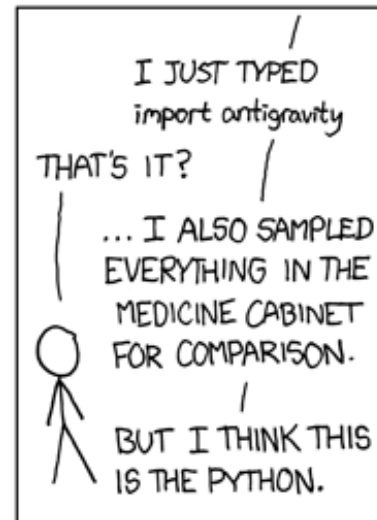
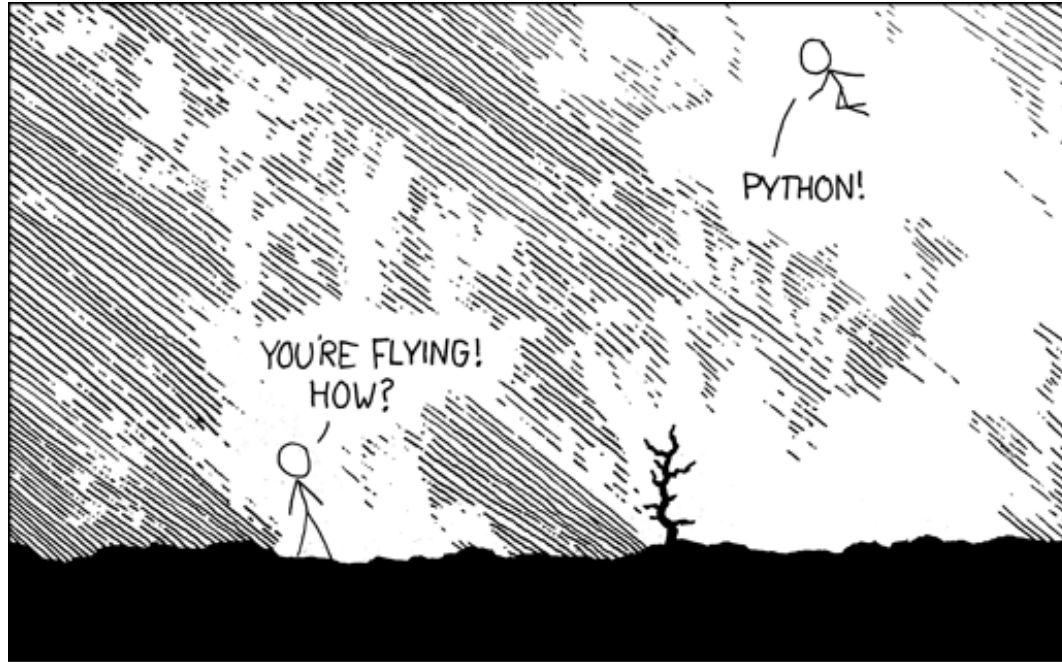


ICGE Programming Module—Python!

























<https://xkcd.com/353/>

Why Python?

- Freely available for all types of computers
- Widely used grade schools to universities to industry
- Powerful object-oriented language, with great built-in types
- Easy to create GUIs and interface to internet
- Fun! Python does a lot of the hard work for you (unlike C++)

My toolkit

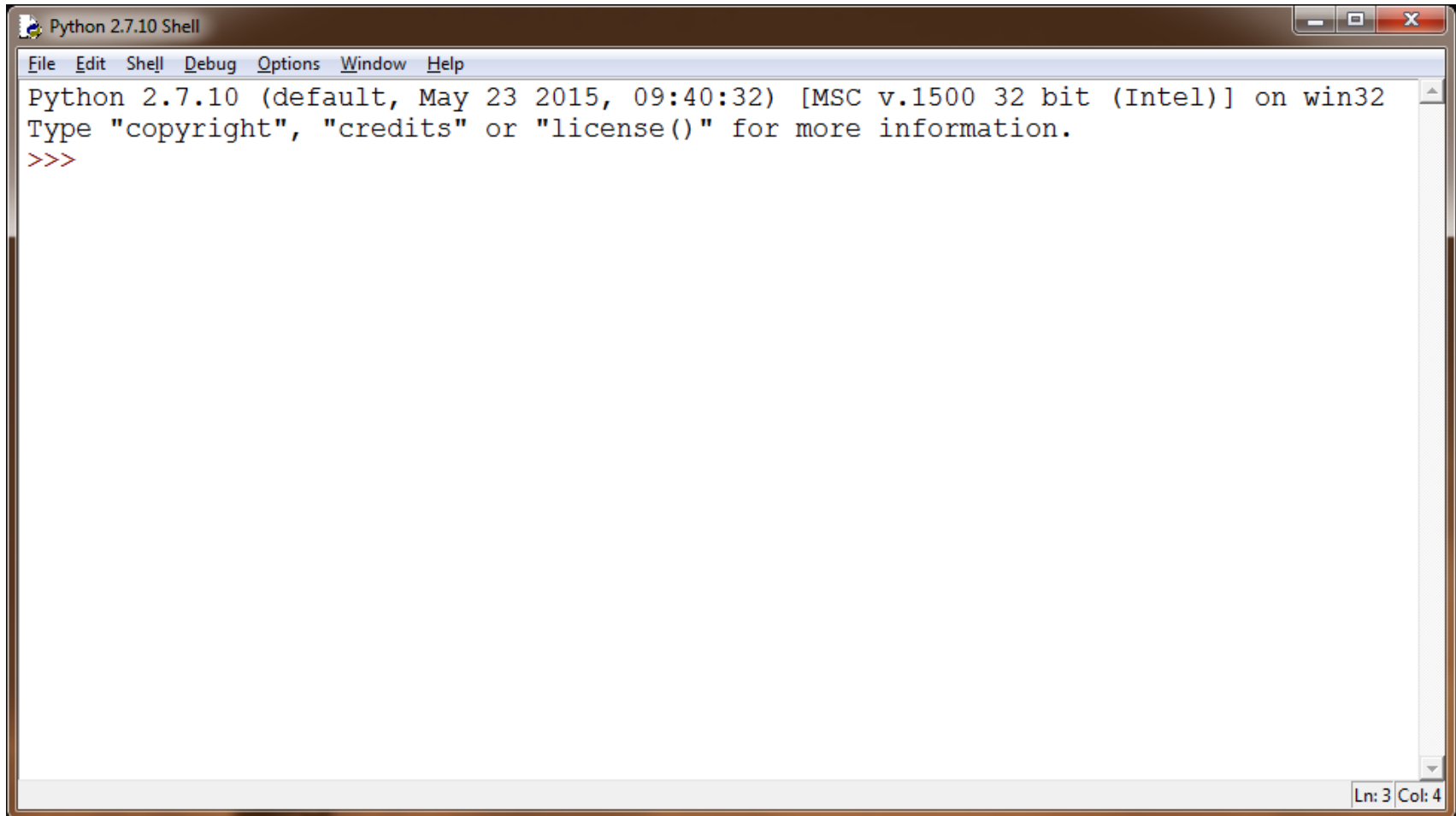


| Language Rank | Types | Spectrum Ranking |
|---------------|--|------------------|
| 1. Python |   | 100.0 |
| 2. C |    | 99.7 |
| 3. Java |    | 99.5 |
| 4. C++ |    | 97.1 |
| 5. C# |    | 87.7 |
| 6. R |  | 87.7 |
| 7. JavaScript |   | 85.6 |
| 8. PHP |  | 81.2 |
| 9. Go |   | 75.1 |
| 10. Swift |   | 73.7 |

If you're going to be using computers in your research—take the time to find the toolkit that works right for you

Python is distributed with a simple integrated development environment (IDE) called **idle**

When you start idle it opens and interactive shell window:



```
Python 2.7.10 Shell
File Edit Shell Debug Options Window Help
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

The screenshot shows a window titled "Python 2.7.10 Shell" with a menu bar containing "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area displays the Python startup message: "Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on win32" followed by "Type 'copyright', 'credits' or 'license()' for more information." and the interactive prompt ">>>". The status bar at the bottom right indicates "Ln: 3 Col: 4".

Just like R, Matlab or other interpreted languages, you can interactively run Python commands in the shell

```
print "Hello world"  
print("Hello world")
```

```
7*1.3
```

```
import math
```

```
math.
```

```
math.sqrt(16*144)
```

```
from math import sqrt
```

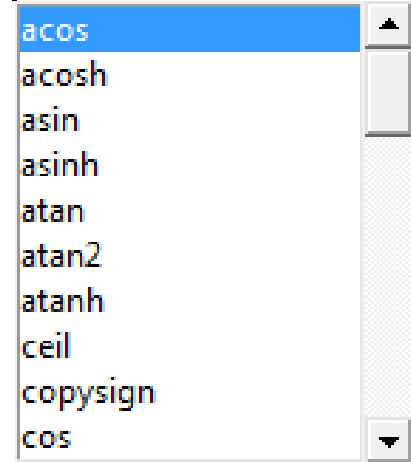
```
sqrt(1024)
```

```
for i in range(1,11):
```

```
    y = i*i
```

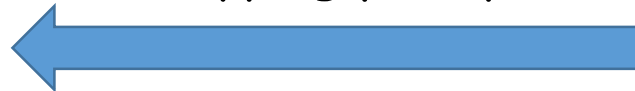
```
    print y
```

```
>>> math.
```



- acos
- acosh
- asin
- asinh
- atan
- atan2
- atanh
- ceil
- copysign
- cos

Typing prompts



```
>>> math.sqrt(
```

```
sqrt(x)
```

A key strength of Python is the power and completeness of its built-in data types (objects)

Lists are a general data container:

```
alist=[1, 9, 3, 7, "a"]
```

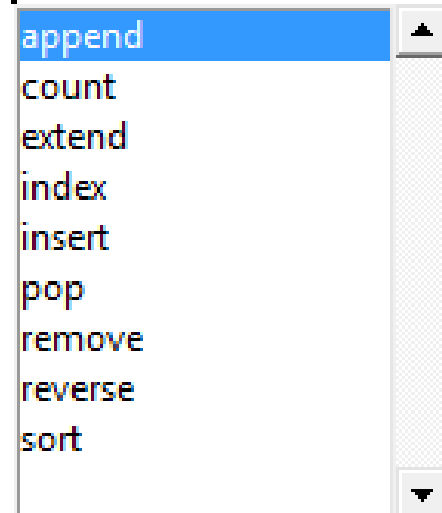
```
alist.count(3)
```

```
alist.pop()
```

```
alist.sort()
```

```
alist.reverse()
```

```
>>> alist.
```



append

count

extend

index

insert

pop

remove

reverse

sort

Lists can be "multidimensional":

```
blist=[[1,2],[3,4]]
```

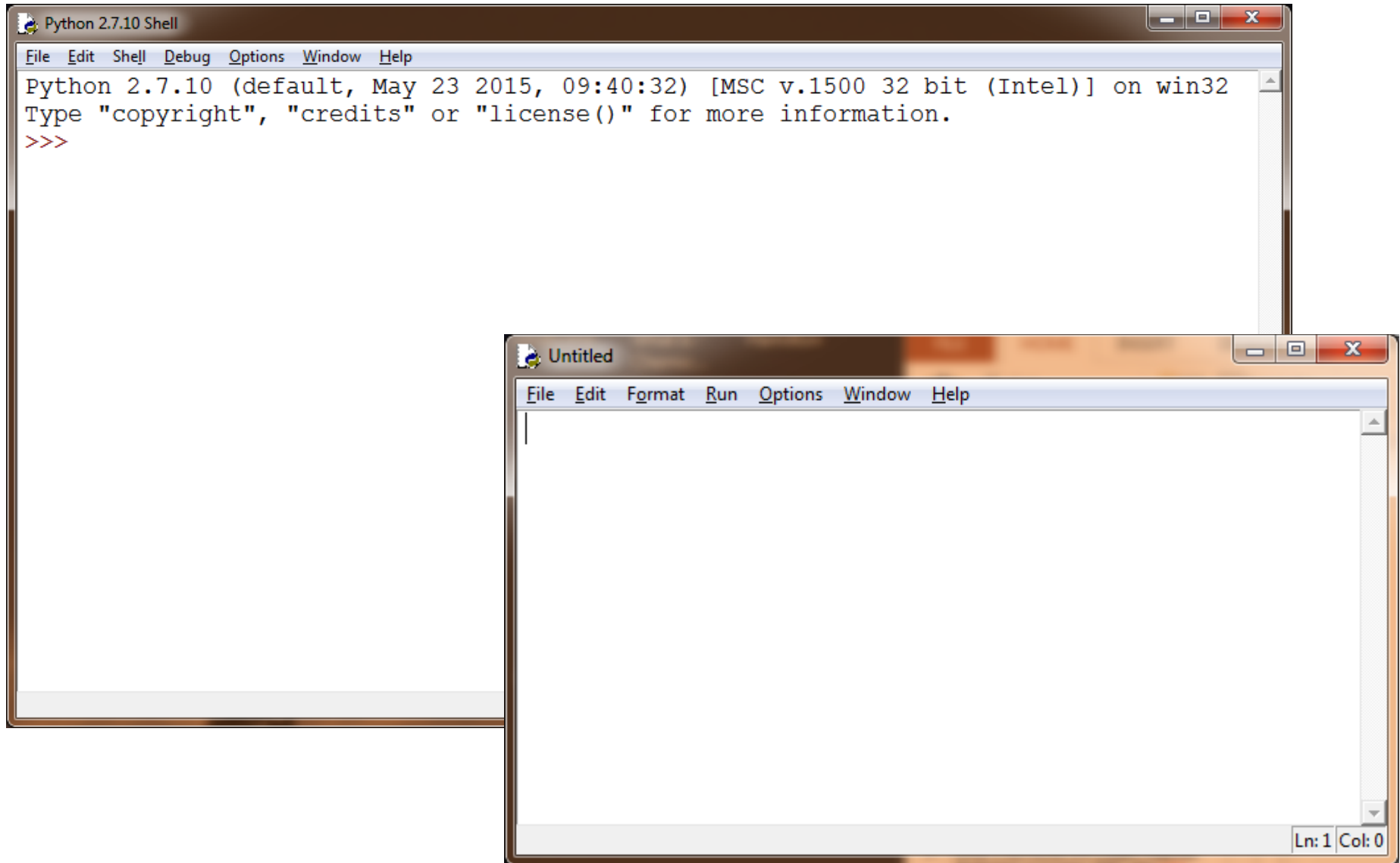
```
blist
```

```
blist[0][1]
```

```
blist[1]
```

```
list2d=[[0 for i in range(10)] for j in range(10)]
```

The idle environment provides an editor to enter Python code without immediately executing it



The idle environment provides an editor to enter Python code without immediately executing it

Enter the following code into the editor box

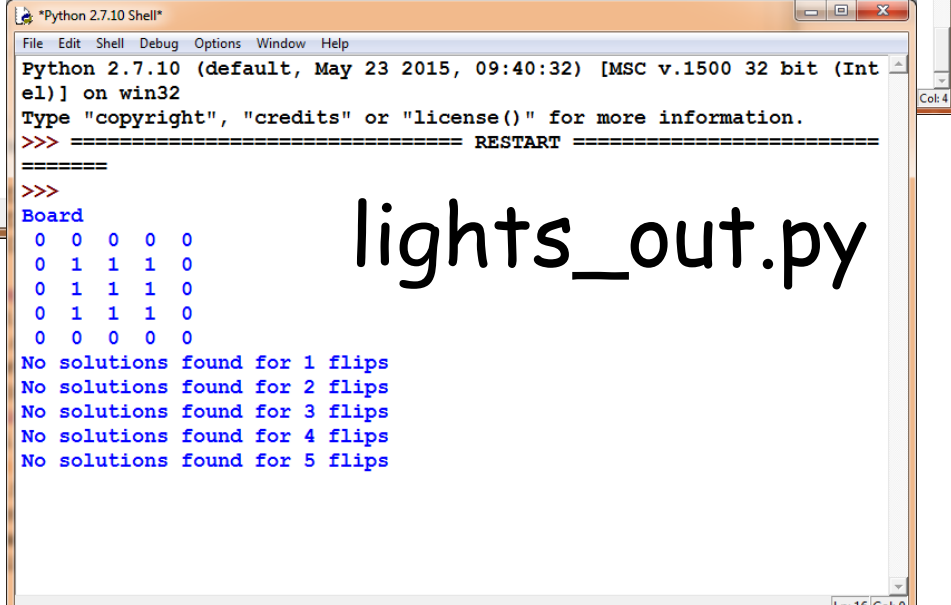
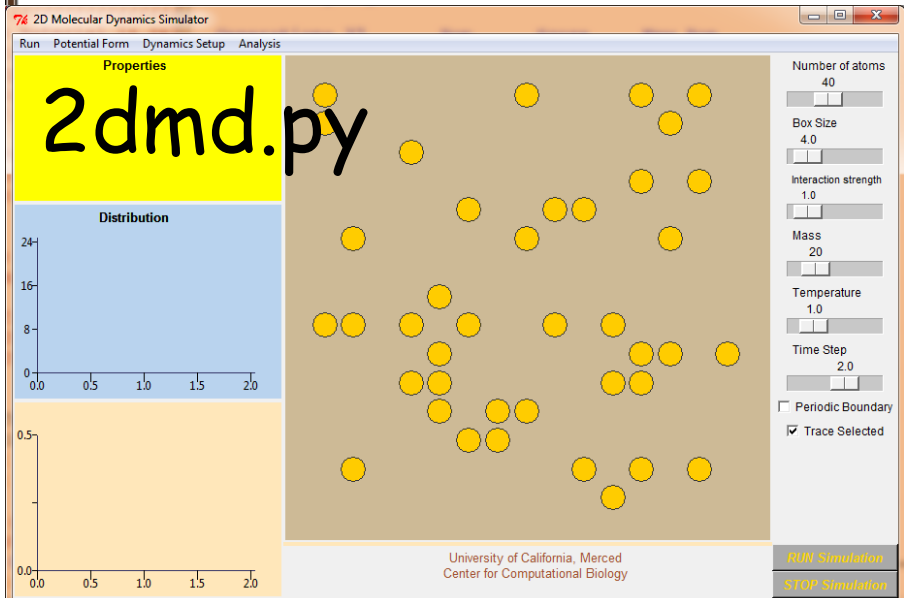
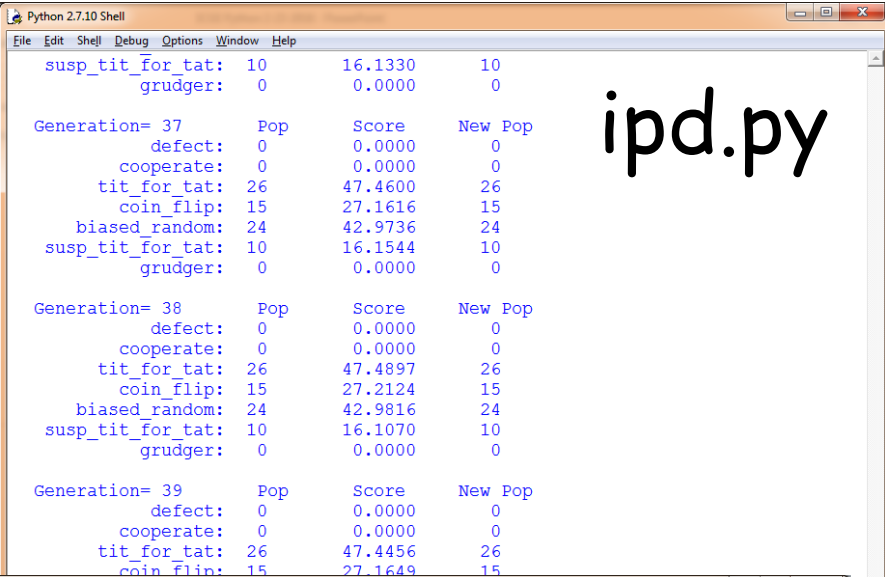
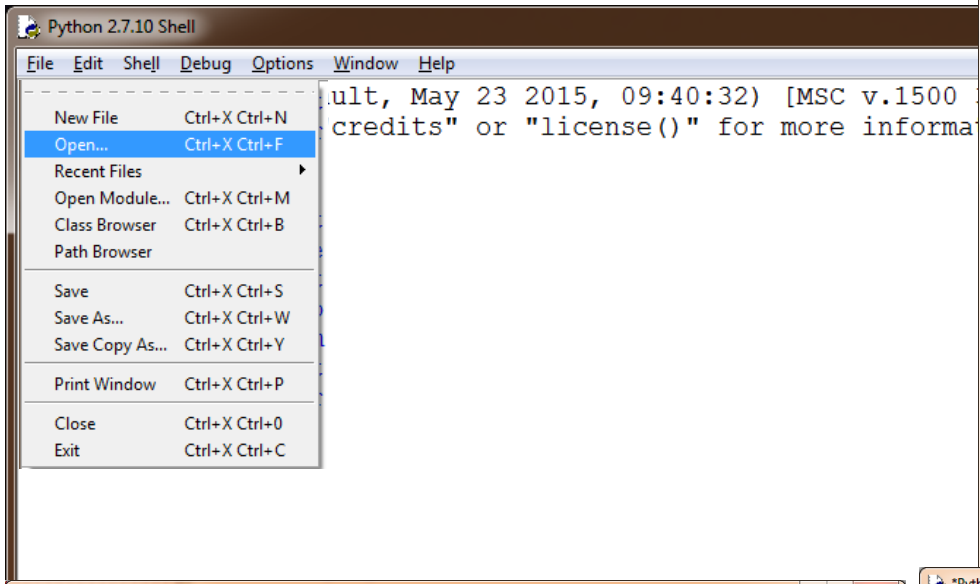
```
from random import choice
ndice=5
ngames=10
for i in range(ngames):
    dice=[]
    for j in range(ndice):
        dice.append(choice(range(1,7)))
    dice.sort()
    lowtwo=dice[0]+dice[1]
    print lowtwo
```

Then select the "Run/Run module" menu option
Save the program as "dice.py"

There's a great online learning tool for Python and other languages lets you visualize code execution

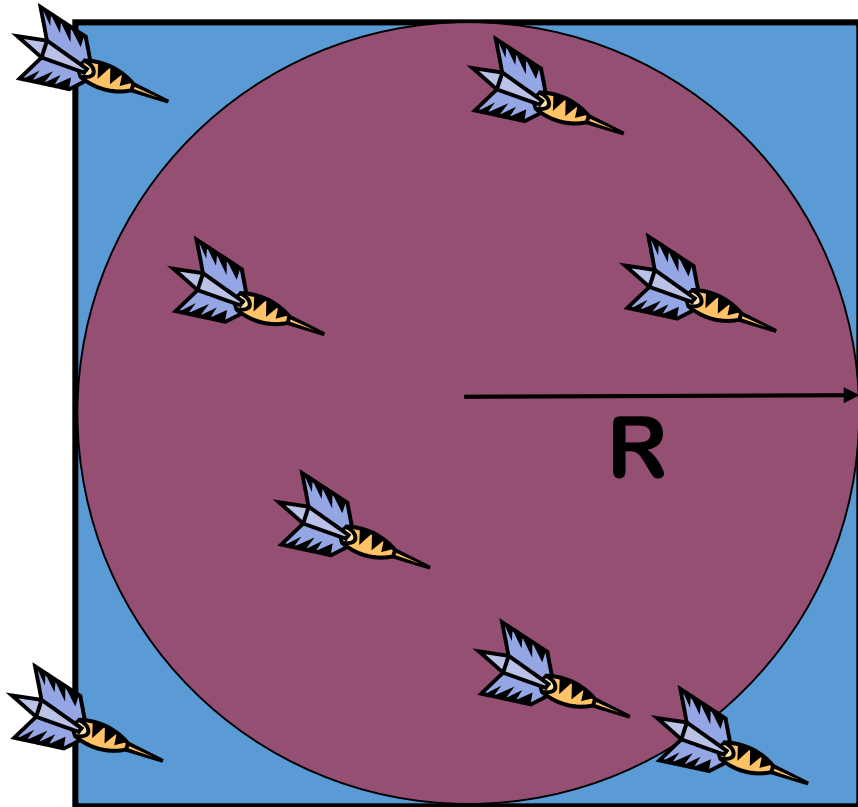
1. Open up the web browser and go to the website:
`http://www.pythontutor.com/`
2. Click on the link entitled “Start visualizing your code now!”
3. Copy and paste the `dice.py` program from your idle window
4. Click the button entitled “Visualize Execution” below the editor box
5. Click the “Forward” button for each step in the program to run
6. As each variable is created or updated, you’ll see that space next to the program.
7. Output will appear in a box below the editor box
8. To run to completion, click the “Last” button”
9. To modify the program, click the “Edit Code” link

You can load existing Python programs into idle and run them. Let's look at a few simulation codes



Monte Carlo simulations use random numbers to statistically sample different outcomes

A simple use of Monte Carlo simulation is to calculate the relative area of a region:



What's the ratio of the area of the circle to the area of the square?

$$\frac{\pi R^2}{(2R)^2} = \frac{\pi}{4} = \frac{\text{Darts in circle}}{\text{Darts in square}}$$

$$\pi = 4 \times \frac{\text{Darts in circle}}{\text{Darts in square}}$$

Here's a simple Python program to simulate this process using virtual darts

```
import random
import math

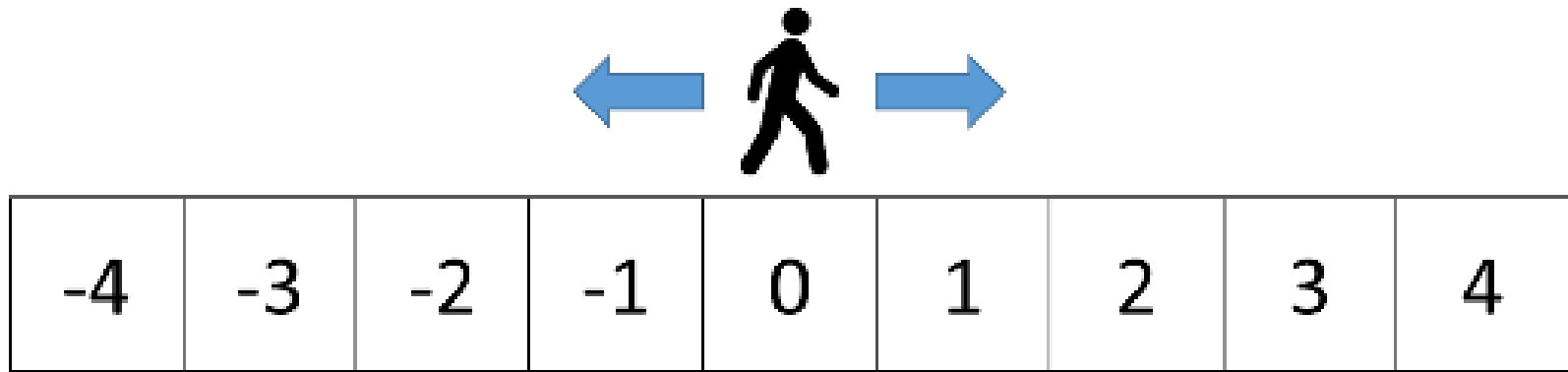
inside=0
trials=1000
for i in range(trials):
    x=random.random()
    y=random.random()
    if (x*x+y*y)<1.0:
        inside+=1

pi=4.*float(inside)/float(trials)
print "N=%d Error=%8.5f"%(trials,pi-math.pi)
```

Indentation matters!

Enter into an idle editing window and then save as "**pi.py**"

Many problems in physics, chemistry and biology essentially boil down to "random walks"



Simplest question to ask—does the walker ever get back home?

Python program for 1-D random walk

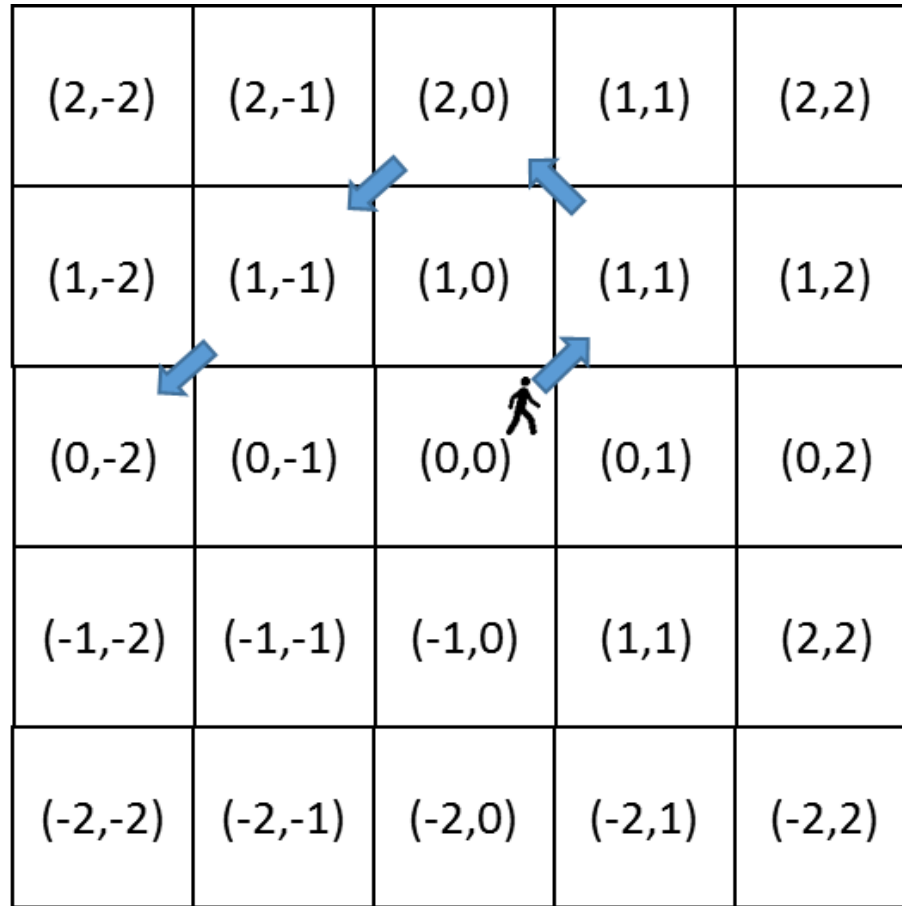
Note double underscores “__”



```
from __future__ import division
from random import choice
trials=1000
steps=1000
gothome=0
for i in range(trials):
    point=0
    for step in range(steps):
        point+=choice((-1,1))
        if point==0:
            gothome+=1
            break
print "Fraction that got home=%f" % (gothome/trials)
```

Save this as "**rwalk1d.py**" and run for different numbers of steps

The problem gets more interesting if the walker moves in 2 or more dimensions



Note that if we randomly change both x & y coordinates by -1 or $+1$, the walker moves diagonally like a checkers piece.

Python program for arbitrary-D random walk making diagonal moves at every step

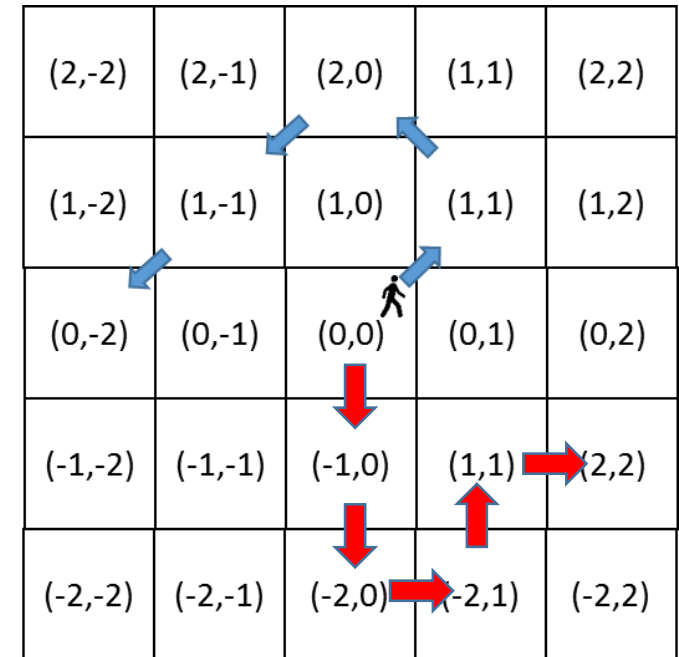
```
from __future__ import division
from random import choice
dim=3
trials=1000
steps=1000
gothome=0
for i in range(trials):
    point=[0]*dim
    for step in range(steps):
        for j in range(dim):
            point[j]+=choice((-1,1))
        if point.count(0)==dim:
            gothome+=1
            break
print "Fract that got home=%f in %d dims" % (gothome/trials,dim)
```

Save this as "**rwalknd.py**" and run for different dimensions

For what #'s of dimensions does the walker make it home?

Exact results for an infinite number of steps moving in only one dimension at a time

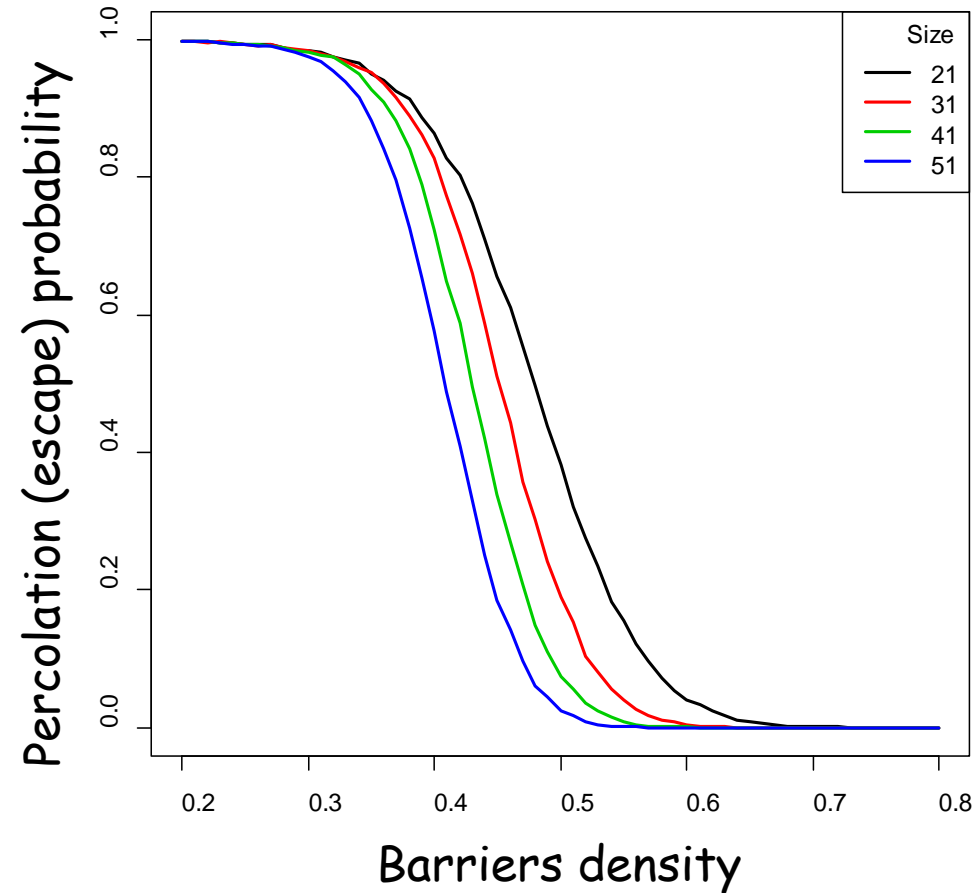
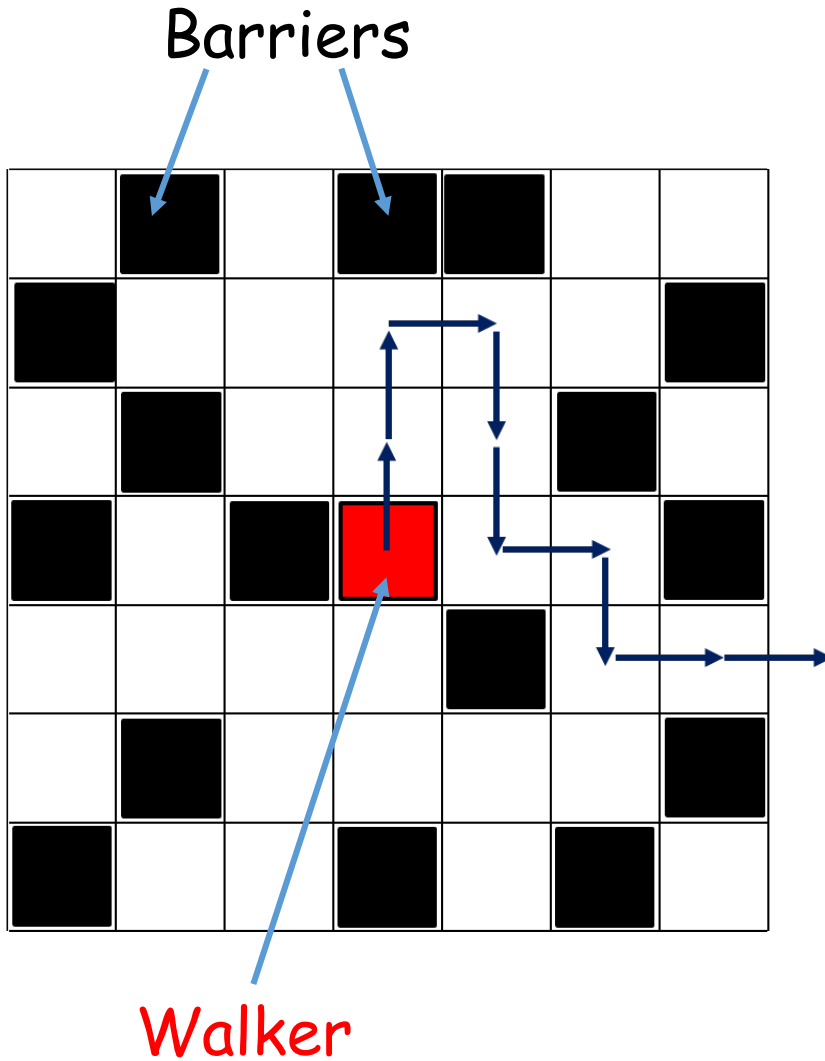
| Dimensions | Prob(get home) |
|------------|----------------|
| 1 | 1.000 |
| 2 | 1.000 |
| 3 | 0.341 |
| 4 | 0.193 |
| 5 | 0.135 |
| 6 | 0.105 |
| 7 | 0.086 |
| 8 | 0.073 |



Possible questions to explore:

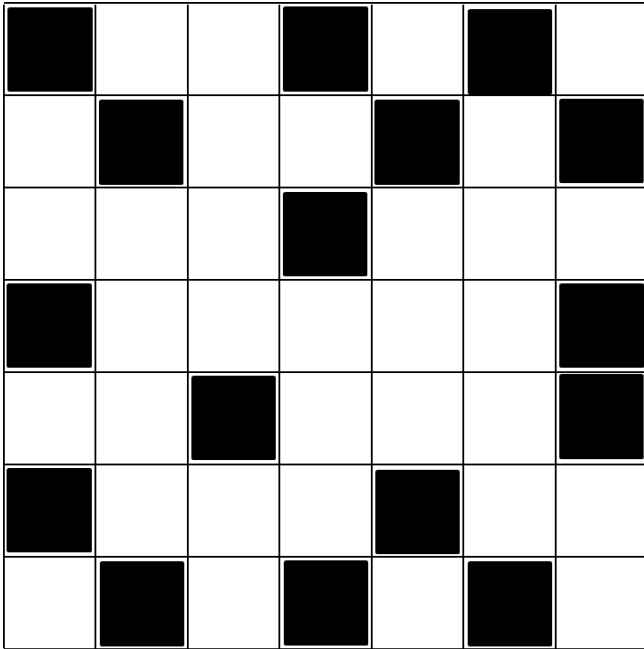
1. How well does `rwalknd.py` agree with the exact results?
2. Do your results match better if you modify the program to step in only one dimension at a time?

Another type of random walk simulation tests the effect of barriers on trapping a walker in space



Simulating percolation is different from random walks because we have to store the barrier locations

The 2D list object we saw previously is a good data structure for storing the barrier locations




```
grid = [[1, 0, 0, 1, 0, 1, 0],  
        [0, 1, 0, 0, 1, 0, 1],  
        [0, 0, 0, 1, 0, 0, 0],  
        [1, 0, 0, 0, 0, 0, 1],  
        [0, 0, 1, 0, 0, 0, 1],  
        [1, 0, 0, 0, 1, 0, 0],  
        [0, 1, 0, 1, 0, 1, 0]]
```

Our random walker will move around "inside" this grid

The first half of the program sets up the simulation and builds a grid filled with barriers at a set density

Open up the program `perc.py` inside `idle`

```
from __future__ import division
from random import choice, random
density=0.5 #Should be a number between 0 and 1
side=21     #Should be an odd number
perc=0
trials=1000
maxtime=1000
steps = [(1,0),(-1,0),(0,1),(0,-1)]
for trial in range(trials):
    count=0
    for x in range(side):
        for y in range(side):
            grid[x][y]=0
            if (random())<density):
                count+=1
                grid[x][y]=1
```



"Non Pythonic"

`grid=[[0 if random() > density else 1 for x in range(side)] for y in range(side)]`

The second half is similar to the random walk, but we need to test for barriers and when the walker leaves

```
# Start particle at center
x,y = int(side/2),int(side/2)
for time in range(maxtime):
    # Randomly move particle
    sx,sy = choice(steps)
    nx = x+sx
    ny = y+sy
    # if new position is occupied try again
    if grid[nx][ny]==1:
        continue
    if nx==0 or ny==0 or nx==(side-1) or ny==(side-1):
        perc+=1
        break
    grid[x][y]=0    #Remove particle from current spot
    grid[nx][ny]=1  #Put particle in new location
    x=nx
    y=ny
print "%4d %5.3f  %5.3f"%(side,density,perc/trials)
```

Some additional Python learning resources

Posted to CatCourses:

- ThinkPython (book)
- Python_quick_reference
- Python_refcard
- Python2.7 Reference

Python classes available at all online course sites:

- Edx
- Coursera
- Software carpentry
- Udacity
- Datacamp

Other WWW resources:

- <http://www.codecademy.com/tracks/python>
- <http://www.afterhoursprogramming.com/tutorial/Python/Overview/>
- <http://www.stavros.io/tutorials/python/> ("Learn Python in 10 Minutes")
- <http://learnpython.org>
- http://www.linuxtopia.org/online_books/programming_books/introduction_to_python/
- <http://wiki.python.org/moin/BeginnersGuide>

If you want a more intensive programming introduction, consider the Scientific Programming Workshop Sponsored

Instructors: Mike Colvin and David Quint

Schedule: 8:30-12:30 for 10 days (June 4-15)

Location: UC Merced campus, SE1-100

Topics:

- Linux/Bash scripting
- Using remote compute servers
- R programming and data analysis
- Python programming
- C/C++ [Optional dep. on interest]



Sponsored by the UC Merced Center
for Cellular and Biomolecular Machines

