Announcements

- First midterm will be in class on Tuesday 10/13
  - Closed book but a sheet (8.5x11) of paper is allowed (both sides)

- Try to get help

- Reading assignment for this week:
  - Chapter 7 of Downey

- Break around 10:15am
Iteration: Repeating
Iteration: repeating things with `for`

- **The `for` loop: Syntax:** `for <variable> in <list>: <block>`
- Repeat the block once for each element in the list

```python
range(4) -> [0, 1, 2, 3]

for i in range(4):
    print i

Output:
0
1
2
3
```

```python
for i in range(4):
    print 'hello'

Output:
hello
hello
hello
hello
```
Iteration: repeating things with `while`

- The `while` loop syntax: `while <condition>: <body>

```python
def countdown(n):
    while n > 0:
        print n
        n = n - 1
    print 'Blastoff!'
countdown(3)
```

Output:
```
3
2
1
Blastoff!
```

1. Evaluate the condition, yielding True or False.
2. If False, exit the while statement and continue execution at the next statement.
3. If True, execute the body and go back to step 1.
def factorial(n):
    f = 1
    while n > 0:
        f = f * n
        n = n - 1
    return f

factorial(6)
720

def factorial(n):
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)

factorial(6)
720

# We will not worry about recursion
# but wanted to mention that there
# is another way of repeating for
# those who are interested.
• Write a function named `isPrime` that tests if a given integer is prime or not.

```python
def isPrime(n):
    i = 2
    while i < n/2:
        if n%i == 0:
            return False
        else:
            i = i + 1
    return True

isPrime(13)
isPrime(27)
```

`isPrime` is an example of an *algorithm*: it is a mechanical process for solving a category of problems (in this case, computing to determine if a number is prime or not).

The process of designing algorithms is intellectually challenging and a key part of programming.
Controlling loops using **break** (next)

- **break** statement breaks out of the **current** loop

```python
while True:
    line = raw_input('> ')
    if line == 'done':
        break
    print line

print 'Done!'
```
Controlling loops using `continue`

- `continue` statement causes Python to skip immediately ahead to the next iteration of a loop

- Example: Write a function called `add_digits` that adds up all the digits in a string ignoring non-digit characters and returns the sum

```python
def sum_digits(s):
    sum = 0
    for c in s:
        if c.isalpha():  # See Section 5.6.1 of
            continue  # the Python Library for isalpha
        sum = sum + int(c)
    return sum

sum_digits('ab2c3d4ef')
```
Exercise (optional)

- The built-in function `eval` takes a string and evaluates it using the Python interpreter. For example:

  ```python
  >>> eval('1+2*3')
  7
  >>> import math
  >>> eval('math.sqrt(5)')
  2.23606797749979
  ```

- Write a function called `evalLoop` that iteratively prompts the user, takes the resulting input and evaluates it using `eval`, and prints the result. It should continue until the user enters ‘done’, and then return the value of the last expression it evaluated.
def evalLoop():
    last = 'None'
    while True:
        line = raw_input('myPython> ')
        if line == 'done':
            return eval(last)
        last = line
        print eval(line)

print evalLoop()
Nested loops

- Loops may be nested in any depth mixing for loops and while loops as needed

```python
def nested_fors():
    for i in range(3):
        for j in range(4):
            print i, j

nested_fors()
```

```python
def nested_whiles():
    i = 0
    while i < 3:
        j = 0
        while j < 4:
            print i, j
            j = j + 1
        i = i + 1

nested_whiles()
```
Nested loops 2

- Very similar to the ones on the previous page, but generate rather different output

```python
def nested_fors2():
    for i in range(3):
        for j in range(4):
            print j,
        print

    nested_fors2()

def nested_whiles2():
    i = 0
    while i < 3:
        j = 0
        while j < 4:
            print j,
            j = j + 1
        print
        i = i + 1

    nested_whiles2()
```
Exercises

• Write a function named square_block that prints the following pattern if its two parameter values are 3 and 10 using either for loops or while loops or combining one each:

    **********
    **********
    **********
    **********

• Write a function named triangle that prints the following pattern if its single parameter value is 5 again using either for loops or while loops or combining one each:

    *
    **
    ***
    ****
    *****
    ******
Output formatting

- Try the following to see what happens
  > print '123'
  > print '1\t2\t3'
  > print '1
2
3'
  > for c in 'apple':
    print c,
  > for c in 'apple':
    print '\b', c,
  > for c in 'apple':
    print '\b' + c,
  > for c in 'apple':
    print '\b' + c
Working with Canopy

- As you work with loops, if you mistakenly create an infinite loop, your Python interpreter (i.e., Canopy) will be happy running infinitely giving you the impression that your program is hung.

- You can get out of that by hitting `control-c` (while holding down the control key, hit c)
Do these before next class

- Finish reading Chapter 5, 6, and 7
  - Try the examples using your Python installation

- Next time: Chapter 8 Strings