Problem Solving

Profiling, Plotting, Etc.
Coursework
Coursework

- hw08 due 4/17
- exam4 4/18–4/20
- hw09 due 4/21 (not posted yet)
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
C SyntaxError
D IndexError
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
x = np.ones(10)
for i in range(10):
    try:
        ???
    except:
        print('Error on step %d.'%err)
        continue

Which of the following candidates for ??? would not produce an error message?

A. \( x += x[i+1] \)
B. \( x[i] /= 0 \)
C. \( x[-i-1] = \text{sum}(x[:i]) \)
D. \( x[10-i] = \text{sum}(x[:i]) \)
```python
x = np.ones(10)
for i in range(10):
    try:
        ???
    except:
        print('Error on step %d.' % err)
        continue
```

Which of the following candidates for `??` would not produce any error message?

A  $x += x[i+1]$ \textit{index error}
B  $x[i] /= 0$ *(surprise!)*
C  $x[-i-1] = \text{sum}(x[:i])$ *
D  $x[10-i] = \text{sum}(x[:i])$ \textit{index error}
_question #3

```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, ) )`
Question #3

```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, ) )
```
Code Performance
In order to compare algorithms, we need a way to measure code run time (called "wallclock time").
In order to compare algorithms, we need a way to measure code run time (called "wallclock time").

The `timeit` module provides three ways to time your code:
Code performance

- In order to compare algorithms, we need a way to measure code run time (called “wallclock time”).
- The `timeit` module provides three ways to time your code:
  - Interpreter: `timeit.timeit('func( n )', number=10000)`
In order to compare algorithms, we need a way to measure code run time (called “wallclock time”).

The `timeit` module provides three ways to time your code:

- **Interpreter:** `timeit.timeit('func( n )', number=10000)`
- **Command line:** `python3 -m timeit 'code'`
In order to compare algorithms, we need a way to measure code run time (called “wallclock time”).

The `timeit` module provides three ways to time your code:

- **Interpreter:** `timeit.timeit('func(n)', number=10000)`
- **Command line:** `python3 -m timeit 'code'`
- **Notebook:** `%timeit func(n)` (this is easiest)
In order to compare algorithms, we need a way to measure code run time (called “wallclock time”).

The `timeit` module provides three ways to time your code:

- **Interpreter:** `timeit.timeit('func( n )', number=10000)`
- **Command line:** `python3 -m timeit 'code'`
- **Notebook:** `%timeit func( n )` (this is easiest)

These run your code many times and return an average time to completion.
The Mandelbrot set is the set of complex numbers $c$ for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from $z = 0$. 
%timeit %run mandelbrot.py

(Change parameters in the file to experiment.)
Code Profiling
Code timing tells you about how long small blocks of code, like functions, are taking.

Code profiling builds a picture of where a large code is spending time—and thus where to improve!
import time

def hello( n ):
    print("Hello from loop %i!
    time.sleep( n/1000 )

def main():
    for i in range( 0,100 ):
        hello( i )
if __name__ == '__main__':
    main()
Profiling

python -m cProfile hello.py
python -m cProfile -s ncalls hello.py
python -m cProfile -s cumtime hello.py
python -m cProfile mandelbrot.py

The problem is that a nontrivial code spends a lot of time running support code which we don’t need to know about!
The problem is that a nontrivial code spends a lot of time running support code which we don’t need to know about!
Solution: sort the output to a file.

```
python -m cProfile -s cumtime mandelbrot.py > profile.log
```
Next steps

- `matplotlib.ipynb` contains directions on better plotting.
Next steps
Next steps

- hw08 due 4/17
- exam4 4/18-4/20
- hw09 due 4/21 (not posted yet)