Movie of the week!? 

Chat Q'n:
How long ago did you see this movie?
What's on cs5's horizon...

Exam – two options – released at 7pm **Wed.**

“Code interview” sign-ups: for Fr/Sa/Su aft/eve
paper (pdf) version: sent Wed. evening, due Sun.

Homework #9 and #10  (Data-structuring!)

Good exam review  ~  Practicing lists, recursion, loops...  *but, nothing new this week is on the exam*

Final project ~ most of next week's hw...
Computing to the **max**

The not-so-subtle art of singling out the best (and worst) of anything...

*a comparison comparison*

\[
\begin{array}{|c|c|}
\hline
'm+ms' & 'coffee' \\
42 & 2 \\
0, 'm+ms' & 4 \\
\hline
\end{array}
\]

**Chat Q'n:**

What is your go-to **unhealthy** snack?

- Battle-tested ciphers & how to break them...

**Loops!**

*Lab ~ hw9pr1: Loops are Life!*

This would make me hungry... but I ate breakfast this morning!
The not-so-subtle art of singling out the best (and worst) of anything...

Computing to the **max**

- 'm+ms' vs 'coffee'
- [0, 42] vs [4, 2]
- [0, 'm+ms'] vs [4, 'coffee']

> or <

Computing with **language**

- Battle-tested ciphers & how to break them...

---

This would make me hungry... but I ate breakfast this morning!

---

**Loops!**

Lab ~ hw9pr1: **Loops are Life!**

- hw9pr2: due Sun. eve.
- hw9pr3: due Sun. eve.
max

A recipe for life?
max

A recipe for life?

and python already has it for us...

The real problem is knowing what we want to maximize!
max

A recipe for life?
and python already has it for us...

The real problem is knowing what we want to maximize!

... or minimize, with min
to the max

Want the highest price?

$$\text{max} \left( \begin{array}{c}
\text{nov} & 475.5 \\
\text{jan} & 458.0 \\
\text{mar} & 441.3 \\
\text{may} & 470.8 \\
\text{jul} & 532.8 \\
\text{sep} & 520.9 \\
\end{array} \right)$$

What if the months are in there, as well?

$$\text{max} \left( \begin{array}{c}
\text{may} & 470.8 \\
\text{jul} & 532.8 \\
\text{sep} & 520.9 \\
\end{array} \right)$$

And I thought $54 was overpriced!
to the max

Want the highest price?

$$\max( \ [475.5, \ 458.0, \ 441.3, \ 470.8, \ 532.8, \ 520.9] \ )$$

What if the months are in there, as well?

$$\max( \ [[[470.8, 'may']], \ [[532.8, 'jul']], \ [[520.9, 'sep']]] \ )$$

$$\max( \ [[['may',470.8]], \ [[jul',532.8]], \ [['sep','520.9']]] \ )$$

Mudd's max?

$$\max( \ MSt)$$

Or Mudd's min?

$$\min( \ MSt)$$
def max(L):
    """ returns the max element from L
    input: L, a nonempty list
    """
    maxsofar = L[0]

    for x in L:
        if x > maxsofar:
            maxsofar = x  # it's the new max

    return maxsofar  # now, the _overall_ max

looping max

[ 7, 10, -2, 42, 15 ]

How many ifs?
def max( L ):
    """ returns the max element from L
    input:  L, a nonempty list
    """
    if len(L) < 2:
        return L[0]  # the only elem.

    maxOfRest = max(L[1:])

    if L[0] > maxOfRest:
        return L[0]  # either it's L[0]
    else:
        return maxOfRest  # or maxOfRest
def maxSS(L):
    """returns L's highest scrabble-scoring element (input: L, a nonempty list)"
    """
    if len(L) < 2: return L[0]  # only 1 elem.

    maxOfRest = maxSS(L[1:])  # rest's max

    if sScore(L[0]) > sScore(maxOfRest):
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """
    if len(L) < 2:
        return L[0]  # only 1 elem.
    maxOfRest = maxSS(L[1:])  # rest's max

    if sScore(L[0]) > sScore(maxOfRest):
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
A more comprehensive solution: \texttt{LoL}

\begin{center}
\texttt{L = ['aliens', 'zap', 'hazy', 'code']}
\end{center}

\begin{center}
def \texttt{maxSS( L )}:
    \begin{verbatim}
    """ returns \texttt{L}'s max-scrabble-score word """
    \end{verbatim}
    \begin{verbatim}
    \texttt{LoL} = [ \texttt{[sScore(w), w]} \texttt{for w in L} ]
    \end{verbatim}
    \begin{verbatim}
    \texttt{bestpair} = \texttt{max( LoL )}
    \end{verbatim}
    \begin{verbatim}
    \texttt{return bestpair[1]}
    \end{verbatim}
\end{center}

This does look funny!
A more comprehensive solution

```python
L = ['aliens', 'zap', 'hazy', 'code']

def maxSS(L):
    """ returns L's max-scramble-score word """
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]
```

Karen Carlson <kcarlson48@gmail.com>
to me
to me

Thanks for the email. I'll write you soon. Glad you made it home safely. Lol, mom
A more comprehensive solution

```python
def maxSS(L):
    """ returns L's max-scrabble-score word ""
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]
```

This does look funny!

```
L = ['aliens', 'zap', 'hazy', 'code']
```
A more **comprehensive** solution

```python
L = [ 'aliens', 'zap', 'hazy', 'code' ]

def maxSS(L):
    """ returns L's max-scrabble-score word """
    LoL = [ [sScore(w), w] for w in L ]
    bestpair = max( LoL )
    return bestpair[1]

I loathe hazy code!
```

L = [ 'aliens', 'zap', 'hazy', 'code' ]
LoL = [ [6, 'aliens'], [14, 'zap'], [19, 'hazy'], [7, 'code'] ]
bestpair = [19, 'hazy']
return bestpair[1]
'hazy'
Everything ... is a max problem?

```python
L = [ 'aliens', 'zap', 'hazy', 'code' ]

def lastrest( L ):
    
    LoL = [ [w[1:], w] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
```

I know the best word here... but does Python?
Everything ... is a max problem?

```
L = [ 'aliens', 'zap', 'hazy', 'code' ]

def lastrest( L ):
    """ another example – what's returned? """

    LoL = [ [w[1:], w] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
```
Everything ... is a max problem?

\[
L = ['aliens', 'zap', 'hazy', 'code']
\]

```python
def lastrest( L ):
    """ another example – what's returned? """

    LoL = [ [w[1:], w] for w in L ]
    LoL = [ [ 'liens', 'aliens' ], [ 'ap', 'zap' ], [ 'azy', 'hazy' ], [ 'ode', 'code' ] ]

    bestpair = max( LoL )
    bestpair = [ 'ode', 'code' ]

    return bestpair[1]
```

I know the best word here... but does Python?
Everything ... is a max problem?

```python
L = ['aliens', 'zap', 'hazy', 'code']

def lastrest(L):
    """ another example – what's returned? """
    LoL = [ [w[1:], w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]
```

I know the best word here... but does Python?

```python
L = ['aliens', 'zap', 'hazy', 'code']
LoL = [['liens', 'aliens'], ['ap', 'zap'], ['azy', 'hazy'], ['ode', 'code']]
bestpair = max(LoL)
bestpair = ['ode', 'code']
return bestpair[1]
'code'
```
Everything ... is a max problem?

```python
L = ['aliens', 'zap', 'hazy', 'code']

def lastrevved(L):
    """ another example - what's returned? """
    LoL = [ [w[::-1], w] for w in L ]

    LoL = [ ['aliens'], ['zap'], ['hazy'], ['code'] ]

    bestpair = max(LoL)

    bestpair =

    return bestpair[1]
```

I know the best word here... but does Python?
def lastrevved( L ):
    """ another example – what's returned? """
    LoL = [ [w[:::-1], w] for w in L ]
    bestpair = max( LoL )
    return bestpair[1]
Everything ... is a max problem?

```python
L = ['aliens', 'zap', 'hazy', 'code']

def lastrevved( L ):
    ''' another example – what's returned? '''
    LoL = [ [w[:::-1], w] for w in L ]

    bestpair = max( LoL )
    bestpair = [ 'yzah', 'hazy' ]

    return bestpair[1]
```

I know the best word here... but does Python?
def lastrevved( L ):
    """ another example – what's returned? """

    LoL = [ [w[:::-1], w] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
Other examples...

What is `bestnumb`?

```python
>>> bestnumb([10,20,30,40,50,60,70])
40
```

What is `mostnumb`?

```python
>>> bestnumb([100,200,300,400])
100
```

```python
>>> bestnumb([1,2,3,4,5,6,7,8,7])
8
```

```python
>>> mostnumb([1,2,3,4,5,6,7,8,7])
7
```
def maxlen (L):
    LoL = [ [len(s),s] for s in L ]
    bstpr = max(LoL)
    return bstpr[1]

def bestnumb(L):
    """ returns the # in L closest to 42 ""
    LoL = [ ]
    bstpr =
    return bstpr[1]

def mostnumb(L):
    """ returns the item most often in L ""
    LoL = [ ]
    bstpr =
    return bstpr[1]

L = [ 'aliens', 'zap', 'hazy', 'code' ]

Extra!
Change exactly three characters in this code so that 3 is returned.

L = [ 3,4,5,7,6,7 ]

Hint: Use this helper function!

def count(e, L):
    """ return # of e's in L ""
    LC = [ 1 for x in L if x == e ]
    return sum(LC)
```python
def maxlen(L):
    LoL = [ [len(s),s] for s in L ]
    bstpr = max( LoL )
    return bstpr[1]
```

1. What is LoL?  
   ```python
   [ [6,'aliens'], [3,'zap'], [4,'hazy'], [4,'code'] ]
   ```

2. What is bstpr?  
   ```python
   [6,'aliens']
   ```

3. What is returned?  
   `'aliens'`

**Extra!** Change exactly **three** characters in this code so that 3 is returned.
def bestnumb( L ):
    """ returns the # closest to 42 in L """

    LoL = [ [abs(x-42),x] for x in L ]

    bstpr = min( LoL )

    return bstpr[1]
def count(e, L):
    """ returns the # of e's in L """
    LC = [ 1 for x in L if x==e ]
    return sum(LC)

def mostnumb(L):
    """ returns the item most often in L """
    LoL = [ [count(e,L),e] for e in L ]
    bstpr = max( LoL )
    return bstpr[1]

L = [6,7,7,8]
LoL = [[1,6], [2,7], [2,7], [1,8]]
bstpr = [2,7]
bstpr[1] = 7

Could you use x here instead of e?
Quiz

L = [ 'aliens', 'zap', 'hazy', 'code' ]

def maxlen(L):
    LoL = [ [len(s), s] for s in L ]

    bstpr = max( LoL )

    return bstpr[1]

1. What is LoL?  

2. What is bstpr?  [6, 'aliens']

3. What is returned?  'aliens'

L = [30, 40, 50]

def bestnumb(L):
    ''' returns the # in L closest to 42 '''

    LoL = [ [abs(x-42), x] for x in L ]

    bstpr = min( LoL )

    return bstpr[1]

L = [3, 4, 5, 6, 7]

def mostnumb(L):
    ''' returns the item most often in L '''

    LoL = [ [count(e, L), e] for e in L ]

    bstpr = max( LoL )

    return bstpr[1]

L = [3, 4, 5, 6, 7]

def count(e, L):
    ''' return # of e's in L '''

    LC = [1 for x in L if x == e]

    return sum(LC)

Extra!

Change exactly three characters in this code so that 3 is returned.

L = [3, 4, 5, 6, 7]
Computing with *language*

ideas / meaning

language / words / phrases

strings

numbers / bits

Python strings are here. "alphabetic processions"
Computing with *language*

- **ideass / meaning**
  - language / words / phrases
    - strings
      - numbers / bits

This week...
- processing language - *how English-y is it?*

Next week...
- open questions in AI...

Eliza, Siri, Tay ... trouble?
Caesar Cipher: \textcolor{blue}{\textit{encipher}} + \textcolor{blue}{\textit{decipher}}

\texttt{encipher(s,n)}

\begin{align*}
\text{encipher( 'I <3 Latin' , 0 )} & \rightarrow \text{I <3 Latin} \\
\text{encipher( 'I <3 Latin' , 1 )} & \rightarrow \text{J <3 Mbujo} \\
\text{encipher( 'I <3 Latin' , 2 )} & \rightarrow \text{K <3 Ncvkp} \\
\text{encipher( 'I <3 Latin' , 3 )} & \rightarrow \text{L <3 Odwlq} \\
\text{encipher( 'I <3 Latin' , 4 )} & \rightarrow \text{M <3 Pexmr} \\
\text{encipher( 'I <3 Latin' , 5 )} & \rightarrow \text{N <3 Qfyns} \\
\vdots \quad \vdots \\
\text{encipher( 'I <3 Latin' , 25 )} & \rightarrow \text{H <3 Kzshm}
\end{align*}
Caesar Cipher: `encipher + decipher`

`encipher(s, n)` should return the string `s` with each *alphabetic* character shifted/wrapped by `n` places in the alphabet.

```
encipher( 'I <3 Latin' , 0 )  returns  'I <3 Latin'
encipher( 'I <3 Latin' , 1 )  returns  'J <3 Mbujo'
encipher( 'I <3 Latin' , 2 )  returns  'K <3 Ncvkp'
encipher( 'I <3 Latin' , 3 )  returns  'L <3 Odwlq'
encipher( 'I <3 Latin' , 4 )  returns  'M <3 Pexmr'
encipher( 'I <3 Latin' , 5 )  returns  'N <3 Qfyns'
  .
  .
  .
encipher( 'I <3 Latin' , 25 ) returns  'H <3 Kzshm'
```
Caesar Cipher: `encipher`

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.!', 25)
'Aycqyp agnfcp? G npcdcp Aycqyp qyjyb.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.!', 15)
'Qosgof qwdvsf? W dfstsf Qosgof gozor.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.!', 4)
'Fdhvdu flskhu? L suhihu Fdhvdu vdogd.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.!', 1)
'Caesar cipher? I prefer Caesar salad.'

>>> encipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla /lclyfaopun dl ohcl slhyulk.', 19)
'An education is what remains after we forget everything we have learned.'
```
Caesar Cipher: \texttt{decipher}

```python
>>> decipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc."
'Caesar cipher? I prefer Caesar salad.'

```python
>>> decipher('Hu lkbjhapvu pz doha ylthpuz hmalv dl mvynla "\lclyfaopun dl ohcl slhyulk."')
'An education is what remains after we forget everything we have learned.'

```python
>>> decipher('Uifz xpsl ju pvu xjui b qfodjm!"

```python
>>> decipher('gv vw dtwvg')

Which is more difficult computationally?
Caesar Cipher: \texttt{decipher}

```python
>>> decipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.\')
'Caesar cipher? I prefer Caesar salad.'

>>> decipher('Hu lkbjhapvu pz deh
\'lclyfaovu\'.
\'An education is what remains after we forget everything we have learned.'

>>> decipher('Uifz xpsl ju pvu xjui b qfodjm!'

>>> decipher('gv vw dtwvg!'

On Wed. ~
More details + ideas...
```

Which is more difficult computationally?
"2D" Loops

A new *dimension* in loops: Nesting!
"2D" Loops

A new *dimension* in loops: **Nesting**!
Nested loops are familiar, too!

```python
for mn in range(60):
    for s in range(60):
        tick()
```
**Nested** loops are familiar, too!

So close!
for y in range(84):
    for m in range(12):
        for d in range(f(m,y)):
            for h in range(24):
                for mn in range(60):
                    for s in range(60):
                        tick()
Nested loops' 2d structure

One hour ~ 3600 seconds

for mn in range(60):
    for s in range(60):
        tick()

hour()
Creating 2d structure ~ in ASCII

```python
for row in range(3):
    for col in range(4):
        print('#')

# # # #
# # # #
# # # #
```

Wait! this needs something more...
Creating 2d structure

```
for row in range(3):
    for col in range(4):
        print("#", end='')
```

Hmmm...
Creating 2d structure

```python
[0, 1, 2]
for row in range(3):
    [0, 1, 2, 3]
        for col in range(4):
            print('#', end='')
    print()
```

row =
col =
col =
col =
col =

col =
col =
col =
col =
col =

col =
col =
col =
col =
col =
Creating 2d structure

```python
for row in range(3):
    for col in range(4):
        if col == row:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

Let's take an alien's-eye view!

```
row = 0
col = 0
col = 1
col = 2
col = 3

row = 1
col = 0
col = 1
col = 2
col = 3

row = 2
col = 0
col = 1
col = 2
col = 3
```
Match!

What code creates the fourth one?

* and ** are extra!

for r in range(3):
    for c in range(6):
        if c >= r:
            print('#', end='')
        else:
            print(' ', end='')
    print()

for r in range(3):
    for c in range(6):
        if c%2 == 1:
            print('#', end='')
        else:
            print(' ', end='')
    print()

for r in range(3):
    for c in range(6):
        if c%2 == r%2:
            print('#', end='')
        else:
            print(' ', end='')
    print()

Match!

What code creates the fourth one?

* and ** are extra!

Name(s) ______________________
**Match!**

What code creates the fourth one?

* and ** are extra!

**answers...**

```python
for r in range(3):
    for c in range(6):
        if c >= r:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

```python
for r in range(3):
    for c in range(6):
        if c%2 == 1:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

```python
for r in range(3):
    for c in range(6):
        if c%2 == r%2:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

**Match!**

What code creates the fourth one?

* and ** are extra!

**answers...**

```python
for r in range(3):
    for c in range(6):
        if c+r <= 4:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

```python
for r in range(3):
    for c in range(6):
        if c%2 == 1:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

```python
for r in range(3):
    for c in range(6):
        if c==r or c+r==4:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```
for d in range(f(m)):
    for m in range(1,13):
        num_bdays(m,d)

What trends appear in this birthday data?

How might we be suspicious of the fairness of this data?!

Data represents the # of babies born in the United States between 1973 and 1999

how many shared birthdays are in CS5?

vizwiz.blogspot.com/2012/05/how-common-is-your-birthday-find-out.html
2D Data

Math + CS: shareful siblings!
2D Dagan!

Math + CS: Shareful Siblings!
2D Dagan+Data!

Math + CS: Shareful Siblings!
2D Dagan+Data!

\[
W = \begin{pmatrix} 2 & 9 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix}
\]
2D data

\[ M = \begin{bmatrix} [2,9], & [1,-2] \end{bmatrix} \]

Math + CS: shareful siblings!
2D data

\[ M = \begin{bmatrix} [2, 9], & [1, -2] \end{bmatrix} \]

Handling 2D data requires no new rules!
Python's two methods for handling data

Lists are handled **by reference**: 
L really holds a *memory address*

Numeric data and strings are handled **by value**: imagine they *hold* the data

$L = [5, 42, 'hi']$; $id(L)$ ...

$s = 'hi'$; $id(s)$ ...

$x = 7$; $id(x)$ ...
Shallow vs. Deep

Python's two methods for copying data

L = [5, 42, 'hi']
M = L
M[0] = 60

What's L[0] ?!

= assignment is "shallow"
Shallow vs. Deep

Python's two methods for copying data

L = [5, 42, 'hi']

M = deepcopy(L)

M[0] = 60

What's L[0]?!

deepcopy is "deep"!

from copy import *

L = [5, 42, 'hi']

M = deepcopy(L)

M[0] = 60

What's L[0]?!
Lists are *Mutable*

You can change **the contents** of lists from within functions that take lists as input.

- Lists are **MUTABLE** objects

Those changes will be visible **everywhere**.

Numbers, strings, etc. are **IMMUTABLE** – they can't be changed, only reassigned.
Different approaches to "rules" ...

Engineers

Physicists

Mathematicians

Different "rules"

mean

Different "worldviews..."
Different approaches to "rules" ...

*Engineers* believe their rules approximate reality;

*Physicists* believe *reality approximates their rules* ...

*Mathematicians* are happy either way!

CS?
Engineers believe their rules approximate reality;
Engineers believe their rules approximate reality; Physicists believe *reality approximates their rules*...
Engineers believe their rules approximate reality;
Physicists believe reality approximates their rules...
Mathematicians are happy either way!

A solid sphere can be split into 5 parts and rigidly reassembled into two solid spheres the same size as the original

Banach-Tarski paradox

the parts are "clouds"

https://www.youtube.com/watch?v=s86-Z-CbaHA @ 18:49
I carved and carved, and the next thing I knew I had two pumpkins.
I carved and carved, and the next thing I knew I had two pumpkins.
Engineers believe their rules approximate reality; 
Physicists believe *reality approximates their rules*... 
*Mathematicians are happy either way!*

*Mathematics reasons about *structural* rules...* 

... *CS reasons about *procedural* ones.*

---

**Math worldv***

- Axioms
- Data
- Definitions

- proofs

- Insights, tools, mathematical truths

---

**CS worldview***

- lists
- for
- while
- variables
- arithmetic operations
- if/else

- programs

- Insights, tools, algorithms
Engineers believe their rules approximate reality;
Physicists believe reality approximates their rules...

Mathematicians are happy either way!

CS?

Don't like your reality? Build a new one!

why settle for gears, when you could have fractal gears?

why settle for structured patterns? Self-propagating patterns!

2D data?

All and only the rules that govern 1D data apply here – no new rules to learn!

~ pure composition
Lists ~ 1D data

\[ A = [42, 75, 70] \]

What does \( A \) "look like"?

1D lists are familiar – but lists can hold ANY kind of data – *including lists!*
Lists ~ 1D data

$A = [42, 75, 70]$
2D data as *lists of lists*...

\[
A = \begin{bmatrix}
[1,2,3,4], & [5,6], & [7,8,9,10,11]
\end{bmatrix}
\]

What does *this* *A* "look like"?

Where's 3?

\[
\text{len}(A) \quad \text{len}(A[0]) \quad \text{len}(A[1])
\]

Replace 10 with 42.
2D data as *lists of lists*...

\[ A = \begin{bmatrix} [1,2,3,4], [5,6], [7,8,9,10,11] \end{bmatrix} \]

Where's 3?  
\[ \text{len}(A) \]  
\[ \text{len}(A[0]) \]

Replace 10 with 42.
2D data as *lists of lists*...

\[ A = \begin{bmatrix} [1, 2, 3, 4], & [5, 6], & [7, 8, 9, 10, 11] \end{bmatrix} \]

Where's 3? \[ \text{len}(A) = 3 \]

\[ \text{len}(A[0]) = 4 \]

Replace 10 with 42.
Rectangular 2D data

\[ A = \begin{bmatrix} [0,0,0,0], [0,0,0,0], [0,0,0,0] \end{bmatrix} \]

\[ A[1][2] = 42 \]

\[ A[r][c] = \text{value} \]
Rectangular 2D data

\[ A = \begin{bmatrix} [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] \end{bmatrix} \]

NROWS = len(A)  # HEIGHT
NCOLS = len(A[0])  # WIDTH

for r in range(0, NROWS):
    for c in range(0, NCOLS):
        if r == c:
            A[r][c] = 4
        else:
            A[r][c] = 2

Nested Loops ~ 2d Data
hw9pr1 (lab): *Conway's Game of Life*

**John Conway**

Geometer @ Princeton 1937-2020

simple rules ~ surprising behavior

**The fantastic combinations of John Conway's new solitaire game "life"**

by Martin Gardner

*Scientific American* 223 (October 1970): 120-123.

1970
Lab Problem: *Conway's Game of Life*

**Grid World**

- red cells are "alive"
- white cells are empty

**Evolutionary rules**

- Everything depends on a cell's eight neighbors
  - Exactly 3 neighbors give birth to a new, live cell.
  - Exactly 2 or 3 neighbors keep an existing cell alive.
  - Any other # of neighbors and the central cell dies…

*Only 2 rules*
Lab Problem

"Parent generation"

Grid World

red cells are "alive"

Evolutionary rules

- Everything depends on a cell's eight neighbors
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...
Lab Problem: Conway's Game of Life

Grid World

red cells are "alive"

white cells are empty

Evolutionary rules

- Everything depends on a cell's eight neighbors
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...
Lab Problem: "Grandchild generation"

**Conway's Game of Life**

Grid World

red cells are "alive"

- Everything depends on a cell's eight neighbors.
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...

Evolutionary rules

What's next?
Lab Problem: *Creating life*

`next_life_generation(A)`

old generation is the input, A

returns the next generation

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

?
Lab Problem: *Creating life*

Stable configurations:
- "rocks"
- Periodic "plants" (period 2)
- Self-propagating "animals" (glider)

Copperhead: 2016
Stable configurations:

Periodic
“rocks”
“plants”
“animals”

Life @ HMC?
Life @ HMC!

Life, universally!
Lab Problem: *Creating life*

Many life configurations expand forever...

What is the largest amount of the life universe that can be filled with cells?

How sophisticated can Life-structures get?

www.ibiblio.org/lifepatterns/
Today we'll be

*Lifing it up*

in lab!

so on over...