Dealing with floats

- Floats approximate real numbers, but useful to understand how
- Decimal number:
  - 302 = 3*10**2 + 0*10**1 + 2*10**0

Remember: ** is Python's exponentiation operator

- Binary number
  - 10011 = 1*2**4 + 0*2**3 + 0*2**2 + 1*2**1 + 1*2**0
  - (which in decimal is 16 + 2 + 1 = 19)
- Internally, computer represents numbers in binary

Converting decimal integer to binary

- Consider example of
  - x = 1*2**4 + 0*2**3 + 0*2**2 + 1*2**1 + 1*2**0
- If we take remainder relative to 2 `(x%2)` of this number, that gives us the last binary bit
- If we then divide x by 2 `(x/2)`, all the bits get shifted left
  - x/2 = 1*2**3 + 0*2**2 + 0*2**1 + 1*2**0 = 1001
- Keep doing successive divisions; now remainder gets next bit, and so on
- Let’s convert to binary form

Doing this in Python

```
if num < 0:
    isNeg = True
    num = abs(num)
else:
    isNeg = False
result = ‘’
if num == 0:
    result = ‘0’
while num > 2:
    result = str(num%2) + result
    num = num/2
if isNeg:
    result = ‘-’ + result
```

Next bit
Shift left

Convert

So what about fractions?

- 3/8 = 0.375 = 3*10**(-1) + 7*10**(-2) + 5*10**(-3)
- So if we multiply by a power of 2 big enough to convert into a whole number, can then convert to binary, then divide by the same power of 2
- 0.375 * (2**3) = 3 (decimal)
- Convert 3 to binary (now 11)
- Divide by 2**3 (shift left) to get <u>0.011</u> (binary)

```
x = float(raw_input('Enter a decimal number between 0 and 1: '))

p = 0
while ((2**p)*x)%1 != 0:
    print('Remainder = ' + str((2**p)*x - int((2**p)*x)))
    p += 1

num = int(x*(2**p))

result = ''
if num == 0:
    result = '0'
while num > 0:
    result = str(num%2) + result
    num = num/2

for i in range(p - len(result)):
    result = '0' + result

result = result[0:-p] + '.' + result[-p:]
print('The binary representation of the decimal ' + str(x) + ' is
' + str(result))
```

Some implications

- If there is no integer p such that x*(2**p) is a whole number, then internal representation is always an approximation
- Suggest that testing equality of floats is not exact
  – Use abs(x-y) < 0.0001, rather than x == y
- Why does print(0.1) return 0.1, if not exact?
  – Because Python designers set it up this way to automatically round