Numbers and their bases

- Information on computers are all represented as numbers. For example, ord('a') == 97, ord('+') == 43 (See ASCII table). All audio, video, and photos are represented as numbers.
- Further more, all information are represented as binary numbers on computers! 42 == 101010, 97 == 1100001
- We'll learn exactly how this works in the next segment of the lectures.
- For now, we want to learn how the numbers can be converted among different bases.

Base 2 | Base 10

THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

101010

Each column represents another power of the base

TENs column ONEs column

42

4 tens + 2 ones

128s column SIXTYFOURs col THIRTYTWOs col. SIXTEENs col. EIGHTs column FOURs column TWOs column ONEs column

_ _ _ _ _ _ _ _

Write 123 in binary...

HUNDREDs column TENs column ONEs column

123

1 hundred + 2 tens + 3 ones

Convert from binary to decimal …

Strategy: add weighted bits together. The weight at each position is 2^k at kth position, k = 0, 1, … from right to left

```
101011 = 1*2^5 + 1*2^3 + 1*2^1 + 1*2^0
       = 32 + 8 + 2 + 1 = 43
```

Convert these two binary numbers to decimal:

32 16 8 4 2 1
110011

128 64 32 16 8 4 2 1
10001000

110011 = 1 + 2 + 16 + 32 = 51

10001000 = 8 + 128 = 136

Convert these two decimal numbers to binary:

28 | **101**

32 16 8 4 2 1 | 128 64 32 16 8 4 2 1

28 = 16 + 8 + 4 = 11100

101 = 64 + 32 + 4 + 1 = 1100101

Lab 4: Computing in binary

base 2 | **base 10**

128 64 32 16 8 4 2 1
10001000 = 100 10 1 **136**

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

←

RIGHT-to-LEFT conversion to binary is surprisingly simple!

```
numToBinary( N )
binaryToNum( S )
```

You'll write these right! (-to-left)

And to ***represent*** binary numbers ? **Strings!**

`'11001101000101011101001001'`

BINARY OPERATIONS

Add these two binary numbers
WITHOUT converting to decimal !

```
   101101
+    1110
---------
```

```
    1
   529
+  742
------
  1271
```

Hint:
Do you remember this algorithm? It's the same!

You'll need to write a function to do this on **hw4pr2**!

MULTIPLY these two binary numbers
WITHOUT converting to decimal !

```
   101101
*    1110
---------
```

```
   529
*   42
------
  1058
 2116
------
 22218
```

Hint:
Do you remember this algorithm? It's the same!

You need to add two binary numbers.

Let's first explore adding
two decimal numbers...

...but entirely as strings!

adding decimal *strings*

e.g., S = '31' T = '11'

```
def add10(s,t):
    """ adds the *strings* S and T
        as decimal numbers """
    if len(s) == 0:
        return t
    if len(t) == 0:
        return s
    eS = s[-1]
    eT = t[-1]
    if eS == '0' and eT == '1':
        return  add10(s[:-1],t[:-1]) + '1'
    if eS == '1' and eT == '1':
        return  add10(s[:-1],t[:-1]) + '2'
    if eS == '2' and eT == '1':
        return  add10(s[:-1],t[:-1]) + '3'
    if eS == '3' and eT == '1':
        return  add10(s[:-1],t[:-1]) + '4'
    # Lot more rules - how many in all?
```

the "end of S" - hence eS
the "end of T" - hence eT

how about for hw4: **addB**?

carrying on

e.g., S = '23' T = '19'

```
def add10(s,t):
    """ adds the *strings* S and T
        as decimal numbers """
    if len(s) == 0:
        return t
    if len(t) == 0:
        return s
    eS = s[-1]
    eT = t[-1]
    ...
    if eS == '3' and eT == '1':
        return  add10(s[:-1],t[:-1]) + '4'
    ...
    # What if we have to carry?
    if eS == '3' and eT == '9':
```

```
 1
 23
 19
 --
 42
```

how about for hw4: **addB**?

carrying on

e.g., S = '23' T = '19'

```
def add10(s,t):
    """ adds the *strings* S and T
        as decimal numbers """
    if len(s) == 0:
        return t
    if len(t) == 0:
        return s
    eS = s[-1]
    eT = t[-1]
    ...
    if eS == '3' and eT == '1':
       return  add10(s[:-1],t[:-1]) + '4'
    ...
    # What if we have to carry?
    if eS == '3' and eT == '9':
       return add10(? , ?) + '2'
```

```
 1
23
19
--
42
```

What goes here??? HINT… you might need TWO calls to **add10**

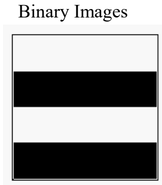
how about for hw4: **addB**?

Now, a word about **hw4pr3**…

REPRESENTING A BINARY IMAGE

Representing a Binary Image

Binary Images



```
11111111
11111111
00000000
00000000
11111111
11111111
00000000
00000000
```

black is 0; white is 1

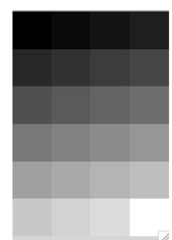
Representing a grey-level image

number of columns   number of rows

```
P2 4 6 255
0 10 20 30 40 50
60 70 80 90 100 110
120 130 140 150 160 170
180 190 200 210 220 255
```

Sample PGM File

Each pixel (picture element) has a value from 0 to 255.



A number of Linux tools can open and edit the PGM images. Try 'inkscape' or 'display.'

Note: Displayed the image by making a 100 pixel by 100 pixel block for each value.

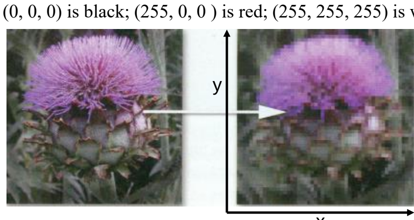
16

Digital Color Image

The continuous picture is broken into a fixed grid of tiny squares (pixel) of same color and intensity. The right image exaggerates the detail of individual pixels.

RGB – each pixel has three values from 0-255 for Red, Green, Blue.

(0, 0, 0) is black; (255, 0, 0) is red; (255, 255, 255) is white



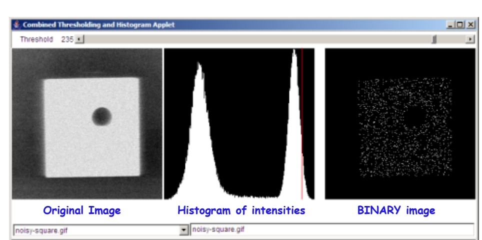
17

Binary Data

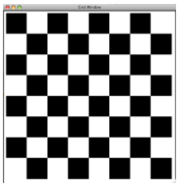
All data is binary…

But some data is ***naturally*** binary!

How about naturally ternary ?



Hw4: binary image compression



Binary Image

```
10101010
01010101
10101010
01010101
10101010
01010101
10101010
01010101
```

Encoding as raw bits

just one big string of 64 characters

```
"1010101001010101101010100101010110101010010101011010101001010101"
```

Hw4: binary image compression

Binary Image

```
00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111
```

Encoding as raw bits

just one big string of 64 characters

If our images tend to have long streaks of unchanging data, how might we represent it ***more efficiently…*** (but still in binary)***?***

```
"0000000000000000111111111111111100000000000000000000000000001111"
```

Hw4: binary image compression

0 is the first digit — There are 16 of them. — 1 is the next digit — Again, there are 16 of them. — 0 is the next digit — There are 28 — 1 is the final digit — There are 4

```
010000110000011100 1100
```

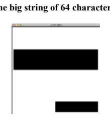
One possible idea

problems??

```
00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111
```

Encoding as raw bits

just one big string of 64 characters



fixed-width image compression

0 is the first digit — There are 16 of them. — 1 is the next digit — Again, there are 16 of them. — and so on… — 28 zeros — 4 ones

```
00010000100100000001110010000100
```

7 bits: # of repeats | 7 bits: # of repeats

8-bit data block | 8-bit data block | 8-bit data block | 8-bit data block

a FIXED-width encoding is needed

8 bits for each "block"

1st bit holds 1 pixel value

7 bits holds the # of those values in a row

```
00000000
00000000
11111111
11111111
00000000
00000000
00000000
00001111
```

original data

original image

- If ***7 bits*** holds the # of consecutive repetitions of data, what is the largest number of `1`s or `0`s that one block can represent?

the pixel → **00010000**

7 bits: # of repeats

8-bit data block

7 bits?

B bits?

hw4 pr3

```
def compress(anImage):
    """ returns the Run-Length-Encoding of the input
        binary image, anImage """
```

What function(s) might help here?

```
def unCompress(compressedImage):
    """ returns the binary image from the
        Run-Length-Encoded, "compressed" input,
        compressedImage """
```

```
from cs5png import *
binaryIm(aImage, ncols, nrows)
```

Can the display image

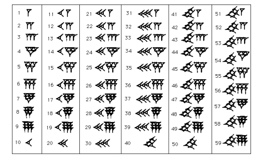
BEYOND BINARY

Off base?

Base 12 – "~~Duodecimal Society~~"
"Dozenal Society"

All this focus on 10 seems so alien!

Base 60 – Ancient Sumeria



Olmec base-20 number representation (East Mexico, 1200 BC-600 AD)

Base 20 – America, precolombus

Echoes of these bases are still bouncing around…

Beyond Binary

base 1 ⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞ digits: 1

base 2 —— 101010 (128 64 32 16 8 4 2 1) digits: 0, 1

base 10 42 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Beyond Binary

base 1 ⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞ digits: 1

base 2 —— 101010 (128 64 32 16 8 4 2 1) digits: 0, 1

base 3 —— (81 27 9 3 1) digits: 0, 1, 2

base 10 42 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Beyond Binary

base 1 ⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞⁞ digits: 1

base 2 —— 101010 (128 64 32 16 8 4 2 1) digits: 0, 1

base 3 —— 1120 (81 27 9 3 1) digits: 0, 1, 2

base 4
base 5
base 6
base 7
base 8
base 9

Which of these *isn't* 42…?

222 132 110 60 52 46

and what are the *bases* of the others?

base 10 42 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

base 11
base 12

Beyond Binary!

base 2 —— 101010 (128 64 32 16 8 4 2 1) digits: 0, 1

base 3 —— 1120 (81 27 9 3 1) digits: 0, 1, 2

base 4 —— 222 (64 16 4 1) digits: 0, 1, 2, 3

base 5 —— 132 (125 25 5 1) digits: 0, 1, 2, 3, 4

base 6 —— 110 (216 36 6 1) digits: 0, 1, 2, 3, 4, 5

base 7 —— 60 (49 7 1) digits: 0, 1, 2, 3, 4, 5, 6

…

base 10 —— 42 (100 10 1) digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

…

base 16 —— 2A (256 16 1) digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Hexadecimal