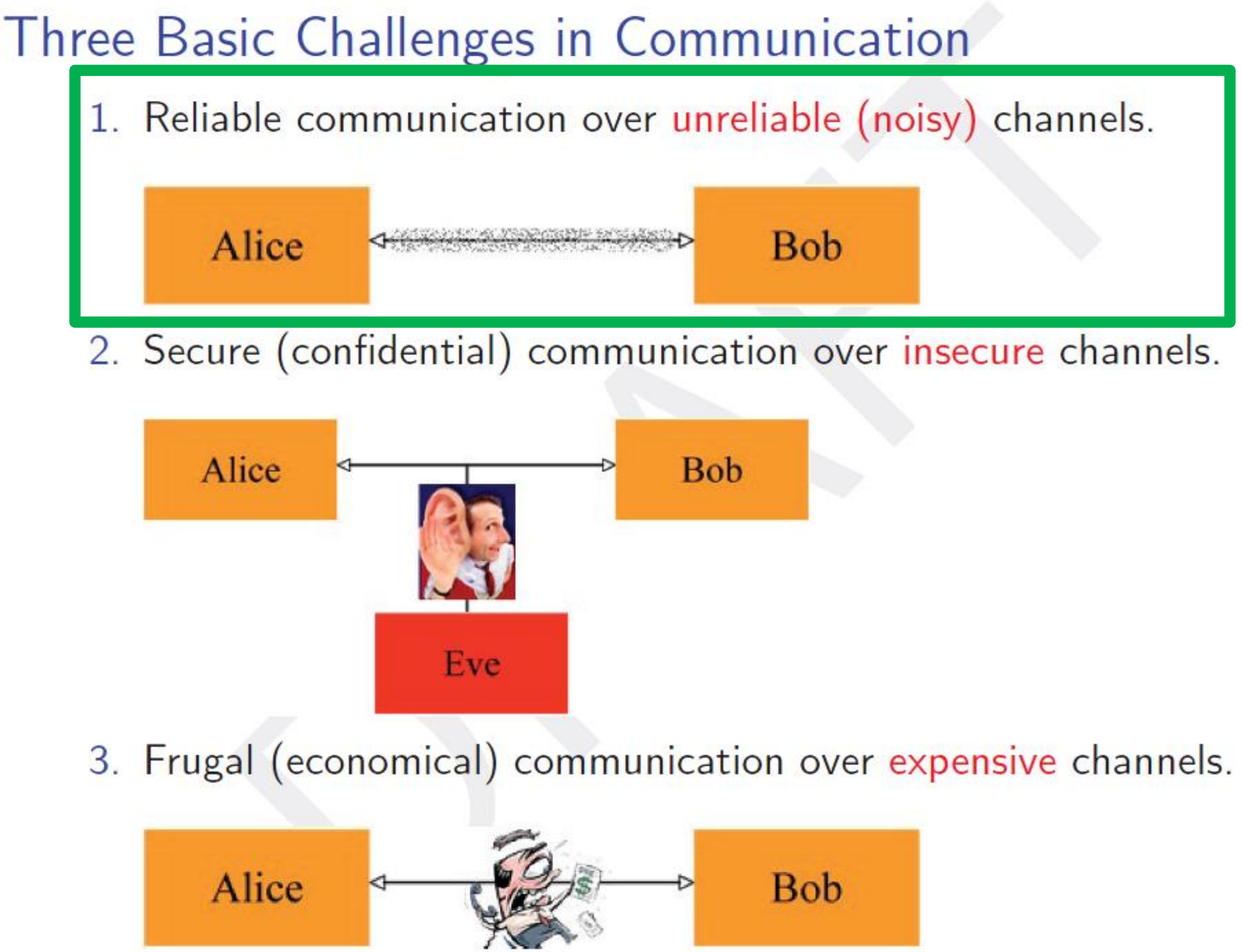
Error detection and correction

The big picture

courtsey of Prof. Benny Chor



Three Basic Challenges in Communication

1. Reliable communication over unreliable (noisy) channels.

Today

2. Secure (confidential) communication over insecure channels.
3. Frugal (economical) communication over expensive channels.

Plan

- Error detection / error correction
- Card magic
- ID (ספרת ביקורת)
- RAID (redundant array of independent disks)
- (yet another) Spell checker

Card magic

- Each card has two sides: **0** and **1**

1. The magician arranges cards in a square, then looks away
2. An audience member flips one of the cards
3. The magician turns back, and reveals which card was flipped

0	1	0	1	0
1	1	1	0	1
1	0	0	1	1
0	0	1	0	1
0	1	0	0	1

Source: http://csu-il.blogspot.co.il/2008/03/blog-post_365.html

Mind reading card trick

- Error correction / error identification
- Error correcting for one card flip
- What if 2 cards flip? 3? 4?
- Applications:
    - Messages between computers
    - Hard disk drive
    - CD
    - Barcode
    - Spelling corraction

XOR – exclusive Or

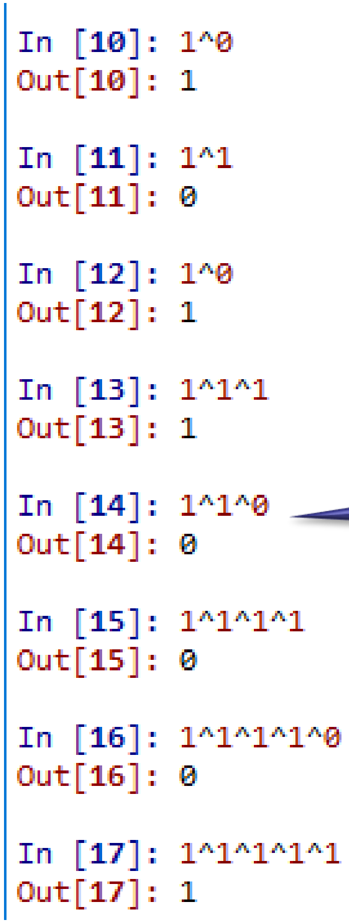
- Logic operator on 2 arguments
- “One or the other but not both“
- Returns:
    - True - one argument is True and other is False
    - False - both arguments are True or both are False
- Examples (for True statements):
    - I’m happy XOR I’m sad
    - It’s raining XOR it’s not raining
- http://en.wikipedia.org/wiki/Exclusive_or

XOR – Exclusive Or

- “One or the other but not both“
- Bitwise operation (Python ^):
- Returns 1 if the bits are different, 0 if bits are identical
  - 1 XOR 1 = 0
  - 1 XOR 0 = 1
  - 0 XOR 1 = 1
  - 0 XOR 0 = 0
  - 1110 XOR 1001 = 0111
- Equivalent to addition without carry
- Useful to calculate parity bit:
- 1 XOR 1 XOR 0 XOR 1 XOR 0
  - 1 means TOTOAL odd number of ones

XOR Examples

- A^0 = A
- A^1 = not(A)



```
In [10]: 1^0
Out[10]: 1

In [11]: 1^1
Out[11]: 0

In [12]: 1^0
Out[12]: 1

In [13]: 1^1^1
Out[13]: 1

In [14]: 1^1^0
Out[14]: 0

In [15]: 1^1^1^1
Out[15]: 0

In [16]: 1^1^1^1^0
Out[16]: 0

In [17]: 1^1^1^1^1
Out[17]: 1
```

Even number of ones

Odd number of ones

XOR For Ints

- What is XOR for non-binary values?

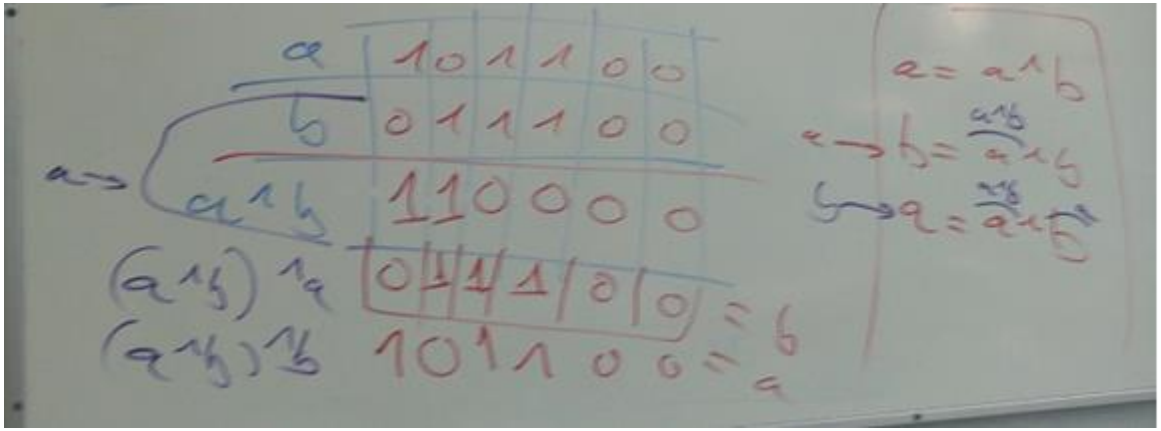
- XOR all the bits, get a new integer
    - Present the ingeter

- E.g. : 6 ^ 5 = 110 ^ 101 = 011
    - 3
- 2^5 = 010 ^ 101 = 111 =$(7)_{10}$

XOR Example – swap 2 vars

- Problem:
- Can you swap between 2 integers without using temp (or a,b=b,a) ?
- a=a^b
- b=a^b
- a=a^b

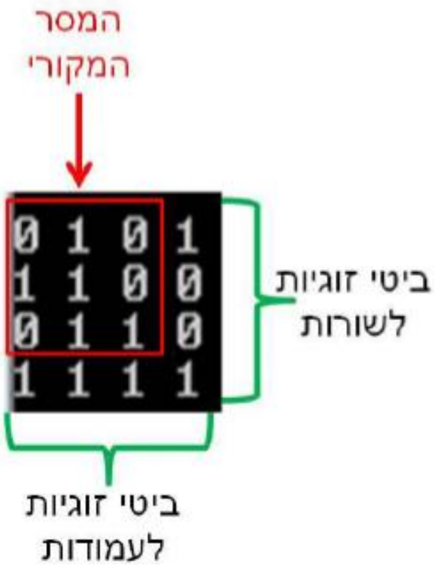
XOR “encodes” the number



a	1	0	1	1	0	0	
b	0	1	1	1	0	0	
a^b	1	1	0	0	0	0	
(a^b)^a	0	1	1	1	0	0	= b
(a^b)^b	1	0	1	1	0	0	= a

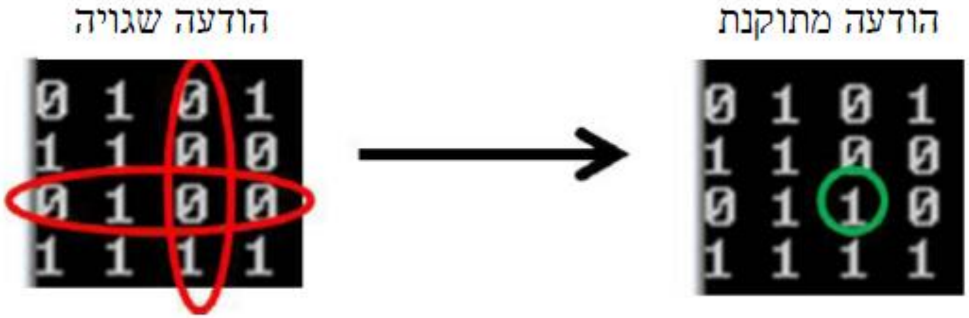
a →
a = a^b
a → b = $\overset{a^b}{a}$^b
b → a = $\overset{a^b}{a}$^$\overset{a}{b}$

Error Correcting



```
0 1 0 1
1 1 0 0
0 1 1 0
1 1 1 1
```

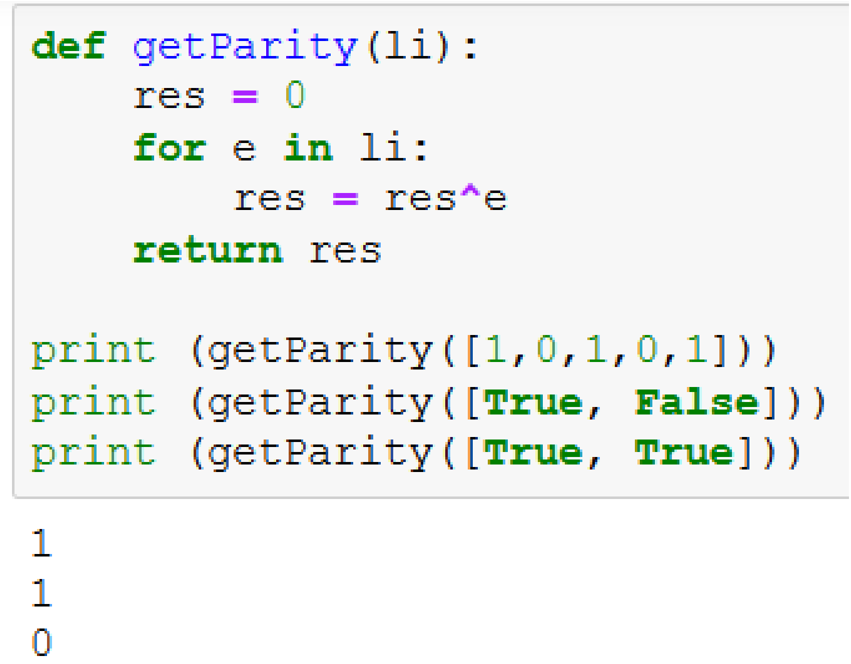
A row (or column)
is intact if-and-only-if
its parity is 0!



```
0 1 0 1
1 1 0 0
0 1 0 0
1 1 1 1
```

```
0 1 0 1
1 1 0 0
0 1 1 0
1 1 1 1
```

Get Parity

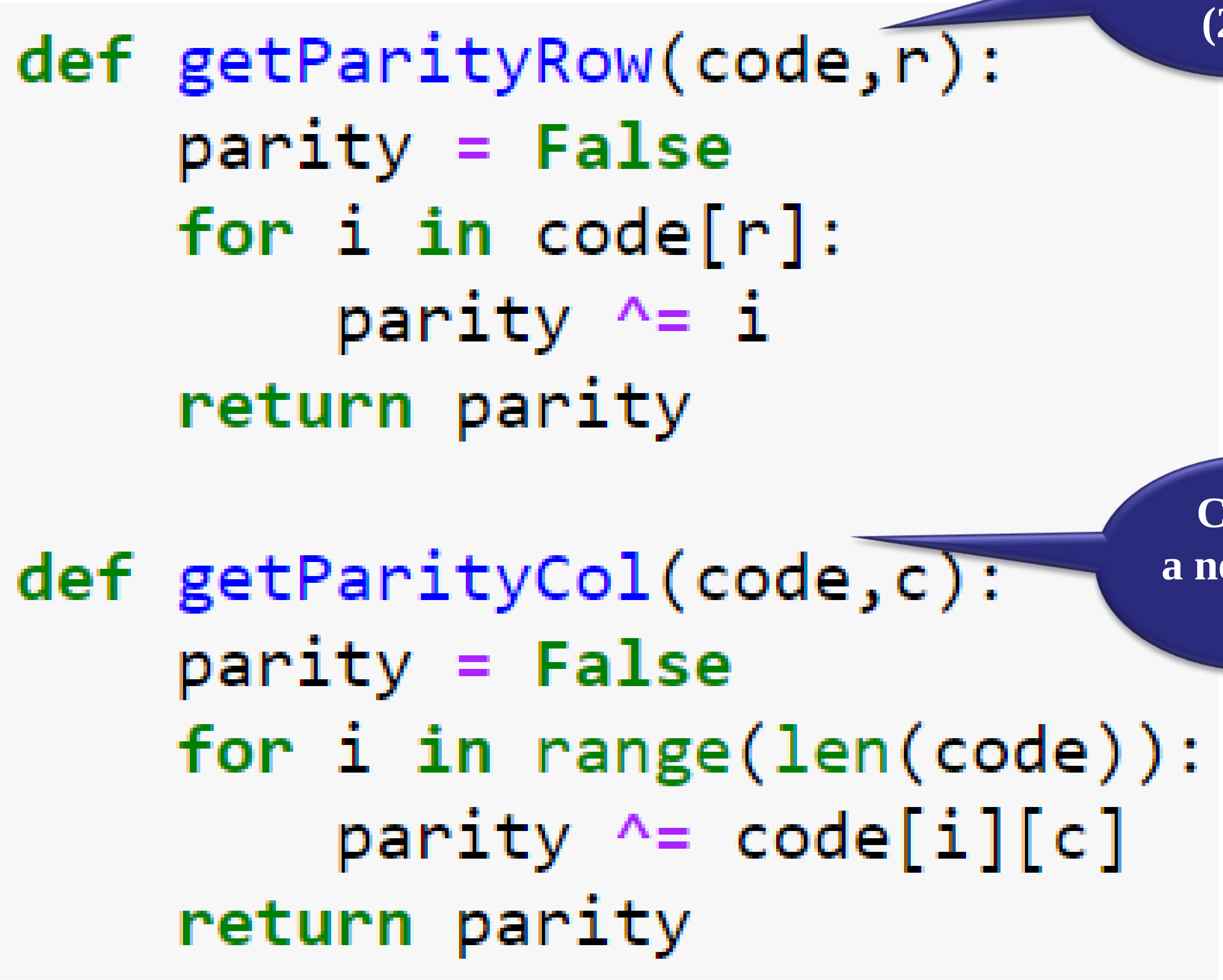


```
def getParity(li):
    res = 0
    for e in li:
        res = res^e
    return res

print (getParity([1,0,1,0,1]))
print (getParity([True, False]))
print (getParity([True, True]))
```

```
1
1
0
```

(Reminder: False = 0, True = 1)

Get Parity



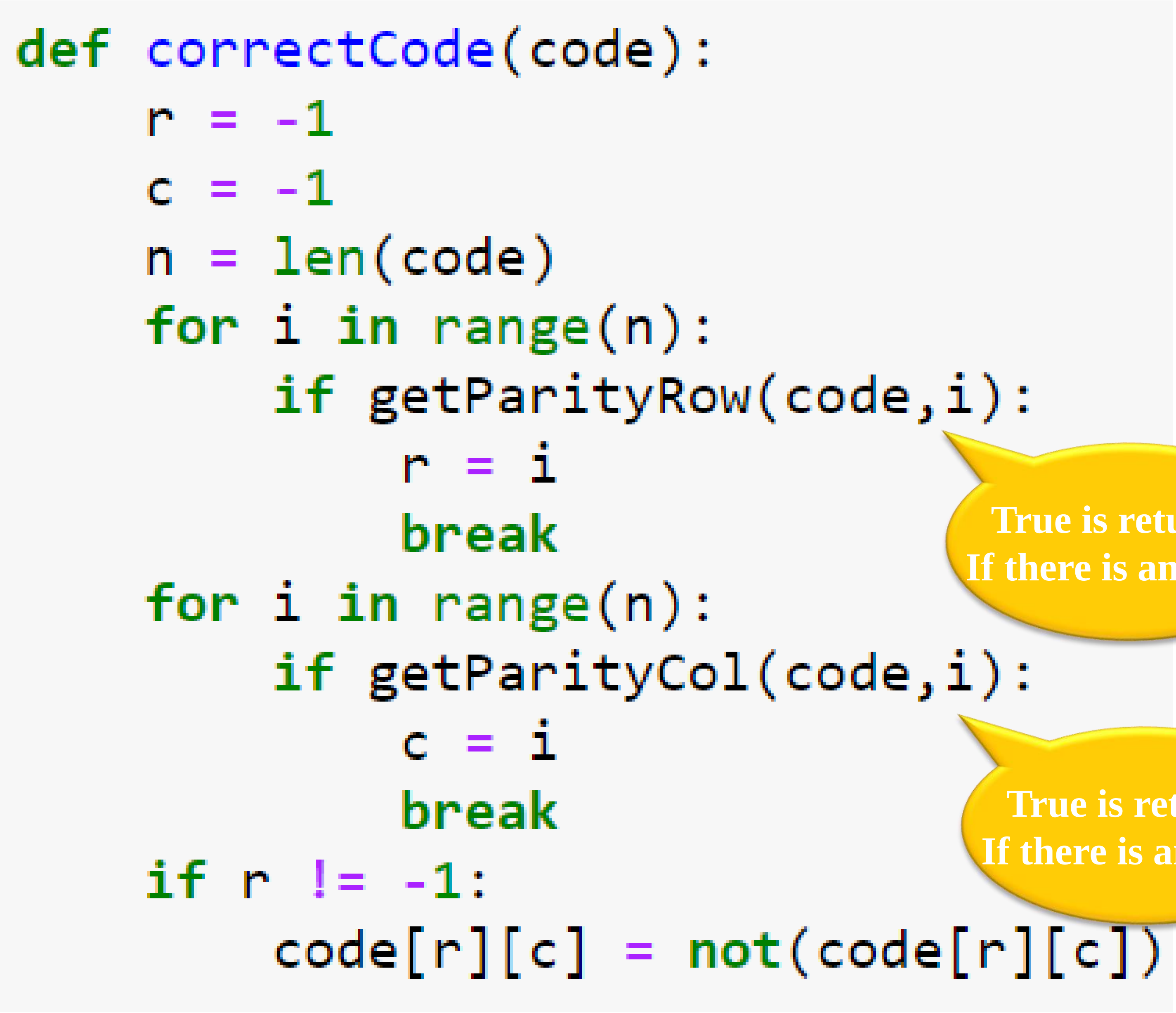
```python
def getParityRow(code,r):
    parity = False
    for i in code[r]:
        parity ^= i
    return parity

def getParityCol(code,c):
    parity = False
    for i in range(len(code)):
        parity ^= code[i][c]
    return parity
```

Code is
a nested list
(2D)

Code is
a nested list
(2D)

(Reminder: False – 0, True - 1)

Correcting Erroneous Code (a single error)



```python
def correctCode(code):
    r = -1
    c = -1
    n = len(code)
    for i in range(n):
        if getParityRow(code,i):
            r = i
            break
    for i in range(n):
        if getParityCol(code,i):
            c = i
            break
    if r != -1:
        code[r][c] = not(code[r][c])
```

True is returned
If there is an error!

True is returned
If there is an error!

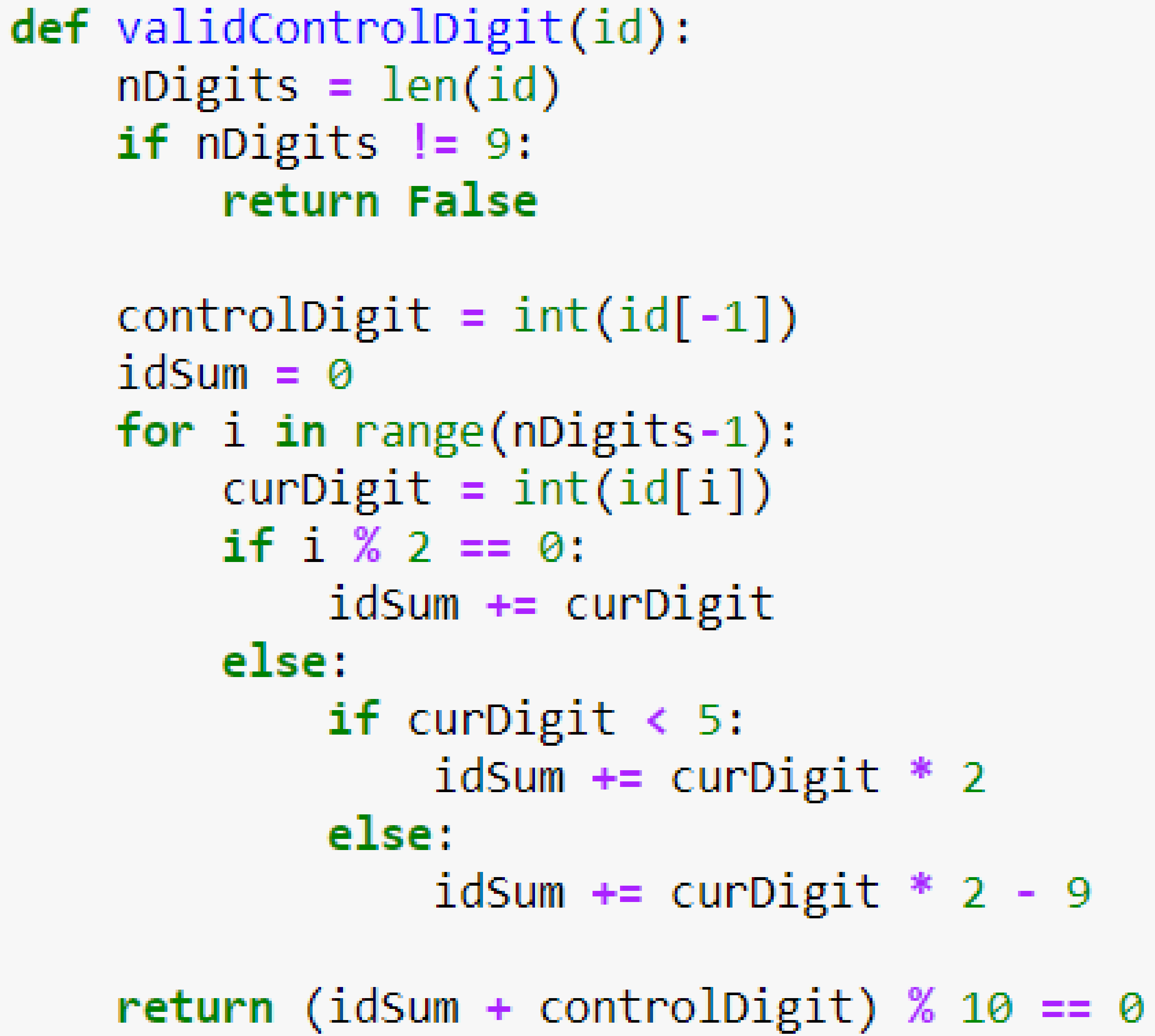
Israeli ID Error Detection

- Israeli ID: unique per person, 9 digits
- Rightmost digit is **control digit**
- How is the control digit defined/checked?
  - Consider first 8 ID digits
  - For every **2nd** digit d:
    - $d < 5$ → replace *d* with $2*d$
    - $d \geq 5$ → replace *d* with $2*d + 1 - 10$
  - Sum up all 8 digits
  - The control digit ***c*** is such that **sum+*c*** is a multiple of 10
- Next slide: example for ID 053326187

Example: 053326187

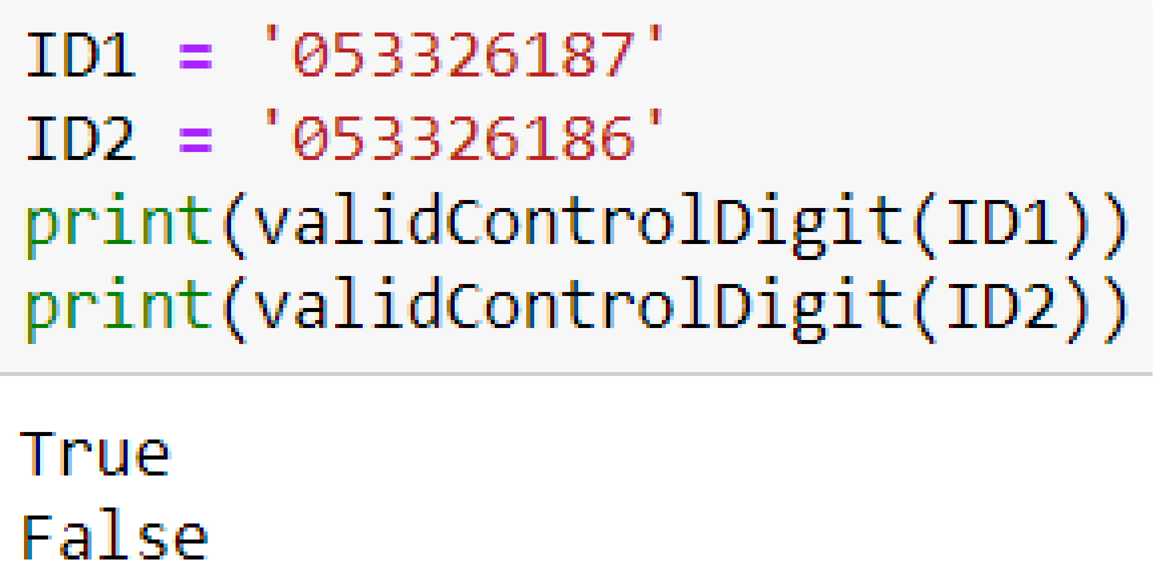
$$\begin{array}{ccccccccccccccccc} 0 & & 5 & & 3 & & 3 & & 2 & & 6 & & 1 & & 8 & & \boxed{7} \\ & & \downarrow & & & & \downarrow & & & & \downarrow & & & & \downarrow & & \\ 0 & + & 1 & + & 3 & + & 6 & + & 2 & + & 3 & + & 1 & + & 7 & = & 23 \end{array}$$

(23 + control_digit) % 10 = 0



```python
def validControlDigit(id):
    nDigits = len(id)
    if nDigits != 9:
        return False

    controlDigit = int(id[-1])
    idSum = 0
    for i in range(nDigits-1):
        curDigit = int(id[i])
        if i % 2 == 0:
            idSum += curDigit
        else:
            if curDigit < 5:
                idSum += curDigit * 2
            else:
                idSum += curDigit * 2 - 9

    return (idSum + controlDigit) % 10 == 0
```

Testing..



```
ID1 = '053326187'
ID2 = '053326186'
print(validControlDigit(ID1))
print(validControlDigit(ID2))
```

```
True
False
```

Exercises

- Write a function called findDigit, which receives an Israeli ID as input, without the control digit, and prints the control digit
  - Use validate **control digit**

- Write a function called findAllDigits, which receives as input an array of IDs **without** digits, and writes all IDs

RAID

- Redundant array of independent disks
- http://en.wikipedia.org/wiki/RAID
- Add XOR disk
- How to fix a flawed disk’s data?

RAID – Example

Disk 1	111	101	001	101	000	110	101	001	010	011
Disk 2	010	111	000	000	001	001	011	110	111	010
Disk 3	010	100	011	011	101	100	001	100	110	010
Disk 4	111	110	111	001	000	010	000	011	101	010
Disk 5	001	101	001	000	111	010	100	111	011	000

RAID – Control Disk

Disk 1	111	101	001	101	000	110	101	001	010	011
Disk 2	010	111	000	000	001	001	011	110	111	010
Disk 3	010	100	011	011	101	100	001	100	110	010
Disk 4	111	110	111	001	000	010	000	011	101	010
Disk 5	001	101	001	000	111	010	100	111	011	000
Disk 6	001	101	100	111	011	011	011	111	111	001

XOR on data disks

RAID – Set Data

Disk 1	111	101	001	101	000	110	101	001	010	011
Disk 2	010	111	000	000	001	001	011	110	111	010
Disk 3	010	100	011	101	101	100	001	100	110	010
Disk 4	111	110	111	001	000	010	000	011	101	010
Disk 5	001	101	001	000	111	010	100	111	011	000
Disk 6	001	101	100	001	011	011	011	111	111	001

RAID – Disk 4 is Faulty

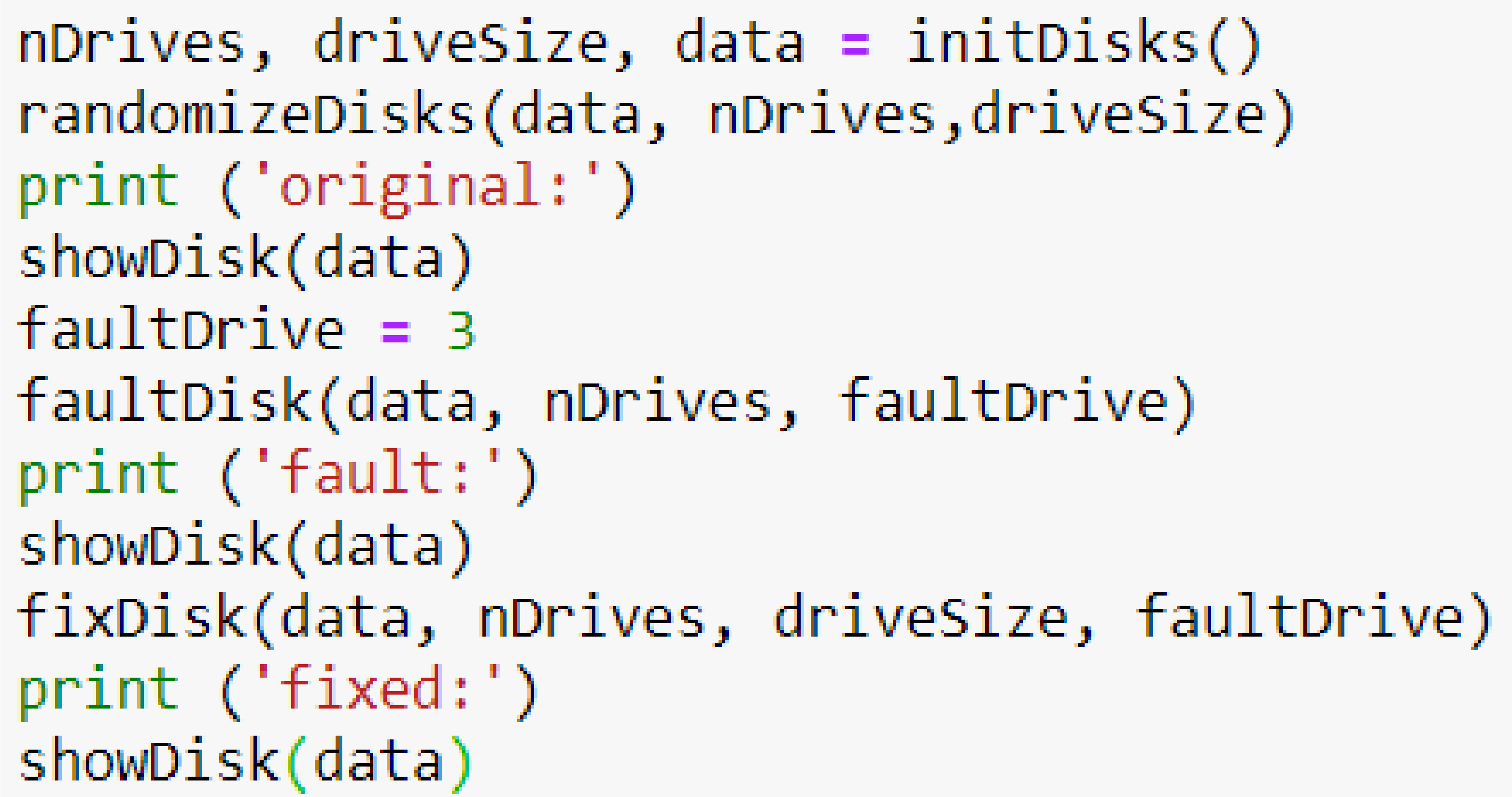
Disk 1	111	101	001	101	000	110	101	001	010	011
Disk 2	010	111	000	000	001	001	011	110	111	010
Disk 3	010	100	011	101	101	100	001	100	110	010
Disk 4	000	000	000	000	000	000	000	000	000	000
Disk 5	001	101	001	000	111	010	100	111	011	000
Disk 6	001	101	100	001	011	011	011	111	111	001

RAID – Fix Disk 4

Disk 1	111	101	001	101	000	110	101	001	010	011
Disk 2	010	111	000	000	001	001	011	110	111	010
Disk 3	010	100	011	101	101	100	001	100	110	010
Disk 4	111	110	111	001	000	010	000	011	101	010
Disk 5	001	101	001	000	111	010	100	111	011	000
Disk 6	001	101	100	001	011	011	011	111	111	001

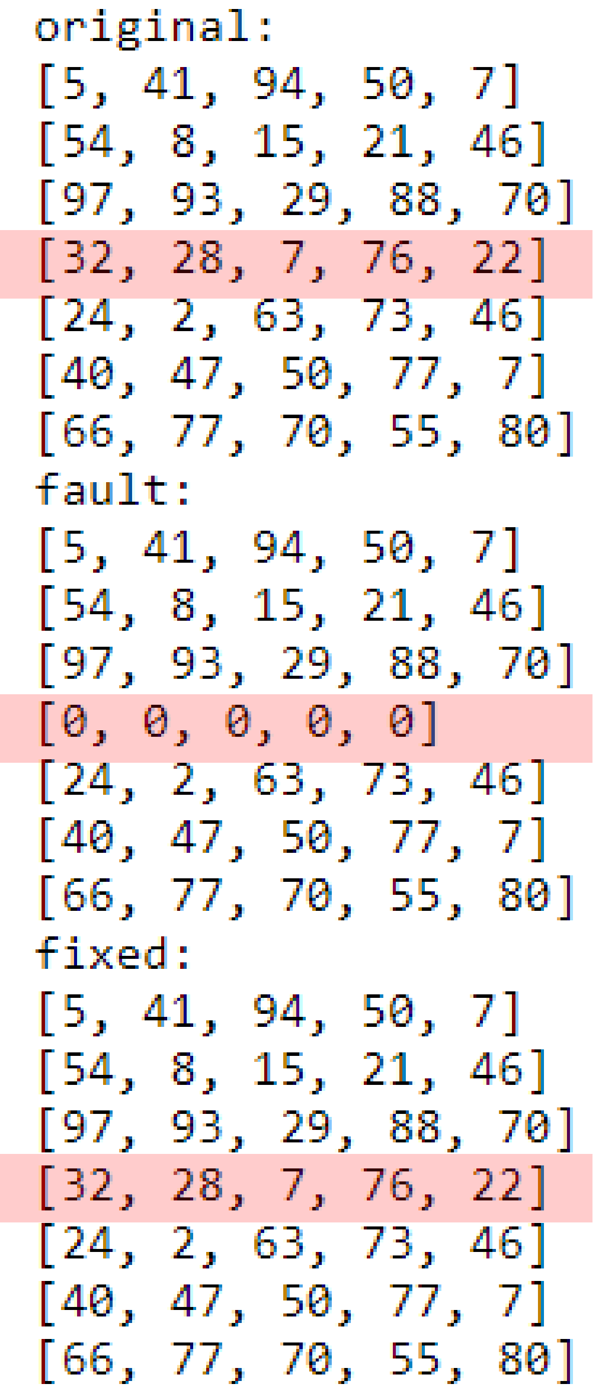
XOR on all other disks

Implementation (Top-Down)



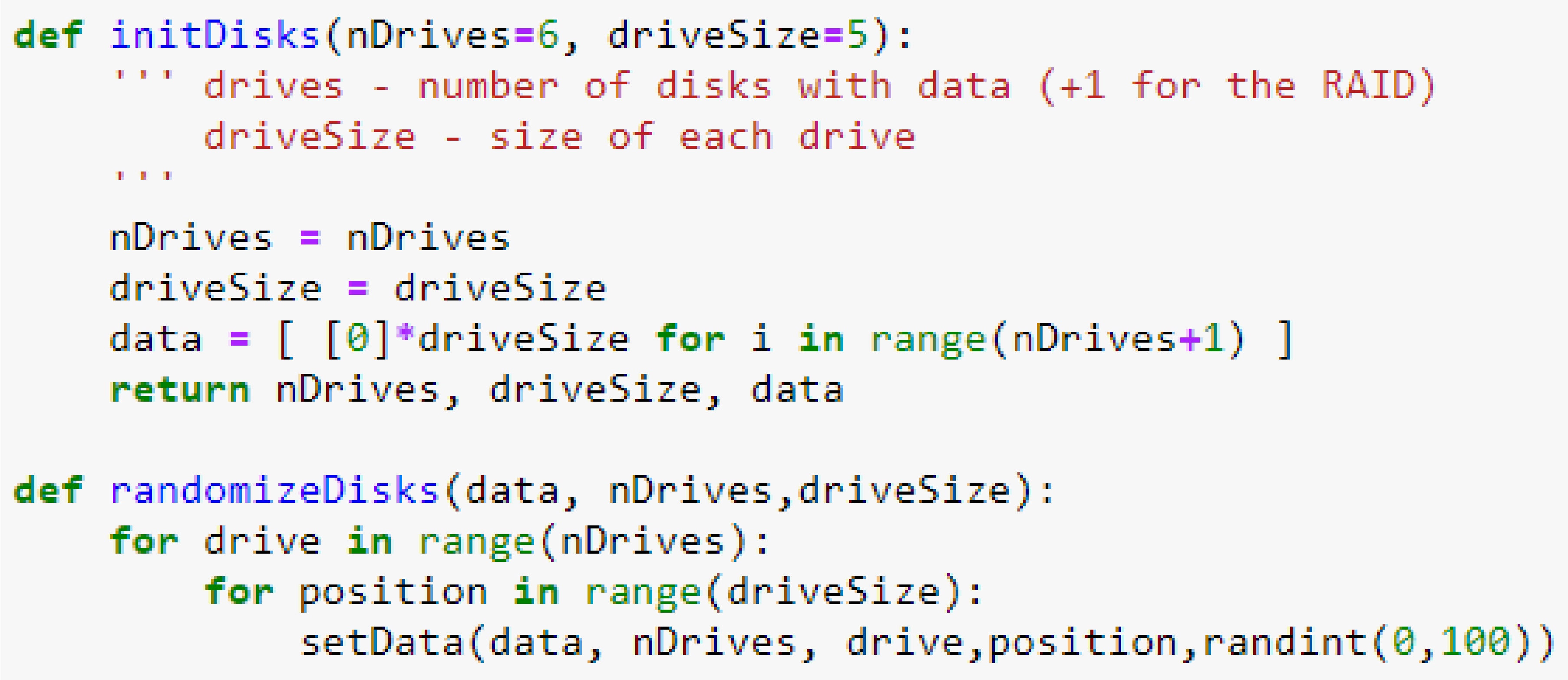
```
nDrives, driveSize, data = initDisks()
randomizeDisks(data, nDrives,driveSize)
print ('original:')
showDisk(data)
faultDrive = 3
faultDisk(data, nDrives, faultDrive)
print ('fault:')
showDisk(data)
fixDisk(data, nDrives, driveSize, faultDrive)
print ('fixed:')
showDisk(data)
```

Output

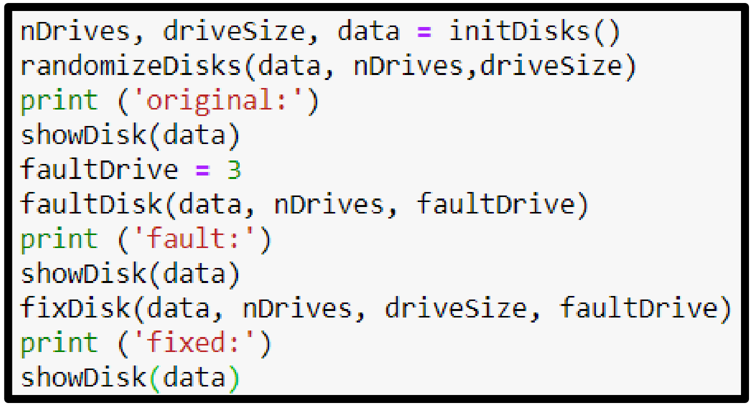


```
original:
[5, 41, 94, 50, 7]
[54, 8, 15, 21, 46]
[97, 93, 29, 88, 70]
[32, 28, 7, 76, 22]
[24, 2, 63, 73, 46]
[40, 47, 50, 77, 7]
[66, 77, 70, 55, 80]
fault:
[5, 41, 94, 50, 7]
[54, 8, 15, 21, 46]
[97, 93, 29, 88, 70]
[0, 0, 0, 0, 0]
[24, 2, 63, 73, 46]
[40, 47, 50, 77, 7]
[66, 77, 70, 55, 80]
fixed:
[5, 41, 94, 50, 7]
[54, 8, 15, 21, 46]
[97, 93, 29, 88, 70]
[32, 28, 7, 76, 22]
[24, 2, 63, 73, 46]
[40, 47, 50, 77, 7]
[66, 77, 70, 55, 80]
```

Initialization

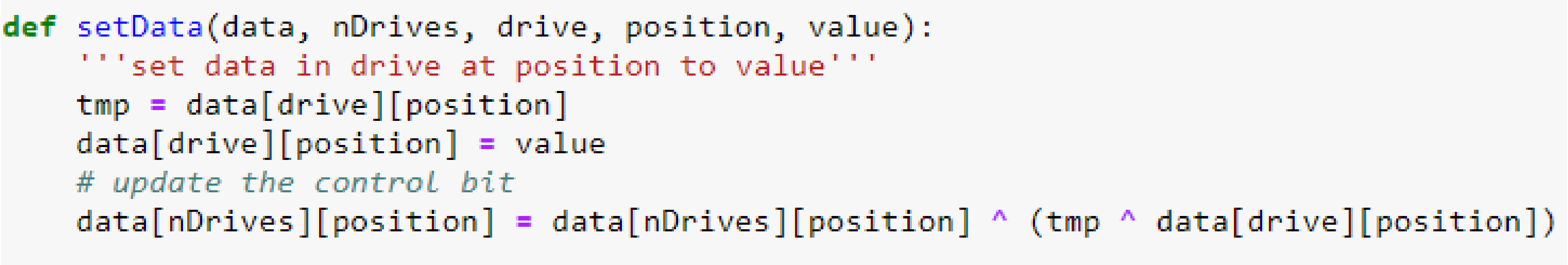


```python
def initDisks(nDrives=6, driveSize=5):
    ''' drives - number of disks with data (+1 for the RAID)
        driveSize - size of each drive
    '''
    nDrives = nDrives
    driveSize = driveSize
    data = [ [0]*driveSize for i in range(nDrives+1) ]
    return nDrives, driveSize, data

def randomizeDisks(data, nDrives,driveSize):
    for drive in range(nDrives):
        for position in range(driveSize):
            setData(data, nDrives, drive,position,randint(0,100))
```

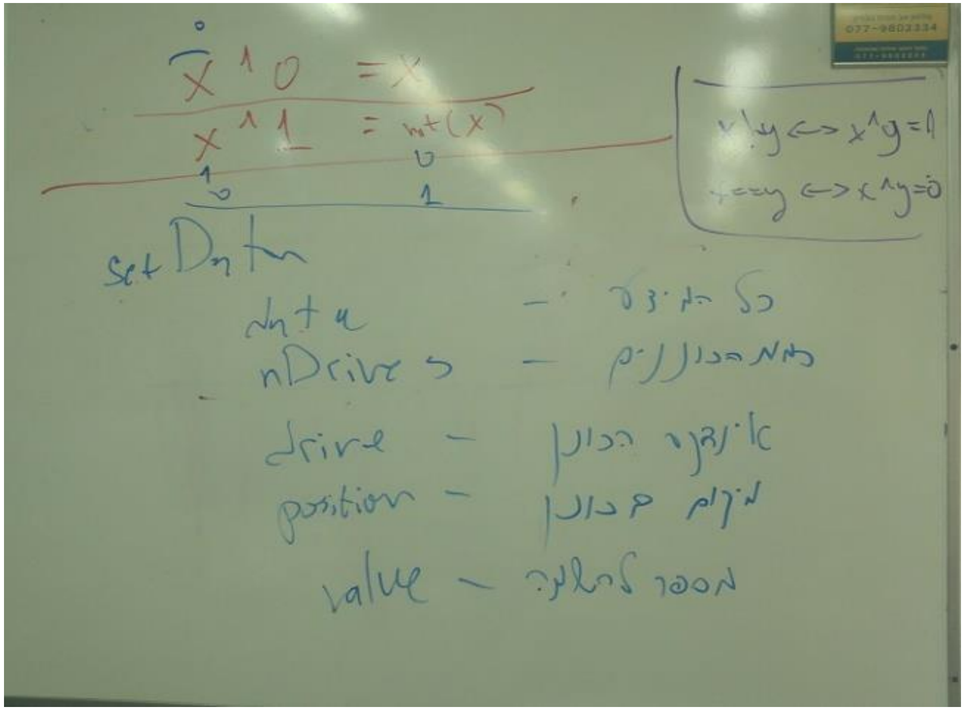


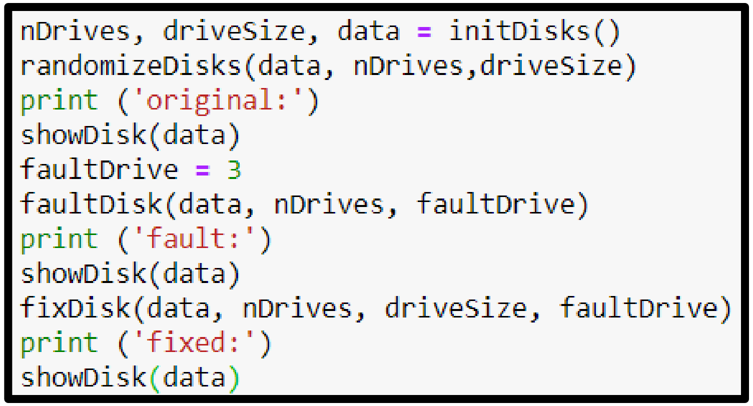
```python
nDrives, driveSize, data = initDisks()
randomizeDisks(data, nDrives,driveSize)
print ('original:')
showDisk(data)
faultDrive = 3
faultDisk(data, nDrives, faultDrive)
print ('fault:')
showDisk(data)
fixDisk(data, nDrives, driveSize, faultDrive)
print ('fixed:')
showDisk(data)
```

Setting Data



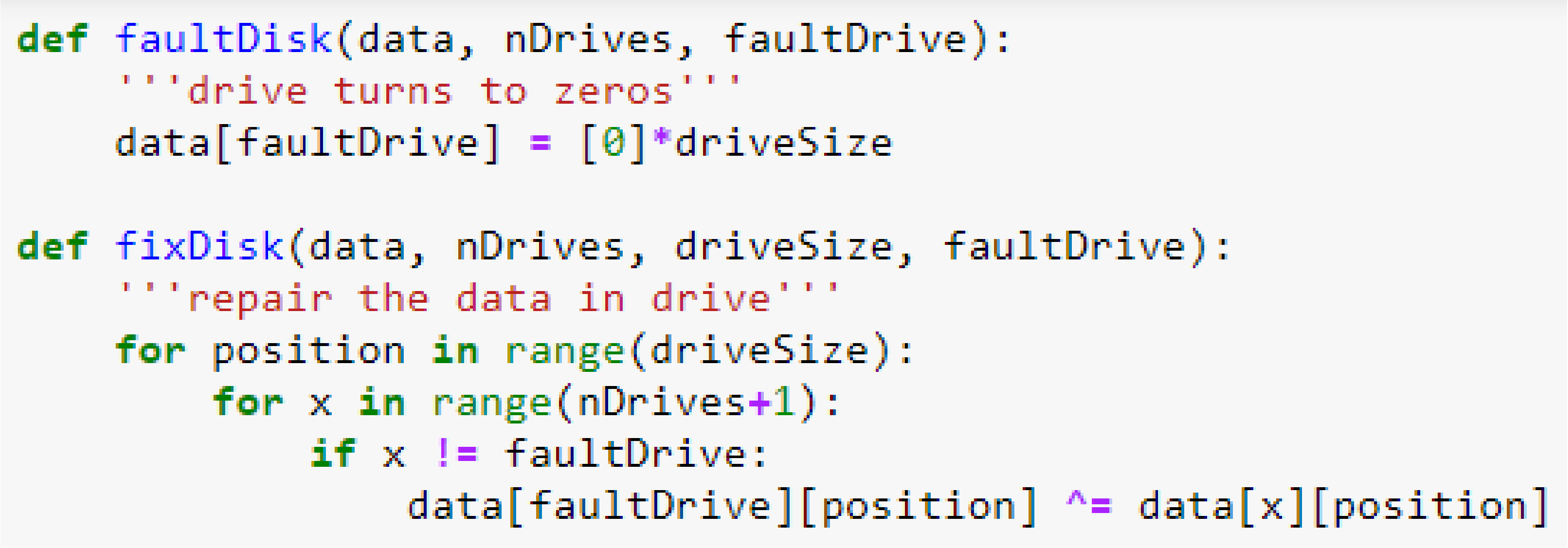
```python
def setData(data, nDrives, drive, position, value):
    '''set data in drive at position to value'''
    tmp = data[drive][position]
    data[drive][position] = value
    # update the control bit
    data[nDrives][position] = data[nDrives][position] ^ (tmp ^ data[drive][position])
```



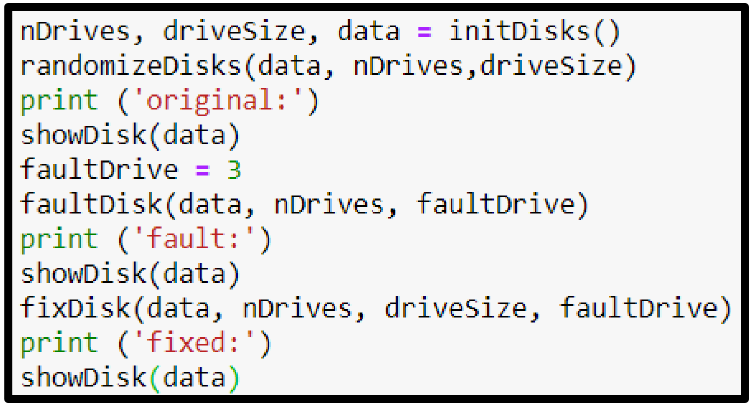


```python
nDrives, driveSize, data = initDisks()
randomizeDisks(data, nDrives,driveSize)
print ('original:')
showDisk(data)
faultDrive = 3
faultDisk(data, nDrives, faultDrive)
print ('fault:')
showDisk(data)
fixDisk(data, nDrives, driveSize, faultDrive)
print ('fixed:')
showDisk(data)
```

Fixing a Faulty Disk

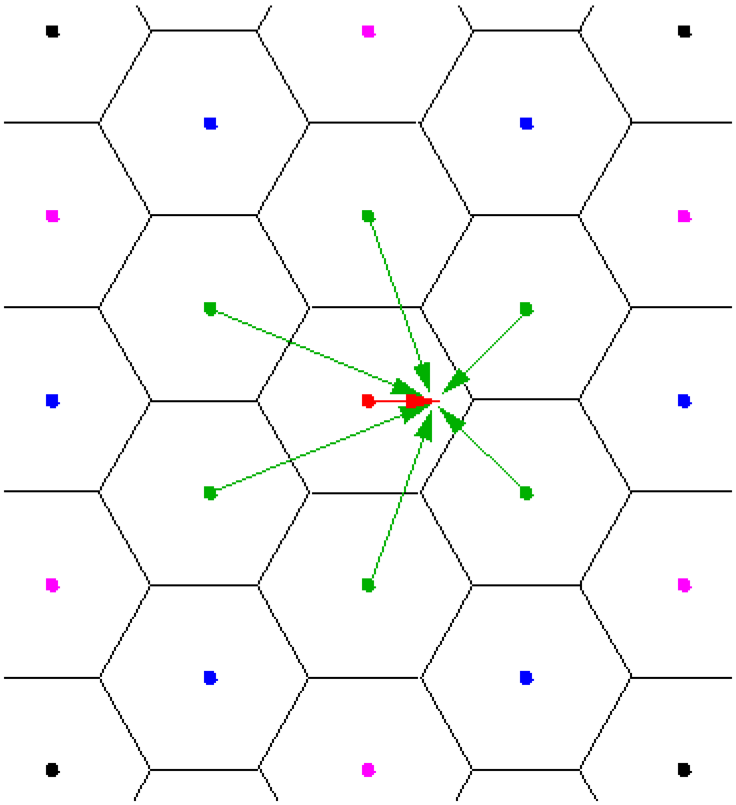


```python
def faultDisk(data, nDrives, faultDrive):
    '''drive turns to zeros'''
    data[faultDrive] = [0]*driveSize

def fixDisk(data, nDrives, driveSize, faultDrive):
    '''repair the data in drive'''
    for position in range(driveSize):
        for x in range(nDrives+1):
            if x != faultDrive:
                data[faultDrive][position] ^= data[x][position]
```



```python
nDrives, driveSize, data = initDisks()
randomizeDisks(data, nDrives,driveSize)
print ('original:')
showDisk(data)
faultDrive = 3
faultDisk(data, nDrives, faultDrive)
print ('fault:')
showDisk(data)
fixDisk(data, nDrives, driveSize, faultDrive)
print ('fixed:')
showDisk(data)
```

Nearest Neighbor



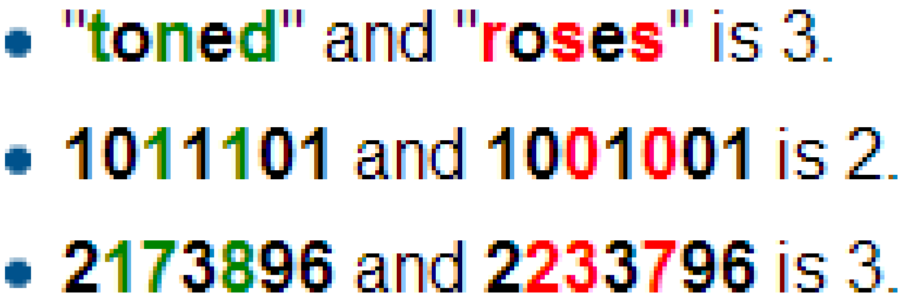
"It's disgraceful the way they build houses so close together!"

The Idea

- Given a word we would like to find the **closest** **correctly-spelled** word
- **Correctly-spelled**: predetermined list of "correct" words
- **Closest**:
  - Nearest neighbor!
  - But how to define a distance metric?
  - How to do it efficiently?

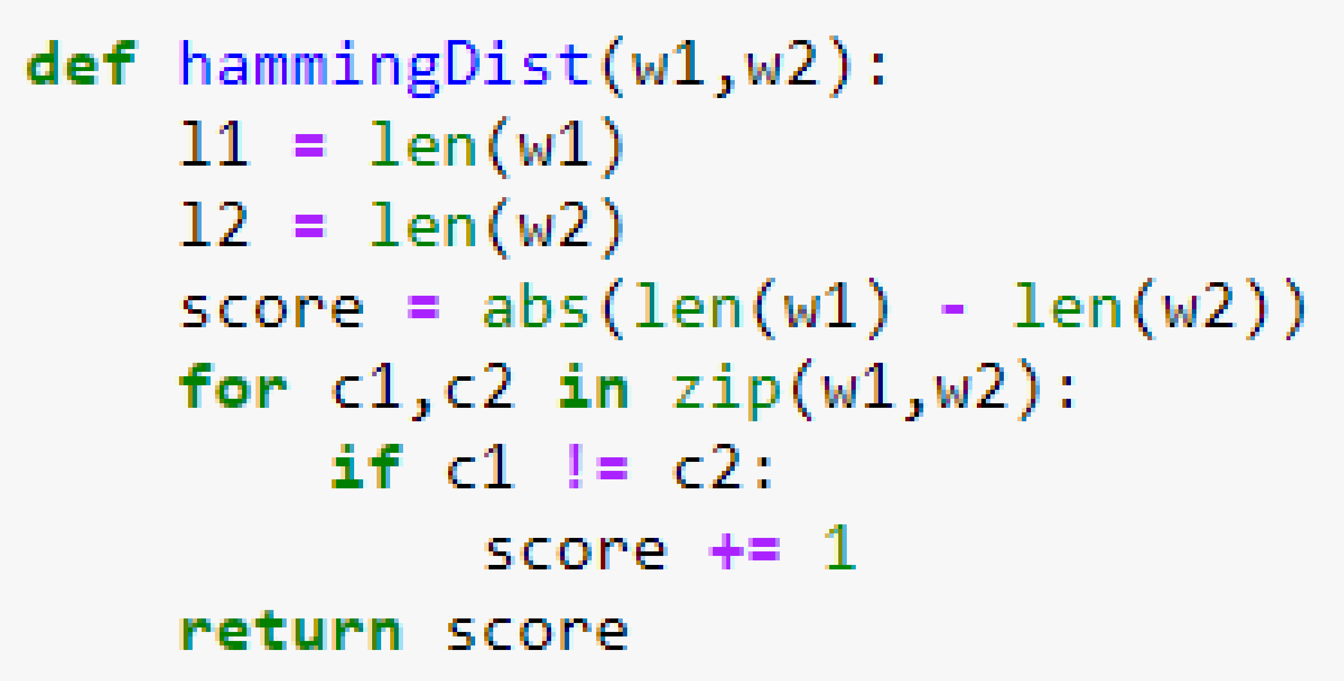
Hamming Distance

- Hamming distance, http://en.wikipedia.org/wiki/Hamming_distance
- The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different
- Examples: the Hamming distance between



- "**toned**" and "**roses**" is 3.
- **1011101** and **1001001** is 2.
- **2173896** and **2233796** is 3.

Hamming Distance Implementation



```
def hammingDist(w1,w2):
    l1 = len(w1)
    l2 = len(w2)
    score = abs(len(w1) - len(w2))
    for c1,c2 in zip(w1,w2):
        if c1 != c2:
            score += 1
    return score
```

Exercises

- Write a function names print closest:
  - Receive an array of strings
  - For each string S write: the closest to S in … (and the closest so S) the farthest from S is…
-