CMPT 120
Intro to CS & Programming I

WEEK 7 (Feb. 24-28)

— *Jérémie O. Lumbroso* —

Lecture 19:
The `while` loop (continued) and more examples

http://www.sfu.ca/~jlumbros/Courses/CMPT120/

Investment, drinking, adding, searching

MORE EXAMPLES OF WHILE LOOPS

`while` loop again

```
while <condition>:
  <actions>
```

- While loops can be used with index variables that are incremented
- But while loops can be used with more complicated stopping conditions
- Or conditions involving an outside event (by the user)

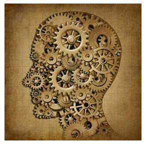
Investment Simulation

Previously given example: how long does it take to double $500, at a yearly interest rate of 2.25%?

http://goo.gl/tX99hi

```
initial = 500           # in dollars
interest_rate = 2.25    # in percent

balance = initial
years = 0

while balance < 2*initial:
  years = years + 1
  interest = interest_rate * balance/100.0
  balance = balance + interest

print "You have", balance
print "You doubled your money in", years, "years!"
```

Is This Predictable?



- How long does it take to double the investment, with a \$500 initial balance? With \$1000? With \$100?
- Always **32 years**: depends on rate, not initial balance.
- Actually it is easy to predict using math, and the notion of **geometric series** which begins:

$$\begin{cases} u_0 &= 500 \\ u_n &= u_{n-1} + 2.25/100 \times u_{n-1} \end{cases}$$

- And the general formula is:

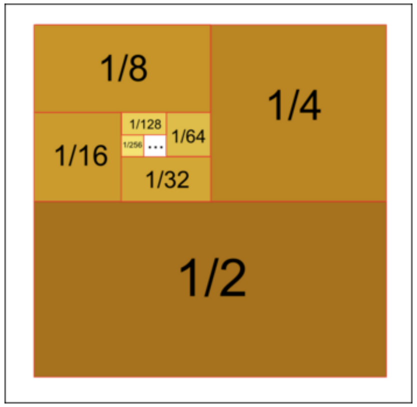
$$u_n = 500 \times 1.0225^n$$

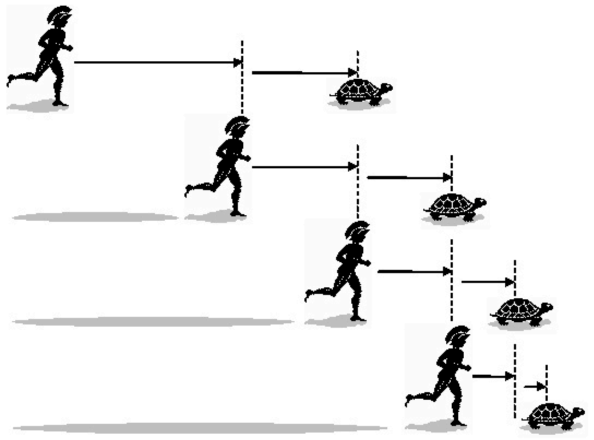
- Thus:

$$\begin{aligned} 500 \times 1.0225^n > 1000 \quad &\Leftrightarrow \quad 1.0225^n > 2 \\ &\Leftrightarrow \quad n > \frac{\log 2}{\log 1.0225} \approx 31.1518 \end{aligned}$$

It's Predictable!

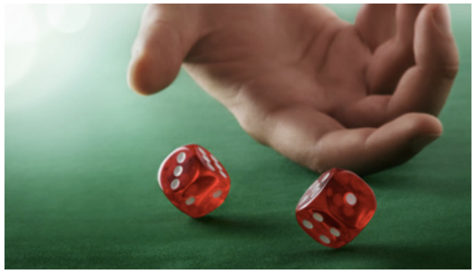
- For this example, the math is relatively simple
- The number of years could easily be determined without programming



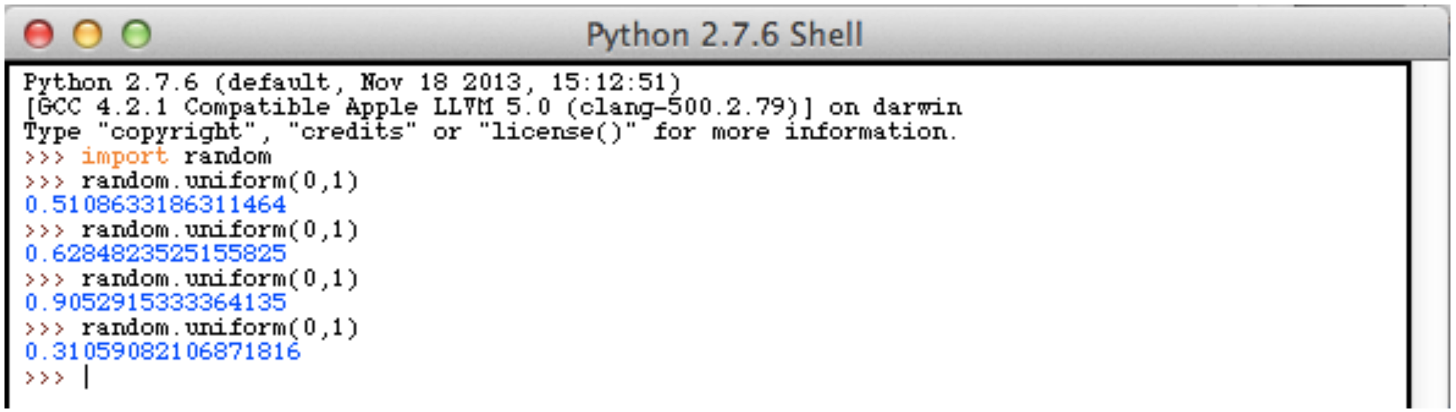


- But what if we introduce randomness?

`import random`



- Random module in Python, for simulations
- `random.randint(a,b)` draws a uniform random int between `a` and `b` (= throw a dice)
- `random.uniform(a,b)` draws a uniform **real** in the interval (a,b)
- **Recall:** in Python shell, type `help(random)` or `help(random.uniform)` to get documentation



```
Python 2.7.6 (default, Nov 18 2013, 15:12:51)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> random.uniform(0,1)
0.5108633186311464
>>> random.uniform(0,1)
0.6284823525155825
>>> random.uniform(0,1)
0.9052915333364135
>>> random.uniform(0,1)
0.31059082106871816
>>> |
```

New Investment Simulation

- Introduce interest rate fluctuations
- Predicting when fluctuations are even a little bit non trivial requires very complicated math probability tools and is much easier to simulate

http://goo.gl/MlrNdZ

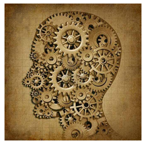
```python
import random

initial = 500           # in dollars
interest_rate = 2.25    # in percent

balance = initial
years = 0

while balance < 2*initial:
  # interest rate now fluctuates randomly
  interest_rate = interest_rate + random.uniform(-0.1,0.1)

  years = years + 1
  interest = interest_rate * balance/100.0
  balance = balance + interest

print "You have", balance
print "You doubled your money in", years, "years!"
```

Crazy Wall St Investment

- What if we can now lose money?
- Introducing: negative rate + fluctuations

```
import random                http://goo.gl/s1PK6r

initial = 500
balance = initial
years = 0

while balance < 2*initial:
  interest_rate = random.uniform(-1,1)
  years = years + 1

  interest = interest_rate * balance
  balance = balance + interest

print "You have", balance
print "You doubled your money in", years, "years!"
```

After running the simulation, how many years does it take?



- **A** Less than 32
- **B** Between 32 and 100
- **C** More than 100
- **D** The program does not work, and just hangs

Going Broke

- In the previous program, if you go broke, you can never double your money
- What to do?
- **Exit the loop** before the doubling condition has been reached

Exit the Loop Before Its Time

- **Solution 1:** add a new condition

```
while balance < 2*initial:        while balance < 2*initial and balance > 1:
  …                                 …
```

- **Solution 2:** use the **break instruction** which instaneously exits the loop and executes the instruction immediately afterwards

```
while balance < 2*initial:        while balance < 2*initial:
  …                                 …
                                    if balance < 1:
                                      print "You went broke!"
                                      break
```

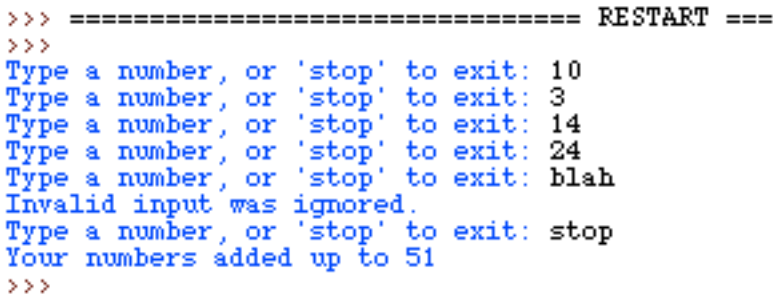
- **Solution 3:** if you are inside a function, you may also exit a loop by simply exiting the function, using the `return` instruction

Waiting on User's Input

Finally another type of unpredictable event that a while loop can test is **user input**

http://goo.gl/TVOe7U

```
total = 0

while True:
  inp = raw_input("Type a number, or 'stop' to exit: ")
  if inp == "stop":
    break
  elif inp.isdigit():
    # if inp is an integer we add it to total
    total = total + int(inp)
  else:
    # if not we print an error
    print "Invalid input was ignored."

# Once we're out, print out total
print "Your numbers added up to", total
```



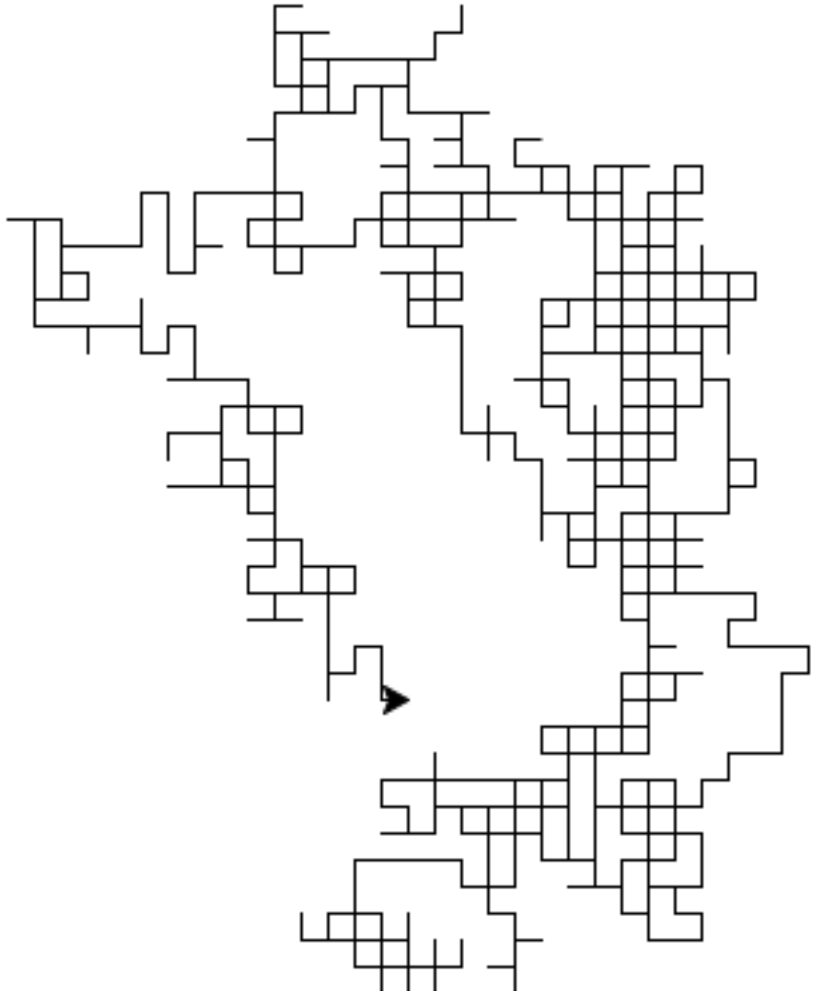
```
>>> ================================ RESTART ===
>>>
Type a number, or 'stop' to exit: 10
Type a number, or 'stop' to exit: 3
Type a number, or 'stop' to exit: 14
Type a number, or 'stop' to exit: 24
Type a number, or 'stop' to exit: blah
Invalid input was ignored.
Type a number, or 'stop' to exit: stop
Your numbers added up to 51
>>>
```

Drunkard's Walk

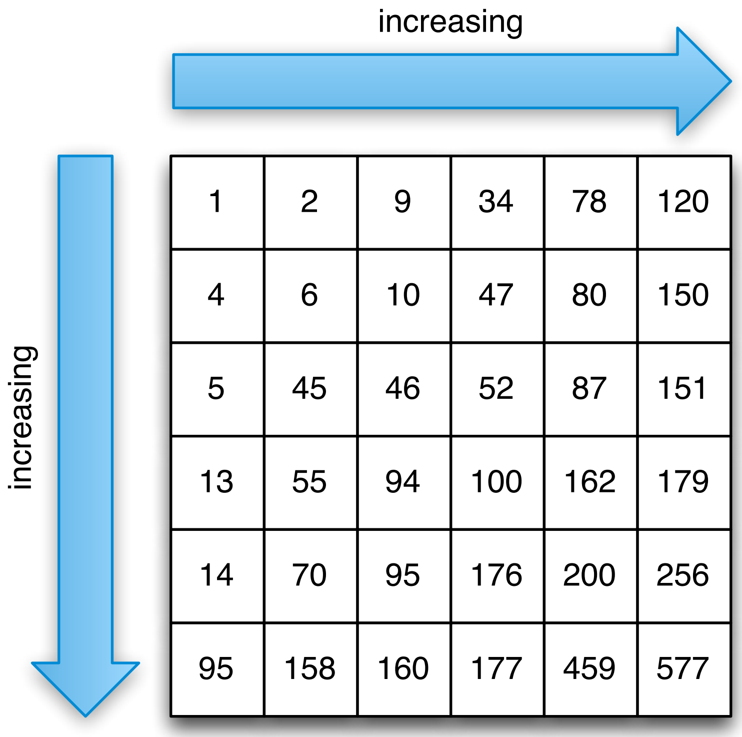
http://goo.gl/CBucNT

```
import random
import turtle

def move_random_direction(step):
  i = random.randint(1,4)
  turtle.left(90*i)
  turtle.forward(step)

turtle.reset()
turtle.speed("fastest")

initial_pos = turtle.position()
move = 0

while move == 0 or turtle.distance(initial_pos) > 1:
  move = move + 1
  move_random_direction(10)

print "Drunkard back home in", move, " moves."

turtle.done()
```

- Drunkard starts at initial position
- Takes a step in any direction at random
- Does he get back home?



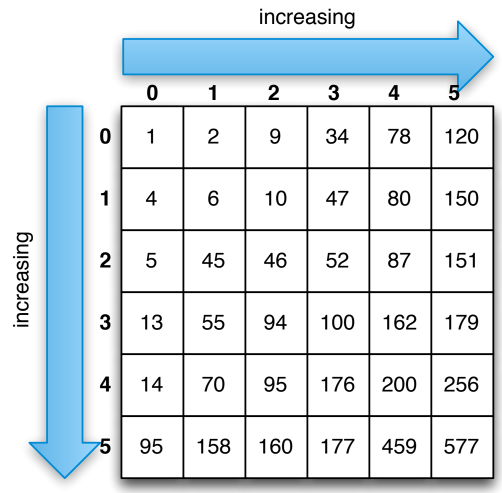
Search in an Ordered Matrix



1	2	9	34	78	120
4	6	10	47	80	150
5	45	46	52	87	151
13	55	94	100	162	179
14	70	95	176	200	256
95	158	160	177	459	577

- `matrix` is a table where rows and columns are **strictly increasing**
- <u>Problem 1:</u> how to find the position (x,y) of a number (assuming that number exists)?
- <u>Problem 2:</u> efficiently?

The Matrix/Table



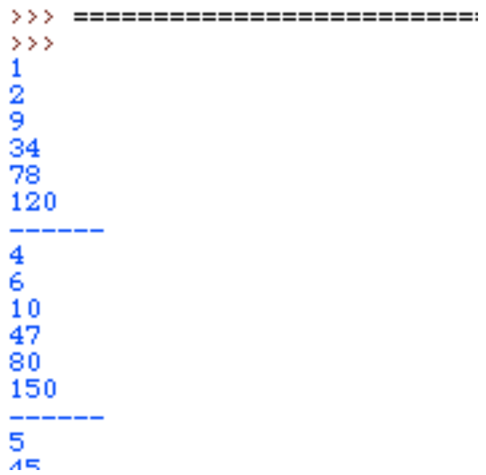
	0	1	2	3	4	5
0	1	2	9	34	78	120
1	4	6	10	47	80	150
2	5	45	46	52	87	151
3	13	55	94	100	162	179
4	14	70	95	176	200	256
5	95	158	160	177	459	577

`http://goo.gl/N6XWXe`

- Assuming you have a table `mat`, access using double index
  - `mat[0][0] == 1` and
  - `mat[0][4] == 120` and
  - `mat[4][4] == 577`

- Or using iteration

```
for row in mat:
    for elem in row:
        print elem
    print "------"
```



```
>>> ==========================
>>>
1
2
9
34
78
120
------
4
6
10
47
80
150
------
5
```

Simple Algorithm

- Go through every element row by row, column by column
- When element is found, return position

```
def search_matrix(mat, e):
  for row_num in range(len(mat)):
    for col_num in range(len(mat)):
      if mat[row_num][col_num] == e:
        return (row_num, col_num)
  return (-1, -1)
```

http://goo.gl/Tv8Yok

- How to modify code **to test the number of comparisons**? How many to find 100?

Smarter Algorithm

- We can use the fact that the matrix is sorted
- Start at the lower left square
- If the element that we are looking for is smaller than the current square, look at the next column
- If the element is larger look at the previous row

```
def faster_search_matrix(mat, e):
  row = 1_______________
  col = 2_______________
  while 3_______________:
    if 4_______________:
      5_______________
    else:
      6_______________
  return (row, col)
```

1.	A	`0`	B	`len(mat)-1`	C	Other
2.	A	`0`	B	`len(mat)-1`	C	Other
3.	A	`mat[row][col] > e`	B	`mat[row][col]!= e`	C	`mat[row][col]==e`
4.	A	`mat[row][col] > e`	B	`mat[row][col]!= e`	C	`mat[row][col]==e`
5.	A	`row = row-1`	B	`col = col-1`	C	Other
6.	A	`row = row-1`	B	`col = col+1`	C	Other

Smarter Algorithm

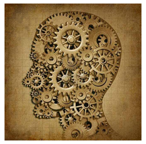
- We can use the fact that the matrix is sorted
- How does this algorithm work?
- How many comparisons to find 100?

```
def faster_search_matrix(mat, e):
  row = len(mat) - 1
  col = 0
  while mat[row][col] != e:
    if mat[row][col] > e:
      row = row - 1
    else:
      col = col + 1
  return (row, col)
```

http://goo.gl/qFDZmq

Pacing and Understanding

How well did you understand today?



- A Too easy, this lecture is way below my abilities
- B Everything went at a good pace, and I am fine
- C Too fast, but I will catch up on my own
- D Too fast, and I need you to slow down
- E I really do not think I can handle this