Three-eyed troubles: GradeScope, Python, & VSCode...

Aliens Attack! Picobot programmer Z. Dodds was subject of a bizarre encounter yesterday with three-eyed aliens. The trinocular tourists, it seems, were conducting experiments that "would help them understand "human thought."

It seems the aliens used a shrinking ray which let them enter the programmer’s head in order to see what was happening. A witness reports deeply disappointed voices emanating from within.

To escape the attack, Dodds had to turn the ray on himself - as he shrank, the aliens quickly flew off, departing so fast that he was unable to use the reverse ray before they left. "No worries," Dodds mused - in fact, this might help me tomorrow...

Automatic translation: if it’s possible for human languages - perhaps for CS, too?

Inside the machine...

What’s behind the scenes (processing + memory):

learning a language ~ syntax
unavoidable, but not the point

... but learning CS ~ semantics
learning how machines think!
Memory!

Random Access Memory

512 MB of memory

bit = smallest amt. of info: 0 or 1

byte = 8 bits

word = 64 bits

a big list of boxes, each with a name, type, location, and value

All languages use **datatypes**

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>What is it?</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>3.14 or 3.0</td>
<td>numeric values with a fractional part, even if the fractional part is .0</td>
</tr>
<tr>
<td>int</td>
<td>42 or 10**100</td>
<td>integers – Python has infinite precision ints!</td>
</tr>
<tr>
<td>bool</td>
<td>True or False</td>
<td>the T/F results from a test or comparison: ==, !=, &lt;, &gt;, &lt;=</td>
</tr>
</tbody>
</table>

"Boolean values"

"Boolean operators"

Operate!

Python operators

It's not worth remembering all these +/-/* things! I'd recommend parentheses over precedence.
% the **mod** operator

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>7</td>
</tr>
</tbody>
</table>

The **mod** operator gives the remainder when `x` is divided by `y`.

For what values of `x` are these `True`?

- `x%2 == 0`
- `x%2 == 1`
- `x%4 == 0`
- `x%4 == 3`

If `x` is a year, what happens on these years?

What happens on these years, football-wise?!

---

// **integer division**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>// 3</td>
</tr>
<tr>
<td>8</td>
<td>// 3</td>
</tr>
<tr>
<td>9</td>
<td>// 3</td>
</tr>
<tr>
<td>30</td>
<td>// 7</td>
</tr>
</tbody>
</table>

`x//y` is `x/y`, rounded-down to an integer.

---

the "equals" operators

`==` `!=` `==` `==`

This is true – but what is it saying!??
Inside the machine...

What’s happening in python:

\[
\begin{align*}
  x &= 41 \\
  y &= x + 1 \\
  z &= x + y \\
  x &= x + y
\end{align*}
\]

What’s happening behind the scenes (in memory):

Computation

Memory (Data Storage)

Are numbers enough for everything?

Yes and no...

You need lists of numbers, as well!

and strings - lists of characters - too.

Both of these are Python sequences...
strings: *textual data*

- **s** = 'scripps'
- **c** = 'college'

**type**

- `type(s)`

**len**

- `len(s)`

**add!**

- `s + c`

**multiply!!**

- `2*s + 3*c`

---

**strings: textual data**

Given \[ \begin{cases} s1 = 'ha' \\ s2 = 't' \end{cases} \]

What are \[ s1 + s2 \]

\[ 2*s1 + s2 + 2*(s1+s2) \]

---

Data, data everywhere...

---

Data, data everywhere...
Data, data everywhere...

1 Yottabyte
1 Zettabyte
1 Exabyte
1 Petabyte

Data produced each year


1 Petabyte, PB == 1000 Terabytes, TB
1 Terabyte, TB == 1000 Gigabytes, GB

References

- Google's users
- Google's users
- Google's users

Big Data?

Big data: The next frontier for innovation, competition, and productivity

The New York Times
Sunday Review | The Opinion Pages

NEWS ANALYSIS
Is Big Data an Economic Big Dud?

Lists ~ collections of any data

M = [ 4, 7, 100, 42, 5, 47 ]

Data's elevation?

G. Garcia Marquez
**Indexing uses [ ]**

\[ s = 'harvey mudd college' \]

<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>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>

**Strings**

| s[0] | is | 'h' |
| s[17] | is |
| s[6] | is | 'e' |

**Negative indices...**

\[ s = 'harvey mudd college' \]

<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>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>-19</td>
<td>-17</td>
<td>-15</td>
<td>-13</td>
<td>-11</td>
<td>-9</td>
<td>-7</td>
<td>-5</td>
<td>-3</td>
<td>-1</td>
<td>-18</td>
<td>-16</td>
<td>-14</td>
<td>-12</td>
<td>-10</td>
<td>-8</td>
<td>-6</td>
<td>-4</td>
<td>-2</td>
</tr>
</tbody>
</table>

Negative indices count **backwards** from the end!

| s[-1] | is | 'e' |
| s[-18] | is |
| s[-7] | is |
| s[-0] | is |

**Slicing**

\[ s = 'harvey mudd college' \]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |

\[ s[ : ] \text{ slices the string, returning a substring} \]

| s[0:6] | is | 'harvey' |
| s[12:18] | is | 'colleg' |
| s[17:] | is | 'ge' |
| s[: ] | is | 'harvey mudd college' |
**Skip-Slicing**

\[ s[\text{start} : \text{end} : \text{stride}] \]

the third index is the stride length  default is +1

\[ s = \text{'harvey mudd college'} \]

\[ s[2:11:2] \text{ is 're ud'} \]

\[ s[17:12] \text{ is} \]

\[ s[17:12:-1] \text{ is 'doe'} \]

\[ s[:::-1] \text{ is} \]

\[ s[1::6] \text{ is} \]

---

**Python slices - it dices...**

(data, at least)

... *but wait*, there's more!

---

**Try it!**

\[ \pi = [3,1,4,1,5,9] \]

\[ L = ['\pi', 'isn't', [4,2]] \]

\[ M = 'You need parentheses for chemistry!' \]

---

**Part 1**

<table>
<thead>
<tr>
<th>What is</th>
<th>( \text{len}(\pi) )</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is</td>
<td>( \text{len}(L) )</td>
<td></td>
</tr>
<tr>
<td>What is</td>
<td>( \text{len}(L[1]) )</td>
<td></td>
</tr>
<tr>
<td>What is</td>
<td>( \pi[2:4] )</td>
<td></td>
</tr>
<tr>
<td>What slice of ( \pi ) is ( [3,1,4] )</td>
<td>( \pi[0:3] )</td>
<td></td>
</tr>
<tr>
<td>What slice of ( \pi ) is ( [3,4,5] )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Part 2**

<table>
<thead>
<tr>
<th>What is</th>
<th>( L[0] )</th>
<th>'\pi'</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is</td>
<td>( L[0][1] )</td>
<td></td>
</tr>
<tr>
<td>What is</td>
<td>( L[0:1] )</td>
<td></td>
</tr>
<tr>
<td>What slice of ( M ) is 'try'?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What slice of ( M ) is 'shoe'?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What is</td>
<td>( M[9:15] )</td>
<td></td>
</tr>
<tr>
<td>What is</td>
<td>( M[::5] )</td>
<td></td>
</tr>
</tbody>
</table>

---

Extra! Mind Muddler

What are \( \pi[0] \times (\pi[1]+\pi[2]) \) and \( \pi[0] \times (\pi[1:2]+\pi[2:3]) \)?

These two are different!

---

**Functioning in Python**

```python
# my own function!
def dbl( x ):
    """ returns double its input, x ""
    return 2 * x
```

This doesn't *look* quite right...
**Functioning in Python**

```python
# my own function!
def dbl(x):
    """ returns double its input, x ""
    return 2*x
```

Some of Python's baggage…

---

**Function Fun!**

```python
def undo(s):
    """ this "undoes" its input, s ""
    return 'de' + s

>>> undo('caf')
'decaf'

>>> undo(undo('caf'))
strings, lists, numbers ... all data are fair game
```

---

Still broken...!
Run these lines:

\[
x = 41 \\
y = x + 1 \\
z = x + y
\]

What are \(x\), \(y\), and \(z\) at this time?

Then run this line:

\[
x = x + y
\]

What are \(x\), \(y\), and \(z\) at this time?

Extra!

\[
a = 11 // 2 \\
b = a \% 3 \\
c = b ** a + b * a
\]

What are the values of \(a\), \(b\), and \(c\) after the 3 lines, at left, run?