



COMP2720: Automating Tools for New Media

Moving between media using
text



Text as the “Unimedia”

- We can map anything to text.
- We can map text back to anything.
- This allows us to do all kinds of transformations:
 - Sounds into **Excel**, and back again.
 - Sounds into pictures.
 - Pictures and sounds into lists (formatted text), and back again.



Why care about media transformations?

- Transformed digital media can be more easily transmitted.
 - For example, transfer of binary files over email is usually accomplished by converting them into text.
- We can encode additional information to check for and even correct errors in transmission.
- It may allow us to use the media in new contexts, like storing it in databases.
- Some transformations of media are made easier when the media are in new formats.



Mapping sound to text

- Sound is simply a series of numbers (sample values).
- To convert them to text means to simply create a long series of numbers.
- We can store them to a file to manipulate them elsewhere.

Copying a sound to text

```
def soundToText(sound,filename):  
    file = open(filename,"wt")  
    for s in getSamples(sound):  
        file.write(str(getSample(s))+"\n")  
    file.close()
```

What's this `str()`?

- `str()` is a function that returns the string representation of something.

```
>>> a = str(11)
```

```
>>> a
```

```
'11'
```

```
>>> b = str(12.4)
```

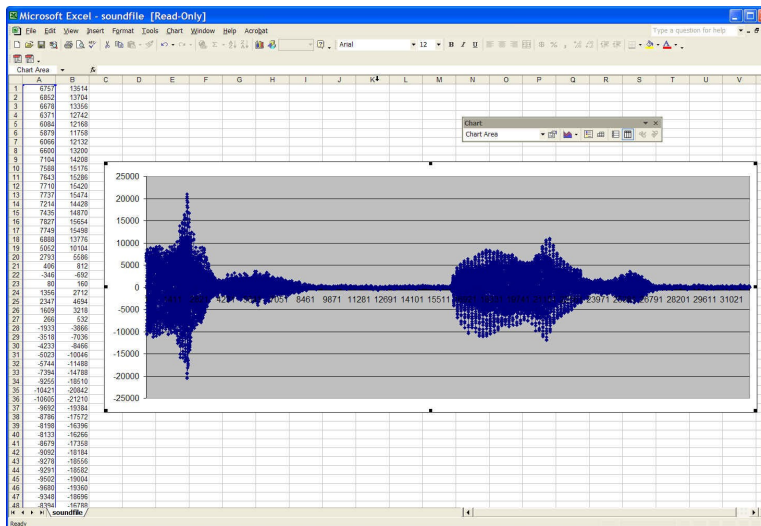
```
>>> b
```

```
'12.4'
```

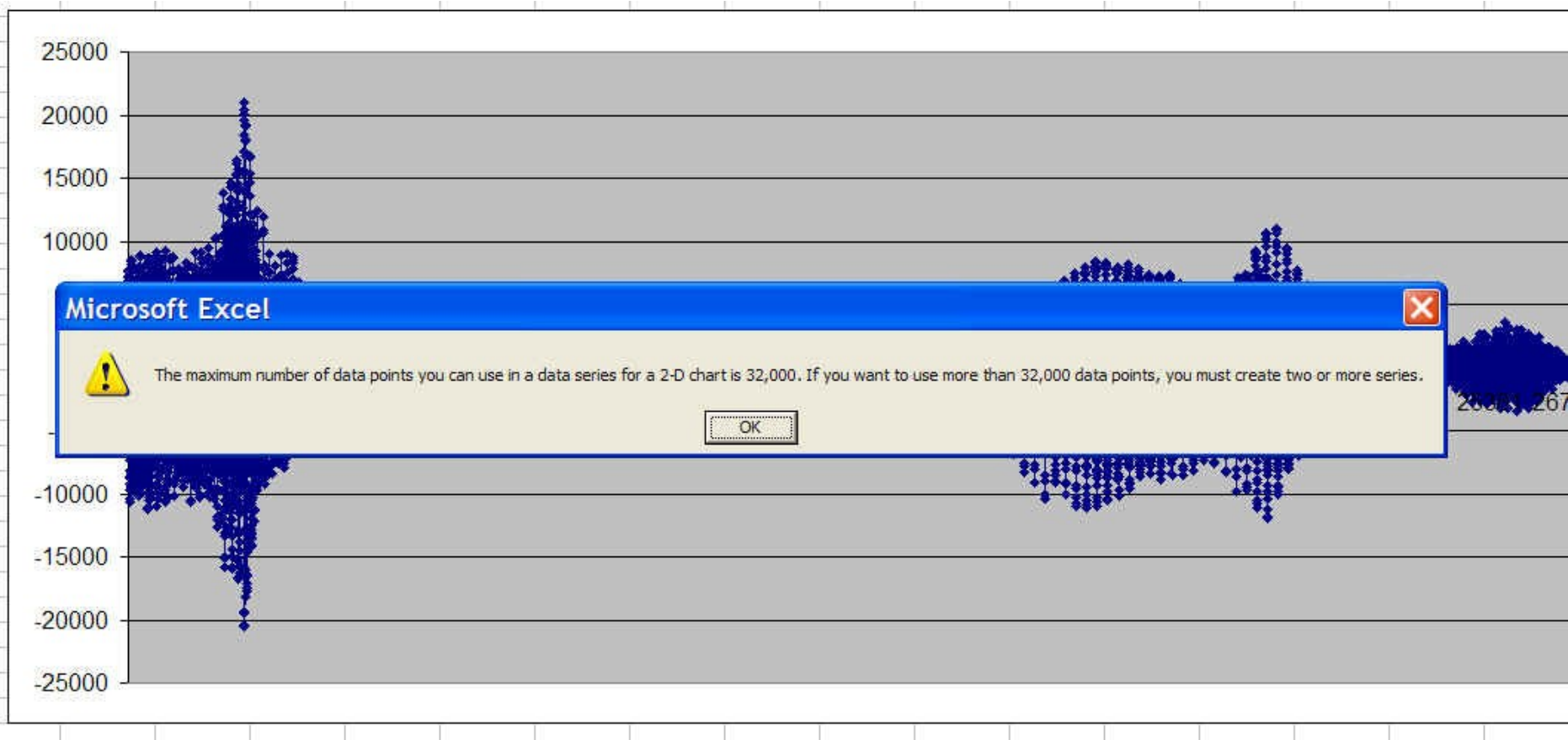
- It's what we use to convert numbers into strings (that we can then add “`\n`” (newline) to).
- You can also convert lists and dictionaries into strings.

We can process the sound in Excel

- We can graph the sound (below).
 - A signal view is simply the graph of the sample values!
- We can add a column and do some modification to the original sound. (Fill down to get them all.)
 - Can increase the volume that way.



Excel has its limitations





Reading text back into a sound

- After we process the sound (as text) in **Excel**, we can save it back to a sound.
 - First, copy the column you want into a new worksheet
 - Then, save the worksheet as a “.txt” file.
 - Get the full pathname of the new “.txt” file to use in JES.



Issues in reading the text back into a sound

- We can't be sure how many numbers are in the file.
- We can't be sure that the numbers will all fit into the sound we've chosen to serve as our target.
- What we want to do is:
 - AS LONG AS we're not out of numbers in the file, and AS LONG AS we still have room in the sound,
 - copy a number out of the file,
 - and put it into a sample in the sound,
 - then go to the next number and the next sample.

Reading the text back as a sound

```
def textToSound(filename):
    # Set up the sound
    sound = makeEmptySound(3)
    soundIndex = 1
    # Set up the file
    file = open(filename,"rt")
    contents=file.readlines()
    file.close()
    fileIndex = 0
    # Keep going until run out sound space or run out of file contents
    while (soundIndex < getLength(sound)) and (fileIndex < len(contents)):
        sample=float(contents[fileIndex]) # Get the file line
        setSampleValueAt(sound,soundIndex,sample)
        fileIndex = fileIndex + 1
        soundIndex = soundIndex + 1
    return sound
```




While loops

- `while` is a new loop for us.
- It takes an expression that is probably *true* to start, but will become *false*.
- A while loop
 - Evaluates the expression.
 - If the expression is *true*, the body of the `while` loop is executed.
 - Then we go back to the beginning and evaluate the expression again.



String to number conversion: float() and int()

- float() (and int()) can take a string as input.
 - As long as the string is a representation of a floating point number (e.g., “34.5”)
- It returns the number that has that string representation.



We could do pictures, but more complicated

- Pictures aren't just a single number for each pixel.
- To recreate a picture in text we need to record, for each pixel:
 - The X and Y positions,
 - the R, G, and B color component values.
- That requires more structured text than simply a long line of numbers.
- Let's do that in just a few minutes.



Mapping from text to anything

- Once we've converted to text (or numbers), we can do anything we want.
- Like, mapping from sound to...pictures!

We simply decide on a representation: How do we map sample values to colors?

```
def soundToPicture(sound):  
    picture = makeEmptyPicture(640,480)  
    soundIndex = 1  
    for p in getPixels(picture):  
        if soundIndex > getLength(sound):  
            break  
        sample = getSampleValueAt(sound,soundIndex)  
        if sample > 1000:  
            setColor(p,red)  
        if sample < -1000:  
            setColor(p,blue)  
        if sample <= 1000 and sample >= -1000:  
            setColor(p,green)  
        soundIndex = soundIndex + 1  
    return picture
```

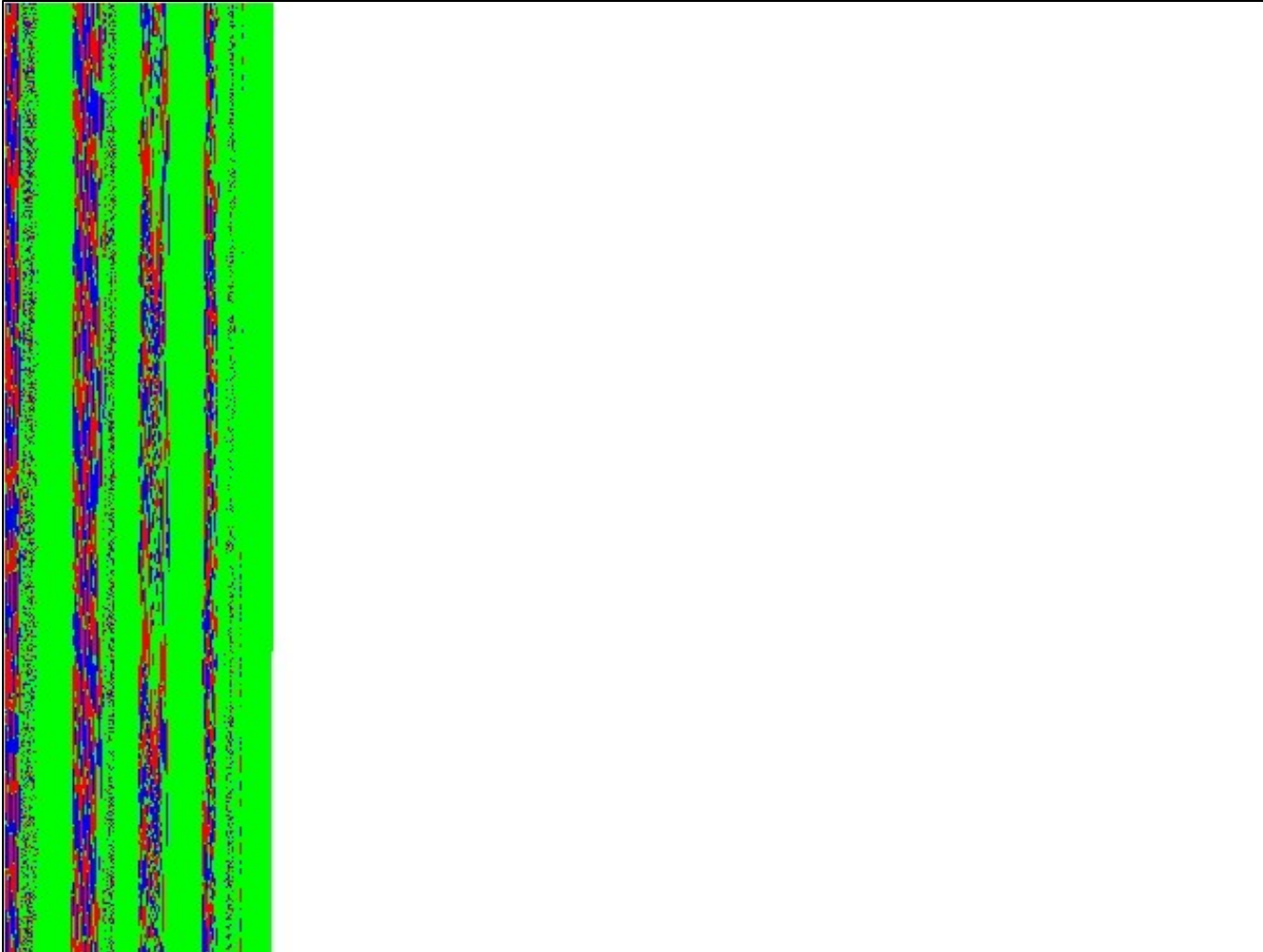
Here's one:

- Greater than 1,000 is red
- Less than -1,000 is blue
- Everything else is green

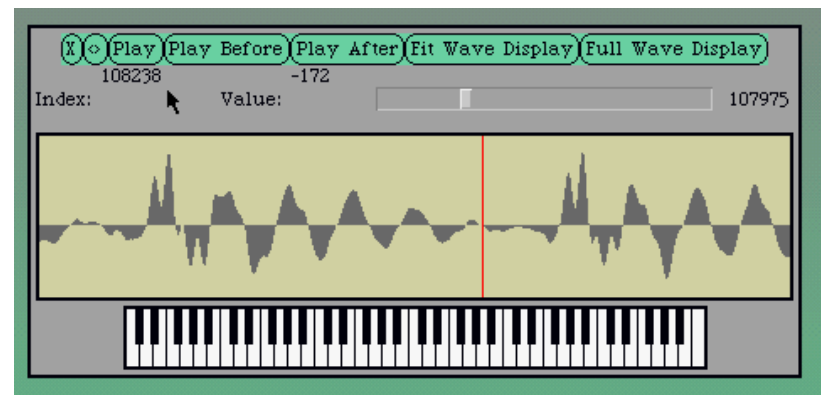
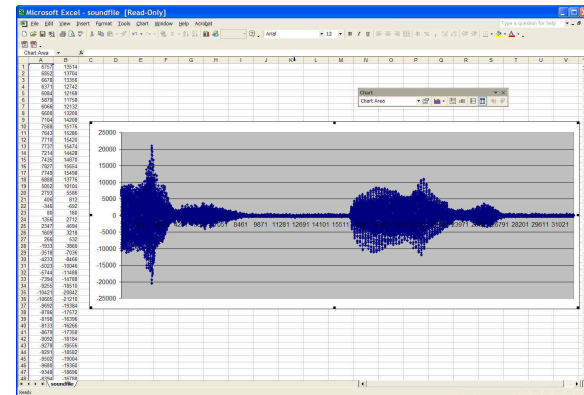
Break

- `break` is yet another new statement.
- It literally means: *“Exit the current loop.”*
- It’s most often used in the block of an `if`
 - “If something extraordinary happens, leave the loop immediately.”
- In our case: *“If we run out of samples before we run out of pixels, STOP!”*

Representing “This is a test”



Any visualisation of sound is merely an encoding



Any visualisation of any kind is merely an encoding

- *A line chart? A pie chart? A scatterplot?*
 - These are just lines and pixels set to correspond to some mapping of the data.
- Sometimes data is lost.
 - Recall the mapping of greyscale.
- Sometimes data is not lost, even if it looks like a dramatic change.
 - Recall creating a *negative* of an image, then taking the negative of a negative to get back to the original.

Lists can do anything!

- Going from sound to lists is easy:

```
def soundToList(sound):
```

```
    list = []
```

```
    for s in getSamples(sound):
```

```
        list = list + [getSample(s)] # Append next sample to end of list
```

```
    return list
```

This really does work

```
>>> list = soundToList(sound)
```

```
>>> print list[0]
```

```
6757
```

```
>>> print list[1]
```

```
6852
```

```
>>> print list[0:100]
```

```
[6757, 6852, 6678, 6371, 6084, 5879, 6066, 6600, 7104, 7588, 7643, 7710, 7737, 7214,  
7435, 7827, 7749, 6888, 5052, 2793, 406, -346, 80, 1356, 2347, 1609, 266, -1933,  
-3518, -4233, -5023, -5744, -7394, -9255, -10421, -10605, -9692, -8786, -8198,  
-8133, -8679, -9092, -9278, -9291, -9502, -9680, -9348, -8394, -6552, -4137,  
-1878, -101, 866, 1540, 2459, 3340, 4343, 4821, 4676, 4211, 3731, 4359, 5653,  
7176, 8411, 8569, 8131, 7167, 6150, 5204, 3951, 2482, 818, -394, -901, -784, -541,  
-764, -1342, -2491, -3569, -4255, -4971, -5892, -7306, -8691, -9534, -9429, -8289,  
-6811, -5386, -4454, -4079, -3841, -3603, -3353, -3296, -3323, -3099, -2360]
```



Can we go from pictures into lists?

- Of course! We just have to decide on a representation.
 - We'll put a list as an element for each pixel.
 - The numbers in the pixel-list will represent
 - The X and Y positions.
 - The Red, Green, and Blue component values.

Pictures to Lists

```
def pictureToList(picture):  
    list = []  
    for p in getPixels(picture):  
        list = list + [[getX(p),getY(p),getRed(p),getGreen(p),getBlue(p)]]  
    return list
```

Why the double brackets?
Because we're putting a sub-list in the list, not just adding a component as we were with sound.

Running pictureToList

```
>>> picture = makePicture(pickAFile())
>>> piclist = pictureToList(picture)
>>> print piclist[0:5]
[[1, 1, 168, 131, 105], [1, 2, 168, 131, 105], [1, 3, 169, 132, 106],
 [1, 4, 169, 132, 106], [1, 5, 170, 133, 107]]
```

Can we go back again? Sure!

```
def listToPicture(list):  
    picture = makeEmptyPicture(640,480)  
    for p in list:  
        if p[0] <= getWidth(picture) and p[1] <= getHeight(picture):  
            setColor(getPixel(picture,p[0],p[1]),makeColor(p[2],p[3],p[4]))  
    return picture
```

We need to make sure that the X and Y fits within our canvas, but other than that, it's pretty simple code.



The numbers could have come from anywhere

- The numbers in the list came from another picture, but we know that they could have come from anywhere!
 - From multiple sounds, one for each of Red, Green, and Blue.
 - From random numbers.




All we're doing is changing encodings

- The basic information isn't changing at all here.
- What's changing is our encoding.
- Different encodings afford us different capabilities.
 - If we go to numbers, we can use Excel.
 - If we go to lists, we can represent structure more easily.

Kurt Gödel

- One of Time magazine's 100 greatest thinkers of the 20th century.
- Proved the “Incompleteness Theorem”.
- By mapping mathematical statements to numbers, he was able to show that there are true statements (numbers) that cannot be proven by any mathematical system.
 - Gödel numbers
- In this way, he showed that no system of logic can prove all true statements.





What to do now

- Read section 10.6 in text book (6 pages)
- Prepare lab 3 (this week)