

Module 16

Error Handling



Acknowledgement

Walker M. White
Cornell University

Motivation

- Suppose we have this code:

```
result = input('Number: ')    # get number from user
x = float(result)             # convert string to float
print('The next number is '+str(x+1))
```

Motivation

- Suppose we have this code:

```
result = input('Number: ')    # get number from user
x = float(result)              # convert string to float
print('The next number is '+str(x+1))
```

- What if user mistypes?

Number: 12a

Traceback (most recent call last):

File "prompt.py", line 13, in <module>

x = float(result)

ValueError: could not convert string to float: '12a'

Ideally Would Handle with Conditional

```
result = input('Number: ') # get number from user
if is_float(result):
    x = float(input)        # convert to float
    print('The next number is ' + str(x+1))
else:
    print('That is not a number!')
```

Does not Exist

Using Try-Except

try:

```
result = input('Number: ') # get number
x = float(result)           # convert to float
print('The next number is ' + str(x+1))
```

except:

```
print('That is not a number!')
```

Using Try-Except

try:

```
result = input('Number: ') # get number
x = float(result)           # convert to float
print('The next number is '+str(x+1))
```

except:

```
print('That is not a number!')
```

Similar to if-else

- But always does the try block
- Might not do **all** of the try block

Python Tutor Example

[Visualize](#)[Execute Code](#)[Edit Code](#)

```
1  try:
2      result = input('Number: ')
3      x = float(result)
4      print('The next number is ' + str(x+1))
5  except:
6      print('That is not a number')
```

Globals

global

result "12a"

Frames

ValueError: could not convert string to float: '12a'

→ line that has just executed

→ next line to execute

A Problematic Function

```
def is_number(s):
```

```
    """Returns: True if string s can be cast to a float
```

```
    Examples: is_number('a') is False
```

```
               is_number('12') is True
```

```
               is_number('12.5') is True
```

```
               is_number('1.2.5') is False
```

```
               is_number('1e-2') is True
```

```
               is_number('0-1') is False
```

```
               is_number('e') is False
```

```
    Precondition: s is a string"""
```

These examples
seem a bit
overwhelming

A Problematic Function

```
def is_number(s):
```

```
    """Returns: True if string s can be cast to a float  
    Precondition: s is a string"""
```

- Complications (It is a mess)
 - Everything must be digit, e, minus, or period
 - Period can only happen once
 - Minus can only happen after e
 - The e can only be second

Taking Advantage of Errors

```
def is_float(s):  
    """Returns: True if string s can be cast to a float  
    Precondition: s is a string"""  
    try:  
        x = float(s)  
        return True  
    except:  
        return False
```

Conversion to a float might fail

If attempt succeeds, string s is a float

Otherwise, it is not

A Design Philosophy Difference



- Conditionals are **asking for permission**
 - Check if a property holds
 - The body proceeds if it is safe
- Try-Except is **asking for forgiveness**
 - Assumes that a property always holds
 - Recovers if it does not
- Python often prefers the **latter**
 - But this is largely unique to Python
 - Only because errors are “relatively” cheap

A Design Philosophy Difference

- Conditionals are **asking for permission**

- Check if a property holds
- The body proceeds if it is safe

- Try-Except is **asking for forgiveness**

- Assumes that...

But still use try-except sparingly.
Only when it simplifies code a lot.

- Python's error handling is largely unique to Python

- Only because errors are “relatively” cheap

Errors and the Call Stack

```
# error.py

def function_1(x,y):
    return function_2(x,y)

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y # crash here

if __name__ == '__main__':
    print function_1(1,0)
```

Errors and the Call Stack

```
# error.py

def function_1(x,y):
    return function_2(x,y)

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y # crash here

if __name__ == '__main__':
    print function_1(1,0)
```

Crashes produce the call stack:

```
Traceback (most recent call last):
  File "error.py", line 20, in <module>
    print(function_1(1,0))
  File "error.py", line 8, in function_1
    return function_2(x,y)
  File "error.py", line 12, in function_2
    return function_3(x,y)
  File "error.py", line 16, in function_3
    return x/y
```

Errors and the Call Stack

```
# error.py
```

```
def
```

Script code.
Global space

```
def function_2(x,y):  
    return function_3(x,y)
```

```
def
```

Where error occurred
(or where was found)

```
if __name__ == '__main__':  
    print function_1(1,0)
```

Crashes produce the call stack:

Traceback (most recent call last):

File "error.py", line 20, in <module>
 print(function_1(1,0))

File "error.py", line 8, in function_1
 return function_2(x,y)

File "error.py", line 12, in function_2
 return function_3(x,y)

File "error.py", line 16, in function_3
 return x/y

Try-Except and the Call Stack



```
def function_1(x,y):  
    try:  
        return function_2(x,y)  
    except:  
        return float('inf')  
  
def function_2(x,y):  
    return function_3(x,y)  
  
def function_3(x,y):  
    return x/y # crash here  
  
print(function_1(1,0))
```

Try-Except and the Call Stack

```
def function_1(x,y):  
    try:  
        return  
        function_2(x,y)  
    except:  
        return float('inf')
```

```
def function_2(x,y):  
    try:  
        return  
        function_3(x,y)  
    except:  
        return 0.0
```

```
def function_3(x,y):  
    return x/y # crash here  
  
print function_1(1,0)
```

Try-Except and the Call Stack

```
def function_1(x,y):  
    try:  
        return function_2(x,y)  
    except:  
        return float('inf')  
  
def function_2(x,y):  
    return function_3(x,y)  
  
def function_3(x,y):  
    return x/y # crash here  
  
print function_1(1,0)
```

- Error “pops” frames off stack
 - Starts from the stack bottom
 - Continues until it sees that current line is in a try-block
 - Jumps to **except**, and then proceeds as if no error

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    third(x)  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    third(x)  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    third(x)  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    third(x)  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    third(x)  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Caught at first'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    third(x)  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Caught at first'
'Ending first'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Caught at second'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Caught at second'
'Ending second'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(2)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Caught at second'
'Ending second'
'Ending first'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

'Starting first.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

'Starting first.'
'Starting second.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

'Starting first.'
'Starting second.'
'Starting third.'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Ending third'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
    print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
    print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Ending third'
'Ending second'

Tracing Control Flow

```
def first(x):  
    print('Starting first.')  
    try:  
        second(x)  
    except:  
        print('Caught at first')  
        print('Ending first')  
  
def second(x):  
    print('Starting second.')  
    try:  
        third(x)  
    except:  
        print('Caught at second')  
        print('Ending second')
```

```
def third(x):  
    print('Starting third.')  
    assert x < 1  
    print('Ending third.')
```

What is the output of **first(0)**?

'Starting first.'
'Starting second.'
'Starting third.'
'Ending third'
'Ending second'
'Ending first'