CS 42 Today

ARTICLE

Received 21 Apr 2011 | Accepted 21 Sep 2011 | Published 18 Oct 2011

DOI: 10.1038/ncomms1516

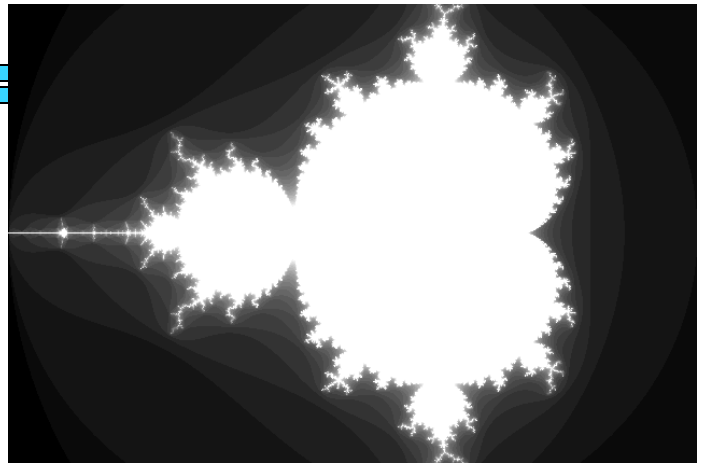
Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology

Baojun Wang[1], Richard I Kitney[1], Nicolas Joly[2,†] & Martin Buck[2]

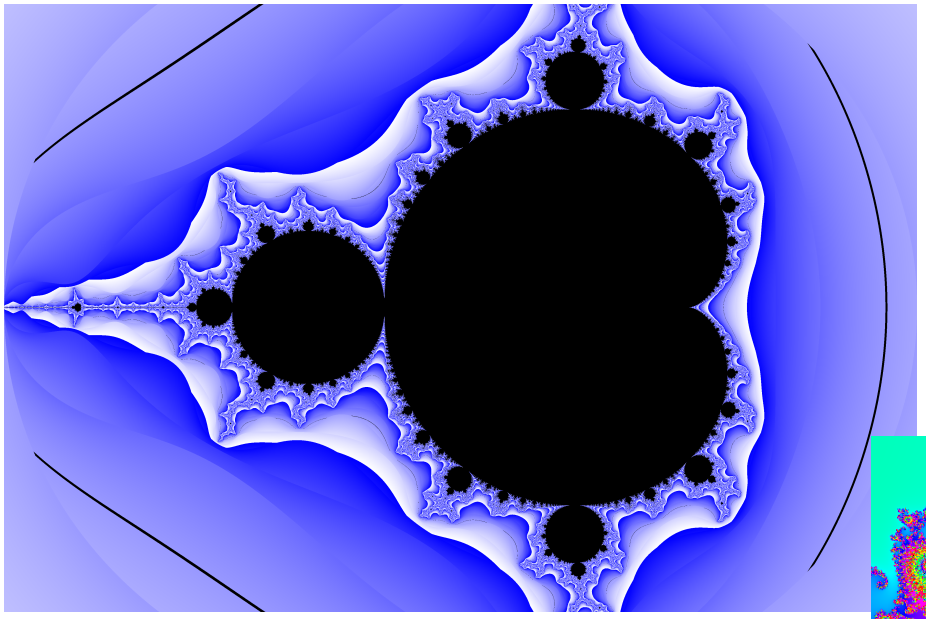
Modular and orthogonal genetic logic gates are essential for building robust biologically based digital devices to customize cell signalling in synthetic biology. Here we constructed an orthogonal AND gate in *Escherichia coli* using a novel hetero-regulation module from *Pseudomonas syringae*. The device comprises two co-activating genes *hrpR* and *hrpS* controlled by separate promoter inputs, and a σ^{54}-dependent *hrpL* promoter driving the output. The *hrpL* promoter is activated only when both genes are expressed, generating digital-like AND integration behaviour. The AND gate is demonstrated to be modular by applying new regulated promoters to the inputs, and connecting the output to a NOT gate module to produce a combinatorial NAND gate. The circuits were assembled using a parts-based engineering approach of quantitative characterization, modelling, followed by construction and testing. The results show that new genetic logic devices can be engineered predictably from novel native orthogonal biological control elements using quantitatively in-context characterized parts.

Mset 2010 hall of fame

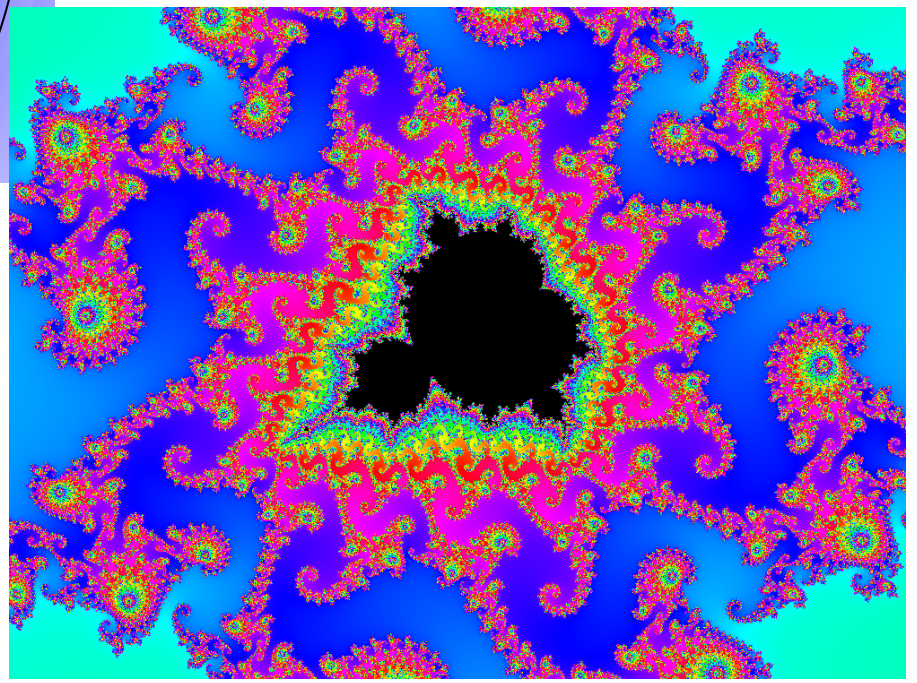
Spencer Alves



Patrick Meehan



Jacob Bandes-Storch



Ben Huppe

Markov Model

Technique for modeling any sequence of natural data

Each item depends ***on only the item immediately before it .***

1st-order Markov Model

The text file:

I like spam. I like toast and spam.
I eat ben and jerry's ice cream too.

For each word, keep track of the words that can follow it (and how often)

The Model:

I: like, like, eat
like: spam., toast
spam.: I, I
toast: and
eat: ben
and: spam., jerry's

ben: and
jerry's: ice
ice: cream
cream: too.
too.:

- What about second-order?
- Third order?

The Price Is Right!

Goal: Buy from a set of 5 items (as many of each as you want) while spending between $9.25 and $10.00.

Step 1: Identify Information To Store

- What information does this program need to keep track of?

Step 1: Identify Information To Store

- How much has the user spent?

  `float: money`
- Which items are available?

  `list of strings: items`
- How much does each available item cost?

  `list of floats: prices`
- Which item did the user currently choose?

  `int: choice`
- How many of the currently chosen item does the user want?

  `int: number`

Step 2: Break Down Functionality

- What are the things that this program needs to do?

Step 2: Break Down Functionality

- Control the overall game play
- Prompt the user to enter a choice and number of items
- Calculate the total spent
- Figure out when the game is over
- Figure out win or lose
- Print a welcome message
- Remove the purchased item from the lists

Attempt 1

```
def playTry1():
    print "Welcome to the price is right!"
    print "Your goal is to buy 5 or fewer items (any number of each)"
    print "and spend between $9.25 and $10"

    items = ["bleach","coke","ramen","ice cream","super ball"]
    prices = [1.35, .75, .25, 3.00, 1.75]
    money = 0.0

    while money < 9.25 and len(items) > 0:
        print "You have spent $", money
        printItems( items )
        choice = input( "Which item would you like to buy? ")
        number = input( "How many "+items[choice]+" would you like?")
        print items[choice], "is $", prices[choice]
        money += prices[choice]*number

        prices = prices[0:choice] + prices[choice+1:]
        items = items[0:choice] + items[choice+1:]

    print "You have spent $", money
    if money >= 9.25 and money <= 10.0:
        print "You win!"
    else:
        print "You lose!"
```

Name: _____ "QUIZ"

```
def playTry1():
    print "Welcome to the price is right!"
    print "Your goal is to buy 5 or fewer items (any number of each)"
    print "and spend between $9.25 and $10"

    items = ["bleach","coke","ramen","ice cream","super ball"]
    prices = [1.35, .75, .25, 3.00, 1.75]
    money = 0.0

    while money < 9.25 and len(items) > 0:
        print "You have spent $", money
        printItems( items )
        choice = input( "Which item would you like to buy? ")
        number = input( "How many "+items[choice]+" would you like?")
        print items[choice], "is $", prices[choice]
        money += prices[choice]*number

        prices = prices[0:choice] + prices[choice+1:]
        items = items[0:choice] + items[choice+1:]

    print "You have spent $", money
    if money >= 9.25 and money <= 10.0:
        print "You win!"
    else:
        print "You lose!"
```

This program works (mostly).
But how could the code be improved
without changing what the user sees?
(Think style, maintainability, readability, …)

Attempt 1: Issues

```
def playTry1():
    print "Welcome to the price is right!"
    print "Your goal is to buy 5 or fewer items (any number of each)"
    print "and spend between $9.25 and $10"

    items = ["bleach","coke","ramen","ice cream","super ball"]
    prices = [1.35, .75, .25, 3.00, 1.75]
    money = 0.0

    while money < 9.25 and len(items) > 0:
        print "You have spent $", money
        printItems( items )
        choice = input( "Which item would you like to buy? ")
        number = input( "How many "+items[choice]+" would you like?")
        print items[choice], "is $", prices[choice]
        money += prices[choice]*number

        prices = prices[0:choice] + prices[choice+1:]
        items = items[0:choice] + items[choice+1:]

    print "You have spent $", money
    if money >= 9.25 and money <= 10.0:
        print "You win!"
    else:
        print "You lose!"
```

Magic numbers!

Attempt 1a

```
def playTry1a():
    LIMIT_MIN = 9.25
    LIMIT_MAX = 10.00
    items = ["bleach","coke","ramen","ice cream","super ball"]
    prices = [1.35, .75, .25, 3.00, 1.75]
    money = 0.0
    print "Welcome to the price is right!"
    print "Your goal is to buy", len(items), "or fewer items…"
    print "and spend between $", LIMIT_MIN, "and $", LIMIT_MAX

    while money < LIMIT_MIN and len(items) > 0:
        print "You have spent $", money
        printItems( items )
        choice = input( "Which item would you like to buy? ")
        number = input( "How many "+items[choice]+" would you like?")
        print items[choice], "is $", prices[choice]
        money += prices[choice]*number

        prices = prices[0:choice] + prices[choice+1:]
        items = items[0:choice] + items[choice+1:]

    print "You have spent $", money
    if money >= LIMIT_MIN and money <= LIMIT_MAX:
        print "You win!"
    else:
        print "You lose!"
```

Attempt 1a: Issue

```
def playTry1a():
    LIMIT_MIN = 9.25
    LIMIT_MAX = 10.00
    items = ["bleach","coke","ramen","ice cream","super ball"]
    prices = [1.35, .75, .25, 3.00, 1.75]
    money = 0.0
    print "Welcome to the price is right!"
    print "Your goal is to buy", len(items), "or fewer items…"
    print "and spend between $", LIMIT_MIN, "and $", LIMIT_MAX

    while money < LIMIT_MIN and len(items) > 0:
        print "You have spent $", money
        printItems( items )
        choice = input( "Which item would you like to buy? ")
        number = input( "How many "+items[choice]+" would you like?")
        print items[choice], "is $", prices[choice]
        money += prices[choice]*number

        prices = prices[0:choice] + prices[choice+1:]
        items = items[0:choice] + items[choice+1:]

    print "You have spent $", money
    if money >= LIMIT_MIN and money <= LIMIT_MAX:
        print "You win!"
    else:
        print "You lose!"
```

Functionality is not broken out

Each function does one specific thing.
It's clear where we go to change aspects of the game.
Code is *self-commenting.*

```
def playBetter():
    """ Play the game """
    [items, prices] = initializeItems()

    LIMIT_HIGH = 10.00
    LIMIT_LOW = 9.50
    money = 0.0

    printIntro(LIMIT_LOW, LIMIT_HIGH, items)

    while not allDone(items, LIMIT_LOW, money):
        print "You have spent $", money
        [items, prices, spent] = playRound(items, prices)
        money += spent

    winOrLose(money, LIMIT_LOW, LIMIT_HIGH)
```

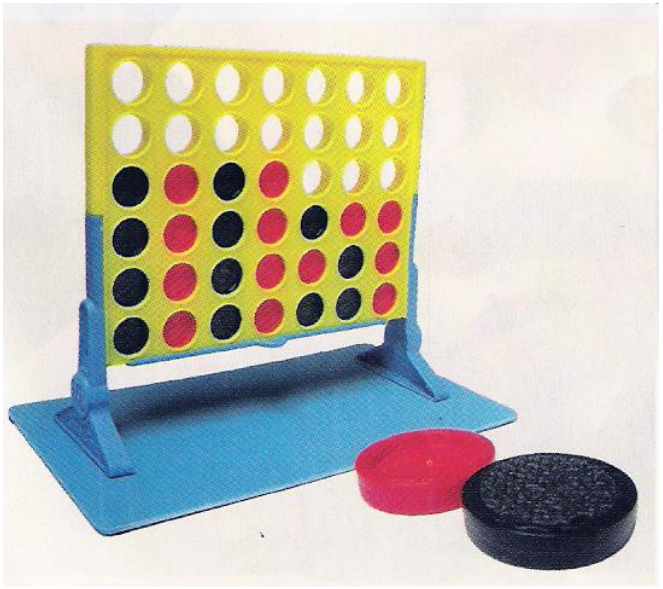
Functions can return more than one thing in a list

```
def playRound(items, prices):
    """ Play one round of the game
        Inputs: A list of items and a list of prices
        Returns: [items, prices, spent] where
                 items is list of items remaining,
                 prices is a list of remaining prices,
                 and spent is the amount spent this round"""
    print "**********************"
    printItems( items )
    choice = input( "Which item would you like to buy? ")
    number = input( "How many " + items[choice] + " would you like? ")
    print items[choice], "is $", prices[choice]

    spent = prices[choice]*number
    prices = prices[0:choice] + prices[choice+1:]
    items = items[0:choice] + items[choice+1:]

    return [items, prices, spent]
```

You CAN AND SHOULD modify your code as you go
to make it cleaner and better organized.
You DON'T have to get it perfect the first time.
(No one else does.)

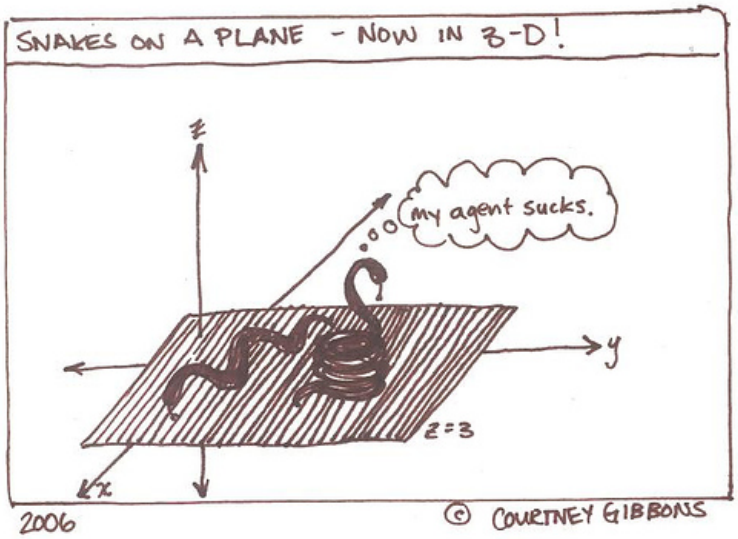
```
def playBetter():
    """ Play the game """
    [items, prices] = initializeItems()

    LIMIT_HIGH = 10.00
    LIMIT_LOW = 9.50
    money = 0.0

    printIntro(LIMIT_LOW, LIMIT_HIGH, items)

    while not allDone(items, LIMIT_LOW, money):
        print "You have spent $", money
        [items, prices, spent] = playRound(items, prices)
        money += spent

    winOrLose(money, LIMIT_LOW, LIMIT_HIGH)
```

Objects, Objects Everywhere



Connect 4!

Spam!

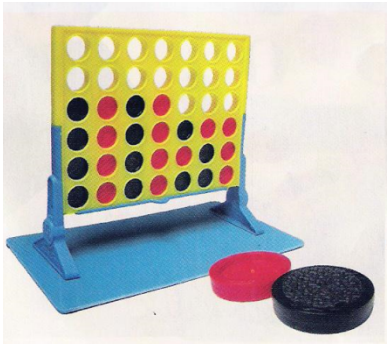


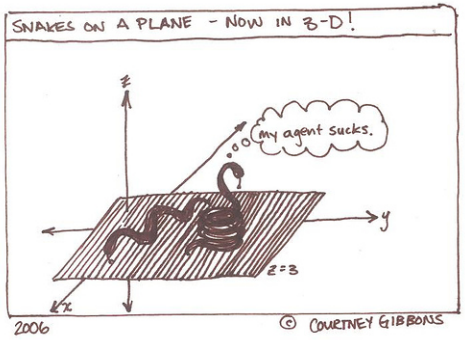
Spam lists? Not going to cut it.

Geometric Objects!

And much more…

OOP: Where Data is King





User-Defined Classes & Objects

A user-defined object is a data structure where:

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

(2) Data elements are chosen & organized by the programmer

(3) *An object can have behaviors built-in by the programmer.*

usually called "methods"

(1) A *class* is a **type** of variable.

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

There will typically be MANY objects of a single class.

Python is Object Oriented

Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects. (In a sense, and in conformance to Von Neumann's model of a ``stored program computer," code is also represented by objects.)
--Python Reference Manual

Some are mutable, e.g.

```
list
dictionary
user-defined classes
```

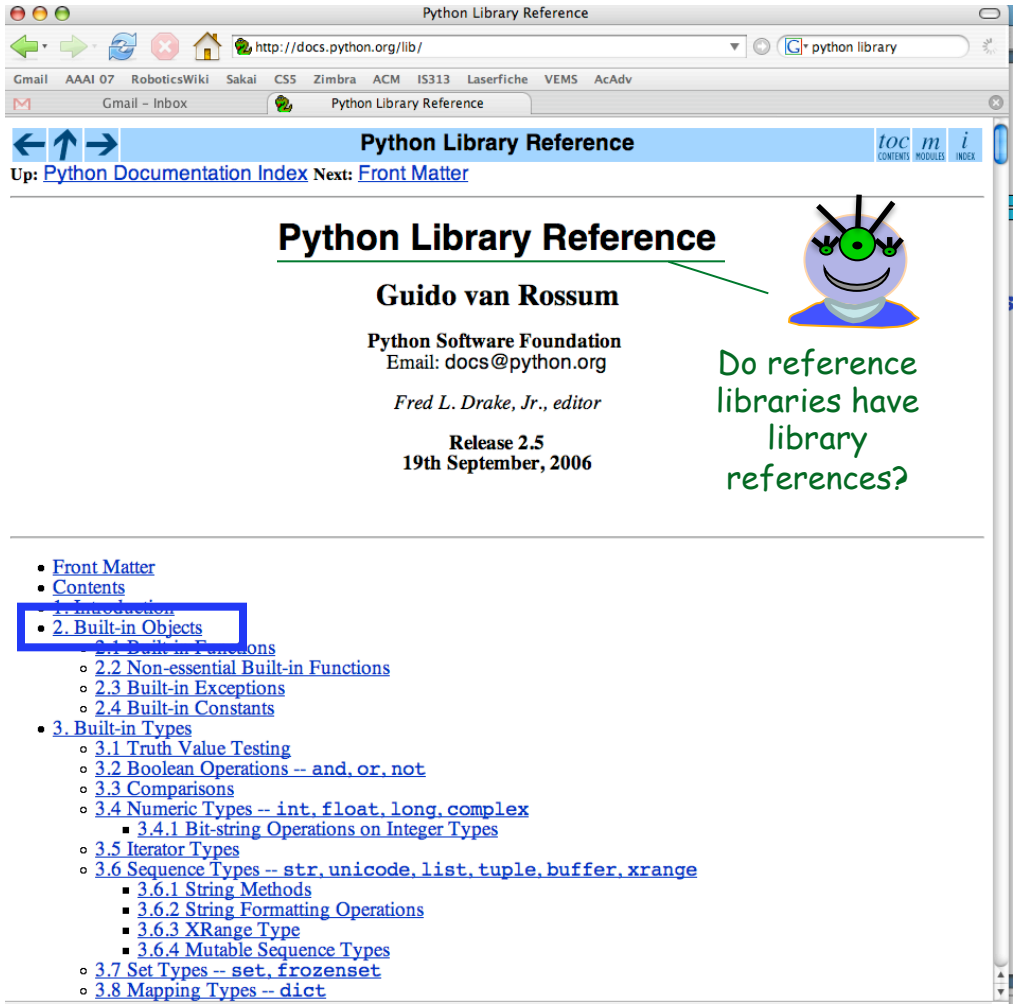
Others are not, e.g.

```
string
tuple
numbers
bool
```

Consequences?

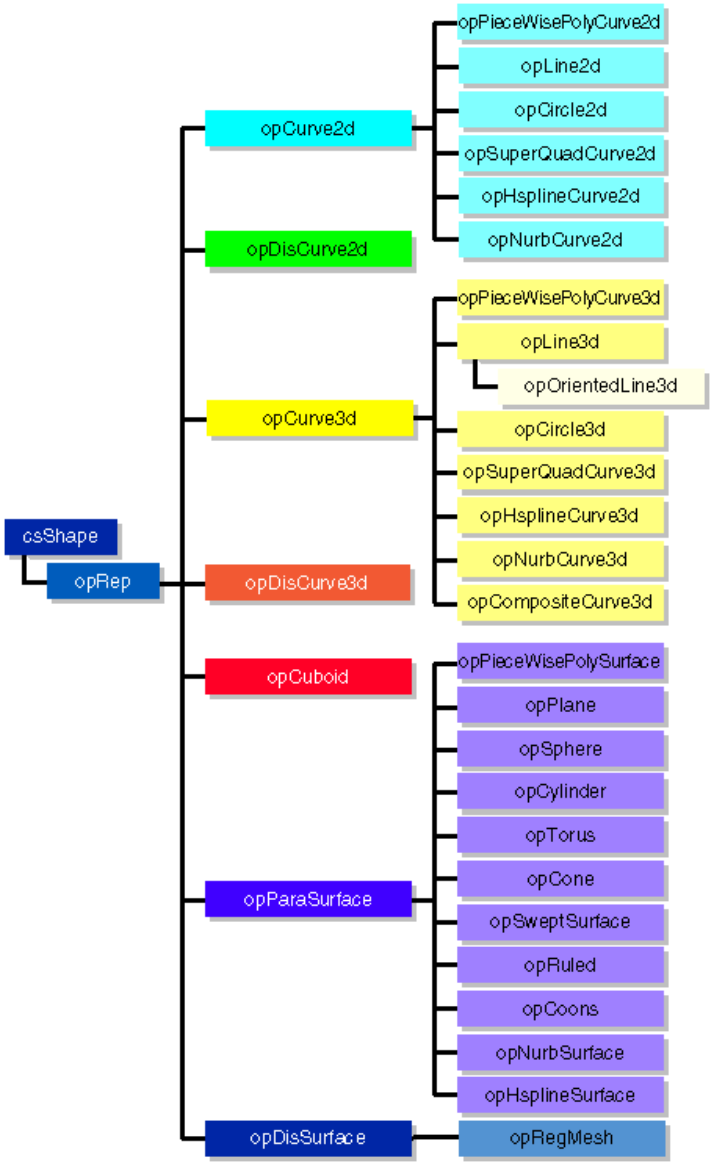
Examples…

Python's class libraries…



http://docs.python.org/lib/

Graphics libraries



Using objects and classes:

A particularly *complex* example…

```
>>> z = 3 + 4j

>>> dir(z)
```

all of the data members and methods of the complex class (and thus the object z !)

```
>>> z.imag
```

a data member of all objects of class **complex**

```
4.0
```

its value for this object, **z**

```
>>> z.conjugate()
```

a method (function) within all objects of class **complex**

```
3-4j
```

its return value for this object, **z**

The `Point` of Python: defining objects and classes

The `Point` of Python: defining objects and classes

syntax and idiosyncrasies

	Java	Python
types	**static**: all types explicitly declared and are checked when compiling	**dynamic**: types are checked at run-time
references	all data handled ***by reference*** except `boolean`, `int`, `long`, `double`, `char`, `float`, `byte`, `short` the 8 "primitive types"	all data is handled ***by reference***

Objects	`= new Object()`
	`this`
	`null`
		`= Object()`
		`self`
		`None`

equality	`s1.equals(s2)`
	compares Objects' contents
	`s1 == s2`
	compares references or primitive types
		`s1 == s2`
		compares Objects' contents
		there are no primitive types

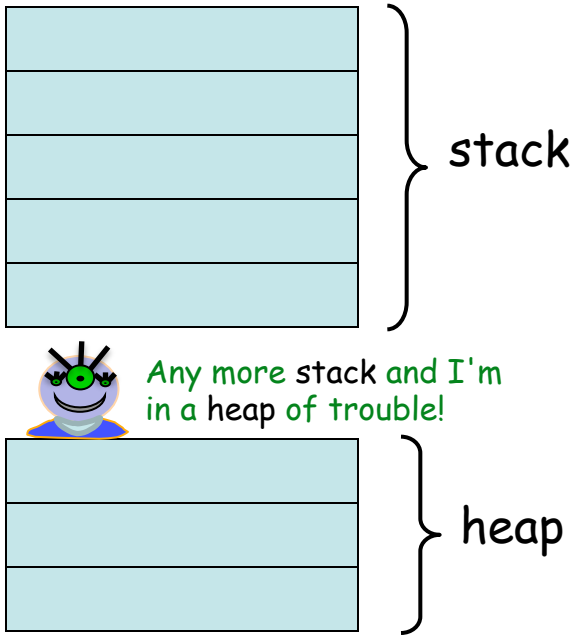
arrays/lists	`int[] A = new int[10];`
	`A` is an array of 10 `int`s: `A[0] A[1]…`
	`A.length` is the array's length
		`A = [0]*10`
		`len(A)`

booleans	`true` and `false` `&&  ||  !`	`True` and `False` `and or not`
Strings	`"strings"` vs. `'c'`haracters	`"string" 'string'` there are no distinct characters

loops, conditions and code blocks	`for (int i=0 ; i<10 ; ++i)`
	`{`
	`  code runs 10 times…`
	`  with i == 0,1,2,…,9`
	`}`
	`if (test)`
	`{`
	`  code;`
	`  code;`
	`}`
	or `while`
		`for i in List:`
		`  i becomes each`
		`  element in List`
		`if test:`
		`  code`
		`  code`

functions	***must*** return one value of the ***declared*** type, unless `void`	return whatever, `None` by default
printing	`System.out.println("x == " + x);` concatenate with +	`print 'x ==', x` concatenate with ,

memory model and management



A `Point` class

The `Point` of Python: objects and classes

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y


def main():
    d = 42.0
    p = Point(d, 42.0)

    q = Point(p.x, 42)
```

Python sure is proud of its `self`!

Constructors are giving me a **HEAP** of trouble!

Draw a picture of the data at the end of main

The point of **Point**

another **main** method for **Point.py**

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y


def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(42.0, 42.0)

    p1 = p2
    p1.x = 60.0
    print "p2.x is", p2.x
```

It will print: p2.x is ____

A. 42.0 B. 60.0 C. 0.0

Draw a picture of the data at the end of main. What will be printed?

The `Point` of Java: objects and classes

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y


def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)
    if p1 == p2:
       print "Equally valid points!"
    else:
       print "They're not equal!"
```

This code will print:

A. "Equally valid points!" B. "They are not equal!"

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __eq__(self, p2):
        return self.x == p2.x and self.y == p2.y


def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)

    if p1 == p2:
       print "Equally valid points!"
    else:
       print "They're not equal!"

    if p1 != p2:
       print "They're not equal!"
    else:
       print "Equally valid points!"
```

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __eq__(self, p2):
        return self.x == p2.x and self.y == p2.y


def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)

    if p1 is p2:
       print "Equally valid points!"
    else:
       print "They're not equal!”

    if p1 is not p2:
       print "They're not equal!"
    else:
       print "Equally valid points!"
```

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __eq__(self, p2):
        return self.x == p2.x and self.y == p2.y


def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)
    if p1 == (0.0, 0.0):
       print "Equally valid points!"
    else:
       print "They're not equal!"
```

This code will print:

A. "Equally valid points!" B. "They are not equal!" C. None of the above

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __eq__(self, p2):
        if type(p2) != type(self):
            return False
        return self.x == p2.x and self.y == p2.y


def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)
    if p1 == (0.0, 0.0):
       print "Equally valid points!"
    else:
       print "They're not equal!"
    print p1
```

```
class Point(object):
    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __eq__(self, p2):
        if type(p2) != type(self):
            return False
        return self.x == p2.x and self.y == p2.y

    def __str__(self):
        s = "(" + str(self.x) + "," + str(self.y) + ")"
        return s



def main():
    p1 = Point(0.0, 0.0)
    p2 = Point(0.0, 0.0)
    if p1 == (0.0, 0.0):
       print "Equally valid points!"
    else:
       print "They're not equal!"
    print p1
```