> ... <expn>)
```

The Begin Special Form

Begin expressions allow sequencing

```
(begin <exp_1> <exp_2> ... <exp_n>)
```

```
(define (repeat k fn)
```

The Begin Special Form

Begin expressions allow sequencing

$$\texttt{(begin <exp}_1\texttt{> <exp}_2\texttt{> ... <exp}_n\texttt{>)}$$

```
(define (repeat k fn)
  (if (> k 0)
```

The Begin Special Form

Begin expressions allow sequencing

```
(begin <exp1> <exp2> ... <expn>)
```

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
```

The Begin Special Form

Begin expressions allow sequencing

$$\texttt{(begin <exp}_1\texttt{> <exp}_2\texttt{> ... <exp}_n\texttt{>)}$$

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))
```

The Begin Special Form

Begin expressions allow sequencing

$$(\texttt{begin } \langle exp_1\rangle\ \langle exp_2\rangle\ \ldots\ \langle exp_n\rangle)$$

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))

(define (tri fn)
```

The Begin Special Form

Begin expressions allow sequencing

```
(begin <exp₁> <exp₂> ... <expₙ>)
```

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))

(define (tri fn)
  (repeat 3 (lambda () (fn) (lt 120))))
```

The Begin Special Form

Begin expressions allow sequencing

$$\texttt{(begin <exp}_1\texttt{> <exp}_2\texttt{> ... <exp}_n\texttt{>)}$$

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))

(define (tri fn)
  (repeat 3 (lambda () (fn) (lt 120))))

(define (sier d k)
```

The Begin Special Form

Begin expressions allow sequencing

$$\texttt{(begin <exp}_1\texttt{> <exp}_2\texttt{> ... <exp}_n\texttt{>)}$$

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))

(define (tri fn)
  (repeat 3 (lambda () (fn) (lt 120))))

(define (sier d k)
  (tri (lambda () (if (= k 1) (fd d) (leg d k)))))
```

The Begin Special Form

Begin expressions allow sequencing

$$\texttt{(begin <exp}_1\texttt{> <exp}_2\texttt{> ... <exp}_n\texttt{>)}$$

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))

(define (tri fn)
  (repeat 3 (lambda () (fn) (lt 120))))

(define (sier d k)
  (tri (lambda () (if (= k 1) (fd d) (leg d k)))))

(define (leg d k)
```

The Begin Special Form

Begin expressions allow sequencing

```
(begin <exp₁> <exp₂> ... <expₙ>)
```

```
(define (repeat k fn)
  (if (> k 0)
      (begin (fn) (repeat (- k 1) fn))
      'done))

(define (tri fn)
  (repeat 3 (lambda () (fn) (lt 120))))

(define (sier d k)
  (tri (lambda () (if (= k 1) (fd d) (leg d k)))))

(define (leg d k)
  (sier (/ d 2) (- k 1)) (penup) (fd d) (pendown))
```

Handling Errors (Back to Python)

Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

- A function receives unexpected argument types

Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

- A function receives unexpected argument types
- Some resource (such as a file) is not available

Handling Errors (Back to Python)

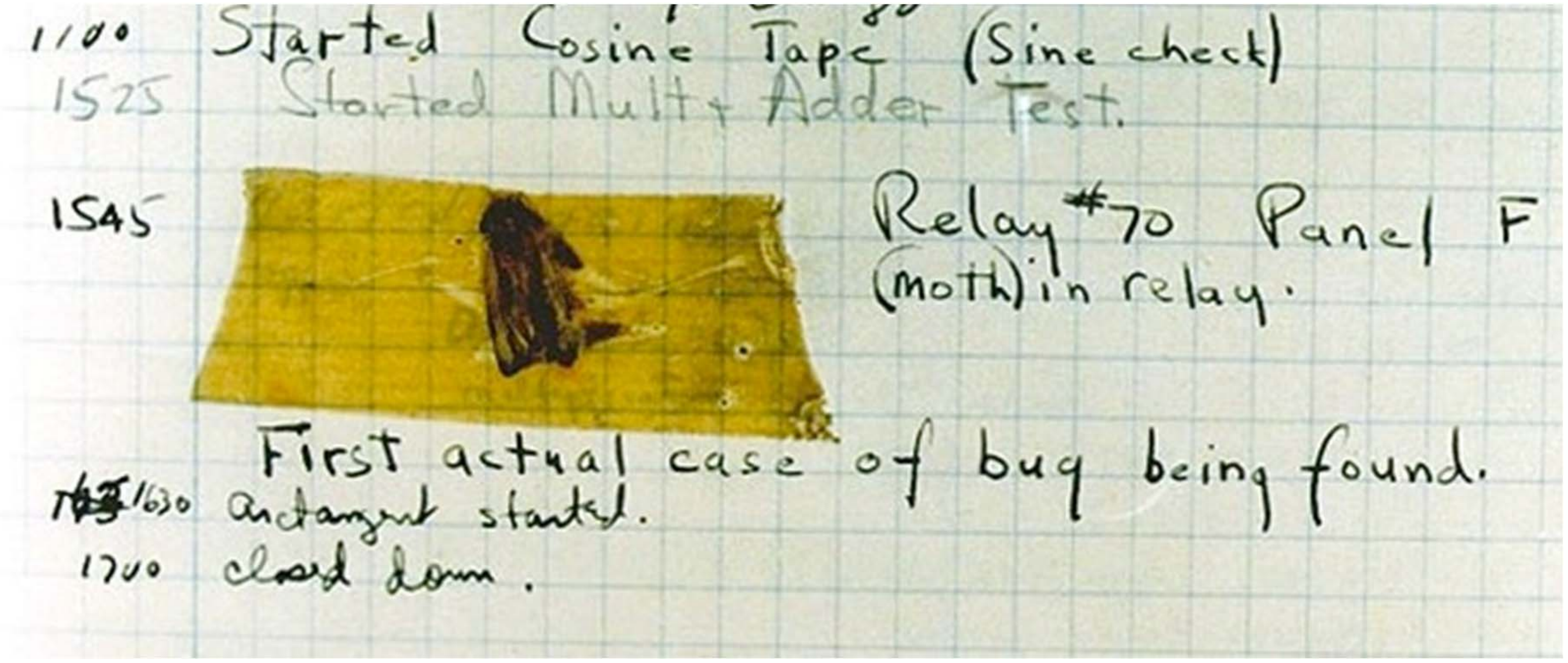
Sometimes, computers don't do exactly what we expect

- A function receives unexpected argument types
- Some resource (such as a file) is not available
- A network connection is lost

Handling Errors (Back to Python)

Sometimes, computers don't do exactly what we expect

- A function receives unexpected argument types
- Some resource (such as a file) is not available
- A network connection is lost



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.

September 9 1947: Moth found in a Mark II Computer

Exceptions

Exceptions

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

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

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

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

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

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

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:

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

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 Statements

Assert statements raise an exception of type **`AssertionError`**

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

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

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

```
python3 -O
```

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

```
python3 -O
```

"O" stands for optimized. Among other things, it disables assertions

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

```
python3 -O
```

"O" stands for optimized. Among other things, it disables assertions

Whether assertions are enabled is governed by the built-in bool **`__debug__`**

Raise Statements

Raise Statements

Exceptions are raised with a *raise statement*

Raise Statements

Exceptions are raised with a *raise statement*

```
raise <expression>
```

Raise Statements

Exceptions are raised with a *raise statement*

```
raise <expression>
```

`<expression>` must evaluate to an exception instance or class.

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`**

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

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

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

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

Try statements handle exceptions

Try Statements

Try statements handle exceptions

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

Try Statements

Try statements handle exceptions

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

Execution rule:

Try Statements

Try statements handle exceptions

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

Execution rule:

- The **<try suite>** is executed first;

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

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

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

Handling Exceptions

Exception handling can prevent a program from terminating

Handling Exceptions

Exception handling can prevent a program from terminating

```
>>> try:
```

Handling Exceptions

Exception handling can prevent a program from terminating

```
>>> try:
        x = 1/0
```

Handling Exceptions

Exception handling can prevent a program from terminating

```
>>> try:
        x = 1/0
    except ZeroDivisionError as e:
```

Handling Exceptions

Exception handling can prevent a program from terminating

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

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 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'>
```

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
```

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
```

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 Do?

How will the Python interpreter respond?

WWPD: What Would Python Do?

How will the Python interpreter respond?

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

>>> invert_safe(1/0)
```

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

```
>>> invert_safe(1/0)
>>> try:
```

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

```
>>> invert_safe(1/0)
>>> try:
        invert_safe(0)
```

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

```
>>> invert_safe(1/0)
>>> try:
        invert_safe(0)
    except BaseException:
```

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

```
>>> invert_safe(1/0)
>>> try:
        invert_safe(0)
    except BaseException:
        print('Handled!')
```

WWPD: What Would Python Do?

How will the Python interpreter respond?

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

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

```
>>> invert_safe(1/0)
>>> try:
        invert_safe(0)
    except BaseException:
        print('Handled!')

>>> inverrrrt_safe(1/0)
```