Numerical Python

Error Handling

CS101 Lecture #21

2016-12-12

Administrivia

Administrivia

- Homework #10 is due Tuesday, Dec. 20.
- Midterm #2 is Monday, Dec. 19 from 7–10 p.m.

Error Handling

Common exceptions

- `SyntaxError`
- `NameError`
- `TypeError`
- `ValueError`
- `IOError`
- `IndexError`
- `KeyError`
- `ZeroDivisionError`
- `IndentationError`
- `Exception`

Exception handling

- Most of the time, we want errors to happen—but we may not want our program to crash (stop executing)!

Exception handling

- Most of the time, we want errors to happen—but we may not want our program to crash (stop executing)!
- We can tell Python to `try` a block of code, and it will run normally `except` if something goes wrong.

Exception handling

- Most of the time, we want errors to happen—but we may not want our program to crash (stop executing)!
- We can tell Python to `try` a block of code, and it will run normally `except` if something goes wrong.

```
d = list( range( 10 ) )
i = 0
while i < len( d )+1:
    try:
        d[ i ] = d[ i ] ** 2.0
        i += 1
    except:
        print( 'An error occurred.' )
        break
```

Exception handling

- The advantage: you can handle the error and execution can proceed normally.
- The disadvantage: the traceback doesn't appear automatically.

Exception handling

- The advantage: you can handle the error and execution can proceed normally.
- The disadvantage: the traceback doesn't appear automatically.
- This also doesn't guard against errors or bugs which don't raise an exception:

```
d = list( range( 10 ) )
i = 0
while i < len( d )+1:
    try:
        d[ i ] = d[ i ] ** 2.0
        i += 1
    except:
        print( 'An error occurred.' )
```

Examples

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Division by zero occurred.")
```

Examples

```
denom = 0
while True:
    try:
        # Read int from console.
        denom = input()

        # Use as denominator.
        i = 1 / float(denom)
    except:
        print("non-numeric value entered")
    else:
        print(i)
    finally:
        if denom == 'q': break
```

Examples

```
try:
    # the main code
except:
    # an error occurs
else:
    # but if no error occurs
finally:
    # in either case, this happens
```

Examples

- If we lose the information on what went wrong, our response may not be appropriate.

Examples

- If we lose the information on what went wrong, our response may not be appropriate.
- What could have gone wrong in the code below?

```
fname = 'spring.data'
try:
    data = open( fname,'r' )
except:
    print( 'Unable to open file "%s".'%fname )
```

Examples

- It is often preferable to handle different kinds of errors separately:

```
fname = 'spring.data'
try:
    data = open( fname,'r' )
except IOError as err:
    print( 'Unable to open file "%s"
      with error "%s".'%(fname,err) )
finally:
    print( 'Done with file I/O code.' )
```

Examples

- Finally, use `try` at the finest degree of precision you can:

```
filename = 'spring.data'
try:
    data = open( filename,'r' )
except IOError as err:
    ...
```

is better than

```
filename = 'spring.data'
try:
    data = open( filename,'r' )
    for line in data:
        ...
except IOError as err:
    ...
```

Examples

```
a = [ 'a','n','y' ]
try:
    a[ 3 ] = '.'
except IndexError:
    pass  # does nothing
a[0][0] = 'b'
```

Which uncaught error will cause this code to terminate?

A `IndexError`
B `TypeError`
C `OSError`

Examples

```
a = [ 'a','n','y' ]
try:
    a[ 3 ] = '.'
except IndexError:
    pass  # does nothing
a[0][0] = 'b'
```

Which uncaught error will cause this code to terminate?

A `IndexError`
B `TypeError` ⋆
C `OSError`

Examples

```
???
try:
    a[ 4 ] *= 2
except TypeError:
    pass
finally:
    print( 'No error arose.' )
```

Which line replacing the `???` will raise an uncaught error?

A `a = '12345'`
B `a = [ 1,2,3,4 ]`
C `a = ( 1,2,3,4,5 )`
D `a = np.ones( ( 10, ) )`

Examples

```
???
try:
    a[ 4 ] *= 2
except TypeError:
    pass
finally:
    print( 'No error arose.' )
```

Which line replacing the ??? will raise an uncaught error?

A `a = '12345'`
B `a = [ 1,2,3,4 ]` ⋆
C `a = ( 1,2,3,4,5 )`
D `a = np.ones( ( 10, ) )`

Configuration Files

Configuration

- We don't like magic numbers, and we prefer not to hard-code values that can change.
- It's also inconvenient to ask the user for input every time.

Configuration

- We don't like magic numbers, and we prefer not to hard-code values that can change.
- It's also inconvenient to ask the user for input every time.
- A *configuration file* allows us to store parameters (like grid size or spacing) where they can easily be changed if necessary.

Configuration

config.ini:

```
dx,1e-3
dy,1e-3
n,1200
```

```
config_file = open( './config.ini','r' )
for line in config_file:
    param = '='.join(line.split(','))
    exec( param )
config_file.close()
```

Configuration

- `exec` accepts Python code as a string and evaluates it.

Configuration

- `exec` accepts Python code as a string and evaluates it.
- This is rather dangerous, so use it carefully!

A note on HW10

- In hw10 we run many simulations.
- A good approach:
  - Create a 2D array for the state variables.
  - Each *row* tracks a different simulation (angle).
  - Each *column* tracks one time step.
  - (You can transpose these as well, but be consistent.)

A note on HW10

```
# Parameters of simulation
n = 1000    # number of data points to plot
m = 20      # number of balls to drop
start = 0.0 # start time of simulation
end   = 2.0 # ending time of simulation
g = -9.8    # acceleration of gravity

# State variable initialization
t = np.linspace(start,end,n+1)           # time in seconds
y = np.zeros((m,n+1),dtype=np.float64)   # height in meters
v = np.zeros((m,n+1),dtype=np.float64)   # velocity in m/s

for i in range(m):
    y[i][0]=i+1
```

A note on HW10

```
for i in range(m): # ball number
    for j in range(1,n+1): # time number
        if y[i][j-1]>0:
            y[i,j] = y[i,j-1] + v[i,j-1] * (t[j]-t[j-1])
            v[i,j] = v[i,j-1] + g * (t[j]-t[j-1])
        else:
            y[i,j] = 0
            v[i,j] = 0

plt.plot( y.transpose() )
plt.show()
```