Homework #10 is due Friday, Nov. 11.
Homework #11 is due Friday, Nov. 18. Will be posted soon.
Midterm #2 is Monday, Nov. 14 from 7–8 p.m. Sections and rooms are posted on the website—same as last time.
Warmup Quiz
Which of the following optimization techniques repeatedly moves to the best neighboring solution?

A. Hill climbing
B. Brute force
C. Random sampling
D. Random walk
Which of the following optimization techniques repeatedly moves to the best neighboring solution?

A Hill climbing ★
B Brute force
C Random sampling
D Random walk
z = 13
n = np.array( (1,0) )
s = np.array( (-1,0) )
e = np.array( (0,1) )
w = np.array( (0,-1) )
d = [ n,s,e,w ]
x = np.array( (z//2,z//2) )
x += d[ np.random.randint( 0,4 ) ]

What does x represent in this code?

A direction of travel
B current location in array
C step in a random direction
D array of possible locations incl. current
z = 13
n = np.array((1,0))
s = np.array((-1,0))
e = np.array((0,1))
w = np.array((0,-1))
d = [n,s,e,w]
x = np.array((z//2,z//2))
x += d[np.random.randint(0,4)]

What does x represent in this code?

A direction of travel
B current location in array ★
C step in a random direction
D array of possible locations incl. current
d = np.zeros( ( 3,4 ) )

if ???:
    print( 'a zero was found in the array' )

Which of the following comparisons should be included to make this code check if the array contains at least one zero?

A d has 0
B 0 in d
C d.any() == 0
D d.all() == 0
E d == 0
d = np.zeros((3,4))

if ???:
    print( 'a zero was found in the array' )

Which of the following comparisons should be included to make this code check if the array contains at least one zero?

A  d has 0
B  0 in d★
C  d.any() == 0★
D  d.all() == 0
E  d == 0
Error Handling
Common exceptions

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

```python
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.
- This also doesn’t guard against errors or bugs which don’t raise an exception:

```python
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.' )
```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Division by zero occurred.")
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
try:
    # the main code
except:
    # an error occurs
else:
    # but if no error occurs
finally:
    # in either case, this happens
If we lose the information on what went wrong, our response may not be appropriate.

What could have gone wrong in the code below?

```python
filename = 'spring.data'
try:
    data = open(filename,'r')
except:
    print( 'Unable to open file "{}".'.format(filename) )
```
It is often preferable to handle different kinds of errors separately:

```python
filename = 'spring.data'
try:
    data = open( filename,'r' )
except IOError as err:
    print( 'Unable to open file "%s" with error "%s".'%(filename,err) )
finally:
    print( 'Done with file I/O code.' )
```
Finally, use try at the finest degree of precision you can:

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

is better than

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

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