Controlling the flow of program execution

One of Python's innovations is to use

indentation:

to mark program blocks.

Other languages use curly braces

```
{
… bunch of code …
}
```

or keywords

```
BEGIN
… bunch of code …
END
```

and DON'T CARE about indentation.

Guido van Rossum launched Python in 1989.

This fussy dependence on hierarchical formatting is almost unique to computer programming languages.

It makes them "brittle". (Code is easily "broken" by teeny-weenie flaws.)

Natural human languages have nothing like this, whether spoken, or in writing. We get away with "Well, you know what I mean!"

So it's a trap for beginners. (Don't worry!)

Example #1: the for-loop

In Python, it steps through a list - any list!

```
nums = [1, 2, 3, 4, 5, 6]
factorial = 1
for x in nums:
    factorial = factorial * x
    print("%d! = %d" % (x,factorial))

= RESTART: C:/Users/Jon/Desktop
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
>>>
```

A more "Pythonic" translation …

```
nums = range(1,7)
factorial = 1
for x in nums:
    factorial *= x
    print("%d! = %d" % (x,factorial))

= RESTART: C:/Users/Jon/Desktop
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
>>>
```

Example #2: functions

```
def var(x_vector):
    sum_x = float(sum(x_vector))
    mean_x = sum_x/float(len(x_vector))

    sum_dev_sq = 0.0
    for x in x_vector:
        sum_dev_sq += (x - mean_x)**2

    variance = sum_dev_sq/float(len(x_vector))

    return variance

nums = [1,2,3,4,5,6]
# Or, nums = range(1,7)

print("variance of nums[] = %.1f" % (var(nums)))

= RESTART: C:/Users/Jon/Desktop
variance of nums[] = 2.9
```

Or better, use the simpler way to calculate var( )

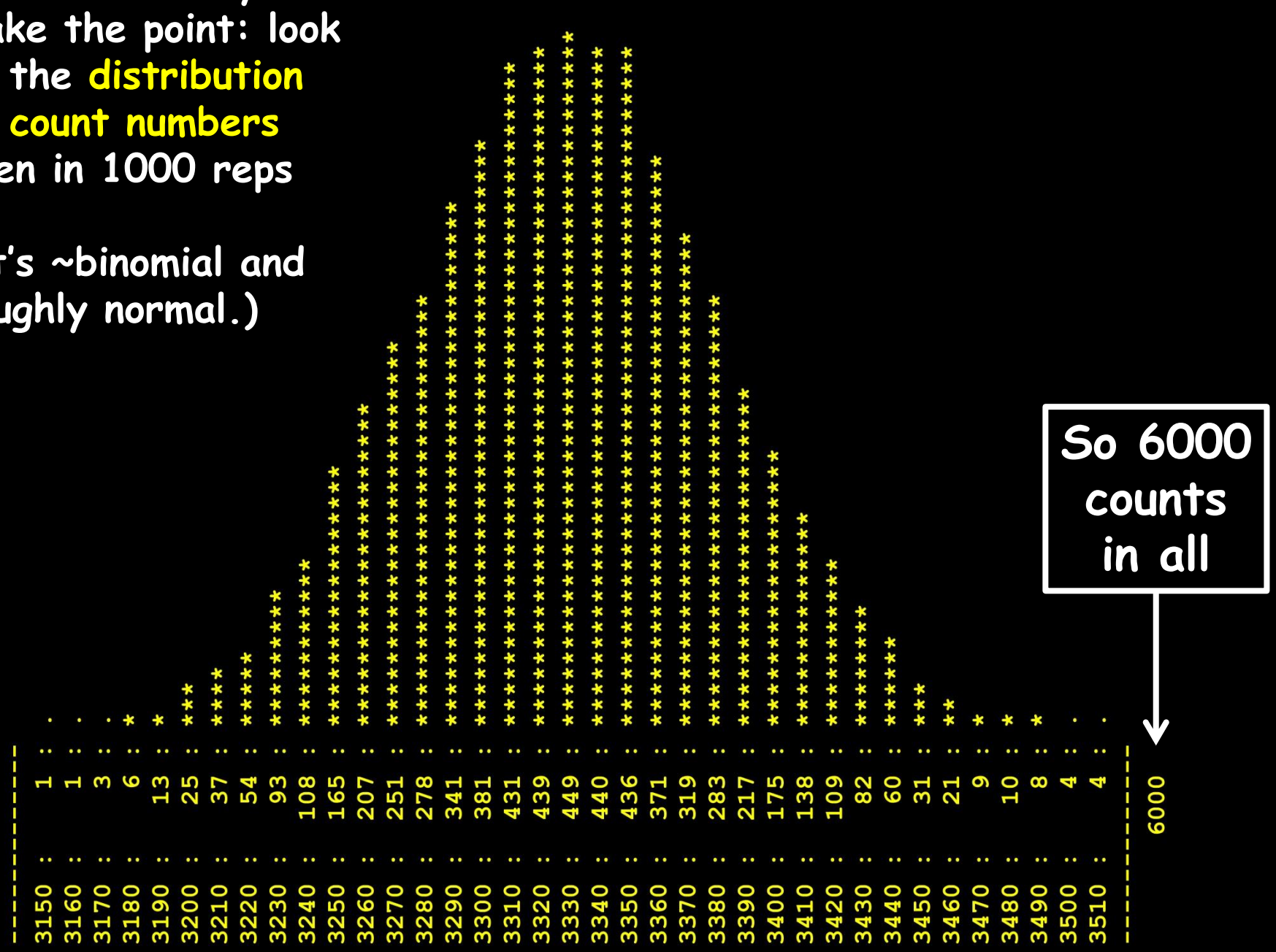
```
def var2(xvec):
    m = msq = 0.0
    for x in xvec:
        m   += x
        msq += x*x

    n = len(xvec)
    m   /= n
    msq /= n

    return msq - m*m          # E(x**2) - [E(x)]**2


print("easier variance of nums[] = %.1f" % (var2(nums)))

= RESTART: C:/Users/Jon/Desktop
easier variance of nums[] = 2.9
```

Example #3: Loops within loops!

```
from random import random                                   # line number  1

# define a function to return the variance of values in xvec                #  3
def var(xvec):                                                              #  4
    m = msq = 0.0                                                           #  5
    for x in xvec:                                                          #  6
        m += x                                                              #  7
        msq += x*x                                                          #  8
    n = float(len(xvec))                                                    #  9
    m /= n                                  # mean                          # 10
    msq /= n                                # mean square                   # 11
    return (msq - m*m)                      # variance                      # 12

w = [3246, 3449, 2897, 2841, 3635, 3932]    # counts from white die         # 14
r = [3407, 3631, 3176, 2916, 3448, 3422]    # counts from red die           # 15

Vw = var(w)                                                                 # 17
Vr = var(r)                                                                 # 18
print("Var(white):", Vw)                                                    # 19
print("Var(red)  :", Vr)                                                    # 20

nreps = 10                          # adjust this to do more replicates     # 22
for rep in xrange(nreps):           # outer loop: replicates of expt.       # 23
    count = [0,0,0,0,0,0]           # count[i] accumulates the numbers of   # 24
                                    #     rolls that showed i+1 spots       # 25
    for roll in range(20000):       # inner loop: rolls of the die          # 26
        spots = int(6.0*random())   # uniform on [0,1,2,3,4,5]  fn(fn())    # 27
        count[spots] += 1           # accumulate spot numbers               # 28

    v = var(count)                  # and here's our function call          # 30

    print("Replicate # %d: var=%f" % (rep, v))   # REMOVE ME later          # 32
```

A different way to make the point: look at the **distribution of count numbers** seen in 1000 reps

(It's ~binomial and roughly normal.)



So 6000 counts in all

```
# Wolf_counts_distribution.py

from random import random

# NO NEED FOR VARIANCE CALCULAION

w = [3246, 3449, 2897, 2841, 3635, 3932]    # counts from white die
r = [3407, 3631, 3176, 2916, 3448, 3422]    # counts from red die


nreps = 1000
counts = [0 for i in range(5000)

for rep in xrange(nreps):              # outer loop: replicates of expt.
    count = [0,0,0,0,0,0]              # count[i] accumulates the numbers of
                                       #     rolls that showed i+1 spots
    for roll in range(20000):          # inner loop: rolls of the die
        spots = int(6.0*random())      # uniform on [0,1,2,3,4,5]  fn(fn())
        count[spots] += 1              # accumulate spot numbers

    for x in range(6):
        counts[count[x]] += 1

# AT THE END, PRINT EACH OBSERVED COUNT NUMBER, AND THE NUMBER OF TIMES IT WAS SEEN

for j in range(2000,4000):
    if counts[j] > 0:
        print("%4d  :  %4d" % (j,counts[j]))

# (Later turn this into a histogram) #
```

OR, keep track of the ***largest variances seen*** for your white and red dice (over all reps).

How?

Initialize a "max_var" memory variable for each.

```
max_V_w = max_V_r = 0
```

Then (in the right place in your program):

```
if Vw > max_V_w:
        max_V_w = Vw
```

… and so on. Then report them at the very end.

MORAL: There are usually many ways to solve a problem!