DeMorgan’s Law, Short circuiting, Images, Tuples
Part 1 of 4

Susan Rodger
October 1, 2020
PFTD

• DeMorgan’s Law
• Short Circuiting
• Images & Tuples
  • Start today, finish next class
Index without error?

```
lst = ["a","b","c","a"]
dex = lst.index("b")
    lst.index("b") is 1
    lst.index("B") ERROR!
    lst.index("B") ??? -1
```

- Use while loop to implement index.
- What is the while loop’s Boolean condition?

```
dex = 0
while BOOL_CONDITION:
    dex += 1
```
Index without error?

```python
lst = ['a', 'b', 'c', 'a']
dex = lst.index('b')
    lst.index('b') is 1
    lst.index('B') ERROR!
    lst.index('B') ?? -1
```

• Use while loop to implement index.
• What is the while loop’s Boolean condition?
  • Whether found value: \( \text{lst}[\text{dex}] == \text{elm} \)
  • Whether reach end of list: \( \text{dex} >= \text{len(lst)} \)
DeMorgan’s Law

• While loop stopping conditions, stop with either:
  • lst[dex] == elm
  • dex >= len(lst)
• While loop needs negation: DeMorgan's Laws
  not (A and B) equivalent to (not A) or (not B)
  not (A or B) equivalent to (not A) and (not B)

while not (lst[dex] == elm or dex >= len(lst)):

while lst[dex] != elm and dex < len(lst):

Why did >= become < ?
Think: DeMorgan’s Law

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>not (A and B)</th>
<th>(not A) or (not B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>not (A or B)</th>
<th>(not A) and (not B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Compsci 101
DeMorgan’s Law, Short circuiting, Images, Tuples
Part 2 of 4

Susan Rodger
October 1, 2020
Think: DeMorgan’s Law

**Fill in the blanks**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>not (A and B)</th>
<th>(not A) or (not B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>not (A or B)</th>
<th>(not A) and (not B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
To solve these problems, and to help figure out exactly how morality functions and can (hopefully) be programmed into an AI, the team is combining the methods from computer science, philosophy, and psychology. “That’s, in a nutshell, what our project is about,” Conitzer asserts.

Vince Conitzer

THE EVOLUTION OF AI: CAN MORALITY BE PROGRAMMED?

- Duke Professor of Philosophy, Economics, Computer Science
- Computational Social Choice
- Computers and Thought Award
Compsci 101
DeMorgan’s Law, Short circuiting, Images, Tuples
Part 3 of 4

Susan Rodger
October 1, 2020
Images

What is Photoshop?
Image Processing

- Convert image into format for manipulating the image
  - Visualization, Sharpening, Restoration, Recognition, Measurement, more
- Resizing, Red-eye Removal, more
- CrashCourse: Navigating Digital Info
Image Library

• PIL: Python Image Library -> Pillow
  • To install run the command below in a terminal
    • Terminal in PyCharm, not “Python Console”
    • `pip install Pillow`
      – If that doesn’t work try:
        – `Python3 -m pip install Pillow`

• Library has extensive API, far more than we need
  • Concepts often apply to every image library
  • Realized in Python-specific code/functions
SimpleDisplay.py

- Access to PIL and Image module
  - What type is `img`?

```python
from PIL import Image

if __name__ == '__main__':
    img = Image.open("images/bluedevil.png")
    img.show()
    print("width %d, height %d" % (img.width, img.height))
```
DeMorgan’s Law, Short circuiting, Images, Tuples
Part 4 of 4

Susan Rodger
October 1, 2020
What is a class in Python?

• Class ≈ module ≈ library (for this CS101)

• Class – Also blueprint/Factory for creating objects
  • We've used int, float, str
  • `<class 'int'>`, `<class 'list'>`
  • Everything is a class in Python3

• Use .dot notation to access object's innards
  • "Hello".lower() is a function aka method
  • `img.width` is an attribute aka field/property
Image Library Basics

• Library can create/open images in different formats, e.g., .png, .jpg, .gif, ...

• Images have properties: width, height, type, color-model, and more
  • Functions and fields access these properties, e.g., \texttt{im.width}, \texttt{im.format}, and more

• Pixels are formed as triples (255,255,255), (r,g,b)
  • In Python these are tuples: immutable sequence
Color Models

• Cameras, Displays, Phones, JumboTron: RGB
  • Additive Color Model: Red, Green, Blue

• Contrast Printers and Print which use CMYK
  • Subtractive: Cyan, Magenta, Yellow, Key/Black
Example: Convert Color to Gray

Process each pixel
Convert to gray
First View of Image for Grayscale

• Image is a collection of pixels
  • Organized in rows: # rows is image height
  • Each row has the same length: image width

• Pixels addressed by (x, y) coordinates
  • Upper-left (0,0), Lower-right (width-1,height-1)
  • Typically is a single (x, y) entity: tuple

• Tuple is immutable, indexed sequence (a, b, c)
Tuple: What and Why?

• Similar to a list in indexing starting at 0
  • Can store any type of element
  • Can iterate over
• Immutable - Cannot mutate/change its value(s)
  • Efficient because it can't be altered
• Consider $x = (5, 6)$ and $y = ([1, 2], 3.14)$
  • What is $x[0] = 7? y[0].append(5)?$