# Lec 18 ~ Python gets *classy*

It's an alien date!

What are **classes** and **objects**?

What are their **methods**?

And why do they matter?

Ann Clark
Sur la Table
Ali Express

# Lec 18 ~ Classes and Objects...

Let's immerse ourselves!

CS-specific **names**

**class**, type, user-defined type, template
**object**, **instance**, **target**, **self**,
**attribute**, 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!

All Python values are **objects**...
Examples:
+ **Student** class (that we define)
+ **str** class (Python-defined)
+ **Date** class (that we define)

# Lists are all you need... (and yet...)

```
# Represent "Hello" as a list of characters
S = [ 'H', 'e', 'l', 'l', 'o' ]
S = [ 72, 101, 108, 108, 111 ]     # or just ASCII values

# Instead of the complex number 3 + 7j
C = [ 3, 7 ]

# Instead of the dictionary { 'a': 1, 'b': 2 }
D = [ [ 'a', 1 ], [ 'b', 2 ] ]

# The time 3:45:02 PM
T = [ 15, 45, 02 ]
```

***What's not to like?***

# Two ways to do complex numbers

*make list type*

```
C = [ 3, 7 ]
Re = C[0]
Im = C[1]
```

*index syntax*

*make complex-number type*

```
C = complex(3, 7)
Re = C.real
Im = C.imag
```

*attribute syntax*

*What's better?*

# Different types “behave” differently!

***doesn't mean the same thing!***

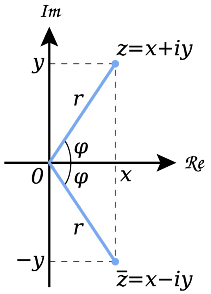
```
S = "Hello" * 2
```

```
N = 21 * 2
```

```
Float1 = 3.14
Float2 = -Float1
```

***negation operation***

complex numbers have additional operations compared to floats



```
Cplex1 = 3 + 7j
Cplex2 = -Cplex1
```

$\texttt{Cplex3} = \overline{\texttt{Cplex1}}$

***complex conjugate operation***

this is how mathematicians write it, can't do overbar in Python!

Normal functions have just **one way** to work

Need a universal way to say “use **your** way to do *whatever*”

# Methods — *Identity*-based functions

*target* *method*

The universal way to say
“use **your** way to do *whatever*”

```
Cplex3 = Cplex1.conjugate()

I = 1234
Bits = I.bit_length()    # How many bits do we need?

S = " harvey mudd college "
S = S.strip()  # remove leading/trailing whitespace
S = S.upper()  # convert to upper case
L = S.split()  # split into words at whitespace

L.sort()                 # sort the list
L.reverse()              # reverse the list
L.remove('COLLEGE')      # remove the word 'COLLEGE'
L.extend(['CS','DEPT']) # add two words to the list
```

# Special Methods

Examples:

```
N = -22
N = N.__add__(1)       # same as N + 2
N = N.__mul__(2)       # same as N * 2
N = N.__neg__()        # same as -N

S = "Hello"
S = S.__add__("World") # same as S + "World"
S = S.__mul__(2)       # same as S * 2
```

Behind the scenes, Python calls a lot of methods!

# Classes and Objects

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

- A *class* is a **type**
- An *object* is an *instance* of that type

Class

Objects

Customizing Python

# Classes and Objects

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

We can define our own new classes!

- A *class* is a **type**
- An *object* is an *instance* of that type

Class

Objects

# Designing a **Student** class !

```
class Student:
```

**Data** in each *instance* (e.g., `self`)

`self.name`    `self.dorm`    `self.year`

**Methods** provided by the class

- needed special methods   `__init`

Let's build-our-own...

ethod we design:   `defer(numyrs)`

# Designing a **student** class !

```
class Student:
```

**Data** in each *instance* (e.g., self)

self.name

self.dorm

self.year

**Methods** provided by the class

- needed special methods `__init__` `__repr__`
- method we design: `newdorm()`
- method we design: `defer(numyrs)`

# One-page example

```python
# we define a Student class  (our own class/type)
#
class Student:
    """ a class representing students """
    # the CONSTRUCTOR method (function)
    # [sets initial data]
    def __init__(self, name, dorm, yr):
        """ this is the constructor """
        self.name = name
        self.dorm = dorm        # self.var = var is common
        self.year = yr          # but not required
        print("Welcome to Claremont,", self.name)    # add some printing

    # the "REAPER" or "REPPER" method (for printing)
    # [let's change from 2025 to '25 or 2021 to '21]
    def __repr__(self):
        """ print uses __repr__ to get a string representation """
        s = self.name + " " + str(self.year) + " (" + self.dorm + ")"
        return s

    # here's a method of our own
    def newdorm(self, dorm):
        """ sets the Student's new dorm """
        self.dorm = dorm

    # here's another method of our own
    def defer(self, numyrs):
        """ defer graduation for numyrs years """
        self.year += numyrs

# Thus ends the Student __class__ (for now)
```

**`Student`** is a *class*

1. constructor, **init**

2. its string **repr**esentation

3. we change and access information via methods

**define**

---

```python
# Next, let's construct several students!
sr = Student("Melissa", "West", 2023 )
jr = Student("Anadel", "New Dorm", 2024 )
so = Student("Nico", "Case", 2025)
fr = Student("Madeline", "Atwood", 2026)
fi = Student("Maya", "Linde", 2026)

za = Student("zach", "The Cafe", 2042)
# all are variables of type Student ("software objects")
```

**use**

**`fr`** and **`so`** are *objects*

as are **`jr`** and **`sr`** and **`fi`** and **`za`**

all are variables

*all are **`self`**!*

# 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's class provides its functions, called ***methods***

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

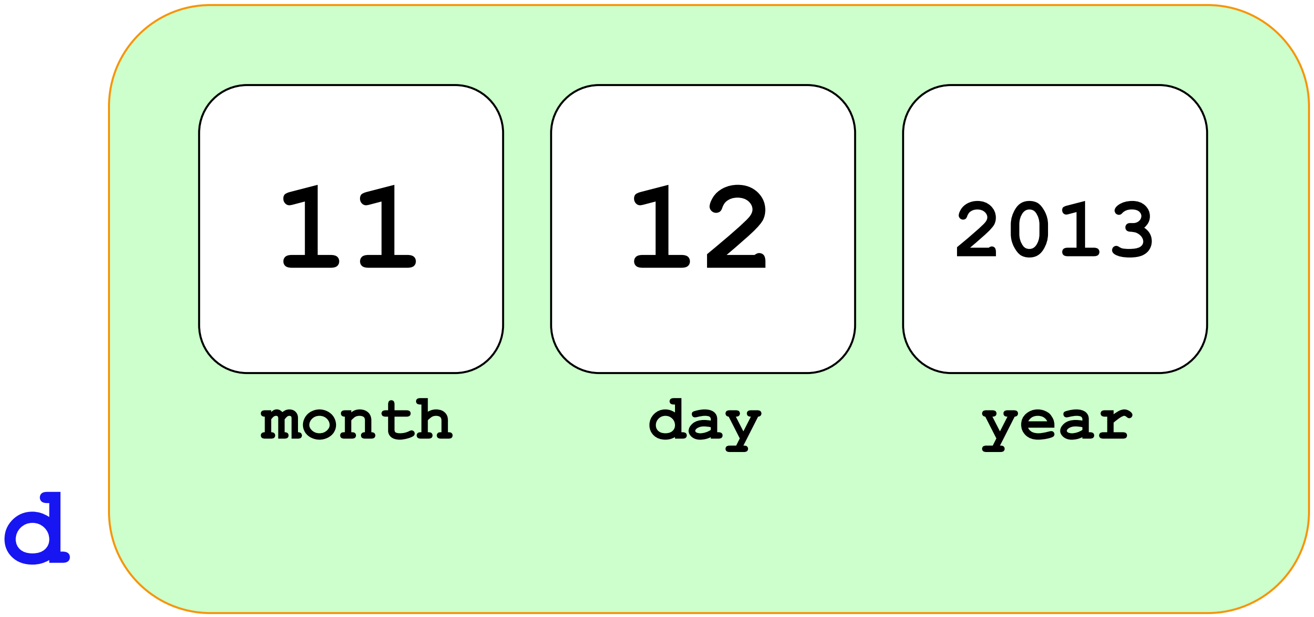
**(4)** Python signals *special methods* with two underscores:

**`__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`



11

12

2013

month

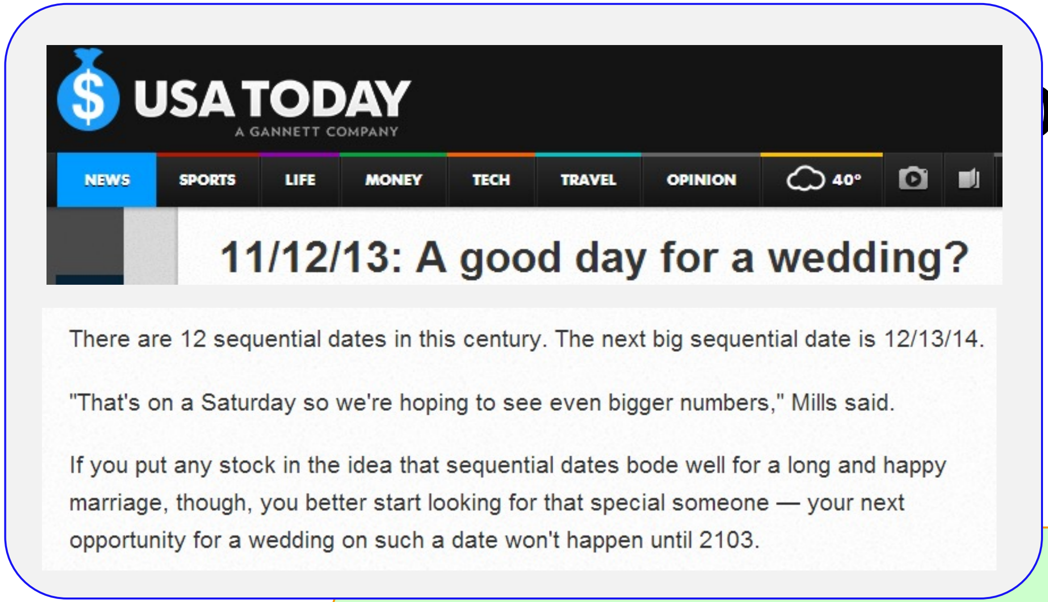
day

year

d

memory location ~ 42042778

# bject, d

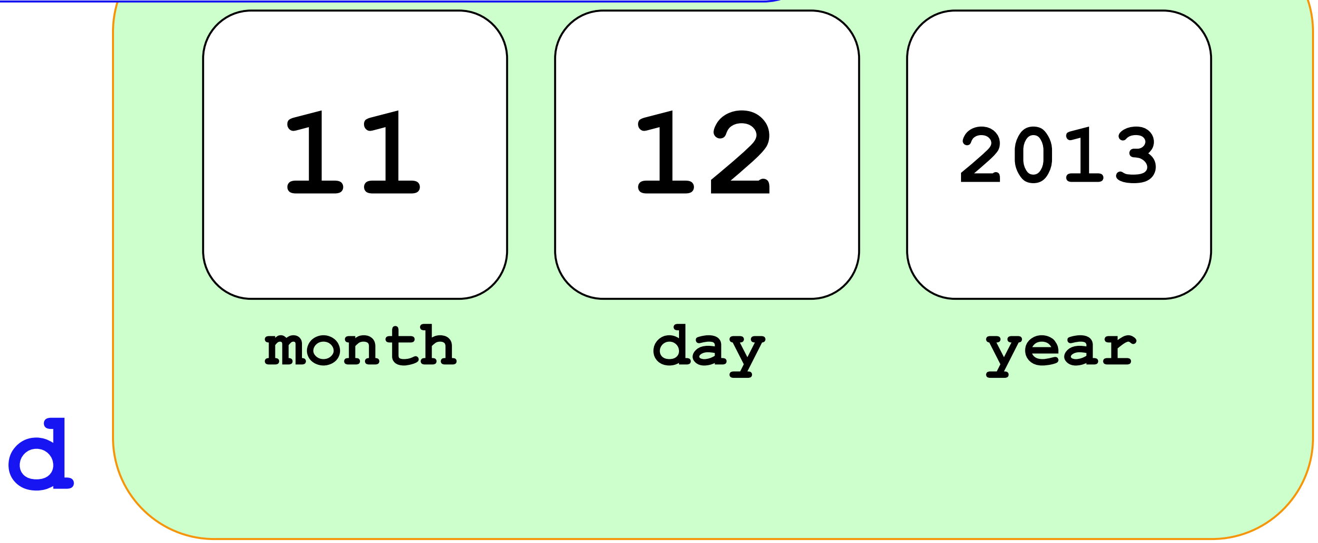


## 11/12/13: A good day for a wedding?

There are 12 sequential dates in this century. The next big sequential date is 12/13/14.

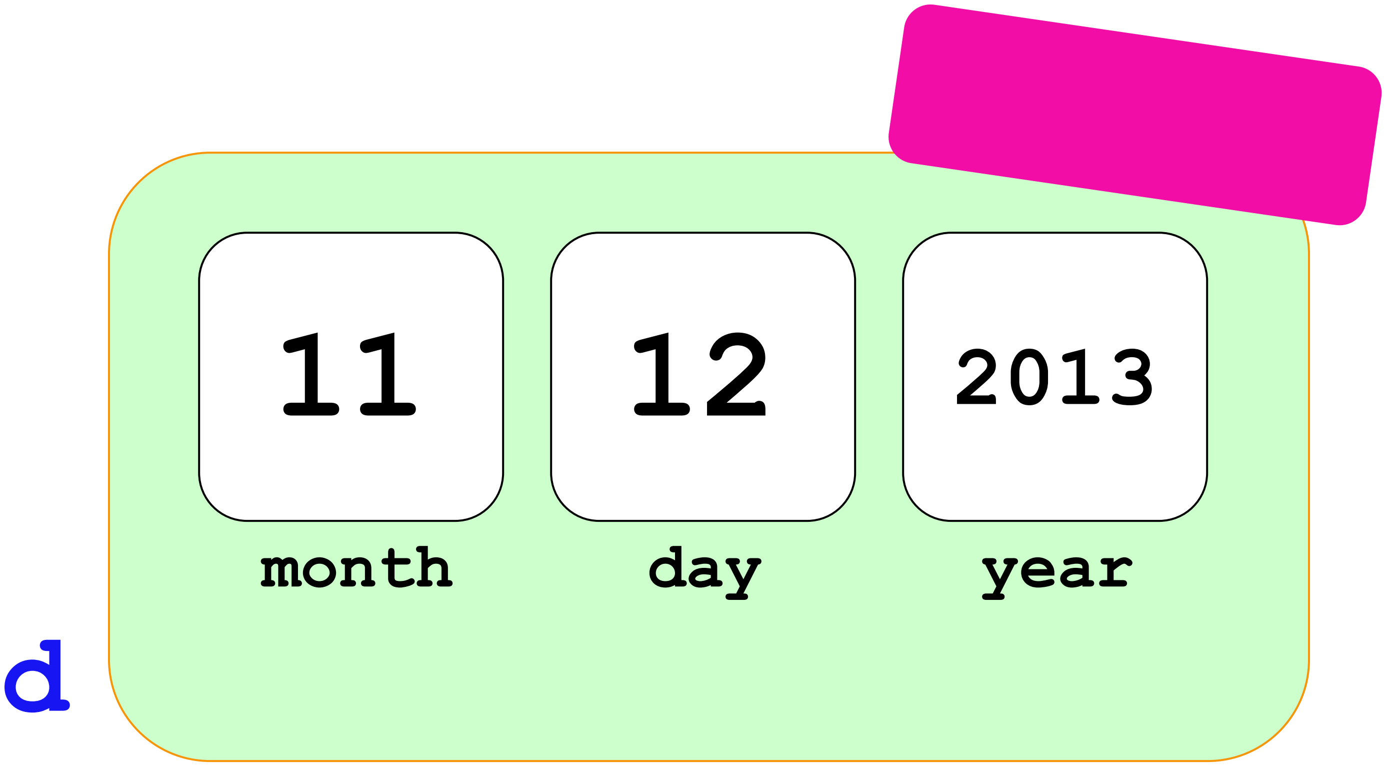
"That's on a Saturday so we're hoping to see even bigger numbers," Mills said.

If you put any stock in the idea that sequential dates bode well for a long and happy marriage, though, you better start looking for that special someone — your next opportunity for a wedding on such a date won't happen until 2103.



memory location ~ 42042778

# A `Date` class and object, `d`

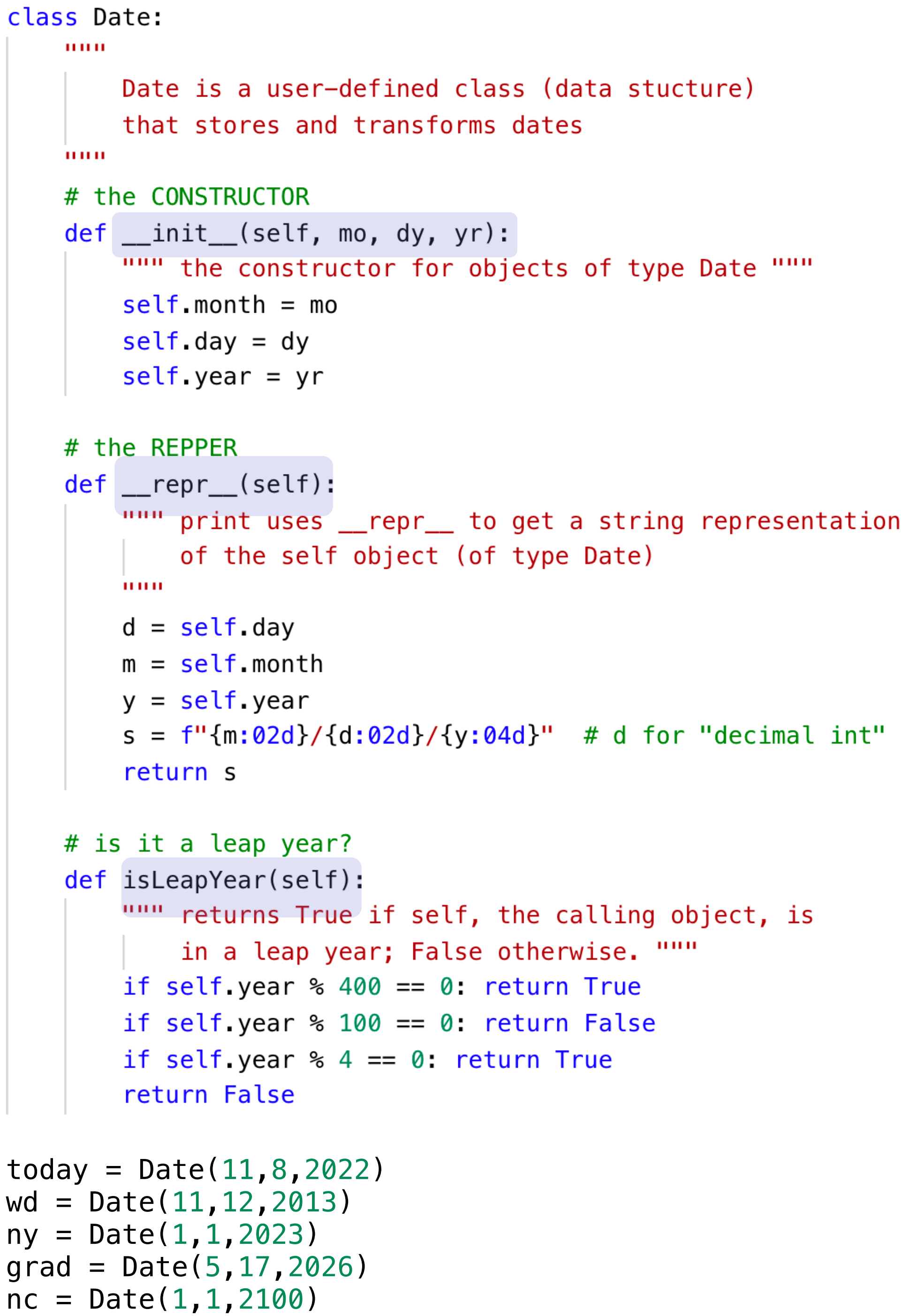


memory location ~ 42042778

# A Date class

# and

# five objects, named...



```
class Date:
    """
        Date is a user-defined class (data stucture)
        that stores and transforms dates
    """

    # the CONSTRUCTOR
    def __init__(self, mo, dy, yr):
        """ the constructor for objects of type Date """
        self.month = mo
        self.day = dy
        self.year = yr

    # the REPPER
    def __repr__(self):
        """ print uses __repr__ to get a string representation
            of the self object (of type Date)
        """
        d = self.day
        m = self.month
        y = self.year
        s = f"{m:02d}/{d:02d}/{y:04d}"  # d for "decimal int"
        return s

    # is it a leap year?
    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

today = Date(11,8,2022)
wd = Date(11,12,2013)
ny = Date(1,1,2023)
grad = Date(5,17,2026)
nc = Date(1,1,2100)
```

# 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**

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

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

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

```
today = Date(11,8,2022)
wd = Date(11,12,2013)
ny = Date(1,1,2023)
grad = Date(5,17,2026)
nc = Date(1,1,2100)
```

Why **self** ?

# The Date class

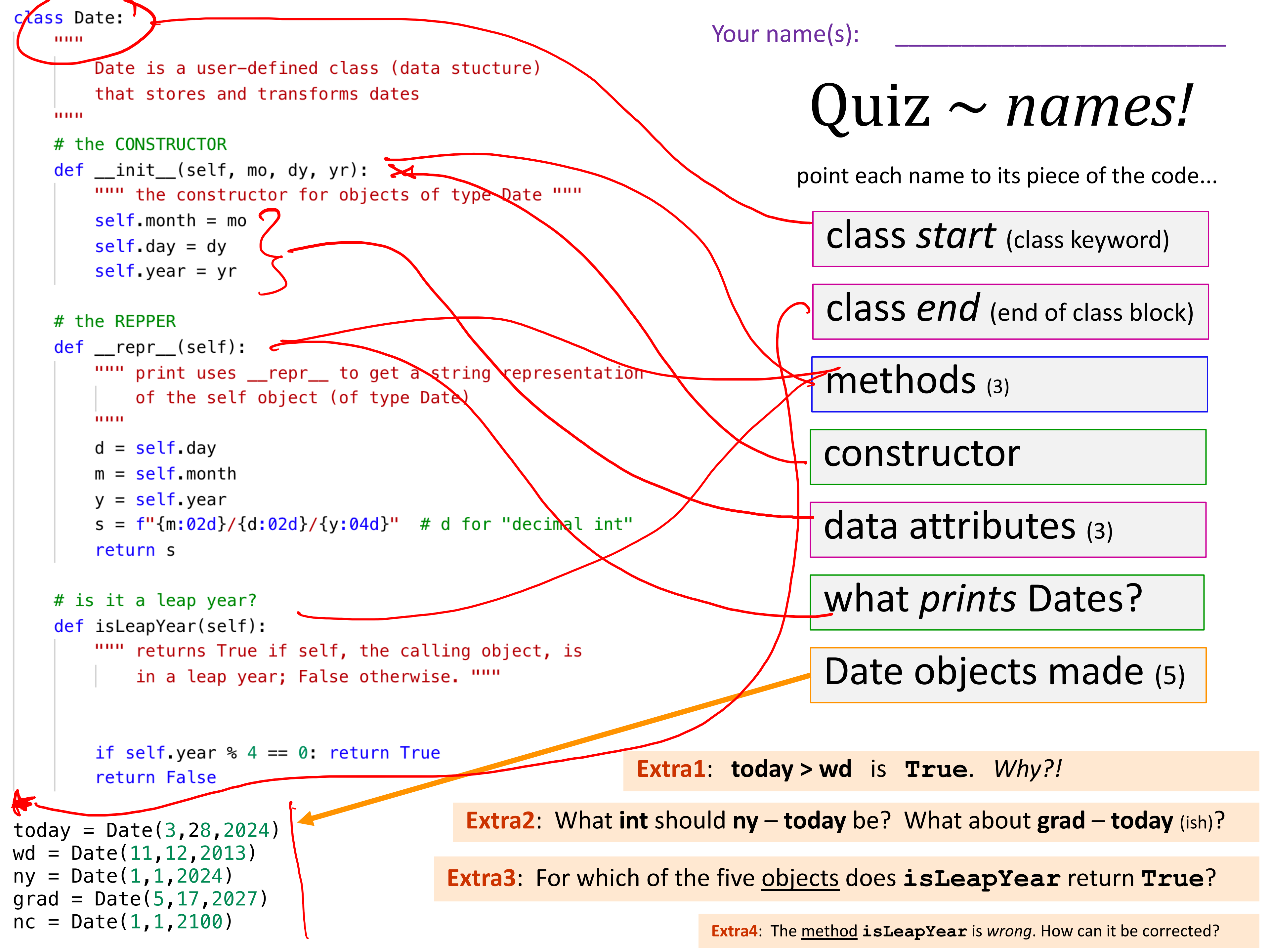
```
class Date:
    """ a blueprint (class) for objects
        that represent calendar days
    """
    def __init__( self, mo, dy, yr ):
        """ the Date constructor """
        self.month = mo
        self.day = dy
        self.year = yr

    def __repr__( self ):
        """ used for printing Dates """
        m = self.month
        d = self.day
        y = self.year
        string = f"{m:02d}/{d:02d}/{y:04d}"
        return string
```

Python's f"strings" are f"antastic"!

This is the **repr** for Date objects

It tells Python how to show these objects.

```
today = Date(11,8,2022)
wd = Date(11,12,2013)
ny = Date(1,1,2023)
grad = Date(5,17,2026)
nc = Date(1,1,2100)
```



Your name(s): ______________________

# Quiz ~ *names!*

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

class *start* (class keyword)

class *end* (end of class block)

methods (3)

constructor

data attributes (3)

what *prints* Dates?

Date objects made (5)

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

    # the REPPER
    def __repr__(self):
        """ print uses __repr__ to get a string representation
            of the self object (of type Date)
        """
        d = self.day
        m = self.month
        y = self.year
        s = f"{m:02d}/{d:02d}/{y:04d}"  # d for "decimal int"
        return s

    # is it a leap year?
    def isLeapYear(self):
        """ returns True if self, the calling object, is
            in a leap year; False otherwise. """


        if self.year % 4 == 0: return True
        return False

today = Date(3,28,2024)
wd = Date(11,12,2013)
ny = Date(1,1,2024)
grad = Date(5,17,2027)
nc = Date(1,1,2100)
```

**Extra1**: **today > wd** is **True**. *Why?!*

**Extra2**: What **int** should **ny – today** be? What about **grad – today** (ish)?

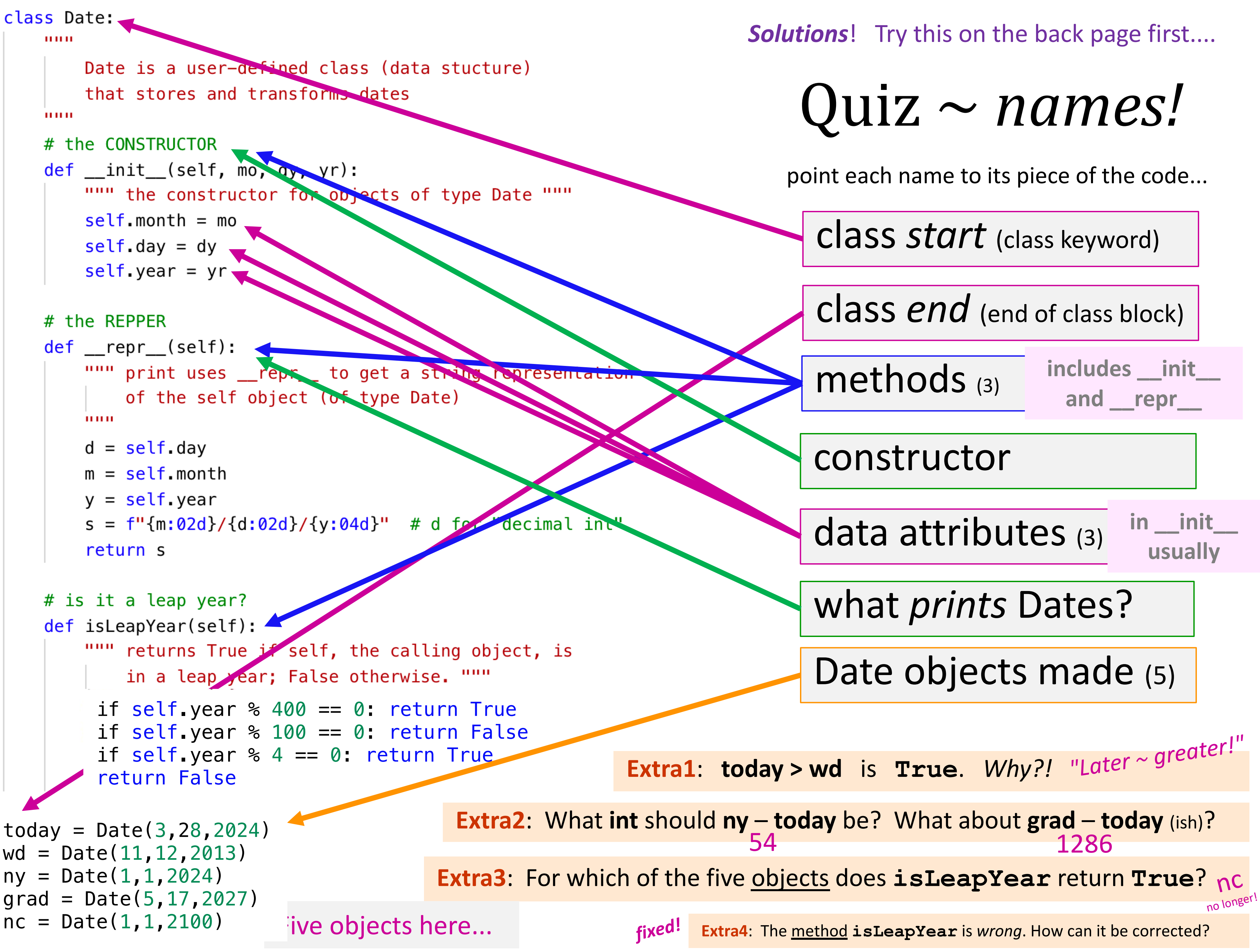
**Extra3**: For which of the five objects does **isLeapYear** return **True**?

**Extra4**: The method **isLeapYear** is *wrong*. How can it be corrected?

*Solutions*! Try this on the back page first....

# Quiz ~ *names!*

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



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

    # the REPPER
    def __repr__(self):
        """ print uses __repr__ to get a string representation
            of the self object (of type Date)
        """
        d = self.day
        m = self.month
        y = self.year
        s = f"{m:02d}/{d:02d}/{y:04d}"  # d for "decimal int"
        return s

    # is it a leap year?
    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

today = Date(3,28,2024)
wd = Date(11,12,2013)
ny = Date(1,1,2024)
grad = Date(5,17,2027)
nc = Date(1,1,2100)
```

ive objects here...

class *start* (class keyword)

class *end* (end of class block)

methods (3)

includes __init__ and __repr__

constructor

data attributes (3)

in __init__ usually

what *prints* Dates?

Date objects made (5)

**Extra1**: **today > wd** is **True**. *Why?!* *"Later ~ greater!"*

**Extra2**: What **int** should **ny – today** be? What about **grad – today** (ish)?
54
1286

**Extra3**: For which of the five objects does **isLeapYear** return **True**? nc no longer!

fixed! **Extra4**: The method **isLeapYear** is *wrong*. How can it be corrected?

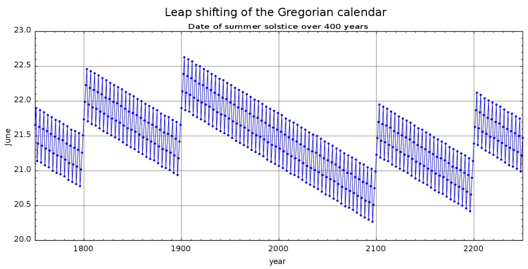
## 2.2.1 What years are leap years?

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.

Calendar seasonal error



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

    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
```

**self** is a name for the method's target object, used by *convention* to make our code clearer to others

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

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

**self** is the *target object* that's ***calling*** the method

```
>>> od = Date(1,1,2020)
>>> print(od)
1/1/20
>>>
True
```

# self?

```
>>> w
>>> p        (wd)
11/09/2021
>>> wd.isLeapYear()
False
```

need access
to the object that calls it:
*that object is* **self**

## `self` is the *target object* that's ***calling*** the method

```
>>> od = Date(1,1,2020)
>>> print(od)
1/1/2020

>>> od.isLeapYear()
True
```

---

```
>>> wd = Date(11,12,2013)
>>> print(wd)
11/09/2021
>>> wd.isLeapYear()
False
```

Every method need access to the object that calls it: *that object is* **`self`**

(there's no way for the class code to know what the variable name will be -- days, months, or years before it's used)

# Lab next week...

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(11,8,2022)
In : wd = Date(11,12,2013)
In : today.diff(wd)
Out: 3283
```

*method*

```
In : today - wd
Out: 3283
```

operator

```
In : wd - today
Out: -3283
```

operator

```
In : eraday = Date(1,1,1)
In : today.diff(eraday)
Out: 738466
```

*method*

```
In : today - eraday
Out: 738466
```

operator

This gives me pause

# Where's the dow?

The dow looks down to me!

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

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

uses a *named* object...

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

uses a *named* object...

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

*unnamed!*

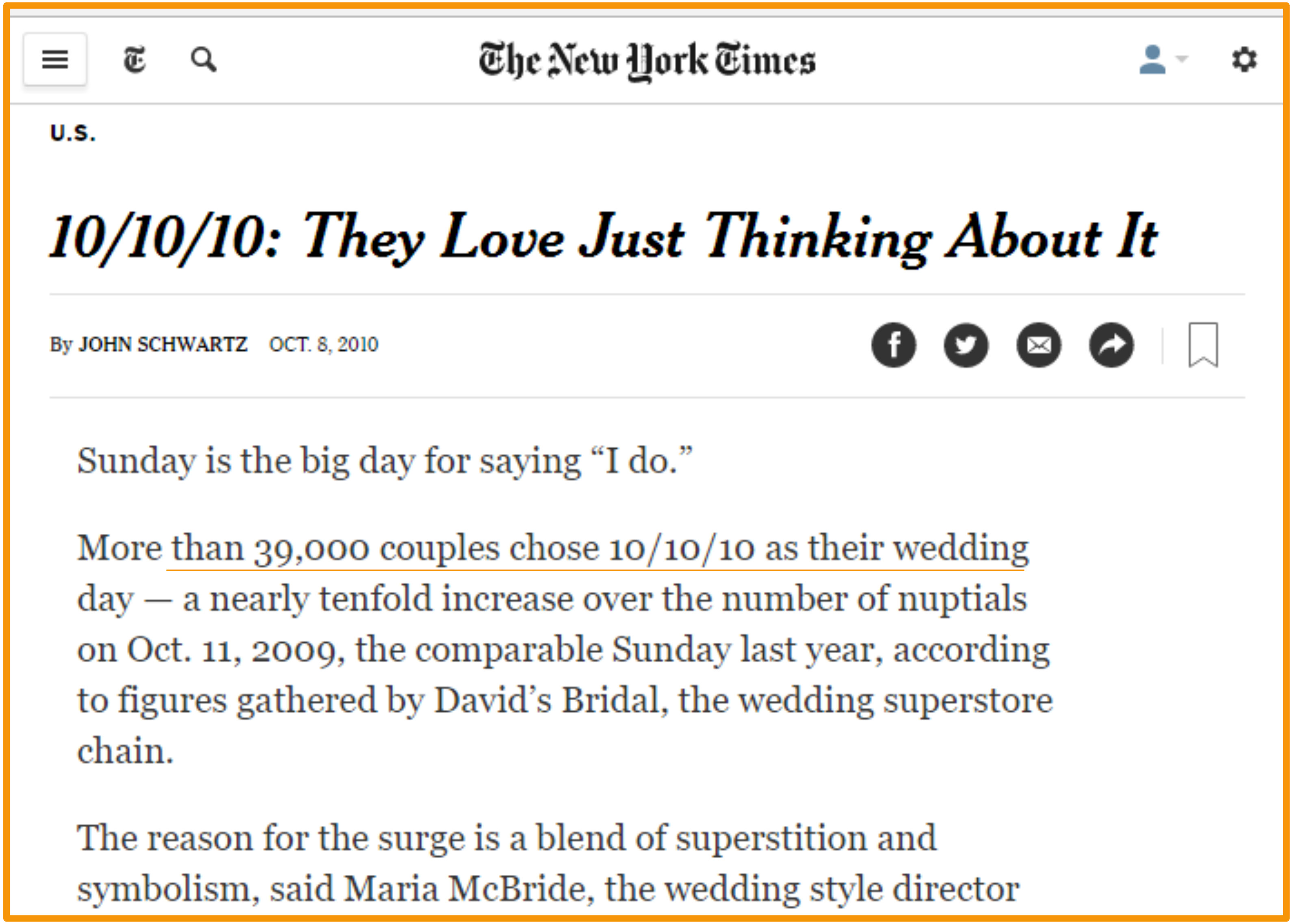
```
In : Date(1,1,2100).dow()
Out: 'Friday'
```

*unnamed!*

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

*popular!*

# Special Dates?



U.S.

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

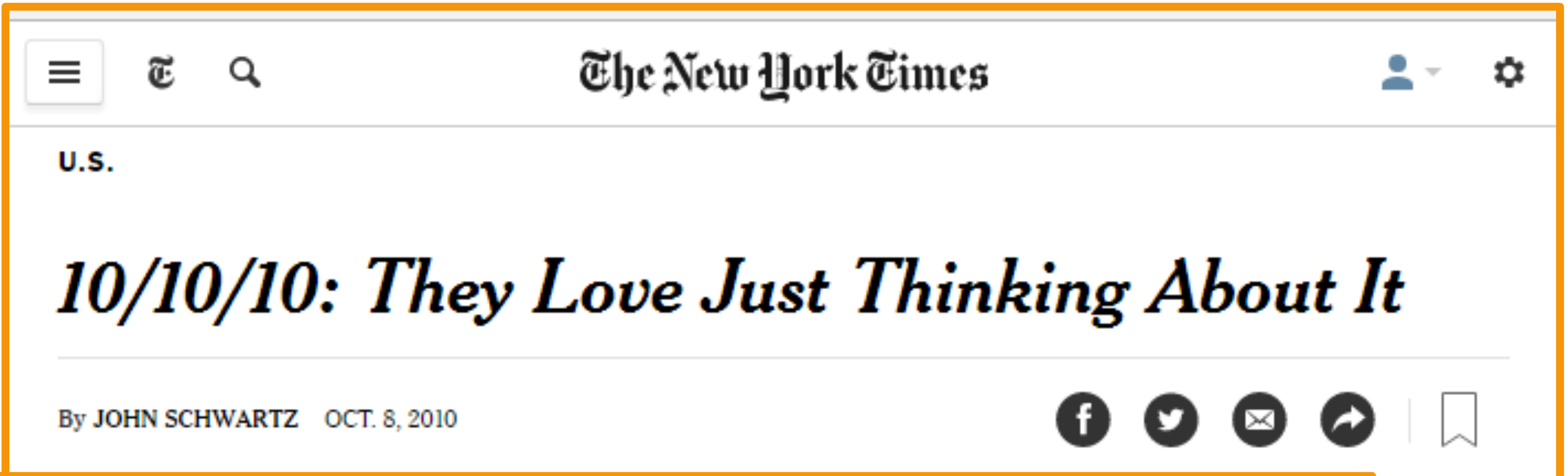
By JOHN SCHWARTZ OCT. 8, 2010

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?

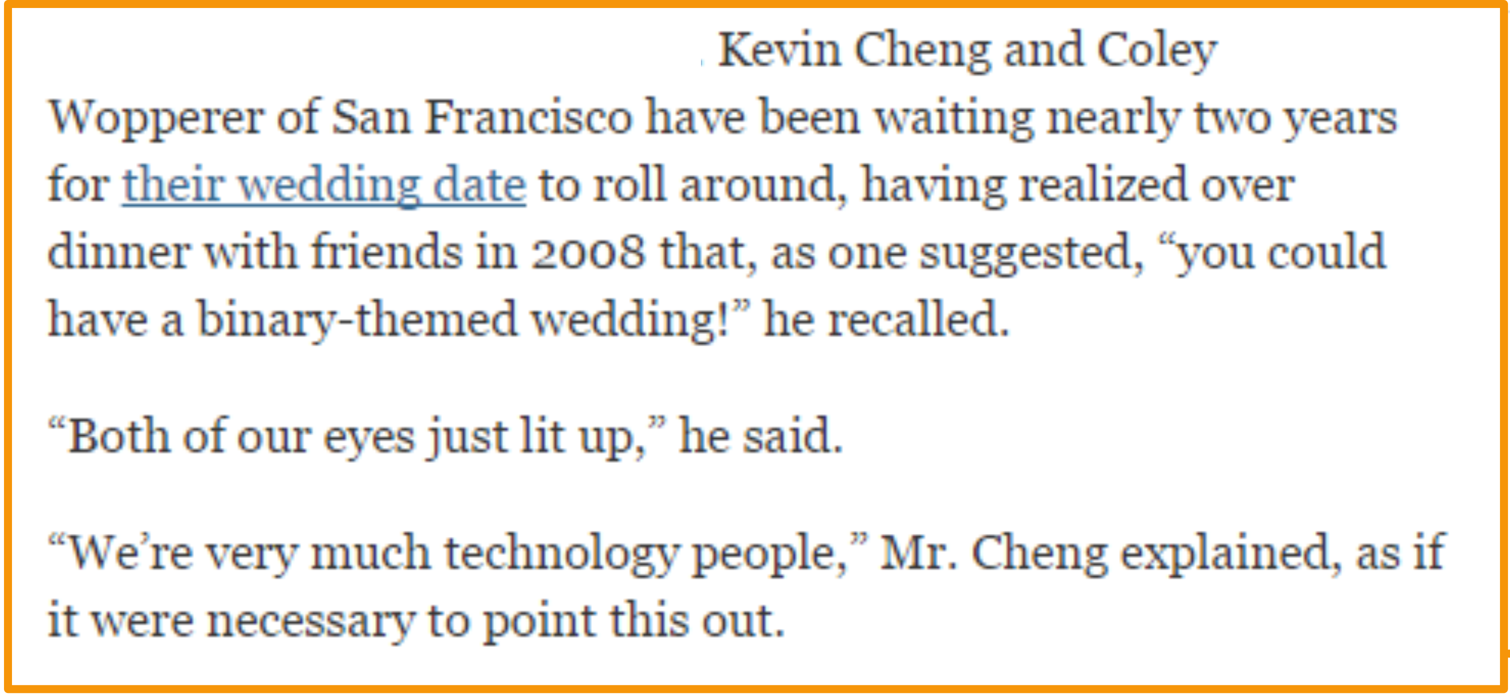


The New York Times

U.S.

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

By JOHN SCHWARTZ OCT. 8, 2010

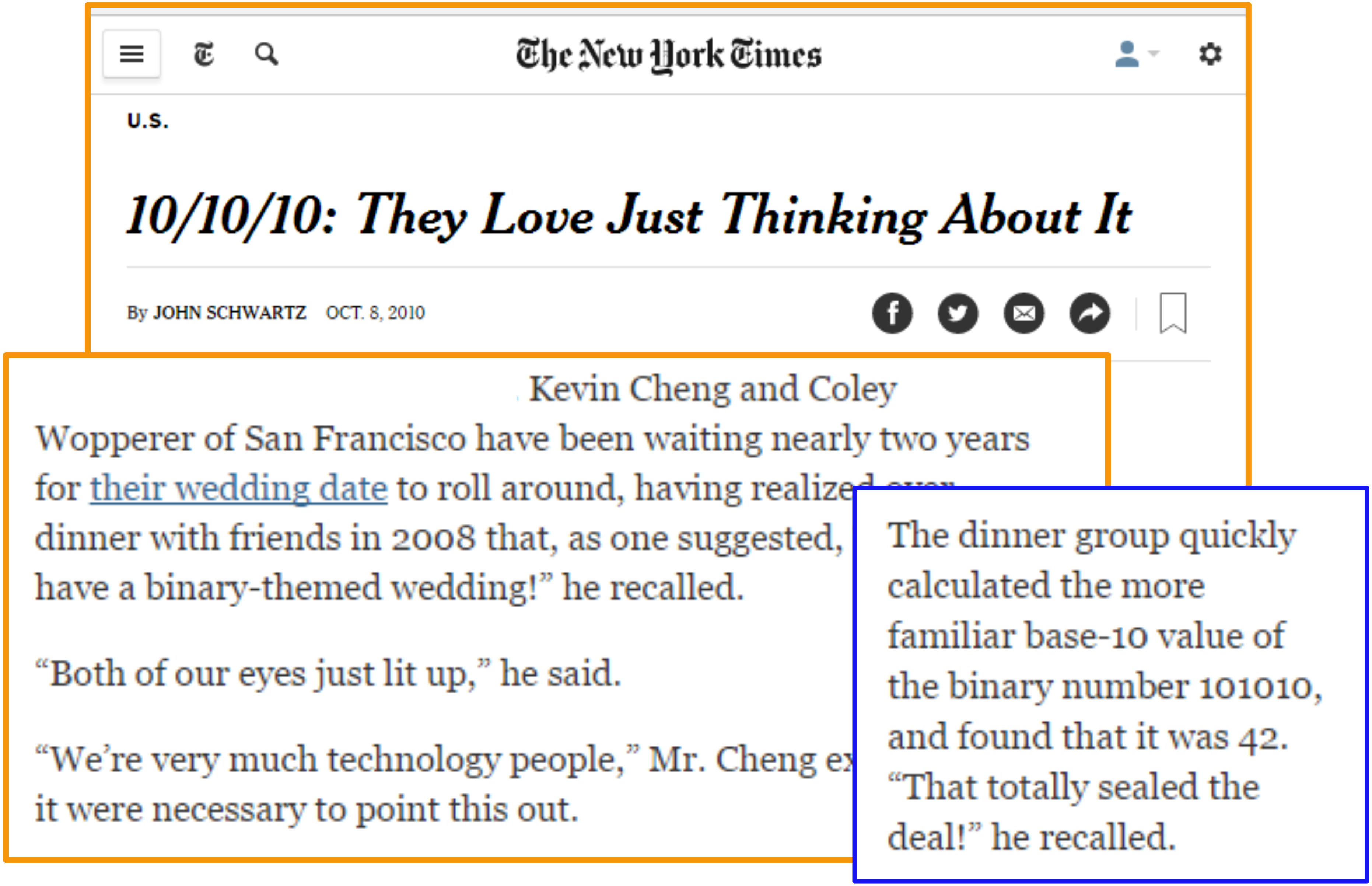


Kevin Cheng and Coley Wopperer of San Francisco have been waiting nearly two years for their wedding date to roll around, having realized over dinner with friends in 2008 that, as one suggested, “you could have a binary-themed wedding!” he recalled.

“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 <u>different</u> 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 <u>False</u> ?*

# 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
```

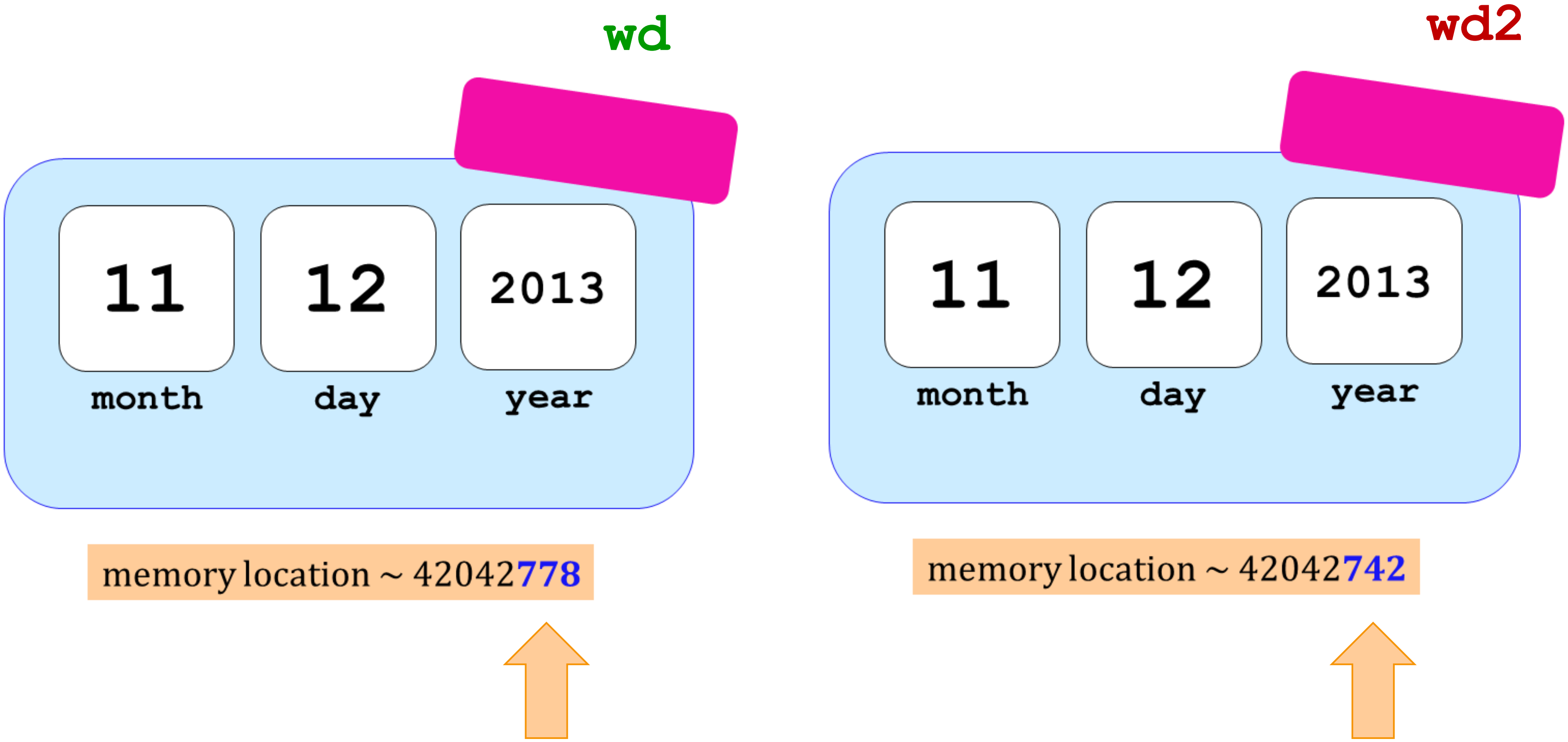
```
>>> wd == wd2
False
```

*How can this be False ?*

Object identity!

`==` compares **`id`***s*!

# 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 both
            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 x
```

```
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 wrote it to!*

*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 both
            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, -

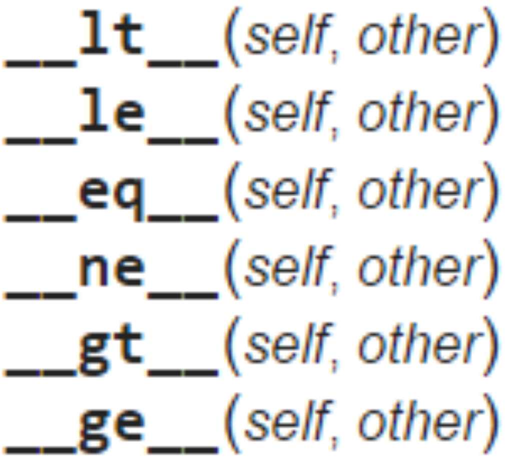
... and many more! Use dir('')

there are two under-scores on each side here

I should **underscore** this unusual syntax!

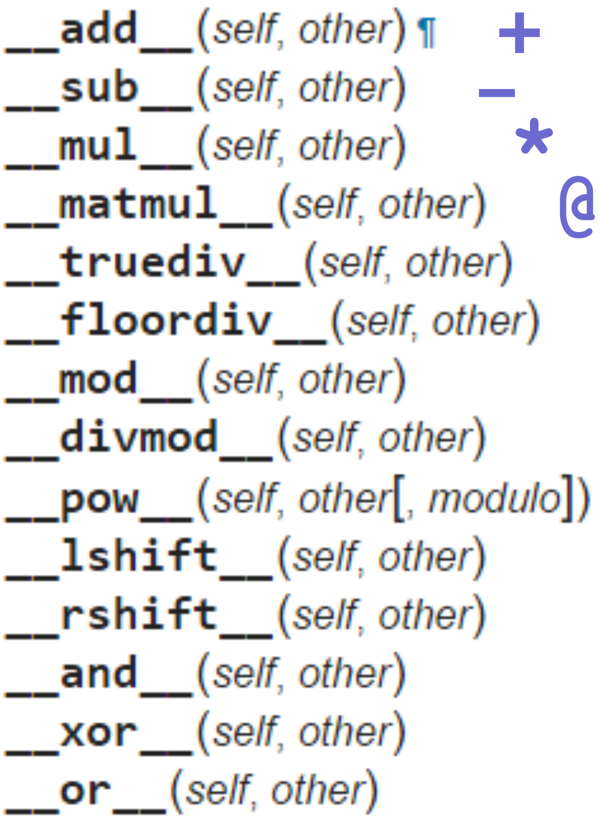
# *More operators!*

**Booleans**

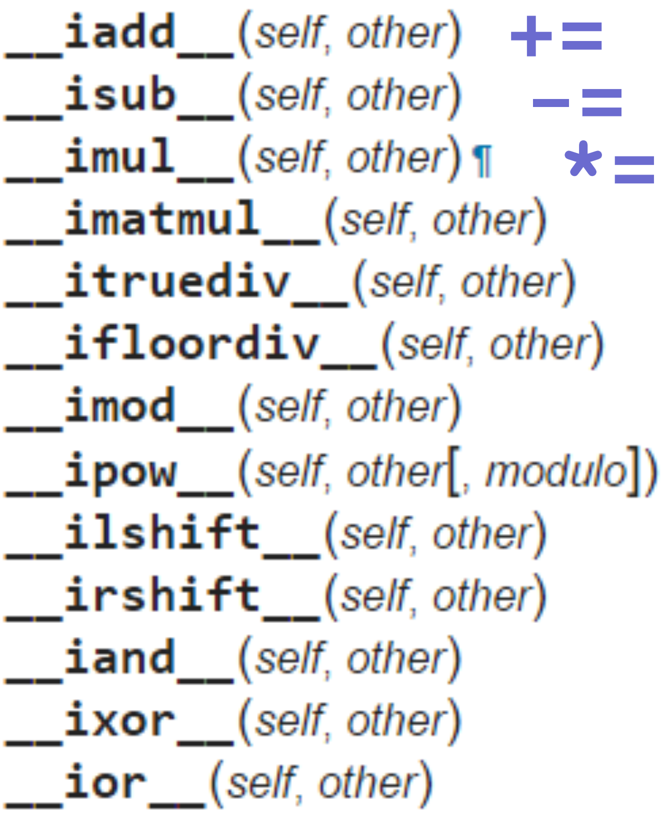


__lt__(self, other)
__le__(self, other)
__eq__(self, other)
__ne__(self, other)
__gt__(self, other)
__ge__(self, other)

**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)



__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 next week!

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!

# 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(11,8,2022).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 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
```

*Say **LESS** !*

# __lt__

<

*that's really* ***LESS*** *!*

```
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 ____.isBefore(____)
```

so LESS really is MORE!

# The two ***most timely*** *methods* ~

```
In1: wd = Date(11,12,2013)
In2: print(wd)
11/12/2013
```

construct with the **CONSTRUCTOR** ...

print uses **__repr__**

```
In1: wd.tomorrow()
In2: print(wd)
11/13/2013
```

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

`d += 1`

wd has changed!

```
In1: wd.yesterday()
In2: print(wd)
11/12/2013
```

**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 hw10pr1 this week!*

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

        self.day += 1
```

add 1 to **self.day**

Try writing tomorrow!

*Use this for hw10pr1 this week!*

```
class Date:

    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. 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
```

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

```
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
```

**Extra** Is there any *more* leap-year craziness available?!

```
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
```

Yes!

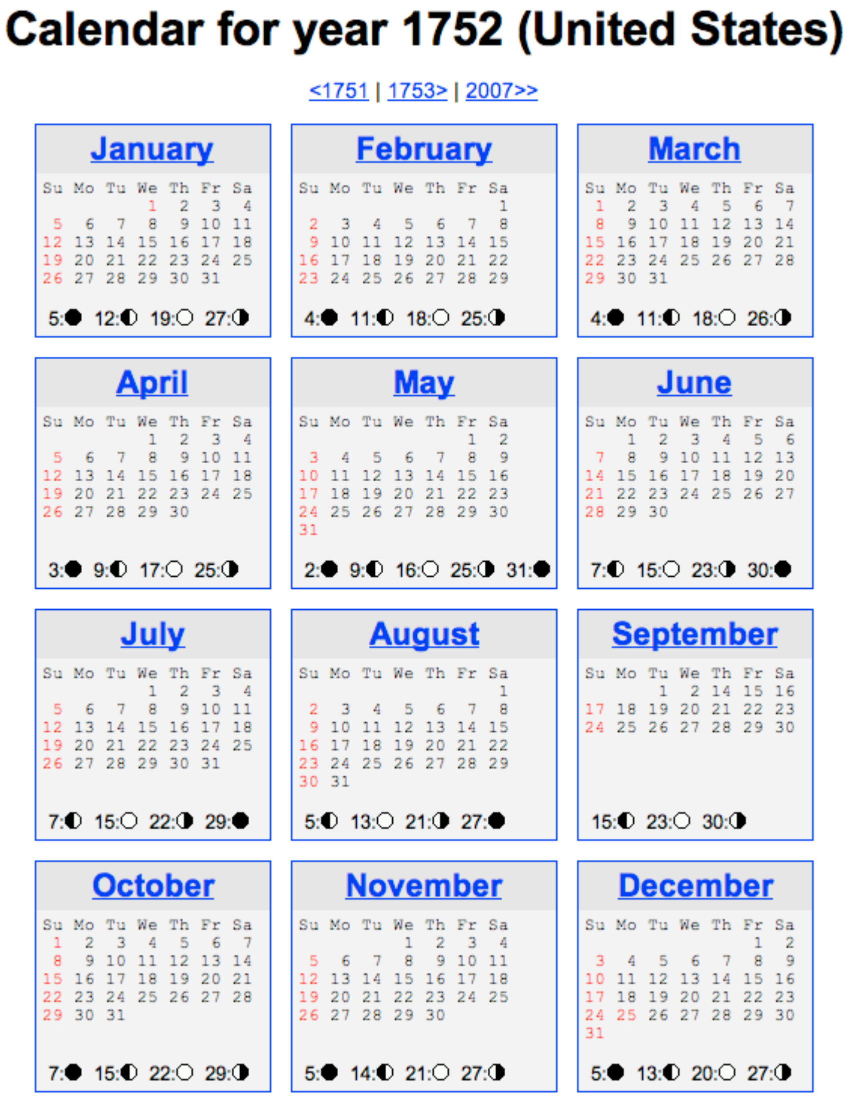
You'll take on yesterday --
tomorrow and today -- in lab...

*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]

    self.day
```

**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

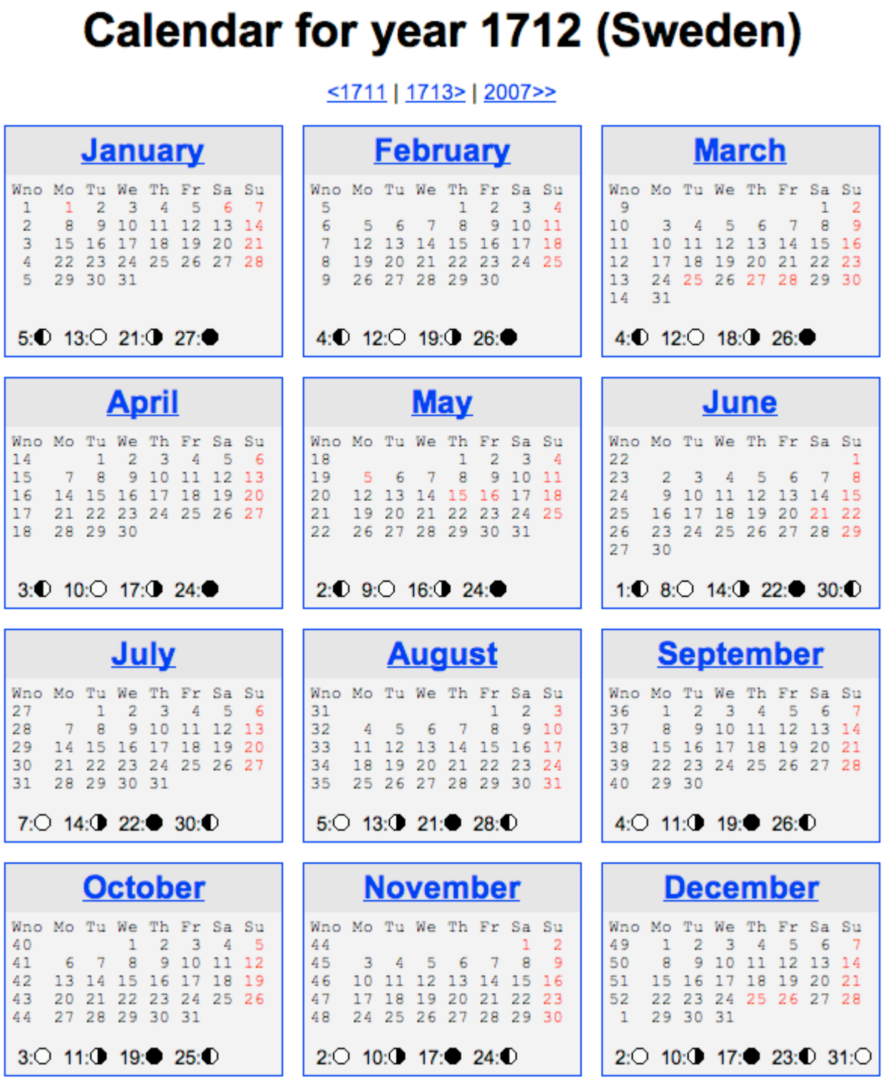
| Su | Mo | Tu | We | Th | Fr | Sa |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | | |

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 |

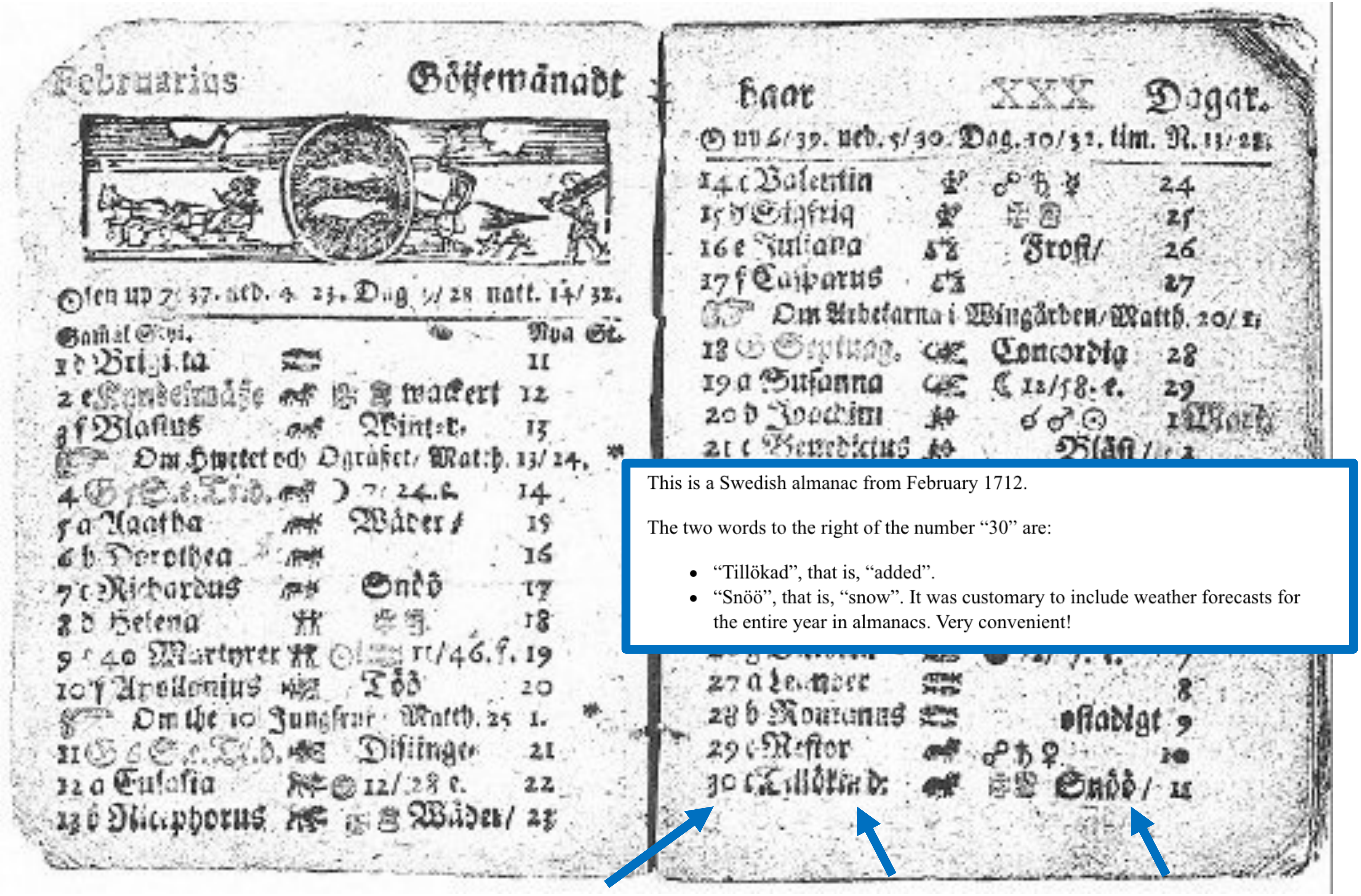
2:○ 10:◑ 17:● 24:◐

### 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:○

# Feb. 30, 1712



Februarius Göjemånadt

Solen up 7/37. ned. 4/23. Dag 9/28. natt. 14/32.

Gamla Styl. Nya St.
1 d Brigitta 11
2 e Kyndelsmäſſe wackert 12
3 f Blaſius Winter. 13
Om Hwetet och Ogräſet/ Matth. 13/24.
4 G 5. S. e. Trid. ☽ 7/24. f. 14
5 a Agatha Wäder/ 15
6 b Dorothea 16
7 c Richardus Snöö 17
8 d Helena 18
9 e 40 Martyrer ☉ 11/46. f. 19
10 f Apollonius Töö 20
Om the 10 Jungfrur Matth. 25 1.
11 G 6 S. e. Tr. d. Diſiingt 21
12 a Eulalia ☉ 12/28 e. 22
13 b Nicephorus Wäder/ 23

haar XXX Dagar.

☉ up 6/39. ned. 5/30. Dag. 10/51. tim. N. 13/28.
14 c Valentin ☍ ♄ ☿ 24
15 d Sigfrid 25
16 e Juliana Froſt/ 26
17 f Caſparus 27
Om Arbetarna i Wingården/ Matth. 20/1.
18 G Septuag. Concordia 28
19 a Suſanna ☾ 12/58. e. 29
20 b Joachim ☌ ♂ ☉ 1 Martz
21 c Benedictus Bläſt/ 2
27 a Leander 8
28 b Romanus oſtadigt 9
29 c Neſtor ☍ ♄ ♀. 10
30 c Tillökad. Snöö/ 11

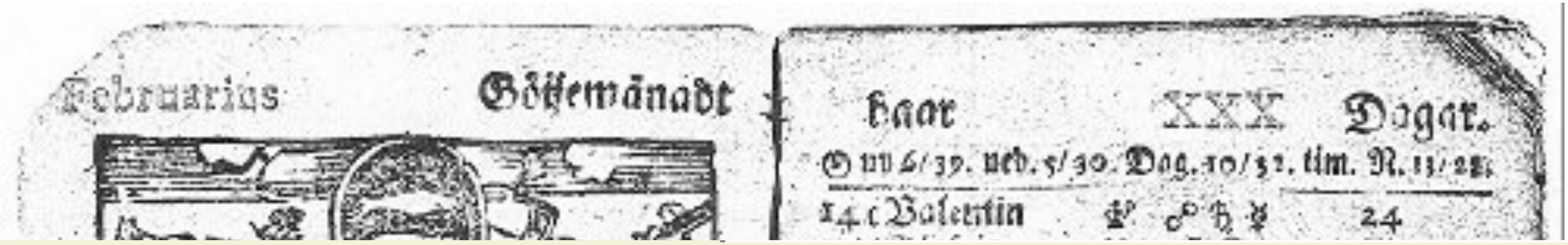
This is a Swedish almanac from February 1712.

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

# Feb. 30, 1712

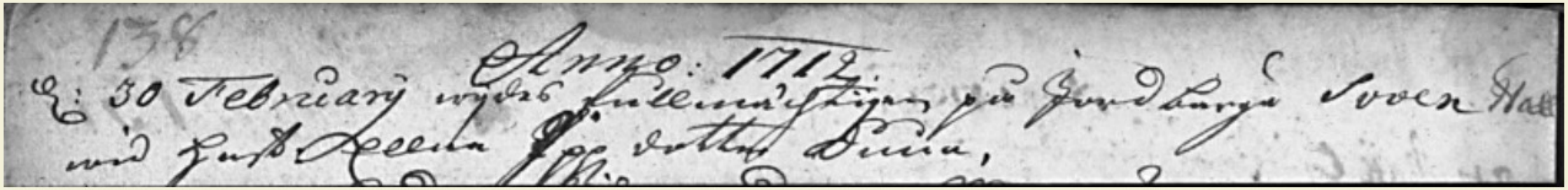


Februarius Göſſemånadt | Haar XXX Dagar.

☉ upp 6/39. ned. 5/30. Dag. 10/51. tim. N. 13/28.

14 c Valentin 24

The image below is a copy from the church registry in St. Petri Parish in the Swedish town of Ystad.[1]

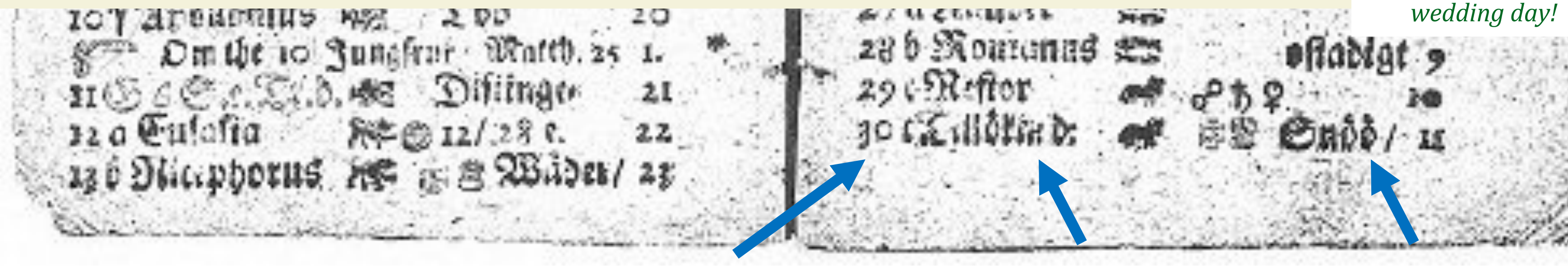


138

Anno: 1712.

d: 30 Februarij wijdes fullmächtigen på Jordbärga Svven Hall wid hust Elena Jäppdotter Duue,

The text reads: *Anno 1712. d: 30 Februarij wijdes fullmächtigen på Jordbärga Svven Hall wid hust Elena Jäppdotter Duue.* (That is: "Anno 1712. On 30 February the clerk Svven Hall of Jordbärga was married to Elena Jäppdotter Duue.")



| | | |
|---|---|---|
| 20 f Arcadius 2 DD 20 | | 27 a ... |
| Om the 10 Jungfrur Matth. 25 1. | | 28 b Romanus oſtadigt 9 |
| 21 ☉ 6 S. e. T. d. Diſiinge 21 | | 29 c Neſtor ♂ ♄ ♀ 10 |
| 22 a Eulaſia ☾ 12/28 e. 22 | | 30 d Tillökt d: Snöö/ 11 |
| 23 b Nicephorus Wäder/ 23 | | |

Now, that's a unique wedding day!

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