Practice: *edit distance*

Spell Checking challenge: what words are *close* to the non—word you typed?

```python
def ED(s1, s2):
    # s1 == s2 => return 0
    # s1 empty => return len(s2)
    # s2 empty => return len(s1)
    # s1[i] == s2[j] => 0 + ED(s1[i:], s2[j:])
    else: 1 + (recursive cost)
```
Practice: edit distance

Base cases:
- \( s_1 = \) “” -> return `len(s2)` # that many insertions
- \( s_2 = \) “” -> return `len(s1)` # that many deletions
- \( s_1 = s_2 \): return 0

Recursive cases:
- \( s_1[0] = s_2[0] \): recurse with \( s_1[1:] \) and \( s_2[1:] \)
- Otherwise: Find the cost if the next step is (insertion, deletion, substitution), then return the best option
Practice: *edit distance*

Base cases:
- $s_1 == "" \rightarrow \text{return } \text{len}(s_2) \# \text{that many insertions}$
- $s_2 == "" \rightarrow \text{return } \text{len}(s_1) \# \text{that many deletions}$
- $s_1 == s_2: \text{return } 0$

Recursive cases:
- $s_1[0] == s_2[0]: \text{recurse with } s_1[1:] \text{ and } s_2[1:]$
- Otherwise: Find the cost if the next step is (insertion, deletion, substitution), then

  \text{return the best option}
**Practice:** edit distance

```python
def ED(s1, s2):
    if len(s1) == 0: return len(s2)
    elif len(s2) == 0: return len(s1)
    elif s1[0] == s2[0]: return ED(s1[1:], s2[1:])
    else:
        insertionCost = 1 + ED(s1, s2[1:])
        deletionCost = 1 + ED(s1[1:], s2)
        substitutionCost = 1 + ED(s1[1:], s2[1:])
        bestCost = min(insertionCost, deletionCost, substitutionCost)
    return bestCost
```

```python
a='abcdefgijklmn'
b= 'abcdefgijklmn'
```
A problem...

ED("spam", "pims")

ED("spam", "ims")

ED("pam", "ims")

ED("pam", "pims")

ED("pam", "ims")
A solution: **memoization**

def fastED(s1, s2, memo={}):
    if (s1, s2) in memo: return memo[(s1, s2)]
    elif len(s1) == 0: return len(s2)
    elif len(s2) == 0: return len(s1)
    elif s1[0] == s2[0]: return fastED(s1[1:], s2[1:])
    else:
        insertionCost = 1 + fastED(s1, s2[1:], memo)
        deletionCost = 1 + fastED(s1[1:], s2, memo)
        substitutionCost = 1 + fastED(s1[1:], s2[1:], memo)
        bestCost = min(insertionCost, deletionCost, substitutionCost)
        memo[(s1, s2)] = bestCost
    return bestCost

dictionary → maps key to value
list → unordered/ordered/ordered/index w/nums
tuple → ordered/index w/nums that can’t grow or change
The Name Challenge Continues!

Stand up – and introduce yourself to EIGHT people to your left!

Around the room: “This is S, T, U, V, W, X, Y and Z and I am A.”
What's new today?

42\text{_{10}} \quad 101010\text{_{2}}
Base-2 Storage & Representation

8 bits = 1 byte = 1 box

The SAME bits can represent different pieces of data, depending on type

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 0000</td>
<td>32</td>
<td>20</td>
<td>(blank)</td>
</tr>
<tr>
<td>0010 0001</td>
<td>33</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>0010 0010</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>0010 0011</td>
<td>35</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>0010 0100</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>0010 0101</td>
<td>37</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>0010 0110</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>0010 0111</td>
<td>39</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>0010 1000</td>
<td>40</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>0010 1001</td>
<td>41</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>0010 1010</td>
<td>42</td>
<td>2A</td>
<td>*</td>
</tr>
<tr>
<td>0010 1011</td>
<td>43</td>
<td>2B</td>
<td>+</td>
</tr>
</tbody>
</table>

chr ord

The same bits are in each container.

But why these bits?
What is 42?

Value!

Syntax.
forty-two

42

tens

ones

value

syntax
forty-two

42

tens

ones

value
<table>
<thead>
<tr>
<th>thirty-twos</th>
<th>sixteens</th>
<th>eights</th>
<th>fours</th>
<th>twos</th>
<th>ones</th>
</tr>
</thead>
</table>

But, a different syntax
forty-two
value

101010
syntax

---

thirty-twos  sixteens  eights  fours  twos  ones

with a binary syntax
forty-two

value

101010

syntax

101010

syntax

thirty-twos

sixteens
eights

fours
twos
ones

with a binary syntax
Base 2
"binary"

101010₂

Base 10
"decimal"

42₁₀

Syntax
the symbols used
(what things look like)

Value
stuff we care about
(what things mean)
Base 2
"binary"

\[ 101010_2 \]

Base 10
"decimal"

\[ 42_{10} \]

Different

Syntax
the symbols used
(what things look like)

Same!

forty-two

Value
stuff we care about
(what things mean)
Base 2
"binary"

Base 10
"decimal"

101010₂

42₁₀
4 tens + 2 ones

writing 123 in binary...
### Binary math

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Decimal math

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Multiplication

- **Binary**
  - 0 * 0 = 0
  - 0 * 1 = 0
  - 1 * 0 = 0
  - 1 * 1 = 1

- **Decimal**
  - 0 * 0 = 0
  - 0 * 1 = 0
  - 1 * 0 = 0
  - 1 * 1 = 1

### Tables of

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>

[www.youtube.com/watch?v=Nh7xapV8-Wk](www.youtube.com/watch?v=Nh7xapV8-Wk)
Quiz

In binary, I’m an 11-eyed alien!

Convert these two binary numbers to decimal:

- 110011
- 10001000

Convert these two decimal numbers to binary:

- 28
- 101

Add these two binary numbers:

- 101101
- 1110

Multiply these binary numbers:

- 101101
- 1110

Extra! Can you figure out the last binary digit (bit) of 53 without determining any other bits? The last two? 3?

Name(s) ____________________
There are 10 kinds of "people" in the universe:

- those who know ternary,
- those who don't, and
- those who think this is a binary joke!
Which one of these isn't 42...?

222  60  54  46  39

and what are the bases of the rest?

42  digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Which one of these *isn't* 42...?

and what are the bases of the rest?

42

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

Everything's 42 if fractional bases are allowed!
<table>
<thead>
<tr>
<th>Base</th>
<th>Number</th>
<th>Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>101010</td>
<td>0, 1</td>
</tr>
<tr>
<td>3</td>
<td>1120</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>4</td>
<td>222</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0, 1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>7</td>
<td>60</td>
<td>0, 1, 2, 3, 4, 5, 6</td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8</td>
</tr>
<tr>
<td>10</td>
<td>42</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>11</td>
<td>39</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A</td>
</tr>
<tr>
<td>16</td>
<td>2A</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F</td>
</tr>
</tbody>
</table>

All 42s!
But *why* base-2?
Ten symbols is too many!

A computer has to differentiate *physically* among all its possibilities.

[0 1 2 3 4 5 6 7 8 9]

ten symbols ~ ten different voltages

*This is too difficult to replicate billions of times*

What digits are these?

Ouch!
Ten symbols is too many!

A computer has to differentiate *physically* among all its possibilities.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

8 4 7 3

What digits are these?

Ouch!
**Two** symbols is easiest!

A computer has to differentiate *physically* among all its possibilities.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ten symbols ~ ten different voltages

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

two symbols ~ two different voltages

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What digits are these?

Easy!
Two symbols is easiest!

A computer has to differentiate physically among all its possibilities.

ten symbols ~ ten different voltages

two symbols ~ two different voltages

What digits are these?
Ternary computers?

50 of these *Setun* ternary machines were made at Moscow U. ~ **1958**

This project was discontinued in 1970... *though not because of the ternary design!*