CS 4 this week

Building classes...

hw10pr2

Connect Four Board class

... vs. using the library

hw10pr3

file and dictionary classes

files
dictionaries

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

Hw #10 due 4/15

Office hours ~ Thursday aft. in LAC
Classes: DIY data

**Class:** a user-defined datatype

**Object:** data or a variable whose type is a class

OOP!

object-oriented programming
Classes: DIY data

**Class:** a user-defined datatype

**Object:** data or a variable whose type is a class

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

```python
d = Date(12, 31, 2018)
d.tomorrow()
print(d)
```

\[ d \text{ would be named } \texttt{self} \text{ inside the Date class...} \]
Why classes?

Python has no Connect-four datatype...

Care for a game?

... and now we can fix that!
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

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

Why self instead of d?

Why is everything so far away?!
Problems with ==

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

>>> 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:

- **wd**
  - month: 11
  - day: 12
  - year: 2013
  - memory location: ~42042778

- **wd2**
  - month: 11
  - day: 12
  - year: 2013
  - memory location: ~42042742

`==` compares memory locations, not contents
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
        self.month == d2.month
            self.day == d2.day:
                return True
    else:
        return False

To use this, write \texttt{wd.equals(wd2)}
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)
Solution: **equals**

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

>>> 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?!
```python
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

To use this, write d == d2

L==k! This is T== C==L!
```
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 underscores on each side here

I should underscore this unusual syntax!
More operators!

**arithmetics**

- `__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)

**in-place arithmetic**

- `__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)

[https://docs.python.org/3/reference/datamodel.html#special-method-names](https://docs.python.org/3/reference/datamodel.html#special-method-names)
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(12, 31, 1999).isBefore(Date(11, 13, 2018))

Date(11, 13, 2018).isBefore(Date(12, 31, 1999))

Why doesn't this function work correctly?!
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
class Date:

def __lt__(self, d2):
    '''this is less than most code! '''
    return self.isBefore(d2)
class Date:

def __lt__(self, d2):
    """ this is less than most code! ""
    return self.isBefore(d2)

def __gt__(self, d2):
    """ this is less than most code! ""
    return d2.isBefore(self)
```python
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 > DIM[self.month]:
        self.day = 1
        self.month += 1

        if self.month == 13:
            self.month = 1
            self.year += 1

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

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, 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
Data design...

(Data Members) What data do we need?

- # rows
- # cols
- pieces ➔ characters 'X', 'O'
- data = the grid of chars

(Methods) What capabilities do we want?

- check for wins ➔ different directions
- strategy: TBD
- add more
- clear
- check for tie (full board)
Our Board object, \( \mathbf{b} \)

```
\textbf{b.data}[0][1] = 'X'
\textbf{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?
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) ]
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) ]

This list comprehension lets us create
    H  independent rows with
    W  independent columns each.
```python
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)*'-'

    return s
```

# what kind of loop will add the col #'s here?
```
class Board:
    def addMove(self, col, ox):
        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
```

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

(2) **Bugs!** Can you fix them?!

Name(s) _____________________________
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

(1) Run b.addMove(3,'O')

(2) Bugs!

Try this on the back page first...
Let's understand this `allowsMove` method ...

If it's in-bounds and not full, return **True**.

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>O</td>
<td></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>O</td>
<td></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>O</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>O</td>
</tr>
</tbody>
</table>

"b3.allowsMove(0) == True"
"b3.allowsMove(1) == True"
"b3.allowsMove(2) == False"
"b3.allowsMove(3) == False"
"b3.allowsMove(4) == True"
"b3.allowsMove(5) == True"
"b3.allowsMove(6) == False"
"b3.allowsMove(7) == False"
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:
        return False

    elif (top row full):
        return False
    else:
        return True

Let's finish this allowsMove method ...

shortcuts!

If col is \textit{out-of-bounds} or \textit{full}, return False.

If it's \textit{in-bounds} and not full, return True.
hw10pr2: Board class

- __init__( self, width, height )
- allowsMove( self, col )
- addMove( self, col, ox )
- delMove( self, col )
- __repr__( self )
- isFull( self )
- winsFor( self, ox )
- hostGame( self )

Which are similar to others? Which requires the most thought?
def winsFor(self, ox):
    """ does ox win? ""
    H = self.height
    W = self.width
    D = self.data

    for row in range(H):
        for col in range(W):
            if D[row][col] == ox:
                if row == col:
                    if row == H - 1 - col:
                        if D[row][col] == ox:
                            return True
    return False

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

Watch out for corner cases!
Why objects and classes?

Elegance: Objects hide complexity!

```python
if b.winsFor( 'X' ) == True:
    rem = self.diff( d2 ) % 7
```

Simple – and INVITING -- building blocks!
CS 4 this week

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

Hw #10 due 11/19

If I had a dictionary, I guess I could look up what it was!
CS 4 this week

Building classes...

 hw10pr2

... vs. using the library

 hw10pr3

Connect Four Board class

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

By SHELLEY PODOLNY  MARCH 7, 2015

Hw #10 due 11/13

Office hours on FRIDAY aft. in LAC 4
Algorithmic Authorship... ?

I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

suppose this text represents my "style" ...

How could a program author new prose in this same style?!
Algorithmic Authorship... !

suppose this text represents my "style" ...
Algorithmic authoring *examples*...

'Cause somethin' like he left knee and a harp," said he had to the whole school? The shouting and then some strange and Mrs. "Well, I know Hagrid; they spotted handkerchief and get him get rid of course, had a gigantic beet with her," he knew what to all he's

Wanna live while we're cool, so tonight What a feeling to be doing what I wish I know we only met but it ain't hard to be nothing left The story of my life I'm watching her eyes smile you flip your eyes You don't know what makes you got stars, they're in the wire She said, "Can I got a feeling to be a dentist

Who's the original human author of each of these?

This is but ourselves. No, faith, My uncle! O royal bed of confession Of your rue for leave to nature; to this time I should weep for thy life is rotten before he is. have sworn 't. Or my blood. I have closely sent for nine; and unprofitable,

The Senators and the date of a written declaration that Purpose, they shall consist of nine States, shall not, when he shall have such Vacancies. The President pro tempore, in the Desire of a Qualification to the Speaker of the Senate. Article 6. When vacancies by the office upon probable
Markov Models

Techniques for modeling *any* sequence of natural data

speech, text, sensor data...

1st-order Markov Model
(defined property)

Each item depends *only* on the one immediately before it.
Lists are *sequential* containers:

\[ L = [ 47, 5, 47, 42 ] \]

elements are looked up by their location, or index, starting from 0

Dictionaries are *arbitrary* containers:

\[ d = \{ 47 : 2, 42 : 1 \} \]

elements (or values) are looked up by a key starting anywhere you want! Keys don't have to be ints!

We need a new data structure!

(A new class...)

keys, values
Lists are *sequential* containers:

\[ L = [47, 5, 47, 42] \]

elements are looked up by their *location*, or *index*, starting from 0

dictionaries are *arbitrary* containers:

\[ d = \{47: 2, 42: 1\} \]

elements (or *values*) are looked up by a *key* starting anywhere you want! *Keys* don't have to be ints!
Dictionaries are *arbitrary* containers:

```
zd = {'rabbit': 1999, 'ox': 1997}
```

Elements (or *values*) are looked up by a *key* starting anywhere you want! *Keys* don't have to be ints!

What's *zd*'s data here?

Now I see the *key* to dictionaries' *value*...
Dictionaries are *arbitrary* containers:

\[
zd = \{ 'rabbit': 1999, 'ox': 1997 \}\]

Elements (or *values*) are looked up by a *key* starting anywhere you want! *Keys* don't have to be ints!

Now I see the *key* to dictionaries' *value*...

<table>
<thead>
<tr>
<th>Animal</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rat</td>
<td>Feb 19 1996–Feb 06 1997</td>
</tr>
<tr>
<td>Ox</td>
<td>Feb 07 1997–Jan 27 1998</td>
</tr>
<tr>
<td>Tiger</td>
<td>Jan 28 1998–Feb 15 1999</td>
</tr>
<tr>
<td>Rabbit</td>
<td>Feb 16 1999–Feb 04 2000</td>
</tr>
<tr>
<td>Dragon</td>
<td>Feb 05 2000–Jan 23 2001</td>
</tr>
<tr>
<td>Snake</td>
<td>Jan 24 2001–Feb 11 2002</td>
</tr>
<tr>
<td>Horse</td>
<td>Feb 12 2002–Jan 31 2003</td>
</tr>
<tr>
<td>Goat</td>
<td>Feb 01 2003–Jan 21 2004</td>
</tr>
<tr>
<td>Monkey</td>
<td>Jan 22 2004–Feb 08 2005</td>
</tr>
</tbody>
</table>

12-year zodiac...
Dictionaries are *arbitrary* containers:

\[
z = \{ 'rabbit': [1999, 1987, 1975, \ldots], \\
'ox': [1997, 1985, 1973, \ldots], \\
'tiger': [1998, 2010, \ldots] \}
\]

What type are the keys?  What type are the values?

*Whose keys? z's keys!*
Dictionaries are *arbitrary* containers:

$$z = \{ '\text{rabbit}': [1999, 1987, 1975, \ldots],$$

$$ '\text{ox}': [1997, 1985, 1973, \ldots],$$

$$ '\text{dragon}': [2000, 1988, 1976, \ldots],$$

$$ \ldots \}$$

Is 'dragon' a key in \( z \)?

Is 1969 in \( z[\text{'dragon'}] \)?
Dictionaries are in

\[
z = \{ \text{'rabbit':[1999,1987,1975,...]}, \\
     \text{'ox':[1997,1985,1973,...]}, \\
     \text{'tiger':[1998,2010,...]}, \ldots \}
\]

Is 'dragon' a key in \(z\)?

if 'dragon' in z

Is 1969 in \(z[\text{'dragon'}]\)?

if 1969 in z[\text{'dragon'}]
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]

d = {}

for w in LoW:
    if w not in d:
        d[w] = 1
    else:
        d[w] += 1

Final d will be...

{}  

{ 'poptarts':1, 'spam':3 }
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]

```python
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]

D = {}
for w in LoW:
  if w not in D:
    D[w] = 1
  else:
    D[w] += 1

d will be...

{ }
{ 'spam':1 }
{ 'poptarts':1, 'spam':2 }
{ 'poptarts':1, 'spam':3 }

vc_print(LoW)
vc_print("a.txt")
```
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]

```python
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]

d = {}
for w in LoW:
    if w not in d:
        d[w] = 1
    else:
        d[w] += 1

print(d)  # Output: {'spam': 2}
print(d)  # Output: {'poptarts': 1, 'spam': 2}
print(d)  # Output: {'poptarts': 1, 'spam': 3}
```

Oldenborg's menu!

but where to get so many words?

FILES!

```
vc_print(LoW)
vc_print("a.txt")
```
Files...

```python
f = open('a.txt')
opens the file and calls it f

text = f.read()
reads the whole file into the string text

f.close()
closes the file (optional)

text
'I like poptarts and 42 and spam.\nWill I

LoW = text.split()
[ 'I', 'like', 'poptarts', ... ]
```

In Python reading files is smooth...
def word_count(filename):
    
f = open(filename)
    text = f.read()
    f.close()

    LoW = text.split()
    print("There are",len(LoW),"words")

What if we wanted the number of different words in the file?

This would be the author's vocabulary count, instead of the total word count.
Vocabulary, anyone?

Shakespeare used **31,534 different words** -- and a grand total of 884,647 words, counting repetitions across all of his works....

http://www.math.cudenver.edu/~wbriggs/qr/shakespeare.html

**Shakespearean coinages**

- gust
- besmirch
- unreal
- superscript
- watchdog
- swagger

- affined
- rooky
- attasked
- out-villained

---

*There's also one contemporary* British author in the Oxford English Dictionary...

Who? with what word?

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html
Vocabulary, anyone?

Shakespeare used **31,534 different words** -- and a grand total of 884,647 words, counting repetitions across all of his works.

http://www.math.cudenver.edu/~wbriggs/qr/shakespeare.html

Shakespearean coinages:

- gust
- besmirch
- unreal
- superscript
- watchdog
- swagger
- affined
- rooky
- attasked
- out-villained
- successful
- unsuccessful

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html

---

'muggle' goes into Oxford English Dictionary

JK Rowling's word for non-wizards - "muggle" - has made it into the new edition of the Oxford English Dictionary (OED).

The draft definition according to the dictionary’s website says:

- **Muggle**: invented by JK (Joanne Kathleen) Rowling (b. 1965), British author of children's fantasy fiction (see quot. 1997).

  In the fiction of JK Rowling: a person who possesses no magical powers. Hence in allusive and extended uses: a person who lacks a particular skill or skills, or who is regarded as inferior in some way.

J. K. Rowling
from filename import defaultdict

def vocab_count( filename ):
    f = open( filename )
    text = f.read()
    f.close()

    LoW = text.split()
    print "There are", len(LoW), "words."

    d = {}

    for w in LoW:
        if w not in d:
            d[w] = 1
        else:
            d[w] += 1

    print "There are", len(d), "_distinct_ words.\n"

    return d  # return d for later use by other code...

Tracking the number of occurrences of each word with a dictionary, d.

Same as before...
Markov Models can be **generative**!

A key benefit of Markov Models is that they can *generate* feasible data!

Original file:

I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

d = create_model('hpwhich.txt')
d = create_model('randj.txt')
d = create_model('oneD.txt')
d = create_model('a.txt')
gt(d,250)
Markov Models can be generative!

A key benefit of Markov Models is that they can generate feasible data!

Original file:
I like poptarts and 42 and spam. Will I get spam and poptarts for the holidays? I like spam poptarts!

Generated text:
I get spam poptarts! I like poptarts and 42 and spam. I like spam and 42 and 42 and 42 and 42 and spam. Will I like poptarts and 42 and poptarts and 42 and poptarts and 42 and 42 and poptarts and spam. I get spam and 42 and 42 and...
Our Markov Model

Try it!

Markov Model
A dictionary!

What are the keys?

What are the values?

What are the missing values?

What is the '$'?

Why do some keys seem missing?

{  
'\$': ['I', 'Will', 'I'],  
'I': ['like', 'get', 'like'],  
'like':  
'poptarts': ['and', 'for'],  
'and': ['42', 'spam.', 'poptarts'],  
'42': ['and'],  
'Will': ['I'],  
'the':  
'spam': ['and', 'poptarts!'],  
'get': ['spam'],  
'for': ['the']  
}
Our Markov Model

Try it!

{  
  '$': ['I', 'Will', 'I'],  
  'I': ['like', 'get', 'like'],  
  'like': ['poptarts', 'spam'],  
  'poptarts': ['and', 'for'],  
  'and': ['42', 'spam.', 'poptarts'],  
  '42': ['and'],  
  'Will': ['I'],  
  'the': ['holidays?'],  
  'spam': ['and', 'poptarts!'],  
  'get': ['spam'],  
  'for': ['the']  
}
Markov-modeling's algorithm

\textbf{LoW} \quad ['I','like','spam. I','eat','poptarts!']

\begin{verbatim}
d = {}
pw = '$'

for nw in LoW:
    if pw not in d:
        d[pw] = [nw]
    else:
        d[pw] += [nw]
pw = __________
\end{verbatim}

cdi_print(PT2)
cdi_print("a.txt")

\textbf{d}'s final form (without quotes)

\$ : [ I, I ]
I : [ like, eat ]
like : [ spam. ]
eat : [ poptarts! ]
Model creation:

1) start with the previous word, `pw` as '$'
2) for each next word, `nw`, in the list of words, add it in ...
3) then change `pw` to `nw` ...
   (a) except if `nw[-1]` was punctuation: change `pw` to...

Generating text:

1) start with `pw` as the '$' string
2) choose a `nw` that follows `pw`, at random.
3) print `nw`,
   (the comma continues on the same line)
4) `pw` gets set to either `nw` or '$'
   or if `nw[-1]` was punctuation: change `pw` to...
We invite you to submit a paper/abstract to The 15th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2011, to be held in Orlando, Florida, USA, on July 19th - July 22nd, 2011 (www.2011iiisconferences.org/wmsci)

If you have any colleagues who might be interested in making a submission to the conference, please feel free to forward this e-mail to them.

Below are the next deadlines for WMSCI 2011 (Check the web site for possible extensions or new set of deadlines):

Papers/Abstracts Submission and Invited Session Proposals: November 25th, 2010
Authors Notifications: January 31st, 2011
Camera-ready, full papers: February 28th, 2011
Dear Zachary Dodds,

We would like to inform you that we extended the submission deadline for your potential contribution in the area "Robotics" or any other included in the 21st World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2017 (http://www.2017iisconf.org/wmsci), to be held on July 8 - 11, 2017, in Orlando, Florida, USA, jointly with:

- The 11th International Multi-Conference on Society, Cybernetics, and Informatics: IMSCI 2017
- The 15th International Conference on Education and Information Systems, Technologies and Applications: EISTA 2017
- The 10th International Multi-Conference on Engineering and Technological Innovation: IMETI 2017

The respective web sites of the above events and the others being jointly organized can be found at the general CFP posted at: http://www.2017iisconf.org/cfp-summer2017.asp

To submit your article, please click the "Authors" tab on the conference website. Submissions for face-to-face and virtual presentations are both accepted.

WMSCI and all its collocated events are being indexed by Elsevier's SCOPUS since 2005. The 2017 proceedings will also be sent to Elsevier’s SCOPUS.
WMSCI 2005

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

http://pdos.csail.mit.edu/scigen/

Markov-generated submission accepted to WMSCI 2005

Not a first-order model … but a third-order model
Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification of virtual machines and real-time theory. To what extent can web browsers be constructed to achieve this purpose?

Certainly, the usual methods for the emulation of Smalltalk that paved the way for the investigation of rasterization do not apply in this area. In the opinions of many, despite the fact that conventional wisdom states that this grand challenge is continuously answered by the study of access points, we...

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-touted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in $\Omega((n + \log n))$ time [22]. In the end, we conclude.

II. ARCHITECTURE

Our research is principled. Consider the early methodology by Martin and Smith; our model is similar, but will actually overcome this grand challenge. Despite the fact that such a claim at first glance seems unexpected, it is buffeted by previous work in the field. Any significant development of secure theory will clearly require that the acclaimed real-time algorithm for the refinement of write-ahead logging by Edward Feigenbaum et al. [15] is impossible; our application is no different. This may or may not actually hold in reality.
There are no one-sided coins...

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

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 mailing list” was recently accepted for publication. Today I received an invitation from the International Conference on 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 suggestions. The text really does not make any sense. After adding the first illustration on nuclear physics from Wikipedia, some references and creating a fake identity (Iris Pear, aka Siri Apple) I submitted the paper which was accepted only three hours later! I know that iOS is a pretty good software, but reaching tenure has never been this close.

UPDATE (27/10/2016): Turns out that conference organizer, OMICS Group, is currently under federal investigation.

http://www.bartneck.de/2016/10/20/ios-just-got-a-paper-on-nuclear-physics-accepted-at-a-scientific-conference/
Have Python write your papers for you...

... you're still the author!

Project deadlines?
Papers due?

Have a \textit{worry-free} weekend!