A whole new class of programming

CS's building blocks: functions and composition

CS 5 overview

behind CS's curtain: circuits, assembly, loops

Designing Data!

CS: theory + practice

Lists ~ 2D data

A = [ [1,2,3,4], [5,6], [7,8,9,10,11] ]

Where's 3? len(A) 3 len(A[0]) 2 Replace 10 with 42.

2 in-a-row?

```python
def two_in_a_row(A):
    NROWS = len(A)
    NCOLS = len(A[0])
    B = deepcopy(A)
    for r in range(0, NROWS):
        for c in range(0, NCOLS):
            if c == NCOLS - 1:
                B[r][c] = False
            elif A[r][c] == A[r][c+1]:
                B[r][c] = True
            else:
                B[r][c] = False
```

Challenge: How could we change the code above to check for two-in-a-row SOUTHWARD or DIAGONALLY?!

A = [ [4, 2, 2, 2], [2, 2, 4, 4], [2, 4, 2, 2] ]

What does two_in_a_row(A) place into B?

First, try it by eye... ... then, on hw9pr2, w/Python!

A = [ [" ", "X", "O", ",", ",", "O" ],
    ["X", "X", "X", ",", ",", "O" ],
    [, "X", ",", "O", "X", ",", "X" ],
    ["X", ",", ",", "O", "O", ",", "" ]
]

inarow_3east('X', 1, 0, A)
Lec 19 ~ 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)

Everything in Python is an object!

Its capabilities depend on its class.

Functions "methods"

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

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.

Designing a student class!

Data contained

| name | year |

Functions contained

- defer(numyrs)
- and others needed by Python __init__, __repr__
Everything is an object! strings, for example:

In : s = str( 42 )   This calls the str constructor.
In : type(s)   Shows the type of s is str
In : dir(s)   Shows all of the methods (functions) of s

A Date class and object, d

Let's try some!
Quizzes ~ naming
point each name to its piece of the code...

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

Extra: when’s the next leap year?  Is 2100 a L.Y.?
Extra: what should ny – today be?  What about nc – d?

You’ll create a Date class with

methods
operators!

What's the diff?

In : today = Date(11,13,2018)
In : wd = Date(11,12,2013)
In : today.diff(wd)
Out: 1827

In : today - wd
Out: 1827

In : wd - today
Out: -1827

In : eraday = Date(1,1,1)
In : today.diff(eraday)
Out: 737010

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.

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

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

In : today = Date(11,13,2018)
In : wd = Date(11,12,2013)
In : today.diff(wd)
Out: 1827

In : today - wd
Out: 1827

In : wd - today
Out: -1827

In : eraday = Date(1,1,1)
In : today.diff(eraday)
Out: 737010

In : today - eraday
Out: 737010
Where's the dow?

```python
In : sml = Date(10,28,1929)
Out: 'Monday'
In : sm2 = Date(10,19,1987)
Out: 'Monday'
In : sm1.dow()
Out: 'Monday'
In : sm2.dow()
Out: 'Monday'
In : Date(1,1,1).dow()
Out: 'Monday'
In : Date(1,1,2100).dow()
Out: 'Friday'
In : Date(10,10,2010).dow()
Out: 'Sunday'
```

The dow looks down to me!

uses a named object...

unnamed!

uses a named object...

unnamed!

popular!

The Date class

def __init__( self, mo, dy, yr ):
    self.month = mo
    self.day = dy
    self.year = yr

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.

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

Special Dates?

```python
>>> d = Date(11,12,2013)
>>> print d
11/12/2013
>>> d.isLeapYear()
False

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

self is the variable calling a method

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

The New York Times

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

By JOHN SCHWARTZ  OCT 1, 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.
Problems with ==

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

Python objects are handled by reference...

== compares references!

How can this be False?

diagram

```python
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
```

To use this, write `wd.equals(wd2)`

DIY operators ...

```python
__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('')`
```

I should underscore this unusual syntax!

More operators!

```python
__add__(self, other) +
__sub__(self, other) -
__matmul__(self, other) @
__rmatmul__(self, other) @
__mod__(self, other) %
__rmod__(self, other) %
__pow__(self, other) **
__rpow__(self, other) **
__floordiv__(self, other) //
__rfloordiv__(self, other) //
```

And many more, use `dir('')`

DIY operators...

I should underscore this unusual syntax!
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(11,13,2018).isBefore(Date(12,31,1999))
Date(12,31,1999).isBefore(Date(11,13,2018))

Why doesn't this function work correctly?!

The 2 most essential methods

>>> wd = Date(11,12,2013)   # construct with the CONSTRUCTOR ...
>>> print(wd)               # print uses __repr__
11/12/2013

>>> wd.tomorrow()           # the tomorrow method returns nothing at all. is it doing anything?
11/13/2013

>>> print(wd)               # wd has changed!
11/12/2013

>>> wd.yesterday()          # yesterday is pretty much just like tomorrow (is this a good thing!?)
11/12/2013

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

__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

See you @ next week's lab ...

... it's a Date!

L.A. street sign with typo from 2006
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

d = Date(11,12,2013)
today = Date(11,13,2018)
y = Date(1,1,2019)
nc = Date(1,1,2100)

Quiz ~ naming

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

class keyword (keyword)

class definition (end)

object creation (4)

methods (3)

constructor

data member (3)

Extra: what prints Dates?

Extra: when's the next leap year? Is 2100 a L.Y.?

Extra: what should ny - today be? What about nc - d?

Your name(s) ________________________________