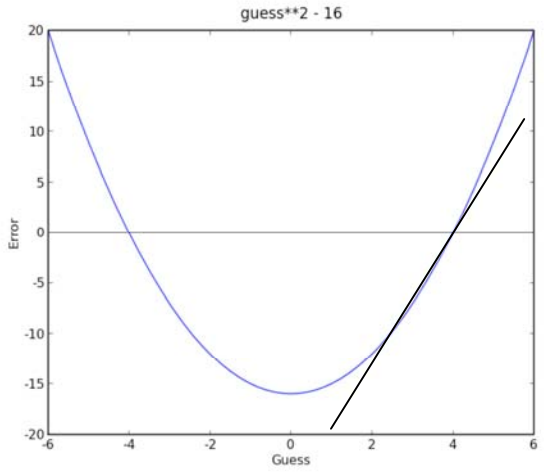
6.00 Introduction to Computer Science and Programming
Fall 2008

6.00 Handout, Lecture 6
(Not intended to make sense outside of lecture)

```
def squareRootBi(x, epsilon):
    """Assumes x >= 0 and epsilon > 0
       Return y s.t. y*y is within epsilon of x"""
    assert x >= 0, 'x must be non-negative, not' + str(x)
    assert epsilon > 0, 'epsilon must be positive, not' + str(epsilon)
    low = 0
    high = x
    guess = (low + high)/2.0
    ctr = 1
    while abs(guess**2 - x) > epsilon and ctr <= 100:
        #print 'low:', low, 'high:', high, 'guess:', guess
        if guess**2 < x:
            low = guess
        else:
            high = guess
        guess = (low + high)/2.0
        ctr += 1
    assert ctr <= 100, 'Iteration count exceeded'
    print 'Bi method.  Num. iterations:', ctr, 'Estimate:', guess
    return guess
-----------------------------------
def squareRootNR(x, epsilon):
    """Assumes x >= 0 and epsilon > 0
       Return y s.t. y*y is within epsilon of x"""
    assert x >= 0, 'x must be non-negative, not' + str(x)
    assert epsilon > 0, 'epsilon must be positive, not' + str(epsilon)
    x = float(x)
    guess = x/2.0
    guess = 0.001
    diff = guess**2 - x
    ctr = 1
    while abs(diff) > epsilon and ctr <= 100:
        #print 'Error:', diff, 'guess:', guess
        guess = guess - diff/(2.0*guess)
        diff = guess**2 - x
        ctr += 1
    assert ctr <= 100, 'Iteration count exceeded'
    print 'NR method.  Num. iterations:', ctr, 'Estimate:', guess
    return guess
```



```
Techs = ['MIT', 'Cal Tech']
print Techs
Ivys = ['Harvard', 'Yale', 'Brown']
print Ivys
Univs = []
Univs.append(Techs)
print Univs
Univs.append(Ivys)
raw_input()
print Univs
raw_input()
for e in Univs:
    print e
    for c in e: print c
raw_input()
Univs = Techs + Ivys
print Univs
raw_input()
Ivys.remove('Harvard')
print Univs
Ivys[1] = -1
print Ivys

L1 = [1, 2, 3]
L2 = L1
L1[0] = 4
print L2

def f(L):
    L[0] = 4
L1 = [1,2,3]
L2 = [1,2,3]
L3 = L1
print L1 == L2
f(L1)
print L1 == L2
print L1
print L2
print L3
```

```
EtoF = {'one': 'un', 'soccer': 'football'}
print EtoF['soccer']
print EtoF[0]
print EtoF
NtoS = {1: 'one', 2: 'two', 'one': 1, 'two': 2}
print NtoS.keys()
print NtoS.keys
del NtoS['one']
print NtoS

L = [['un', 'one'], ['deux', 'two']]
def keySearch(L, k):
    for elem in L:
        if elem[0] == k: return elem[1]
    return None

print keySearch(L, 'deux')
```

```
L1 = [1,2,3]
L2 = L1[:]  #makes a copy of L1
```

```
L = [['un', 'one'], ['deux', 'two']]
def keySearch(L, k):
    for elem in L:
        if elem[0] == k:
            return elem[1]
    return None
```