Debugging Code
When Things Go Wrong
Debugging

- How do I know it isn’t working?
- What do I expect it to do?
- What is my code doing instead? Why?
Common exceptions

- SyntaxError
- NameError
- TypeError
- ValueError
- FileNotFoundError
- IndexError
- KeyError
- ZeroDivisionError
- IndentationError
- Exception
Common exceptions

- SyntaxError—check missing colons or parentheses
- NameError—check for typos, function definitions
- TypeError—check variable types (coerce if necessary)
- ValueError—check function parameters
- FileNotFoundError—check that files exist
Common exceptions

- **IndexError**—don’t reference nonexistent list elements
- **KeyError**—similar to an IndexError, but for dictionaries
- **ZeroDivisionError**—three guesses...
- **IndentationError**—check that spaces and tabs aren’t mixed
- **Exception**—generic error category
Types of Bugs

- A few working definitions:
  - **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
A few working definitions:

- **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
- **Errors**—exceptions which cause the program to be unrunnable (cannot be handled at run time)
A few working definitions:

- **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
- **Errors**—exceptions which cause the program to be unrunnable (cannot be handled at run time)
- **Traceback**—listing of function calls on the stack at the time the exception arises
Types of Bugs

- A few working definitions:
  - **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
  - **Errors**—exceptions which cause the program to be unrunnable (cannot be handled at run time)
  - **Traceback**—listing of function calls on the stack at the time the exception arises
  - **Bugs**—errors and exceptions, but also miswritten, ambiguous, or incorrect code which in fact runs but does not advertise its miscreancy
# calculate squares
d = list(range(10))
i = 0
while i < 10:
    d[i] = d[i] ** 2.0

Which error would this code produce?

A SyntaxError  
B IndexError  
C NameError  
D None of the above.
# calculate squares

d = list(range(10))
i = 0
while i < 10:
    d[i] = d[i] ** 2.0

Which error would this code produce?

A SyntaxError
B IndexError
C NameError
D None of the above. ✫(infinite loop)
import numpy as np

tmax = 10.0

dt = 0.01

nt = int(tmax/dt) + 1

x = np.zeros((nt,))

for i in range(0,nt):
    vx = x[i-1] / np.sin(i)
    x[i+1] = x[i] + vx * dt

Which uncaught error will halt this code?

A ZeroDivisionError
B TypeError
C SyntaxError
D IndexError

When Things Go Wrong
import numpy as np

tmax = 10.0
dt = 0.01
nt = int( tmax/dt ) + 1
x = np.zeros( (nt,) )
for i in range( 0,dt ):
    vx = x[ i-1 ] / np.sin( i )
    x[ i+1 ] = x[ i ] + vx * dt

Which uncaught error will halt this code?

A  ZeroDivisionError
B  TypeError *(range error)*
C  SyntaxError
D  IndexError
Debugging
Flon’s Axiom: “There is not now, nor has there ever been, nor will there ever be, any programming language in which it is the least bit difficult to write bad code.”
This code should find all words whose first two letters are the same:

```python
for line in open( "words.txt" ):
    line = line.strip()
    if len( line ) >= 2:
        a,b = line[1:3]
        if a == b:
            print( line )
```
This code should find all words whose first two letters are the same:

```python
for line in open( "words.txt" ): 
    line = line.strip()
    if len( line ) >= 2:
        print( line )
        a,b = line[1:3]
        print( a,b )
        if a == b:
            print( line )
```
This code should find all words whose first two letters are the same:

```python
for line in open( "words.txt" ):  
    line = line.strip()  
    if len( line ) >= 2:  
        a,b = line[0:2]  
        if a == b:  
            print( line )
```
When do things go wrong?
Debugging strategies

- When do things go wrong?
- Three categories of problems:
  - before the code runs
  - while the code is running
  - in the results
Debugging strategies

- Start early.
Debugging strategies

- Start early.
- Read the problem statement carefully.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
- Add print statements.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
- Add print statements.
- Break the program down into functions.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
- Add print statements.
- Break the program down into functions.
- Document functions before writing them.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
- Add print statements.
- Break the program down into functions.
- Document functions before writing them.
- Explain it to someone else.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
- Add print statements.
- Break the program down into functions.
- Document functions before writing them.
- Explain it to someone else.
- Make no assumptions! If your thinking is not precise, your code will not be precise.
Debugging strategies

- Start early.
- Read the problem statement carefully.
- Chart the flow of the program.
- Add print statements.
- Break the program down into functions.
- Document functions before writing them.
- Explain it to someone else.
- Make no assumptions! If your thinking is not precise, your code will not be precise.
- Start over from scratch. Take a fresh look at the problem.
Consider the following series statement for a Bessel function of the first kind,

\[ J_0(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!m!} \left( \frac{x}{2} \right)^{2m} . \]
from scipy.misc import factorial2 as fact

def term( m,x ):
    return ( ( -1 ) ** m ) / ( fact( m ) * fact( m ) ) * 0.5*x ** 2*m

value    = 0.5
max_term = 20
my_sum   = 0.0
for i in range( 1,max_term ):
    my_sum += term( i,value )
# The following is a reference case (calibration)
from scipy.special import j0 as bessel
A working version:

```python
from scipy.misc import factorial as fact

def term( m, x ):
    return (( -1 ) ** m) / ( fact( m ) ** 2 ) * (x/2)**(2*m)

value = 1.0
max_term = 40
my_sum = 0.0
for i in range( 0, max_term ):
    my_sum += term( i, value )
print ( my_sum)
```
It is often preferable to handle different kinds of errors separately:

```python
filename = 'spring.data'
try:
    data = open(filename,'r')
except FileNotFoundError as err:
    print( 'Unable to open file "{}" with error "{}".'%(filename,err) )
finally:
    print( 'Done with file I/O code.' )
```
Examples

```python
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  KeyError
Examples

```python
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  KeyError
Examples

```python
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, ) )
???
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, ) )
What makes a good Python code?
What makes a good Python code?

import this
Document your code!
Every function should have a docstring.

def warning( msg ):
    '''Display a warning message.'''
    print( 'Warning: %s'%msg)
- Document your code!
- Every function should have a docstring.

```python
def warning( msg ):
    '''Display a warning message.'''
    print( 'Warning: %s'%msg)
```

- Docstrings explain what the function does and what its parameters are.
- They always are triple-quoted strings on the first line of the function block.
Document your code!
Every function should have a docstring.

def warning( msg ):
    '''Display a warning message.'''
    print( 'Warning: %s'%msg)

Docstrings explain what the function does and what its parameters are.
They always are triple-quoted strings on the first line of the function block.

help(warning)
Style

- Use descriptive variable names.
Style

- Use descriptive variable names.
- Why do we write comments?
Use descriptive variable names.
Why do we write comments?
For the person who next looks at the code!

```python
x_vals = [0, 0.1, 0.2, 0.3, 0.4]  # meters
faraday = 96485.3328959  # coulombs,  
    # electric charge
```
Style

- Use functions to structure code.
- This makes code more readable (and debuggable!).