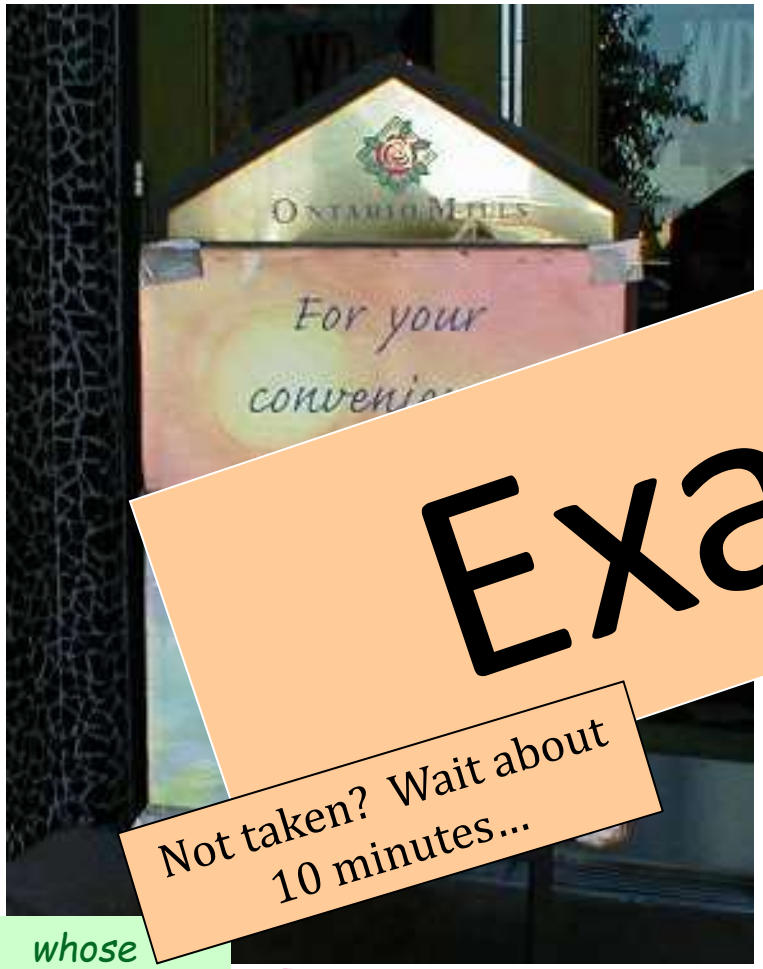
A whole new **`class`** of programming



whose convenience?

Not taken? Wait about 10 minutes...

Exams...

September						
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					

CS's building blocks: functions and composition

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

...s
...uits,
...ops

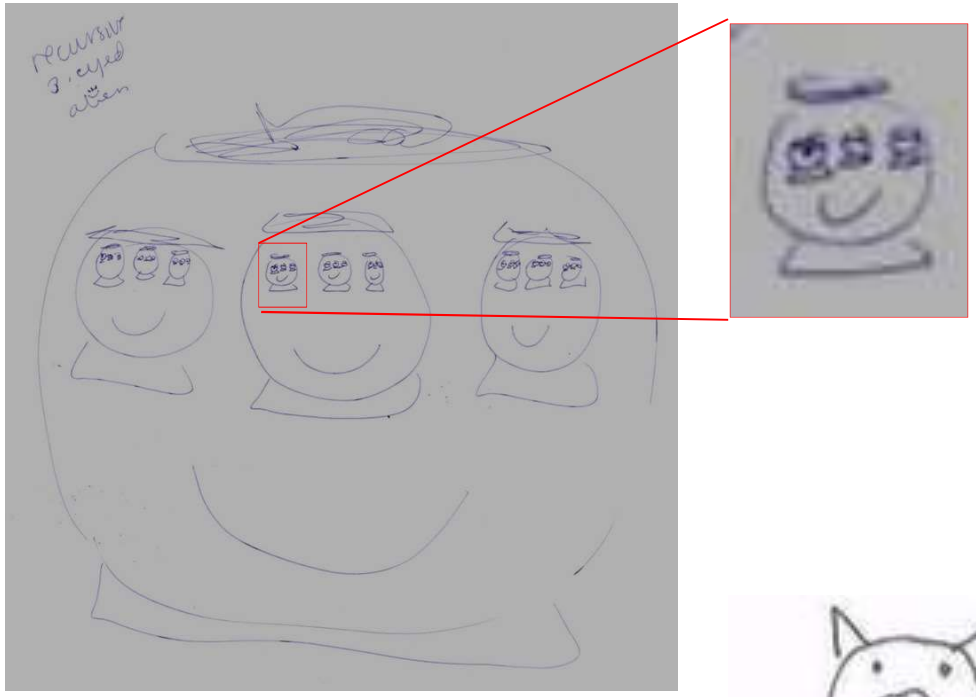
Designing Data!

The **Date** class

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				

Solutions... (*keep within this term's CS5*)

Exams...

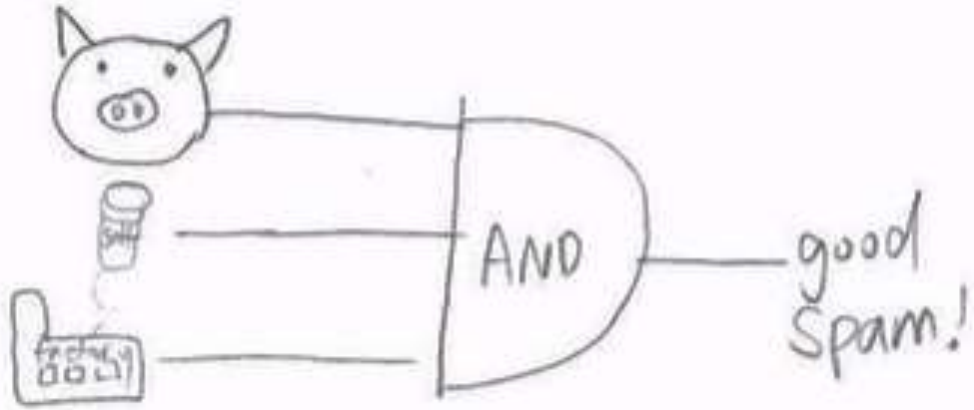


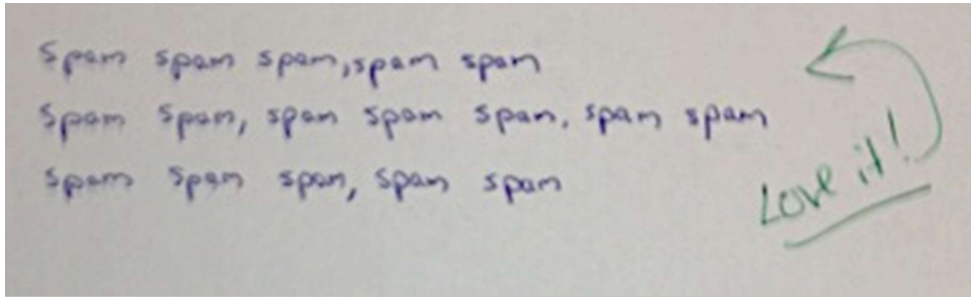
If you're upset
There is no need to fret
All these topics blend together in your he
just go to bed
if ~~grades~~ CS61 is just a wash
just remember
pass/fail frosh ← pass/fail!! ☺

(up to +4.2 points, give or take a bit…) +4.2

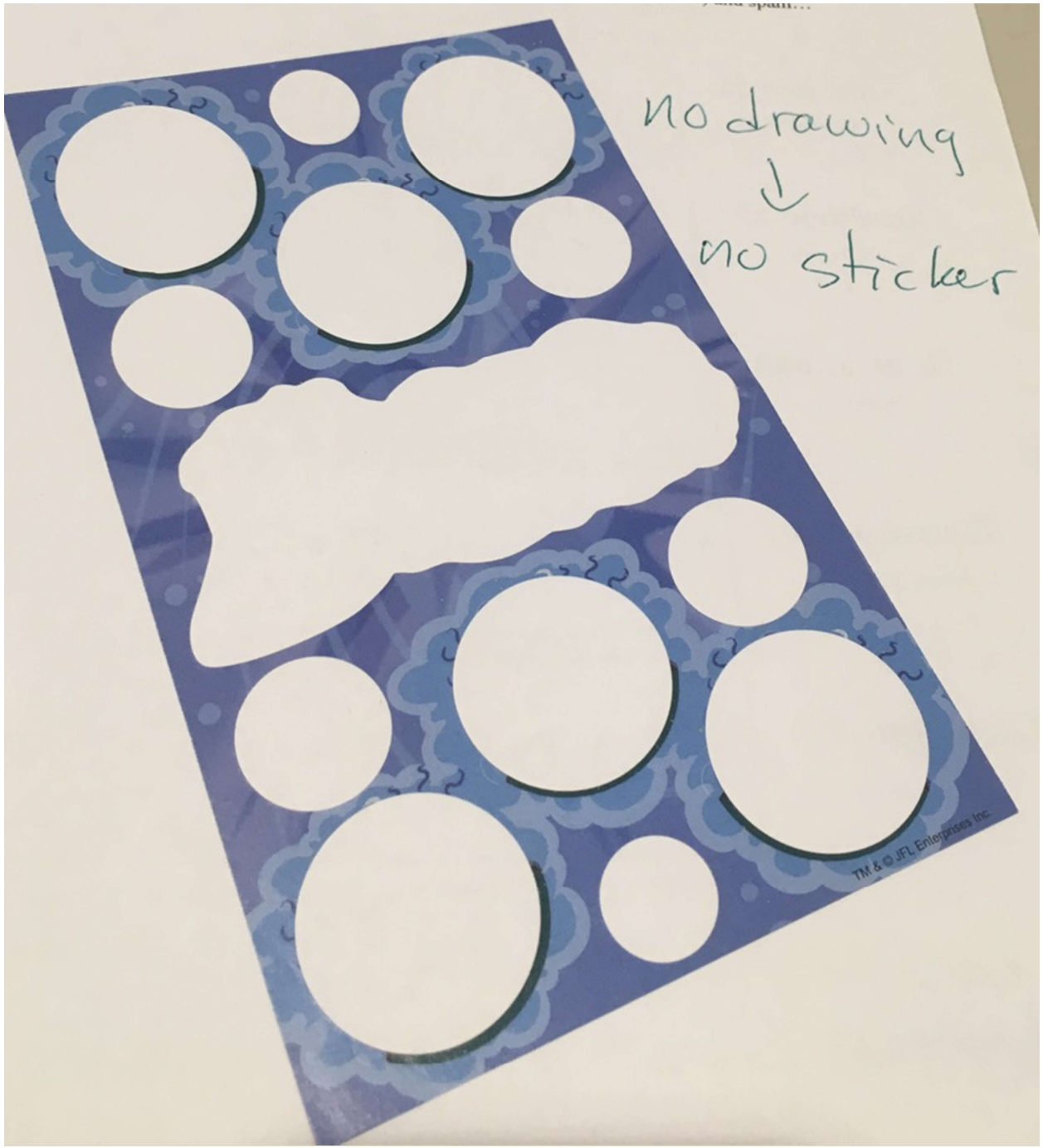
Create a sketch, poem, or other artistic rendering that captu
recursion, circuits, assembly language, three-eyed aliens, lo

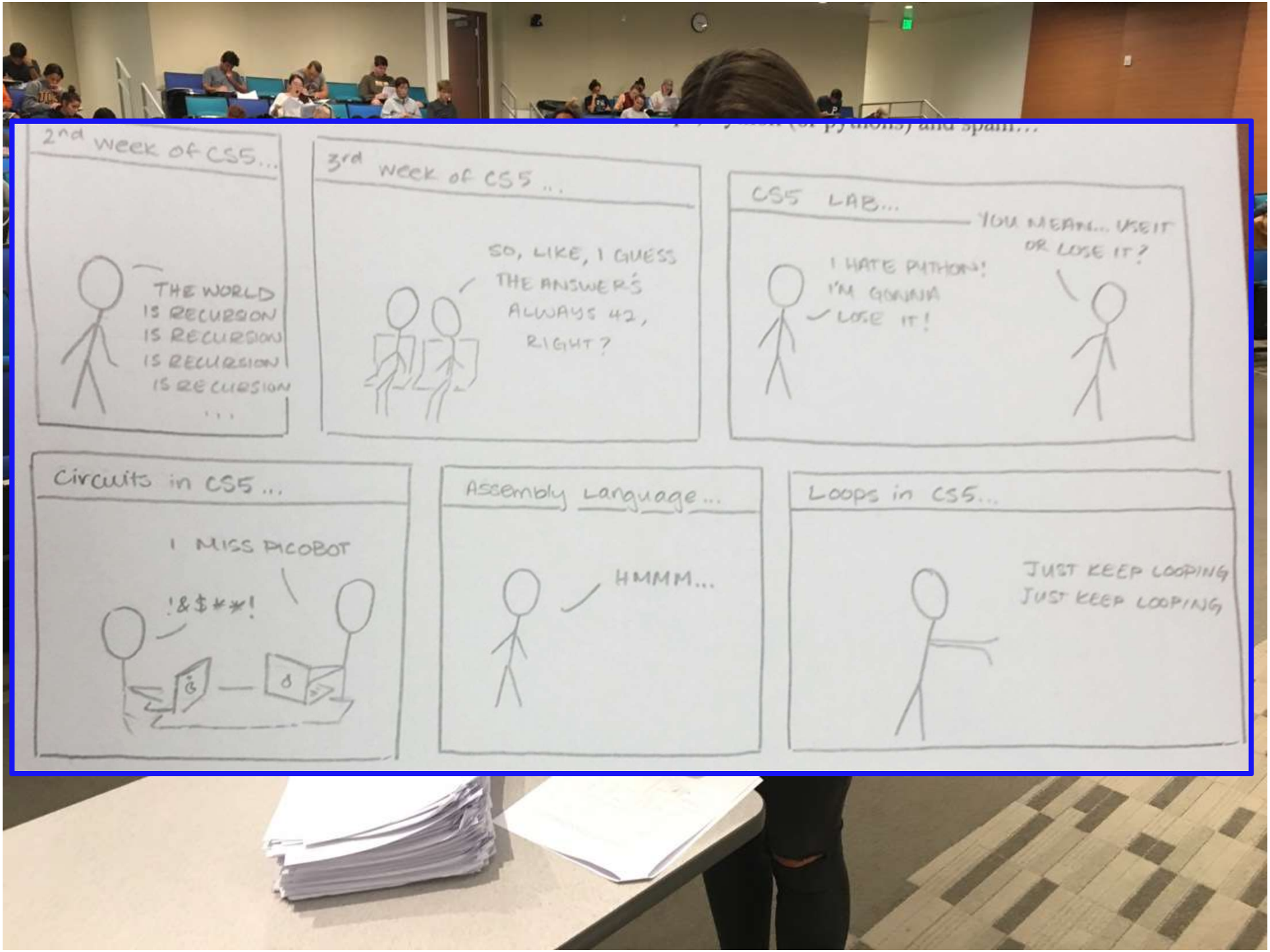
Roses are red
Violets are blue
Please give me full credit
My favorite number is forty-two

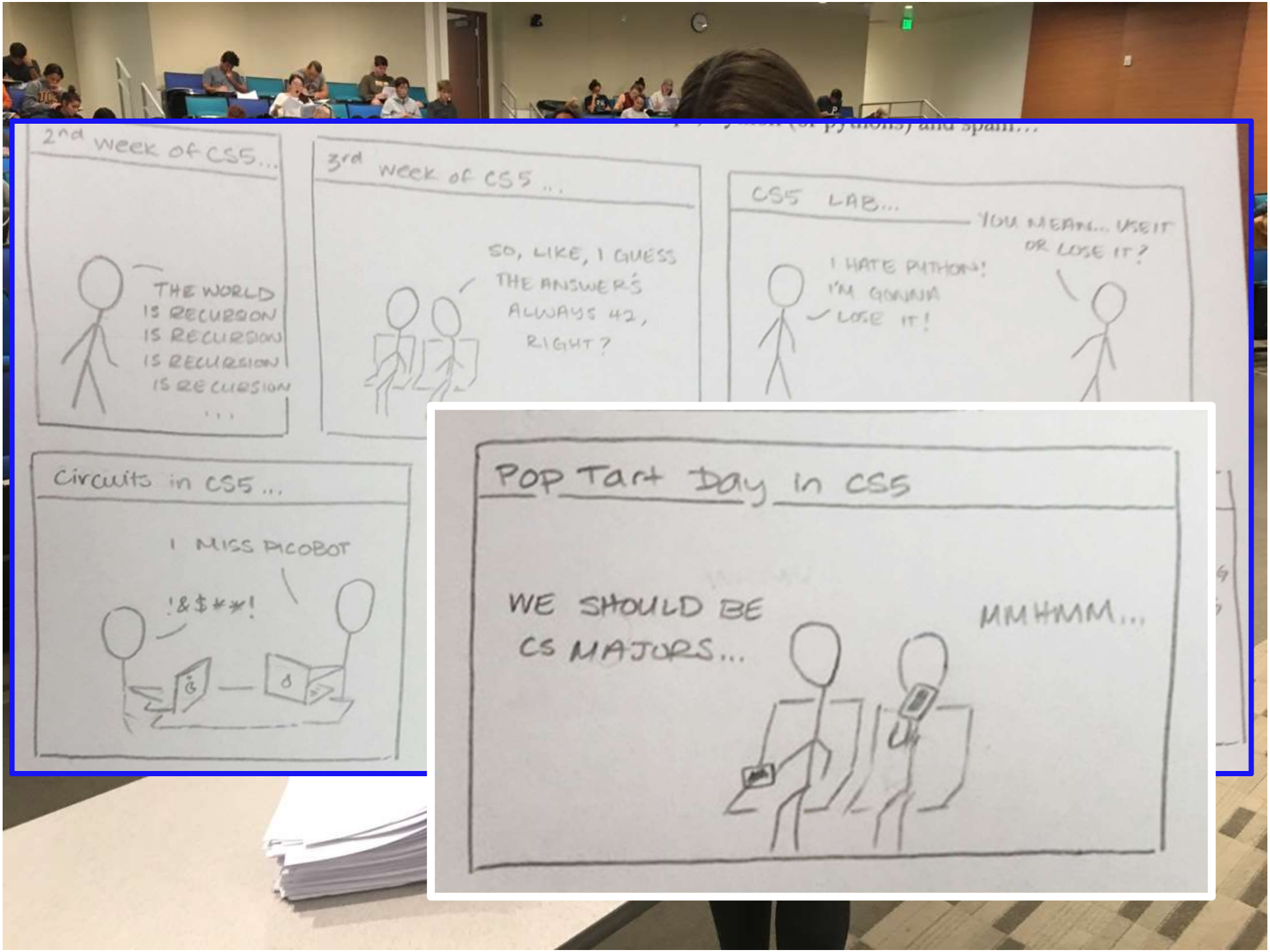




Concerns? Let me know!


no drawing
↓
no sticker
TM & © JFL Enterprises Inc.


2nd week of CS5...
THE WORLD IS RECURSION IS RECURSION IS RECURSION IS RECURSION ...
3rd week of CS5...
SO, LIKE, I GUESS THE ANSWER'S ALWAYS 42, RIGHT?
CS5 LAB...
I HATE PYTHON! I'M GONNA LOSE IT!
YOU MEAN... USE IT OR LOSE IT?
Circuits in CS5...
I MISS PICOBOT
!&$**!
Assembly Language...
HMMM...
Loops in CS5...
JUST KEEP LOOPING
JUST KEEP LOOPING


2nd week of CS5...
THE WORLD IS RECURSION IS RECURSION IS RECURSION IS RECURSION ...
3rd week of CS5...
SO, LIKE, I GUESS THE ANSWER'S ALWAYS 42, RIGHT?
CS5 LAB...
I HATE PYTHON! I'M GONNA LOSE IT!
YOU MEAN... USE IT OR LOSE IT?
Circuits in CS5...
I MISS PICOBOT
!&$**!
Pop Tart Day in CS5
WE SHOULD BE CS MAJORS...
MMHMM...

A whole new **`class`** of programming

whose convenience?

September

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					

CS's building blocks: functions and composition

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		

behind CS's curtain*: circuits, assembly, loops*

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

Designing Data!

The **Date** class

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				

CS: theory + practice

Lec 18 ~ Classes and Objects...

CS-specific **names**

class, type, user-defined type, template
object, **instance**, **self**, variable, container
method, function
  constructor, initializer, **`__init__`**
  `__repr__`, printer

CS-specific **topics**

syntax needed to define a **class**
syntax needed to create an **object**
the use of **`self`** to refer to a specific **object**
  + within the definition of a **class**!

Also!

Midterm exams...
All Python variables are objects...
Examples
+ **Student** class (that we define)
+ **str** class (Python-defined)
+ **Date** class (that we define)

Classes and Objects

An object-oriented programming language allows you to build your **own customized types** of variables.

(1) A *class* is a **type**

(2) An *object* is one such **variable**.

(instance)

There will typically be MANY objects of a single class.

Classes and Objects

Customizing Python

(1) A *class* is a **type**

(2) An *object* is one such **variable**.

(instance)

There will typically be MANY objects of a single class.

Everything in Python is an **object**!

variable

Its capabilities depend on its **class**.

functions
"methods"

type

example

```
s.split()
```

object *method or function*

dot

what's more, you can build your own...

Everything is an object!

strings, for example:

Calls the **`str`** ***constructor***.

```
In : s = str("the claremont colleges")
In : type(s)
In : dir(s)
```

Shows all of the methods (functions) of **`s`**

['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

Let's try it!

Designing a **`student`** class !

Data contained

`name` `year`

Functions contained

- two needed by Python `__init__` `__repr__`

Let's build-our-own...

Designing a **`student`** class !

Data contained

`name`

`year`

Functions contained

- two needed by Python **`__init__`** **`__repr__`**
- a function we design: **`defer(numyrs)`**

One-page example

```
# defining our own Student class
class Student:
    """ a class representing students """
    # the CONSTRUCTOR method (function)
    # [sets initial data]
    def __init__( self, name, yr ):
        """ this is the constructor """
        self.name = name
        self.year = yr

    # the "REAPER" method (for printing)
    # [let's change from 2021 to '21]
    def __repr__( self ):
        """ the not-so-grim reaper: for printing """
        s = self.name + str(self.year)
        return s

    # here's a method of our own
    # (not one of Python's __special__ ones)
    def defer( self, numyrs ):
        """ defer for numyrs years """
        self.year += numyrs

# This is the end of the Student class
```

`Student` is a class

1. constructor, **init**

2. its string **repr**esentation

3. change things via methods

define

```
# Now, let's construct two students:
fr = Student("Frosh A.", 2023 )
sr = Student("Senior B.", 2020 )
```

`fr` and **`sr`** are objects

use

Objects

Like a list, an object is a container, but much more customizable:

(1) Its data elements have *names chosen by the programmer*.

(2) An object contains its own functions, called ***methods***

(3) Inside methods, objects refer to *themselves* as `self`

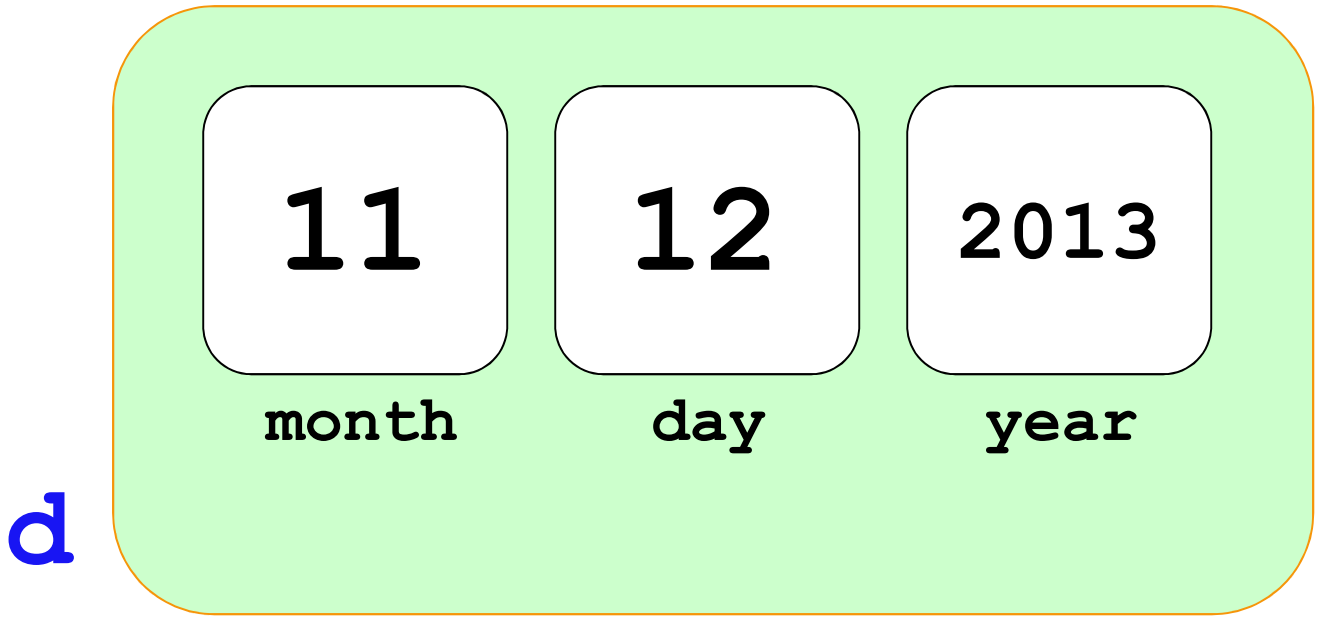
(4) Python signals special methods with two underscores:

`__init__` is called the ***constructor***; it creates new objects

`__repr__` tells Python how to print its objects

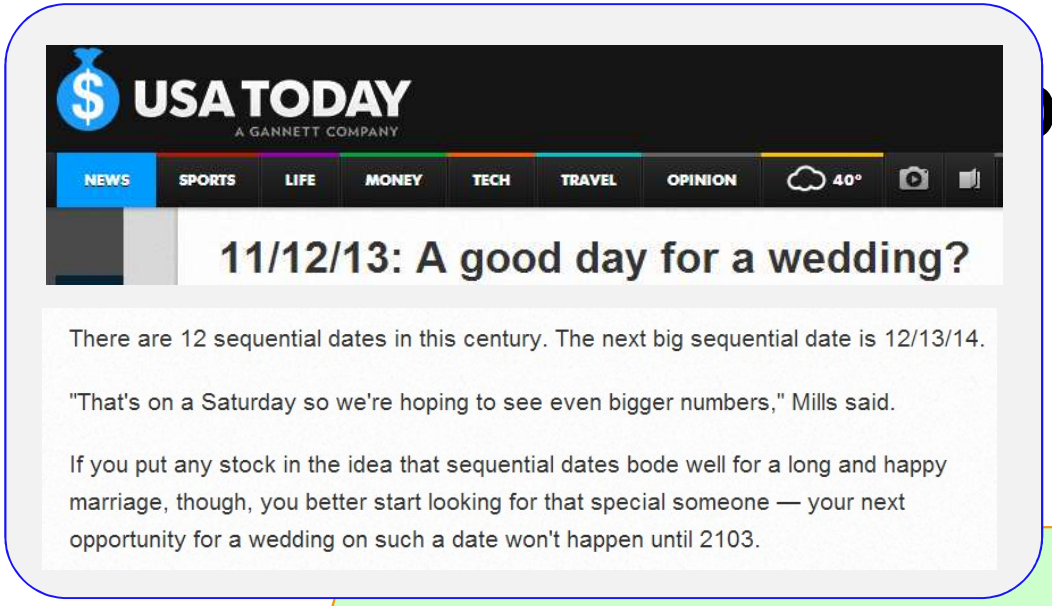
I guess we should doubly ***underscore*** *these two methods!*

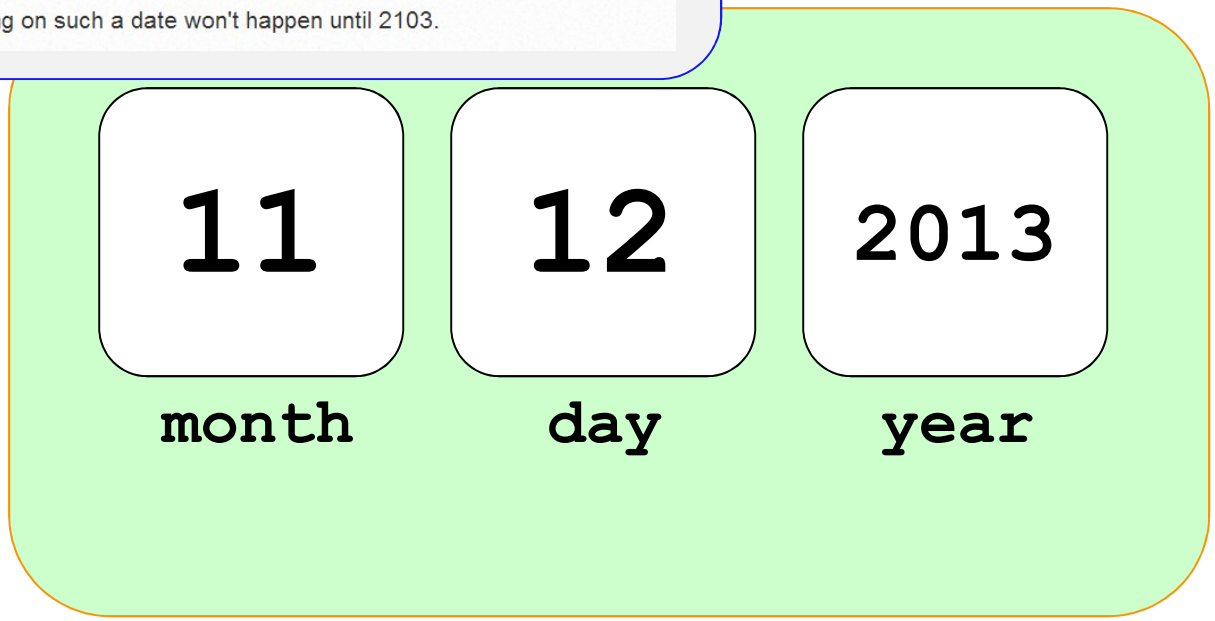
A `Date` class and object, `d`



memory location ~ 42042778

bject, d

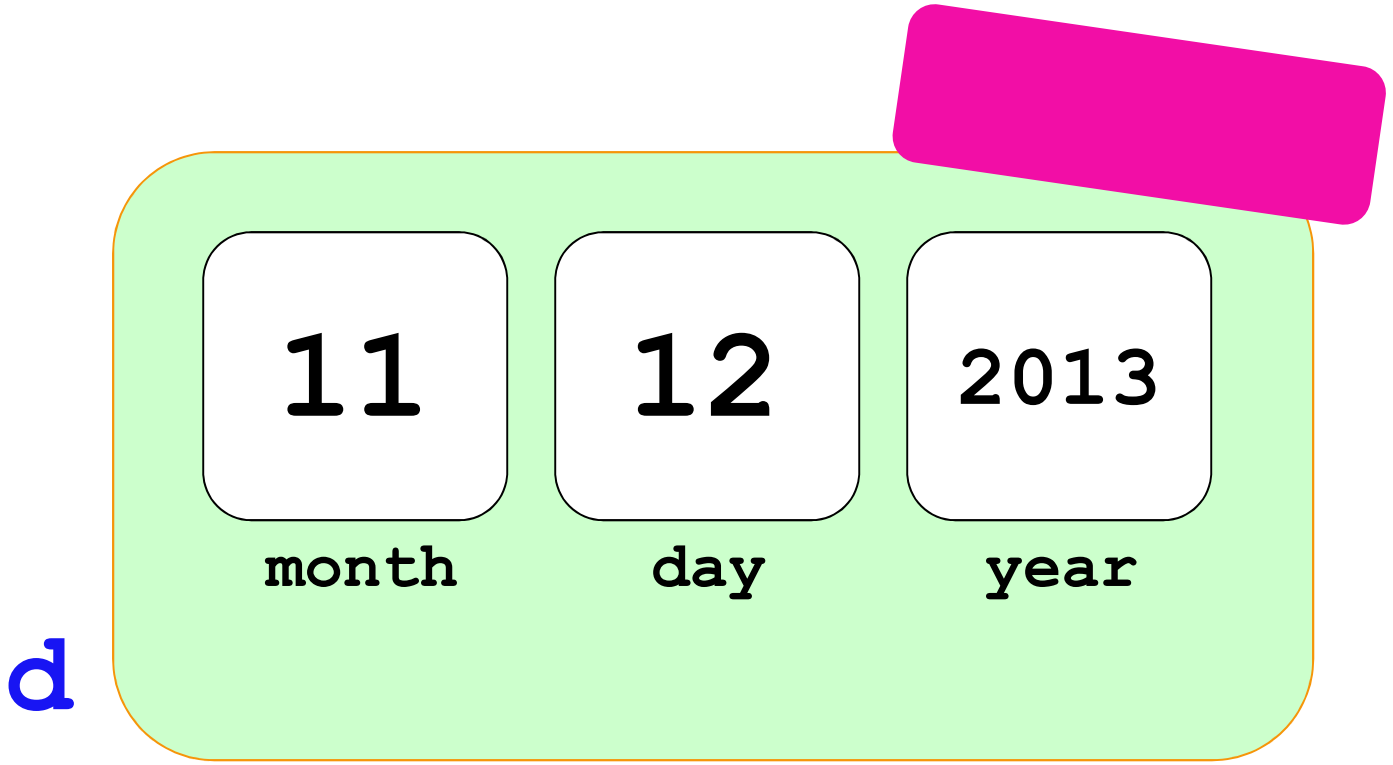




d

memory location ~ 42042778

A `Date` class and object, `d`



memory location ~ 42042778

```
class Date:
    """
        Date is a user-defined data structure --
        a class that stores and manipulates dates
    """
    def __init__(self, mo, dy, yr):
        """ the constructor for objects of type Date """
        self.month = mo
        self.day = dy
        self.year = yr

    def __repr__(self):
        """ This method returns a string representation for the
            object of type Date that calls it (named self).

            It's called by the print statement!
        """
        s =  "{:02d}/{:02d}/{:04d}".format(self.month, self.day,  self.year)
        return s

    def isLeapYear(self):
        """ Returns True if self, the calling object, is
            in a leap year; False otherwise. """
        if self.year % 400 == 0: return True
        if self.year % 100 == 0: return False
        if self.year % 4 == 0: return True
        return False
```

```
wd = Date(11,12,2013)
today = Date(11,12,2019)
ny = Date(1,1,2020)
nc = Date(1,1,2100)
```

A **Date** class and four objects, named…

The `Date` class

```
class Date:
    """ a blueprint (class) for objects
        that represent calendar days
    """
```

This is the start of a new type called Date
It begins with the keyword **class**

This is the **constructor** for Date objects
As is typical, it assigns input data to the data members.

```
def __init__( self, mo, dy, yr ):
    """ the Date constructor """
    self.month = mo
    self.day = dy
    self.year = yr
```

These are data members – they are the information inside every Date object.

The **Date** class

```
class Date:
    """ a blueprint (class) for objects
        that represent calendar days
    """
    def __init__( self, mo, dy, yr ):
        """ the Date constructor """
        self.month = mo
        self.day = dy
        self.year = yr

    def __repr__( self ):
        """ used for printing Dates """
        s = "{:02d}/{:02d}/{:04d}".format(self.month, self.day, self.year)
        return s
```

Why is everything so far away?!

This is the **repr** for Date objects
It tells Python how to print these objects.

Why **self** instead of **d** ?

2.2.1 What years are leap years?

The Gregorian calendar has 97 leap years every 400 years:

> Every year divisible by 4 is a leap year.
> However, every year divisible by 100 is not a leap year.
> However, every year divisible by 400 is a leap year after all.

So, 1700, 1800, 1900, 2100, and 2200 are not leap years. But 1600, 2000, and 2400 are leap years.

```
class Date:
    def __init__( self, mo, dy, yr ):
    def __repr__(self):

    def isLeapYear( self ):
        """ here it is """
        if self.year%400 == 0: return True
        if self.year%100 == 0: return False
        if self.year%4 == 0: return True
        return False
```

(constructor)

(for printing)

```
In : wd = Date(11,12,2013)
In : wd.isLeapYear()
Out: False
```

```
In : d = Date(1,1,2020)
In : d.isLeapYear()
Out: True
```

Your name(s)! _____

Quiz ~ *names!*

point each name to its piece of the code...

```
class Date:
    """
        Date is a user-defined data structure --
        a class that stores and manipulates dates
    """
    def __init__(self, mo, dy, yr):
        """ the constructor for objects of type Date """
        self.month = mo
        self.day = dy
        self.year = yr

    def __repr__(self):
        """ This method returns a string representation for the
            object of type Date that calls it (named self).

            It's called by the print statement!
        """
        s =  "{:02d}/{:02d}/{:04d}".format(self.month, self.day,  self.year)
        return s

    def isLeapYear(self):
        """ Returns True if self, the calling object, is
            in a leap year; False otherwise. """
        if self.year % 400 == 0: return True
        if self.year % 100 == 0: return False
        if self.year % 4 == 0: return True
        return False
```

- class keyword (keyword)
- class definition (end)
- object definitions (4)
- methods (3)
- constructor
- data member (3)
- what *prints* Dates?

```
wd = Date(11,12,2013)
today = Date(11,12,2019)
ny = Date(1,1,2020)
nc = Date(1,1,2100)
```

Extra: what should **today > wd** return? **today > ny** ?

Extra: what should **ny – today** be? What about **nc – wd**?

```
class Date:
    """
        Date is a user-defined data structure --
        a class that stores and manipulates dates
    """
    def __init__(self, mo, dy, yr):
        """ the constructor for objects of type Date """
        self.month = mo
        self.day = dy
        self.year = yr

    def __repr__(self):
        """ This method returns a string representation for the
            object of type Date that calls it (named self).

            It's called by the print statement!
        """
        s =  "{:02d}/{:02d}/{:04d}".format(self.month, self.day,  self.year)
        return s

    def isLeapYear(self):
        """ Returns True if self, the calling object, is
            in a leap year; False otherwise! """
        if self.year % 400 == 0: return True
        if self.year % 100 == 0: return False
        if self.year % 4 == 0: return True
        return False
```

try this on the back first...

Quiz ~ *names!*

point each name to its piece of the code...

- class keyword (keyword)
- class definition (end)
- object definitions (4)
- methods (3) — also __init__ and __repr__
- constructor
- data member (3)
- what *prints* Dates?

```
wd = Date(11,12,2013)
today = Date(11,12,2019)
ny = Date(1,1,2020)
nc = Date(1,1,2100)
```

Four objects here...

True

False

Extra: what should **today > wd** return? **today > ny** ?

Extra: what should **ny – today** be? What about **nc – wd**?

50

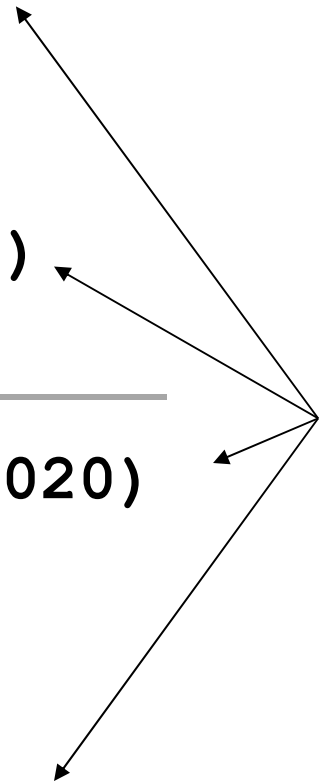
31,461

`self`

is the variable *calling* a method
(object)

```
>>> wd = Date(11,12,2013)
>>> print wd
11/12/2013

>>> wd.isLeapYear()
False
```

```
>>> d = Date(1,1,2020)
>>> print d
01/01/2020
>>> d.isLeapYear()
True
```



These methods need access to the object that calls them: it's **`self`**

Lab today – or `tomorrow()`

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)`	

methods

operators!

Prof. Benjamin !

no computer required…

What's the **`diff`**?

```
In : today = Date(11,12,2019)
In : wd = Date(11,12,2013)
In : today.diff(wd)
Out: 2191
```

method

```
In : today - wd
Out: 2191
```

operator

```
In : wd - today
Out: -2191
```

operator

```
In : eraday = Date(1,1,1)
In : today.diff(eraday)
Out: 737374
```

method

```
In : today - eraday
Out: 737374
```

operator

This gives me pause

Where's the dow?

```
In : sm1 = Date(10,28,1929)
In : sm2 = Date(10,19,1987)

In : sm1.dow()
Out: 'Monday'

In : sm2.dow()
Out: 'Monday'
```

uses a *named* object...

uses a *named* object...

```
In : Date(1,1,1).dow()
Out: 'Monday'

In : Date(1,1,2020).dow()
Out: 'Wednesday'

In : Date(10,10,2010).dow()
Out: 'Sunday'
```

unnamed!

unnamed!

popular!

The dow looks down to me!

Special Dates?

The New York Times

U.S.

10/10/10: They Love Just Thinking About It

By JOHN SCHWARTZ OCT. 8, 2010

Sunday is the big day for saying "I do."

More than 39,000 couples chose 10/10/10 as their wedding day — a nearly tenfold increase over the number of nuptials on Oct. 11, 2009, the comparable Sunday last year, according to figures gathered by David's Bridal, the wedding superstore chain.

The reason for the surge is a blend of superstition and symbolism, said Maria McBride, the wedding style director

Special Dates?

The New York Times

U.S.

10/10/10: They Love Just Thinking About It

By JOHN SCHWARTZ OCT. 8, 2010

Kevin Cheng and Coley Wopperer of San Francisco have been waiting nearly two years for their wedding date to roll around, having realized over dinner with friends in 2008 that, as one suggested, "you could have a binary-themed wedding!" he recalled.

"Both of our eyes just lit up," he said.

"We're very much technology people," Mr. Cheng explained, as if it were necessary to point this out.

Special Dates?

The New York Times

U.S.

10/10/10: They Love Just Thinking About It

By JOHN SCHWARTZ OCT. 8, 2010

Kevin Cheng and Coley Wopperer of San Francisco have been waiting nearly two years for their wedding date to roll around, having realized over dinner with friends in 2008 that, as one suggested, have a binary-themed wedding!" he recalled.

"Both of our eyes just lit up," he said.

"We're very much technology people," Mr. Cheng ex it were necessary to point this out.

The dinner group quickly calculated the more familiar base-10 value of the binary number 101010, and found that it was 42. "That totally sealed the deal!" he recalled.

Problems with ==

```
>>> wd = Date(11,12,2013)
>>> wd
11/12/2013
```

this constructs a different Date object, but with the same mo/dy/yr

```
>>> wd2 = Date(11,12,2013)
>>> wd2
11/12/2013
```

```
>>> wd == wd2
False
```

How can this be False ?

Problems with ==

```
>>> wd = Date(11,12,2013)
>>> wd
11/12/2013
```

this constructs a different Date object, but with the same mo/dy/yr

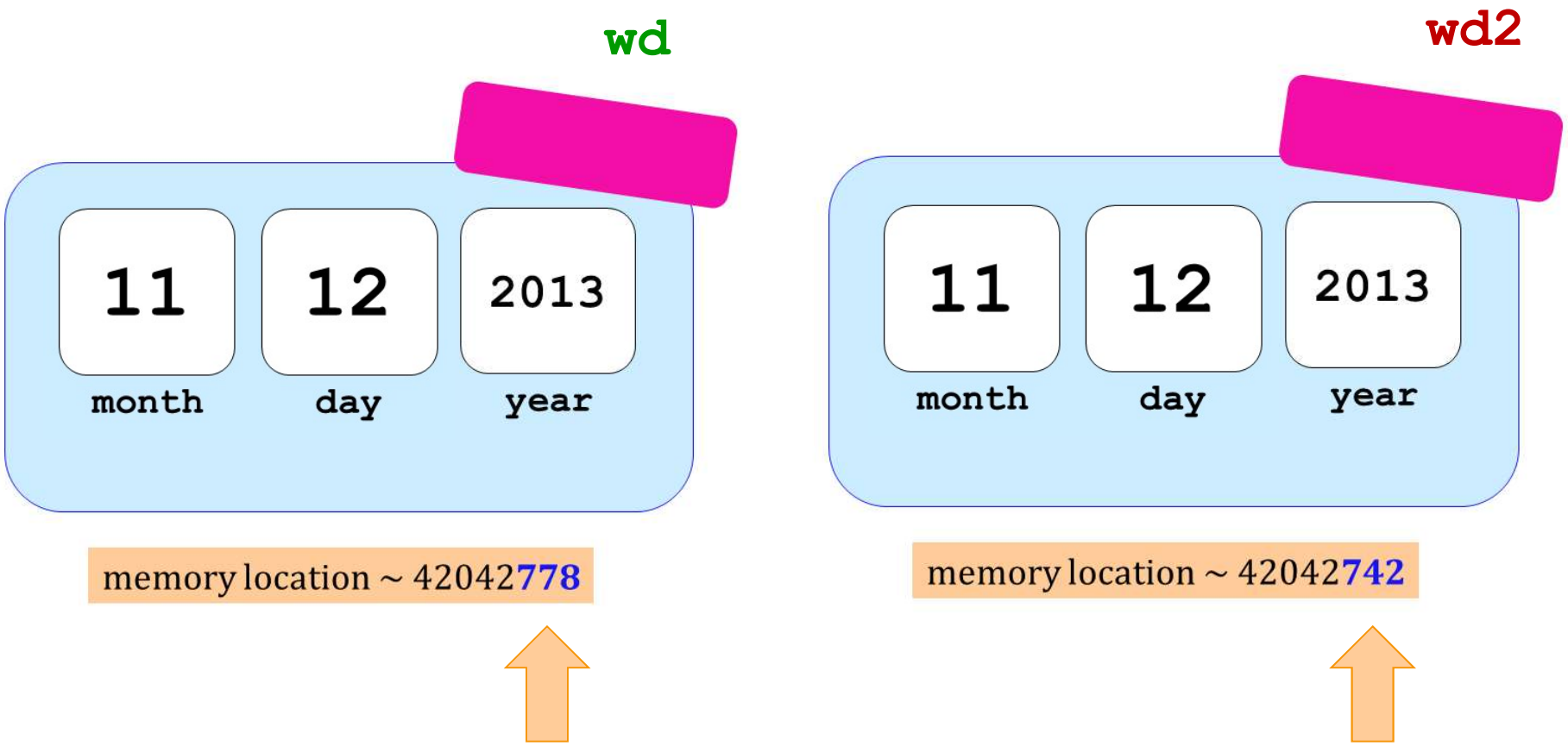
```
>>> wd2 = Date(11,12,2013)
>>> wd2
11/12/2013
```

```
>>> wd == wd2
False
```

How can this be False ?

Python objects are handled by reference…
== compares references!

Two `Date` objects:



== compares **memory locations**, not contents

originals underneath…

equals

```
class Date:

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

    def equals(
        """ returns
            represent
            False oth
        """
        if self.year =
           self.month
           self.day ==
                return
        else:
                return
```

Let's write *our own* equality-tester

```
wd.equals(wd2)
```

```
wd2.equals(wd)
```

equals

```
class Date:

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

    def equals(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
```

`wd.equals(wd2)`

`wd2.equals(wd)`

which goes where?

Solution: **equals**

```
>>> wd = Date(11,12,2013)
>>> wd
11/12/2013
```

this constructs a different Date object, but with the same mo/dy/yr

```
>>> wd2 = Date(11,12,2013)
>>> wd2
11/12/2013
```

```
>>> wd.equals(wd2)
True
```

.equals compares mo/dy/yr – *because we asked it to!*

But *who* is this convenient for?!

__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
```

L==k! This is T== C==L!

redefined for *our convenience*!

To use this, write `d == d2`

DIY operators ...

__eq__(self, other) defines the equality operator, ==
__ne__(self, other) defines the inequality operator, !=
__lt__(self, other) defines the less-than operator, <
__gt__(self, other) defines the greater-than operator, >
__le__(self, other) defines the less-or-equal-to operator, <=
__ge__(self, other) defines the gr.-or-equal-to operator, >=

__add__(self, other) defines the addition operator, +
__sub__(self, other) defines the subtraction operator, -

... and many more! Use dir('')

there are two under-
scores on each side here

I should **underscore** this unusual syntax!

More operators!

arithmetic

```
__add__(self, other)      +
__sub__(self, other)      -
__mul__(self, other)      *
__matmul__(self, other)   @
__truediv__(self, other)
__floordiv__(self, other)
__mod__(self, other)
__divmod__(self, other)
__pow__(self, other[, modulo])
__lshift__(self, other)
__rshift__(self, other)
__and__(self, other)
__xor__(self, other)
__or__(self, other)
```

Booleans

```
__lt__(self, other)
__le__(self, other)
__eq__(self, other)
__ne__(self, other)
__gt__(self, other)
__ge__(self, other)
```

```
__iadd__(self, other)     +=
__isub__(self, other)     -=
__imul__(self, other)     *=
__imatmul__(self, other)  @=
__itruediv__(self, other)
__ifloordiv__(self, other)
__imod__(self, other)
__ipow__(self, other[, modulo])
__ilshift__(self, other)
__irshift__(self, other)
__iand__(self, other)
__ixor__(self, other)
__ior__(self, other)
```

in-place arithmetic

https://docs.python.org/3/reference/datamodel.html#special-method-names

Lab today – or `tomorrow()`

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)
```

methods

operators!

Prof. Benjamin !

no computer required…

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
```

When doesn't this function work?!

```
Date(12,31,1999).isBefore(Date(11,12,2019))
```

```
Date(11,12,2019).isBefore(Date(12,31,1999))
```

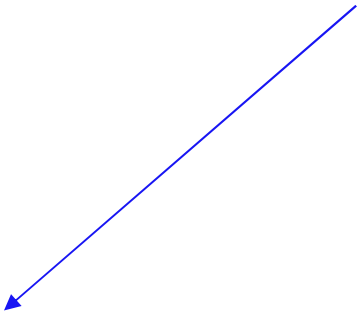
isBefore

(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 < d2?  (earlier)  """

        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 ____.isBefore(____)
```

The 2 *most essential methods*

```
>>> wd = Date(11,12,2013)
>>> print wd
11/12/2013
```

construct with the **CONSTRUCTOR** …

print uses **__repr__**

```
>>> wd.tomorrow()
>>> print wd
11/13/2013
```

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

wd has changed!

`d += 1`

```
>>> wd.yesterday()
>>> print wd
11/12/2013
```

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

`d -= 1`

Some methods return a value; others change the object that call it!

```
class Date:
```

Don't hand this in... *Use for hw10pr1 this week!*

```
    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

        if
                self.day
                self.month
                if
```

first, add 1 to `self.day`

DIM looks pretty bright to me!

test if we have gone "out of bounds!"

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

Don't return anything. This **CHANGES** the date object that calls it.

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
```

the "Luke trick"!

Don't hand this in... *Use for hw10pr1 this week!*

```
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]
```

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

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:○

See you @ this week's lab …

class-y

… it's a `Date`!

L.A. street sign from 2006… *with typo?!*

real or otherwise!