Three-eyed aliens causing most computer troubles...

Alien Attack?! Picobot programmer G. Koenning was subjected to a bizarre encounter with three-eyed aliens yesterday. The trimocular tourists, it seems, were conducting experiments that would help them understand "how humans think."

The aliens apparently used a shrinking ray that let them enter the programmer’s head to see what was happening. A witness reports deeply disappointed voices emanating from within.

To escape the attack, Koenning 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. "I don't mind," Koenning mused. In fact, this might help me hide from students tomorrow,"

Homework #2
0) Reading + response
1) Lab: data
2) Lab: functions
3) The fun in functions!

Read sections 2.3-2.5

learning a language ~ syntax
unavoidable, but not the point

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

Memory!

Random Access Memory

41
name: x
type: int
LOC: 312

42
name: y
type: int
LOC: 324

83
name: z
type: int
LOC: 336

83
name: int
LOC: 348

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

512 MB of memory

on or off →

bit = smallest amt. of info.: 0 or 1

byte = 8 bits

word = 64 bits

All languages use data types

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>What is it?</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>3.14 ≈ 3.0</td>
<td>numeric values with a fractional part, even if the fractional part is 0.0</td>
</tr>
<tr>
<td>int</td>
<td>42 ≈ 10**100</td>
<td>integers – Python has infinite-precision ints!</td>
</tr>
<tr>
<td>bool</td>
<td>True = False</td>
<td>T/F results from a test or comparison: &lt;=, !&lt;, &lt;, &lt;&gt;, &gt;=</td>
</tr>
</tbody>
</table>

Hey! Someone can't spell "Boolean values!"

"Boolean operators"

George Rehbe

type(x)
**Python operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>parens</td>
</tr>
<tr>
<td>**</td>
<td>power</td>
</tr>
<tr>
<td>-</td>
<td>negate</td>
</tr>
<tr>
<td>* / % //</td>
<td>times, mod, divide</td>
</tr>
<tr>
<td>+ -</td>
<td>add, subtract</td>
</tr>
<tr>
<td>&gt; == &lt;</td>
<td>compare</td>
</tr>
<tr>
<td>=</td>
<td>assign</td>
</tr>
</tbody>
</table>

It's hard to remember all these **/\* things! At first, prefer parens/parentheses over precedence.

---

**the mod operator**

<table>
<thead>
<tr>
<th>x % y</th>
<th>is the remainder when x is divided by y</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 % 3</td>
<td>9 % 3</td>
</tr>
<tr>
<td>8 % 3</td>
<td>30 % 7</td>
</tr>
</tbody>
</table>

For what values of x are these True?

- x % 2 == 0
- x % 2 == 1
- x % 4 == 0 (If x is a year, what happens on these years?)
- x % 4 == 3 (What happens on these years, football-wise?)

---

**integer division**

<table>
<thead>
<tr>
<th>x // y</th>
<th>is x/y, rounded down to an integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 // 3</td>
<td></td>
</tr>
<tr>
<td>8 // 3</td>
<td></td>
</tr>
<tr>
<td>9 // 3</td>
<td></td>
</tr>
<tr>
<td>30 // 7</td>
<td></td>
</tr>
</tbody>
</table>

Decomposition of 31 into 7's:

31 == (4) * 7 + (3)

Decomposition of x into y's:

x == (x // y) * y + (x % y)

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**  \(\rightarrow\)  **Memory (Data Storage)**

**Strings: Textual Data**

**Strings**

\[
s = 'scripps' \\
c = 'college'
\]

**Type...**

\[
type(s)
\]

**Len**

\[
len(s)
\]

**Add!**

\[
s + c
\]

**Multiply!!**

\[
2*s + 3*c
\]

**Lists ~ Ordered Collections of Any Data**

\[
L = [3.14, [2, 40], 'third', 42]
\]

- **Len(L)**
- **L[0]**
- **L[0:1]**

**Indexing Uses [ ]**

\[
s = 'harvey mudd college'
\]

**Index**

\[
\begin{align*}
s[0] &\text{ is } 'h' \\
s[17] &\text{ is }
\end{align*}
\]

**Slicing**

\[
\begin{align*}
s[6] &\text{ is } e' \\
s[ ] &\text{ is } 'e'
\end{align*}
\]

- **Top-level length**
  - Only counts top-level elements
- **Indexing**
  - Could return a different type
- **Slicing**
  - Always returns the same type, and always returns a substructure!
**Slicing**

s = 'harvey mudd college'

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

s[start:end] slices the string, returning a substring.

- The first index is the first character.
- The second index is ONE AFTER the last character.

**Negative indices...**

s = 'harvey mudd college'

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

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

\[s[-1] \text{ is } 'e'\]

\[s[-18] \text{ is }\]

\[s[-7] \text{ is }\]

\[s[0] \text{ is } 'h'\]

\[s[0:6] \text{ is } 'harvey'\]

\[s[12:18] \text{ is } 'colleg'\]

\[s[17:] \text{ is } 'ge'\]

\[s[:] \text{ is } 'harvey mudd college'\]

**Skip Slicing**

s = 'harvey mudd college'

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

s[start:end:step] the third index is the stride length. default is +1

**Slicing**

s = 'harvey mudd college'

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

\[s[15:-1] \text{ is }\]

\[s[:2] \text{ is } 'mud'\]

\[s[0:8:2] \text{ is } 'hre'\]

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

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

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

\[ s[\text{start} : \text{end}+1 :] \]

the third index is the stride length
default is +1

\[ s = 'harvey mudd college' \]

\[ s[0:8:2] \text{ is 'hre' } \]
\[ s[17:12:-1] \text{ is } \]
\[ s[:::-1] \text{ is 'doe'} \]
\[ s[1::6] \text{ is } \]
Functioning in Python

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

Some of Python's baggage...

Function Fun!

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

```python
>>> undo('caf')
'decaf'
```

```python
>>> undo(undo('caf'))
```

strings, lists, numbers ...
all data are fair game
how works

"Quiz"

Run these lines:

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

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

Then run this line:

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

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

Extra!

\[
\begin{align*}
a &= 11 \ // 2 \\
b &= a \ % \ 3 \\
c &= b^{**} (a+b) * a
\end{align*}
\]

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