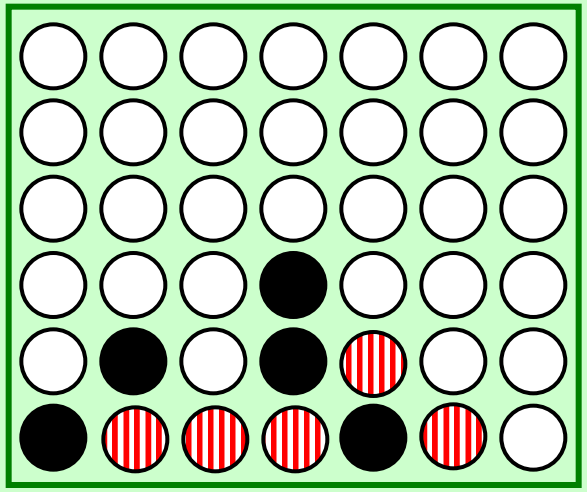
CS 4 this week

Hw #10 due **04/09**

Building classes...

`hw10pr2`

Connect Four **`Board`** class



... vs. *using* classes!

`hw10pr3`

the `file` and `dictionary` classes

files

dictionaries

If I had a dictionary, I guess I could look up what was!

CS 60?

Same as CS5 ← → 2 x the work → Much harder

2+ languages

Racket

```
(define (whoami n)
    (if (
```

```
whoami(n-1);
```

On the subject of classes...

Runtime!

O(N)	O(N^2)	O(2^N)
fast!	medium	slow...

How efficient is **whoami** ?

CS 60?

Same as CS5 ← → 2 x the work ← → Much harder

2+ languages

Racket

```
(define (whoami n)
   (if (= n 0)
     1
     (* n (whoami (- n 1)))))
```

Who is **whoami** ?

Java

```
public static int whoami(int n)
{
   if (n<2) return 1;
   else return n * whoami(n-1);
}
```

Runtime!

$O(N)$	$O(N^2)$	$O(2^N)$
fast!	medium	slow...

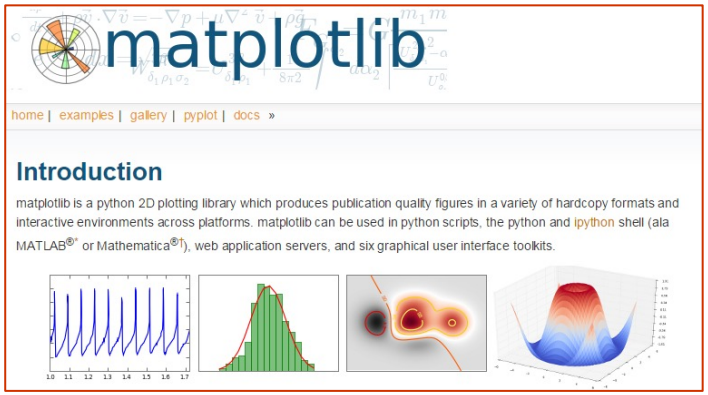
How efficient is **whoami** ?

CS 35?

Same as CS5 ← → Much harder

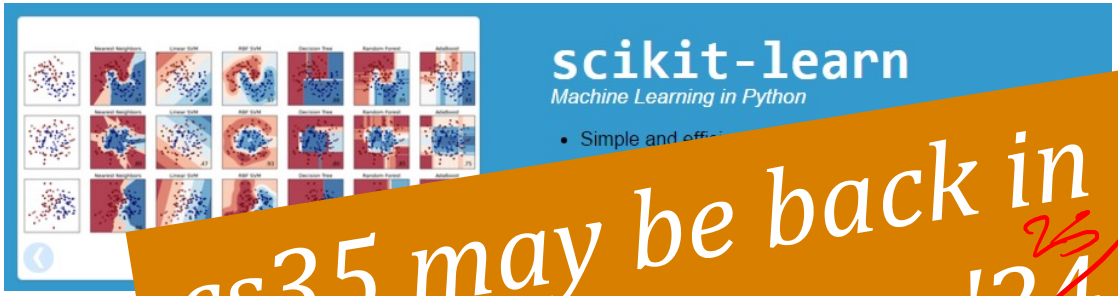
2 x the work

2-3 libraries *each week…*



VPython
3D Programming for Ordinary Mortals

wk11



~~24~~ 25

cs35 may be back in Fall '23 (or Spring '24…)

Classes: DIY data

design-it-yourself!

Class: a (new) datatype

Object: a single instance of a class

OOP

object-oriented programming

... is the organization for most of ...

Python's Libraries

Classes: DIY data

Class: a (new) datatype

Object: a single instance of a class

```
d = Date( 11, 11, 2021 )
d.tomorrow()
print(d)
```

constructor

object

method

uses repr

d will be **self** inside the **Date** class's source code...

Method: a function defined *in a class* called *by an object*

`self`: in a class, the name of the object calling a method

Constructor: the **`__init__`** function for creating a new object

repr: the **`__repr__`** function returning a string to print

data attribute: the data in **`self`**: **`self.day, self.month, self.year`**

Lab **on Friday**

You'll create a **`Date`** class with

```
yesterday(self)  ──►  -= 1
tomorrow(self)   ──►  += 1
addNDays(self, N) ──►  += N
subNDays(self, N) ──►  -= N
isBefore(self, d2) ──►  <
isAfter(self, d2) ──►  >
diff(self, d2)   ──►  -
dow(self)        ──►
```

Prof. Benjamin !

no computer required…

methods

operators!

Standardizing dates was a big project!

Current date and time expressed according to ISO 8601 [refresh]

Date	2022-04-05
Date and time in UTC	2022-04-05T17:09:23+00:00 2022-04-05T17:09:23Z 20220405T170923Z
Week	2022-W14
Week with weekday	2022-W14-2
Date without year	--04-05[1]
Ordinal date	2022-095

Not all years are the same!

Calendar for year 1752 (United States)

<1751 | 1753> | 2007>>

January

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

5:● 12:◐ 19:○ 27:◑

February

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

4:● 11:◐ 18:○ 25:◑

March

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

4:● 11:◐ 18:○ 26:◑

April

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

3:● 9:◐ 17:○ 25:◑

May

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

2:● 9:◐ 16:○ 25:◑ 31:●

June

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

7:◐ 15:○ 23:◑ 30:●

July

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

7:◐ 15:○ 22:◑ 29:●

August

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

5:◐ 13:○ 21:◑ 27:●

September

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

15:◐ 23:○ 30:◑

October

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

7:● 15:◐ 22:○ 29:◑

November

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

5:● 14:◐ 21:○ 27:◑

December

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

5:● 13:◐ 20:○ 27:◑

Calendar for year 1712 (Sweden)

<1711 | 1713> | 2007>>

January

Wno	Mo	Tu	We	Th	Fr	Sa	Su
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31				

5:◐ 13:○ 21:◑ 27:●

February

Wno	Mo	Tu	We	Th	Fr	Sa	Su
5				1	2	3	4
6	5	6	7	8	9	10	11
7	12	13	14	15	16	17	18
8	19	20	21	22	23	24	25
9	26	27	28	29	30		

4:◐ 12:○ 19:◑ 26:●

March

Wno	Mo	Tu	We	Th	Fr	Sa	Su
9					1	2	
10	3	4	5	6	7	8	9
11	10	11	12	13	14	15	16
12	17	18	19	20	21	22	23
13	24	25	26	27	28	29	30
14	31						

4:◐ 12:○ 18:◑ 26:●

April

Wno	Mo	Tu	We	Th	Fr	Sa	Su
14		1	2	3	4	5	6
15	7	8	9	10	11	12	13
16	14	15	16	17	18	19	20
17	21	22	23	24	25	26	27
18	28	29	30				

3:◐ 10:○ 17:◑ 24:●

May

Wno	Mo	Tu	We	Th	Fr	Sa	Su
18				1	2	3	4
19	5	6	7	8	9	10	11
20	12	13	14	15	16	17	18
21	19	20	21	22	23	24	25
22	26	27	28	29	30	31	

2:◐ 9:○ 16:◑ 24:●

June

Wno	Mo	Tu	We	Th	Fr	Sa	Su
22							1
23	2	3	4	5	6	7	8
24	9	10	11	12	13	14	15
25	16	17	18	19	20	21	22
26	23	24	25	26	27	28	29
27	30						

1:◐ 8:○ 14:◑ 22:● 30:◐

July

Wno	Mo	Tu	We	Th	Fr	Sa	Su
27		1	2	3	4	5	6
28	7	8	9	10	11	12	13
29	14	15	16	17	18	19	20
30	21	22	23	24	25	26	27
31	28	29	30	31			

7:○ 14:◑ 22:● 30:◐

August

Wno	Mo	Tu	We	Th	Fr	Sa	Su
31					1	2	3
32	4	5	6	7	8	9	10
33	11	12	13	14	15	16	17
34	18	19	20	21	22	23	24
35	25	26	27	28	29	30	31

5:○ 13:◑ 21:● 28:◐

September

Wno	Mo	Tu	We	Th	Fr	Sa	Su
36	1	2	3	4	5	6	7
37	8	9	10	11	12	13	14
38	15	16	17	18	19	20	21
39	22	23	24	25	26	27	28
40	29	30					

4:○ 11:◑ 19:● 26:◐

October

Wno	Mo	Tu	We	Th	Fr	Sa	Su
40		1	2	3	4	5	
41	6	7	8	9	10	11	12
42	13	14	15	16	17	18	19
43	20	21	22	23	24	25	26
44	27	28	29	30	31		

3:○ 11:◑ 19:● 25:◐

November

Wno	Mo	Tu	We	Th	Fr	Sa	Su
44						1	2
45	3	4	5	6	7	8	9
46	10	11	12	13	14	15	16
47	17	18	19	20	21	22	23
48	24	25	26	27	28	29	30

2:○ 10:◑ 17:● 24:◐

December

Wno	Mo	Tu	We	Th	Fr	Sa	Su
49	1	2	3	4	5	6	7
50	8	9	10	11	12	13	14
51	15	16	17	18	19	20	21
52	22	23	24	25	26	27	28
1	29	30	31				

2:○ 10:◑ 17:● 23:◐ 31:○

Not all years are the same!

Calendar for year 1752 (United States)

<1751 | 1753> | 2007>>

January

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

5:● 12:◐ 19:○ 27:◑

February

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

4:● 11:◐ 18:○ 25:◑

March

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

4:● 11:◐ 18:○ 26:◑

April

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

3:● 9:◐ 17:○ 25:◑

May

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

2:● 9:◐ 16:○ 25:◑ 31:●

June

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

7:◐ 15:○ 23:◑ 30:●

July

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

7:◐ 15:○ 22:◑ 29:●

August

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

5:◐ 13:○ 21:◑ 27:●

September

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

15:◐ 23:○ 30

October

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

7:● 15:◐ 22:○ 29:◑

November

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

5:● 14:◐ 21:○ 27:◑

December

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

5:● 13:◐ 20:○ 27:◑

Calendar for year 1712 (Sweden)

<1711 | 1713> | 2007>>

January

Wno	Mo	Tu	We	Th	Fr	Sa	Su
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31				

5:◐ 13:○ 21:◑ 27:●

February

Wno	Mo	Tu	We	Th	Fr	Sa	Su
5				1	2	3	4
6	5	6	7	8	9	10	11
7	12	13	14	15	16	17	18
8	19	20	21	22	23	24	25
9	26	27	28	29	30		

4:◐ 12:○ 19:◑ 26:●

March

Wno	Mo	Tu	We	Th	Fr	Sa	Su
9						1	2
10	3	4	5	6	7	8	9
11	10	11	12	13	14	15	16
12	17	18	19	20	21	22	23
13	24	25	26	27	28	29	30
14	31						

12:○ 18:◑ 26:●

April

Wno	Mo	Tu	We	Th	Fr	Sa	Su
14		1	2	3	4	5	6
15	7	8	9	10	11	12	13
16	14	15	16	17	18	19	20
17	21	22	23	24	25	26	27
18	28	29	30				

3:◐ 10:○ 17:◑ 24:●

May

Wno	Mo	Tu	We	Th	Fr	Sa	Su
18				1	2	3	4
19	5	6	7	8	9	10	11
20	12	13	14	15	16	17	18
21	19	20	21	22	23	24	25
22	26	27	28	29	30	31	

2:◐ 9:○ 16:◑ 24:●

June

Wno	Mo	Tu	We	Th	Fr	Sa	Su
22							1
23	2	3	4	5	6	7	8
24	9	10	11	12	13	14	15
25	16	17	18	19	20	21	22
26	23	24	25	26	27	28	29
27	30						

1:◐ 8:○ 14:◑ 22:● 30:◐

July

Wno	Mo	Tu	We	Th	Fr	Sa	Su
27		1	2	3	4	5	6
28	7	8	9	10	11	12	13
29	14	15	16	17	18	19	20
30	21	22	23	24	25	26	27
31	28	29	30	31			

7:○ 14:◑ 22:● 30:◐

August

Wno	Mo	Tu	We	Th	Fr	Sa	Su
31					1	2	3
32	4	5	6	7	8	9	10
33	11	12	13	14	15	16	17
34	18	19	20	21	22	23	24
35	25	26	27	28	29	30	31

5:○ 13:◑ 21:● 28:◐

September

Wno	Mo	Tu	We	Th	Fr	Sa	Su
36	1	2	3	4	5	6	7
37	8	9	10	11	12	13	14
38	15	16	17	18	19	20	21
39	22	23	24	25	26	27	28
40	29	30					

4:○ 11:◑ 19:● 26:◐

October

Wno	Mo	Tu	We	Th	Fr	Sa	Su
40			1	2	3	4	5
41	6	7	8	9	10	11	12
42	13	14	15	16	17	18	19
43	20	21	22	23	24	25	26
44	27	28	29	30	31		

3:○ 11:◑ 19:● 25:◐

November

Wno	Mo	Tu	We	Th	Fr	Sa	Su
44						1	2
45	3	4	5	6	7	8	9
46	10	11	12	13	14	15	16
47	17	18	19	20	21	22	23
48	24	25	26	27	28	29	30

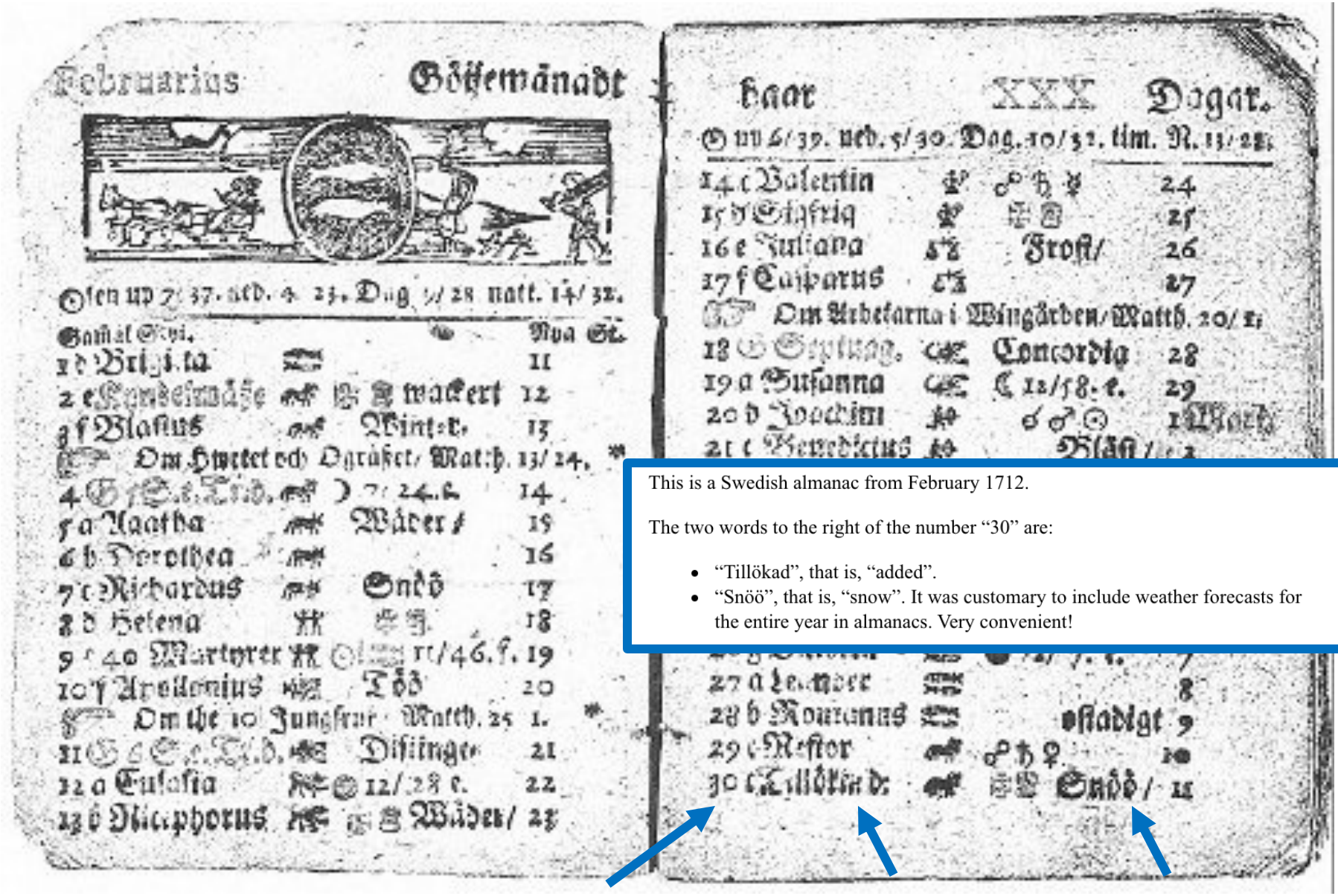
2:○ 10:◑ 17:● 24:◐

December

Wno	Mo	Tu	We	Th	Fr	Sa	Su
49	1	2	3	4	5	6	7
50	8	9	10	11	12	13	14
51	15	16	17	18	19	20	21
52	22	23	24	25	26	27	28
1	29	30	31				

2:○ 10:◑ 17:● 23:◐ 31:○

Feb. 30, 1712



This is a Swedish almanac from February 1712.

The two words to the right of the number "30" are:

- "Tillökad", that is, "added".
- "Snöö", that is, "snow". It was customary to include weather forecasts for the entire year in almanacs. Very convenient!

https://www.tondering.dk/claus/cal/gregorian.php

__eq__

```
class Date:

    def __init__( self, mo, dy, yr ):
    def __repr__(self):
    def isLeapYear(self):

    def __eq__(self, d2):
        """ returns True if they
            represent the same date;
            False otherwise
        """
        if self.year == d2.year and \
           self.month == d2.month and \
           self.day == d2.day:
                return True
        else:
                return False
```

Not recursion?!
Perish the thought!

redefined for *our convenience*!

To use this, write `d == d2`

isBefore

(with bugs!)

```
class Date:

    def isBefore(self, d2):
        """ True if self is before d2, else False """
        if self.year  < d2.year:
            return True
        elif self.month < d2.month:
            return True
        elif self.day   < d2.day:
            return True
        else: return False
```

```
Date(11,9,2021).isBefore(Date(12,31,1999))
```

No wonder I was late to all my millenium parties!

isBefore

(correct)

```
class Date:

    def isBefore(self, d2):
        """ True if self is before d2, else False """
        if self.year  < d2.year:
            return True

        elif self.month < d2.month :
            return True

        elif self.day < d2.day :
            return True

        else:
            return False
```

I <3 Elf! But what about Elif?

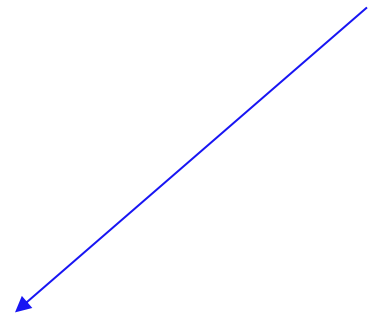
isBefore

(correct)

```
class Date:

    def isBefore(self, d2):
        """ True if self is before d2, else False """
        if self.year  < d2.year:
            return True

        elif self.month < d2.month and self.year == d2.year :
            return True

        elif self.day < d2.day and self.year == d2.year \
                               and self.month == d2.month :
            return True

        else:
            return False
```

I <3 Elf! But what about Elif?

__lt__ <



```
class Date:

    def __lt__(self, d2):
        """ if self is before d2, this should
            return True; else False """

        if self.isBefore(d2) == True:
            return True
        else:
            return False
```

*I want **LESS** !*

__lt__

<


```
class Date:

    def __lt__(self, d2):
        """ is self less than d2? (before) """
        return self.isBefore(d2)
```

__lt__ <

```
class Date:

    def __lt__(self, d2):
        """ is self less than d2? (before) """
        return self.isBefore(d2)
```

__gt__ >

```
    def __gt__(self, d2):
        """ is self greater than d2? (after) """
        return d2.isBefore(self)
```

LESS really is MORE!

The two *most essential methods*

```
In1: wd = Date(2,22,2022)
In2: print(wd)
2/22/2022
```

construct with the **CONSTRUCTOR** …

print uses **__repr__**

```
In1: wd.tomorrow()
In2: print(wd)
2/23/2022
```

the **tomorrow** method returns nothing at all. Is it doing anything?

wd has changed!

```
d += 1
```

```
In1: wd.yesterday()
In2: print(wd)
2/22/2022
```

yesterday is pretty much just like **tomorrow** (is this a good thing!?)

```
d -= 1
```

yesterday does not return anything!
But, it does *change* the date that calls it ("self")

```
class Date:
```

Try writing tomorrow!

Use this for lab!

```
    def tomorrow(self):
        """ moves the self date ahead 1 day """

        DIM = [0,31,28,31,30,31,30,31,31,30,31,30,31]

        self.day += 1
```

first, add 1 to `self.day`

DIM looks pretty bright to me!

```
        if self.day > DIM[self.month]:
```

test if we have gone "out of bounds!"

```
            self.day = 1
            self.month += 1
            if self.month > 12:
                self.year += 1
                self.month = 1
```

then, adjust the month and year, but only as needed Use another if!

Don't return anything. We **CHANGE** the date object itself.

Extra How could we make this work for leap years, too?


```
class Date:

  def tomorrow(self):
    """ moves the self date ahead 1 day """
```

better as a *variable*!

```
    DIM = [0,31,fdays,31,30,31,30,31,31,30,31,30,31]

    self.day += 1       # add 1 to the day!

    if self.day > DIM[self.month]:    # check day
        self.month += 1
        self.day = 1

        if self.month > 12:           # check month
            self.year += 1
            self.month = 1
```

```
class Date:

  def tomorrow(self):
    """ moves the self date ahead 1 day """

    if self.isLeapYear() == True:    fdays = 29
    else: fdays = 28

    DIM = [0,31,fdays,31,30,31,30,31,31,30,31,30,31]

    self.day += 1      # add 1 to the day!

    if self.day > DIM[self.month]:    # check day
        self.month += 1
        self.day = 1

        if self.month > 12:           # check month
            self.year += 1
            self.month = 1
```

```
class Date:

  def tomorrow(self):
    """ moves the self date ahead 1 day """

    fdays = 28 + self.isLeapYear()      # What ?!

    DIM = [0,31,fdays,31,30,31,30,31,31,30,31,30,31]

    self.day += 1       # add 1 to the day!

    if self.day > DIM[self.month]:    # check day
        self.month += 1
        self.day = 1

        if self.month > 12:           # check month
            self.year += 1
            self.month = 1
```

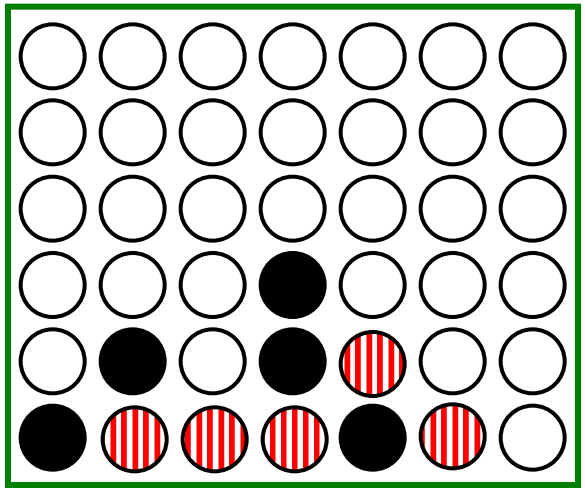
Use for lab!

```
class Date:

  def yesterday(self):
    """ moves the self date backwards 1 day """

    fdays = 28 + self.isLeapYear()     # Yay!

    DIM = [0,31,fdays,31,30,31,30,31,31,30,31,30,31]

    self.day
```

For lab: how will "wrap-around" work in this case? *What cases do we need to worry about?!*

Why classes?

Python has no Connect-four datatype…



```
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |X |  |  |  |
|  |X |  |X |O |  |  |
|X |O |O |O |X |  |O |
-----------------------
 0  1  2  3  4  5  6
```

b2

Care for a game?

… now we can fix that!

Data design…

(**Data Members**) What data do we need?

Attributes

(Fields)

(**Methods**) What capabilities do we want?

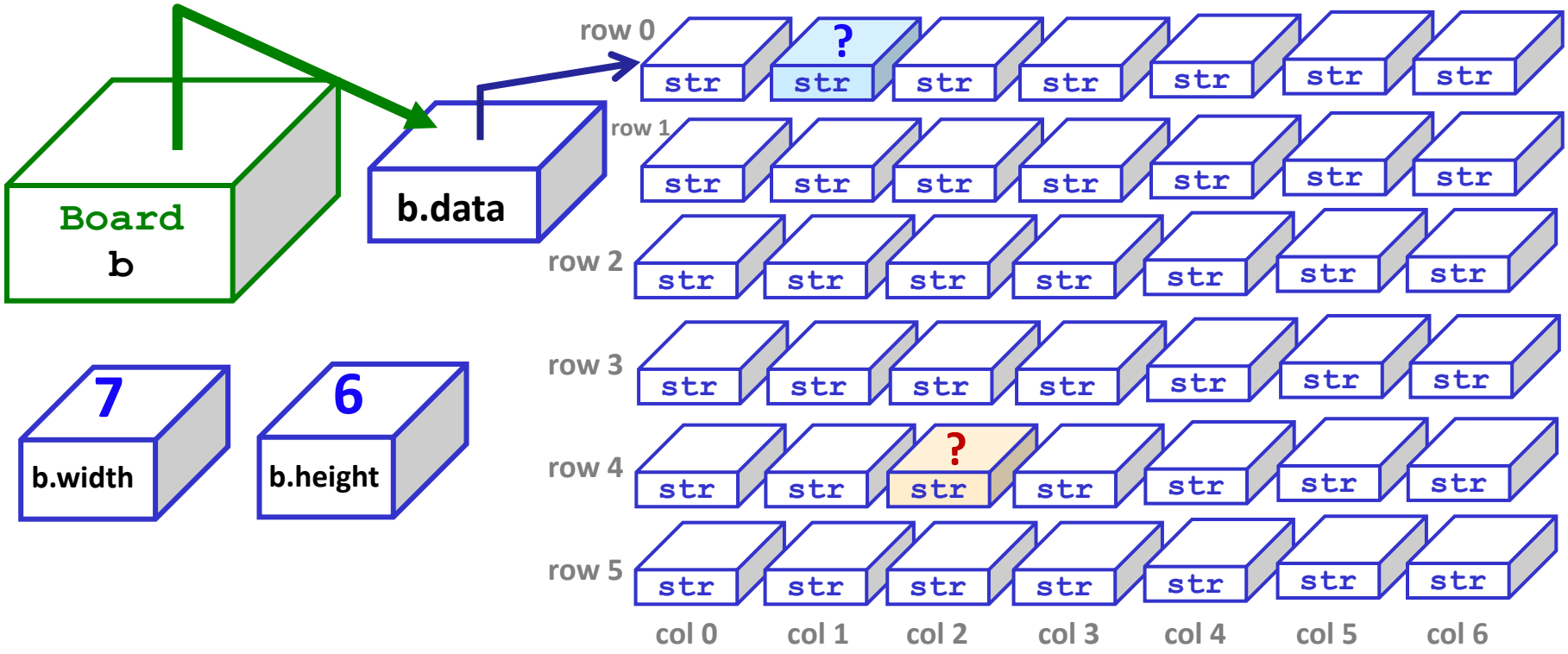
Our Board object, **b**

b.data[0][1] = 'X'

b.data[4][2] = 'O'

How could we set **?** to **'X'** and **?** to **'O'**

Our Board object, `b`

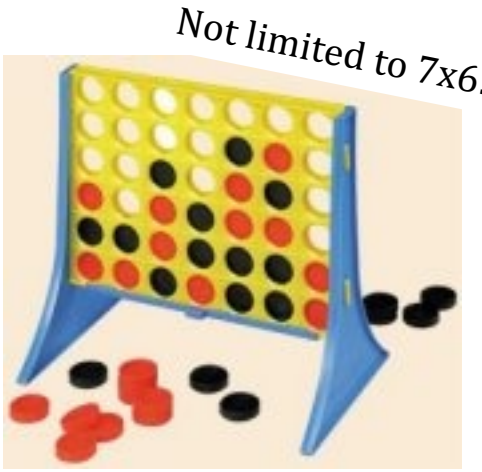



```
b.data[0][1] = 'X'
```

```
b.data[4][2] = 'O'
```

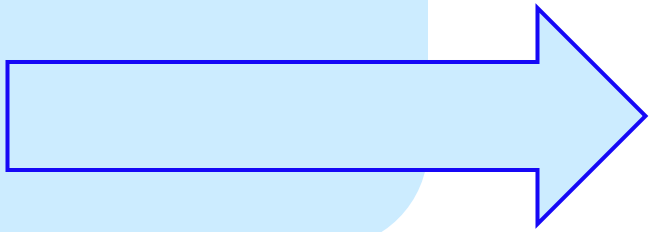
How could we set ? to **'X'** and ? to **'O'**

Data design…



(**Data Members**) What data do we need?

(**Methods**) What capabilities do we want?



__init__

the "constructor"

```
class Board:
    """ a datatype representing a C4 board
        with an arbitrary number of rows and cols
    """

    def __init__( self, width, height ):
        """ the constructor for objects of type Board """
        self.width = width
        self.height = height
        W = self.width
        H = self.height
        self.data = [ [' ']*W for row in range(H) ]
```

For lab, use the starter code—scroll down in the homework writeup!

I <3 LCs!

Maybe here it should be I <4 LCs?

__init__

the "constructor"

```
class Board:
    """ a datatype representing a C4 board
        with an arbitrary number of rows and cols
    """

    def __init__( self, width, height ):
        """ the constructor for objects of type Board """
        self.width = width
        self.height = height
        W = self.width
        H = self.height
        self.data = [ [' ']*W for row in range(H) ]
```

Convenient but optional!

A space, *not* an empty string!

This list comprehension lets us create **H** independent rows with **W** independent columns each.

I <3 LCs!

Maybe here it should be I <4 LCs?

__repr__

the "repr"

```
def __repr__(self):
    """ this method returns a string representation
        for an object of type Board
    """
    H = self.height
    W = self.width
    s = ''
    for r in range( H ):
        s += '|'
        for c in range( W ):
            s += self.data[r][c] + '|'
        s += '\n'

    s += (2*W+1)*'-'

    # what kind of loop will add the col #'s here?

    return s
```

r in [0,1,2,3,4,5]

c in [0,1,2,3,4,5,6]

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X|O| | |
|X|O|O|O|X| |O|
---------------
 0 1 2 3 4 5 6
```

Name(s) _____

"Quiz"

```
class Board:

  def addMove(self, col, ox):
      """ buggy version! """
      H = self.height

      for row in range(0,H):
          if self.data[row][col] != ' ':
              self.data[row-1][col] = ox
              return
```

a C4 board

col #

'X' or 'O'

shortcut

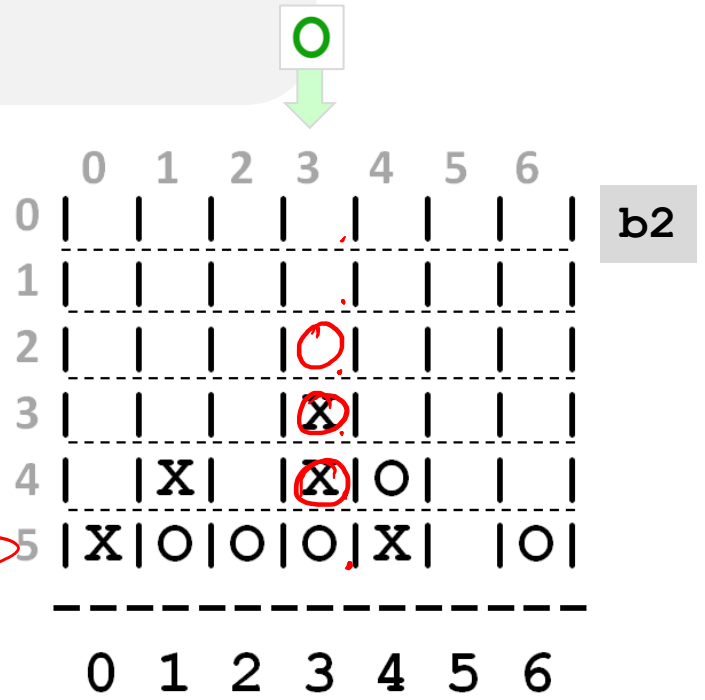
row: [0,1,2,3,4,5]

self.data[H-1][col] = ox

(1) Run **b2.addMove(3,'O')**

Draw how the C4 board will now look...

Warning: Three board spaces will change!!!



(2) ***There are 2 bugs in the code above...***

Can you find/fix them?!

b2.addMoveBUG(3,'O')

Quiz

Try on the back page first!

```
class Board:

    def addMove(self, col, ox):
        """ correct version! """
        H = self.height
        for row in range(0,H):
            if self.data[row][col] != ' ':
                self.data[row-1][col] = ox
                return

        self.data[H-1][col] = ox
```

a C4 board

col #

'X' or 'O'

shortcut

row: [0,1,2,3,4,5]

stop the loop! stop the function!

it gets here ***only*** when the column is empty – this line sets the _bottom_ spot to the correct checker

(1) Run `b2.addMove(3,'O')`

Draw how the C4 board will now look...

Warning: Three board spaces will change!!!

b

	0	1	2	3	4	5	6
0							
1							
2							
3				X			
4		X		X	O		
5	X	O	O	O	X		O
	0	1	2	3	4	5	6

(2) ***There are 2 bugs in the code above...***

Can you find/fix them?!

Try this on the other page first...

b2.addMoveBUG(3,'O')

Quiz

Try on the back page first!

```
class Board:

    def addMove(self, col, ox):
        """ correct version! """
        H = self.height
        for row in range(0,H):
            if self.data[row][col] != ' ':
                self.data[row-1][col] = ox
                return
```

a C4 board

col #

'x' or 'o'

shortcut

row: [0,1,2,3,4,5]

stop the loop!
stop the function!

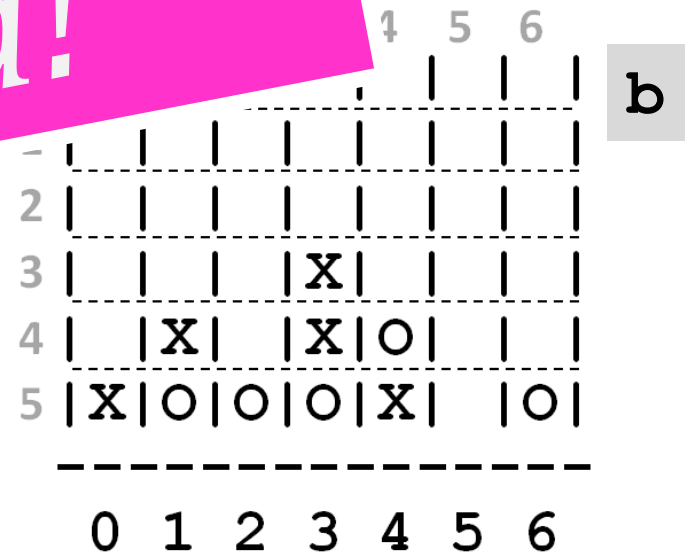
the column is he _bottom_ checker

Pass those Northward!

There are 2 bugs in the code above...

Can you find/fix them?!

Try this on the other page first...



b2.addMoveBUG(3,'O')

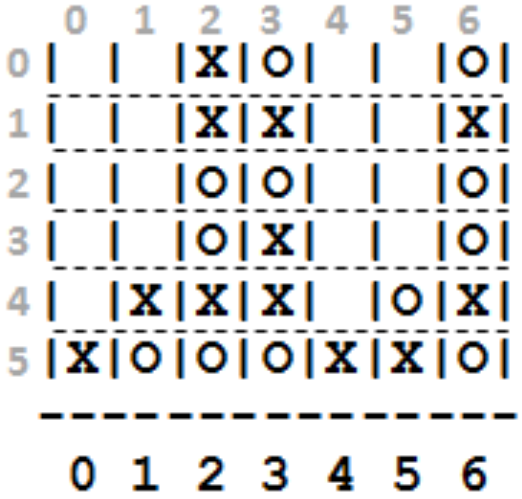
Let's understand the **allowsMove** method ...

```
class Board:

    def allowsMove(self, col):
        """ True if col is in-bounds + open
            False otherwise """
```

a C4 board

col #

b3.allowsMove



b3

```
b.allowsMove(0) == True
b.allowsMove(1) == True
b.allowsMove(2) == False
b.allowsMove(3) == False
b.allowsMove(4) == True
b.allowsMove(5) == True
b.allowsMove(6) == False
b.allowsMove(7) == False
```

If `col` is ***out-of-bounds*** or ***full***, return **False**.

If `col` is *in-bounds and not full*, return **True**.

Let's *finish* this **allowsMove** method ...

a C4 board

col #

```
class Board:

    def allowsMove(self, col):
        """ True if col is in-bounds + open
            False otherwise """
        H = self.height
        W = self.width
        D = self.data

        if col >= W or col < 0:
            return False

        elif D[0][col] != ' ':
            return False
        else:
            return True
```

shortcuts!

out of bounds?

col full?

Allowed!

If **col** is ***out-of-bounds*** or ***full***, return **False**.

If **col** is *in-bounds and not full*, return **True**.

b3.allowsMove

b3

```
  0 1 2 3 4 5 6
0|   |   |X|O|   |   |O|
1|   |   |X|X|   |   |X|
2|   |   |O|O|   |   |O|
3|   |   |O|X|   |   |O|
4|   |X|X|X|   |O|X|
5|X|O|O|O|X|X|O|
  0 1 2 3 4 5 6
```

```
b.allowsMove(0) == True
b.allowsMove(1) == True
b.allowsMove(2) == False
b.allowsMove(3) == False
b.allowsMove(4) == True
b.allowsMove(5) == True
b.allowsMove(6) == False
b.allowsMove(7) == False
```

hw10pr2: **Board** class

✓ the "constructor"	`__init__( self, width, height )`
✓ checks if allowed	`allowsMove( self, col )`
✓ places a checker	`addMove( self, col, ox )`
removes a checker	`delMove( self, col )` ← to write...
✓ outputs a string	`__repr__( self )`
checks if any space is left	`isFull( self )` ← to write...
checks if a player has won	`winsFor( self, ox )` ← to write...
the game...	`hostGame( self )` ← to write...

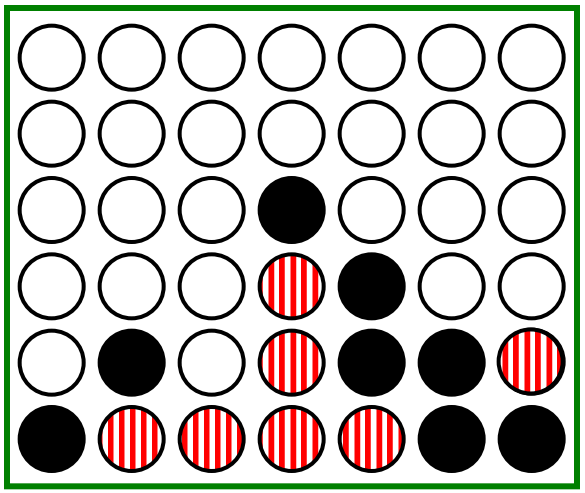
Which are similar to others?

Which requires the most thought?

winsFor(self, ox)

X O

b4



```
b.winsFor('X')
```

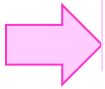
or 'O'

```
def winsFor(self, ox):
    """ does ox win? """
    H = self.height
    W = self.width
    D = self.data

    for row in range(
      for col in range(
            if
            if
            if
            if
```

Is this familiar?!

```
>>> b4.winsFor( 'X' )
True
>>> b4.winsFor( 0 )
False
>>> b4.winsFor( '0' )
False
>>> b4.winsFor( 'O' )
True
```

Watch out for *corner and edge cases*!



Why objects and classes?

Simple – and *INVITING* --
building blocks!

```
if b.winsFor( 'X' ) == True:
```

Elegance: Objects *hide complexity!*

```
dow = self.diff( wd ) % 7
```

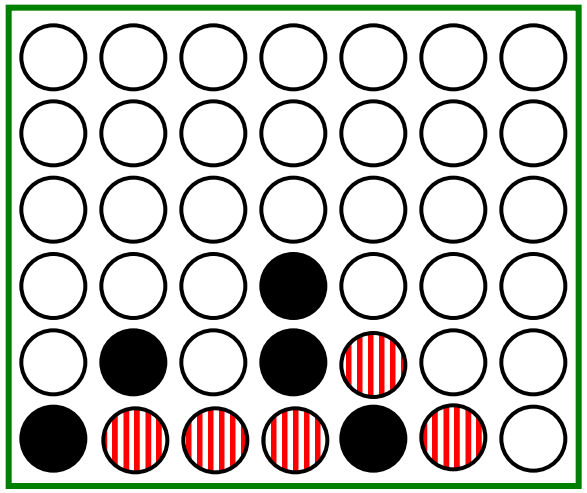
CS 4 this week

Hw #10 due **11/14**

Building classes...

`hw10pr2`

Connect Four **Board** class



... vs. using the library

`hw10pr3`

files and the dictionary class

files

dictionaries

Office hours ~ ***FRIDAY*** aft. in LAC

If I had a dictionary, I guess I could look up what it was!

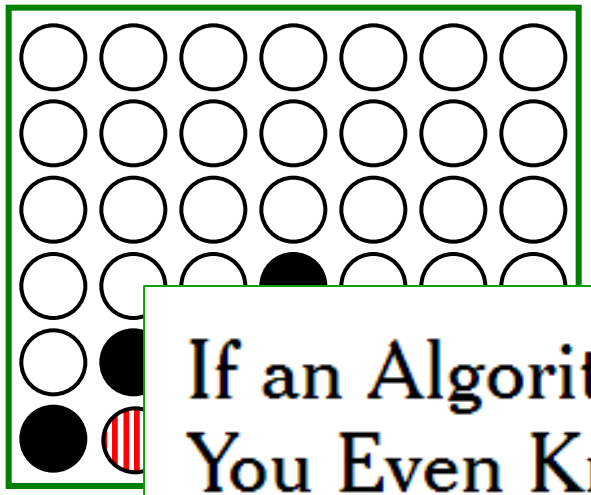
CS 4 this week

Hw #10 due **11/14**

Building classes...

hw10pr2

Connect Four **Board** class



... vs. using the library

hw10pr3

files and the dictionary class

If an Algorithm Wrote This, How Would You Even Know?

By SHELLEY PODOLNY MARCH 7, 2015

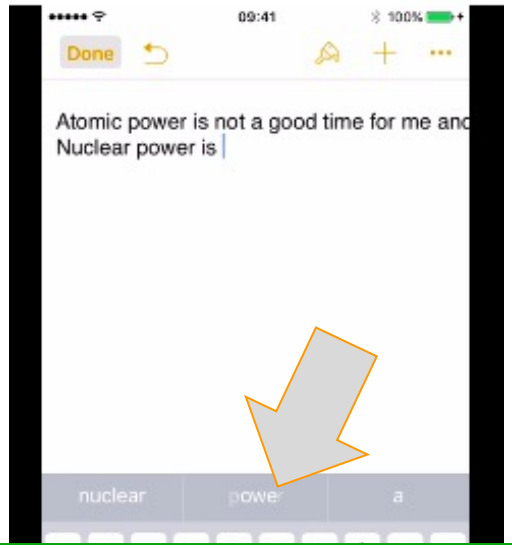
282

Office hou

iOS Just Got A Paper On Nuclear Physics Accepted At A Scientific Conference

Posted by Christoph Bartneck on Oct 20, 2016 in Featured, Research | 7 comments

Automatically generating scientific articles has become easy with dedicated software such as SCIgen. Even a paper that only repeated the sentence "Get me of your fucking mailing list" was recently accepted for publication. Today I received an invitation from the International Confere Atomic and Nuclear Physics to submit a paper. Since I have practically no knowledge of Nuclear Physics I resorted to iOS auto-complete function to help me writing the paper. I started a sentence with "Atomic" or "Nuclear" and then randomly hit the auto-complete sug The text really does not make any sense. After adding the first illustration on nuclear phy n Wikipedia, some references and creating a fake identity (Iris Pear, aka Siri that iOS is a pretty



UPDATE (27/10/2

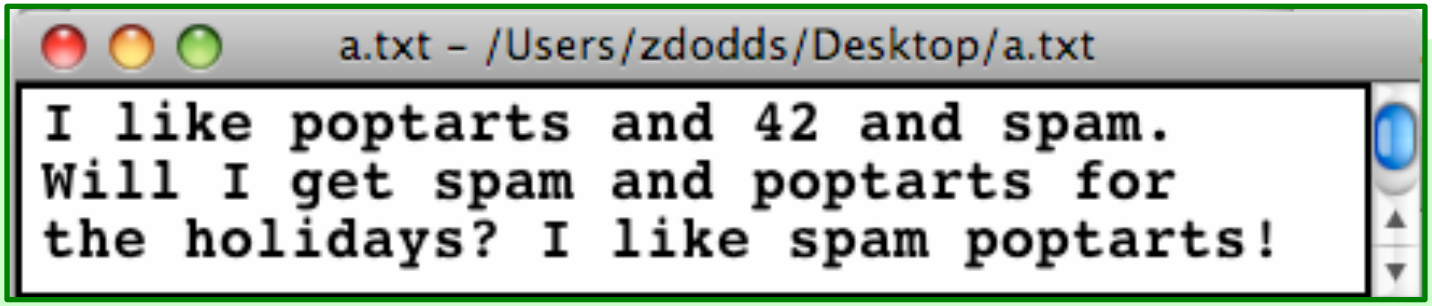
investigation.

If an Algorithm Wrote This, How Would You Even Know?

By SHELLEY PODOLNY MARCH 7, 2015

http://www.bartneck.de/2016/10/20/ios-just-got-a-paper-on-nuclear-physics-accepted-at-a-scientific-conference/

Algorithmic Authorship... ?

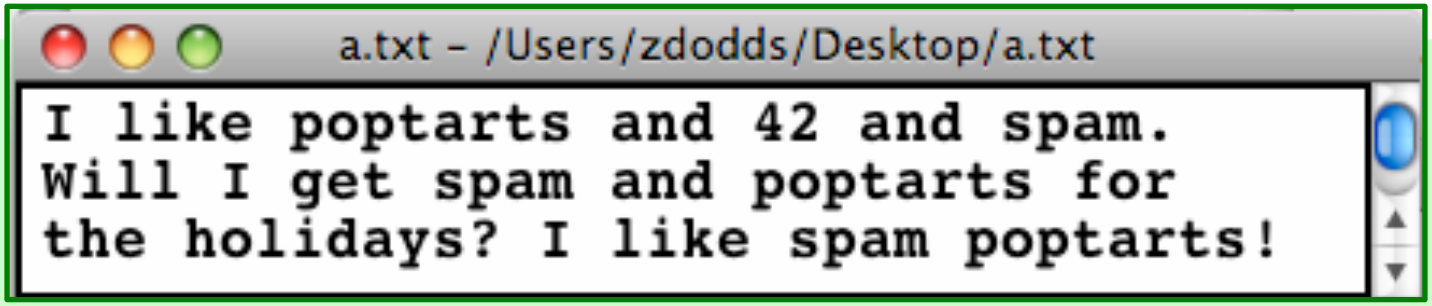


suppose this text represents my "style" ...

How could a *program* author new prose in this "style"?!

"Style" seems like the wrong word here...

Algorithmic Authorship... ?



suppose this text represents my "style" ...

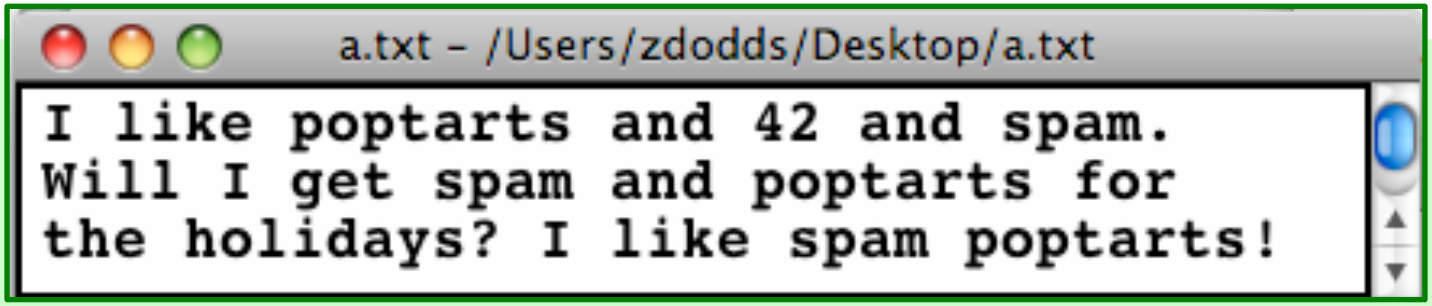
```
I like spam and 42 and poptarts and
poptarts and poptarts and 42 and spam.
```

```
I get spam for 42 poptarts! Will I spam
the holidays?
```

Which of these has more "characteristic" style?

Why?

Algorithmic Authorship... !



suppose this text represents my "style" ...

```
I like spam and 42 and poptarts and
poptarts and poptarts and 42 and spam.
```

How are **first words** chosen to start new sentences?

What are the **next words** that follow follow each?

What would be a reasonable **test for sentence-ending**?