Three-eyed aliens causing most computer troubles...

Alien 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 “how humans think.”

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 shrunk, the aliens quickly flew off, departing so fast that he was unable to use the reverse ray before they left. “I don’t mind,” Dodds mused – in fact, this might help me tomorrow…

see three-eyed alien attack, p. 42
Alien Attack?! Picobot programmer Z. Dodds was subject of a bizarre encounter yesterday. The three-eyed tourists, it seems, were conducting experiments that would help them understand “how humans think.”

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. “I don’t mind,” Dodds mused – in fact, this might help me tomorrow…

see three-eyed alien attack, p. 42
Three-eyed aliens causing most computer troubles...

Alien 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 "how humans think." 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. "I don't mind," Dodds mused – in fact, this might help me tomorrow… "

3) Putting the fun in functions!

1) Lab: data
2) Lab: functions
3) Putting the fun in functions!

[[ I'm out of town on Mon. 5/29 ]]

Memorial Day – Monday, May 29

class will be OFFICE HOURS/HW help for HW #1 (data + functions)

We'll have at least two (maybe three) tutors here to help with HW #1.

Come for any or all of it!
during the semester?

Friday afternoon "grutor" hours!
Never know what's going to happen!?
Picobot tutoring gets real!

Picobot results?!
The *challenge* of programming...

**syntax**
How it looks

**semantics**
What it does

**intent**
What it should do

human-typed input → machine-produced output → human-desired output

Python, Picobot, or any language at all
learning a language $\sim$ syntax
unavoidable, but not the point

... but learning CS $\sim$ semantics
learning how machines think!
Inside the machine...

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

Computation

Data Storage

variables ~ boxes

id, del
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: (empty), type: int, LOC: 348

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

512 MB of memory

- **bit**: smallest amt. of info.: 0 or 1 (False, True)
- **byte**: 8 bits
- **word**: 64 bits
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, <em>even if the fractional part is .0</em></td>
</tr>
<tr>
<td>int</td>
<td>42 or 10**100</td>
<td>integers – Python has <em>infinite precision ints!</em></td>
</tr>
<tr>
<td>bool</td>
<td>True or False</td>
<td>the T/F results from a test or comparison: <code>==, !=, &lt;, &gt;, &lt;=, &gt;=</code></td>
</tr>
</tbody>
</table>

Hey! Someone can’t spelle!

"Boolean values"

"Boolean oeptrators"

George Boole

```
type(x)
```
Operate!

higher precedence

```
( )
**
-
*
/
%
/
+
-
>
==
<
=
```
O-per-ate!

(higher precedence)

\[
\begin{array}{c}
( \quad ) \\
** \\
- \\
* / \% // \\
+ - \\
> == < \\
= \\
\end{array}
\]
Python operators

parens ( )

power **

negate -

times, mod, divide * / % //

add, subtract + -

compare > == <

assign =

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

- 7 % 3
- 8 % 3
- 9 % 3
- 30 % 7

\( x \% y \) is the remainder when \( x \) is divided by \( y \)

For what values of \( x \) are these True?

\[
\begin{align*}
x \% 2 &= 0 \\
x \% 2 &= 1 \\
x \% 4 &= 0 \\
x \% 4 &= 3 \\
\end{align*}
\]

If \( x \) is a year, what happens on these years!? What happens on these years, football-wise!?
### Integer Division

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 // 3</td>
<td>2</td>
</tr>
<tr>
<td>8 // 3</td>
<td>2</td>
</tr>
<tr>
<td>9 // 3</td>
<td>3</td>
</tr>
<tr>
<td>30 // 7</td>
<td>4</td>
</tr>
</tbody>
</table>

The expression $\lfloor x / y \rfloor$ is $x / y$, rounded-down to an integer.

**Decomposition of 31 into 7's:**

Why?

$$31 = (4) \times 7 + (3)$$

**Decomposition of x into y's:**

$$x = \lfloor x / y \rfloor \times y + (x \% y)$$

- $\lfloor x / y \rfloor$: number of full $y$'s in $x$
- $(x \% y)$: remainder after "taking" all of the full $y$'s in $x$
integer division

\[
\begin{array}{c|c}
7 \text{ \(\text{\textbackslash \\} \text{ }\)} \text{ } 3 & 2 \\
8 \text{ \(\text{\textbackslash \\} \text{ }\)} \text{ } 3 & 2 \\
9 \text{ \(\text{\textbackslash \\} \text{ }\)} \text{ } 3 & 3 \\
30 \text{ \(\text{\textbackslash \\} \text{ }\)} \text{ } 7 & 4 \\
\end{array}
\]

\[
\text{x} \text{\textbackslash \\} \text{ } \text{y} \text{ is } \frac{x}{y}, \text{ rounded-down to an integer}
\]

Decomposition of 31 into 7's:

\[
31 \text{ \(\text{\textbackslash =} \text{ }\)} (4) \times 7 + (3)
\]

Why?

Decomposition of \(x\) into \(y\)'s:

\[
x \text{ \(\text{\textbackslash =} \text{ }\)} (x \text{ \(\text{\textbackslash \\} \text{ }\)} \text{ } y) \times y + (x \text{ \(\text{\textbackslash \%} \text{ }\)} y)
\]

\# of full \(y\)'s in \(x\)

remainder after "taking" all of the full \(y\)'s in \(x\)
the "equals" operators

This is true – *but what is it saying!*?
I want

the "equals" operators

SET equals isn't equal to TEST equals

I want ==== !
the "equals" operators

SET equals isn't equal to TEST equals

I want ===!
how = works

Try it!

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:

\[
x = x + y
\]

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?
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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
<td>312</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
<td>324</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
<td>312</td>
</tr>
<tr>
<td>x</td>
<td>int</td>
<td>324</td>
</tr>
</tbody>
</table>

id, del
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 \[
\begin{align*}
    s1 &= 'ha' \\
    s2 &= 't'
\end{align*}
\]

What are \(s1 + s2\)

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

What did you say?!?
Data, data everywhere...
Data, data everywhere...

1 Zettabyte

1 Exabyte

1 Petabyte

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

Data produced each year

Logarithmic scale

100-years of HD video + audio

References


(life in video) 60 PB: in 4320p resolution, extrapolated from 16MB for 1:21 of 640x480 video
(w/sound) – almost certainly a gross overestimate, as sleep can be compressed significantly!

Big Data?

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

Is Big Data an Economic Big Dud?
Quantifying Trading Behavior in Financial Markets Using Google Trends

Tobias Preis, Helen Susannah Moat & H. Eugene Stanley

Profit and loss for an investment strategy based on the volume of the search term debt, the best performing keyword in our analysis, with $\Delta t = 3$ weeks, plotted as a function of time (blue line). This is compared to the “buy and hold” strategy (red line) and the standard deviation of 10,000 simulations using a purely random investment strategy (dashed lines). The Google Trends strategy using the search volume of the term debt would have yielded a profit of 326%.

strategy:

- more-than-average searches for debt: sell
- around-average searches for debt: hold
- fewer-than-average searches for debt: buy
Figure S38. Profit and loss for an investment strategy based on the volume of the search term 'fun' with $\Delta t = 3$ weeks.

Figure S39. Profit and loss for an investment strategy based on the volume of the search term 'water' with $\Delta t = 3$ weeks.
'fun'

If you torture the data enough, it will confess.

'water'

- R. Coates, statistician
Data's *elevation*?
List ~ ordering of any data

\[ M = [ 4, 7, 100, 42, 5, 47 ] \]
List ~ ordering of any data

Square brackets tell Python you want a list.

Commas separate elements.

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

len(M)  M[0]  M[0:3]

top-level length  indexing  slicing
Lists ~ collections of any data

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

- `len(L)`
  - **top-level length**
  - only counts *top-level* elements

- `L[0]`
  - **indexing**
  - could return a different type

- `L[0:1]`
  - **slicing**
  - always returns the same type, and always returns a substructure!

- `L[2][1:3]`
Indexing uses \[ \] Strings

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

Strings

s = 'harvey mudd college'

Some German words are so long that they have a perspective. For example
- Freundschaftsbezeichungen.
- Dilettantenaufdringlichkeiten.
- Stadtverordnetenversammlungen.

These things are not words, they are alphabetical processions.

- Mark Twain
Indexing uses [ ]

s = 'harvey mudd college'

index

s[0] is 'h'

s[17] is

s[6] is

s[ ] is 'e'

Strings

Read as "s-of-zero" or "s-zero"
Negative indices...

```
s = 'harvey mudd college'
```

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'

s[ : ] 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'
Slicing

\[ s = 'harvey mudd college' \]

\[
\begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\
\end{array}
\]

\[ s[\quad : \quad] \] \textit{slices} the string, returning a \textit{substring}.

- \[ s[0:6] \] is \textit{'harvey'}
- \[ s[12:18] \] is \textit{'colleg'}
- \[ s[17:] \] is \textit{'ge'}
- \[ s[:] \] is \textit{'harvey mudd college'}
Slicing

\[ s = 'harvey mudd college' \]

What are these slices?

\[ s[15:-1] \]

is 'mud'

and these?

\[ s[:2] \]

is 'e'

Don't wor'e' - Be hap'e'!
Skip-Slicing

\[ s[\text{start} : \text{end} + 1 : \text{end} - 1] \]\n
the third index is the stride length

default is +1

\[ s = 'harvey mudd college' \]

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

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

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

\[ s[1::6] \text{ is } \]
pi = [3, 1, 4, 1, 5, 9]
L = [ 'pi', "isn't", [4, 2] ]
M = 'You need parentheses for chemistry !'

What is \text{len}(\text{pi}) \quad 6
What is \text{len}(\text{L})
What is \text{len}(\text{L}[1])
What is \text{pi}[2:4]
What slice of \text{pi} is [3, 1, 4] \quad \text{pi}[0:3]
What slice of \text{pi} is [3, 4, 5]

What is \text{L}[0] \quad 'pi'
What is \text{L}[0][1]
What is \text{L}[0:1]
What slice of \text{M} is 'try'? 
What slice of \text{M} is 'shoe'? 
What is \text{M}[9:15]
What is \text{M}[:5]

These two are different! 
\text{pi}[0]*(\text{pi}[1]+\text{pi}[2]) \quad \text{pi}[0]*(\text{pi}[1:2]+\text{pi}[2:3])

Extra! Mind Muddler

Try it! These three are all different!
pi = [3,1,4,1,5,9]
L = [ 'pi', "isn't", [4,2] ]
M = 'You need parentheses for chemistry !'

Part 1

What is \(\text{len}(\pi)\) \quad 6
What is \(\text{len}(L)\) \quad 3
What is \(\text{len}(L[1])\)
What is \(\pi[2:4]\) \quad [4,1]
What slice of \(\pi\) is \([3,1,4]\) \quad \pi[:3]
What slice of \(\pi\) is \([3,4,5]\) \quad \pi[:2]

Part 2

What is \(L[0]\) \quad 'pi'
What is \(L[0][1]\) \quad 'i'
What is \(L[0:1]\) \quad ['pi']
What slice of \(M\) is 'try'? \(M[31:34]\) or \(M[-5:-2]\)
What slice of \(M\) is 'shoe'? 
What is \(M[9:15]\) \quad 'parent'
What is \(M[::5]\)

Extra! Mind Muddlers

These two are different!

\(\pi[0]*(\pi[1]+\pi[2])\) and \(\pi[0]*(\pi[1:2]+\pi[2:3])\) ?

15 \quad [1,4,1,4,1,4]
Python slices - it dices...

(data, at least)

...but wait, there's more!
Python slices - it dices...

Python functions

... but wait, there's more!

(data, at least)
Functioning in Python

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

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

```python
# Putting the "fun" into Python functions!

def dbl( x):
    """ returns double its input, x """
    return 2*x
```

More colorful ~
still broken...!
Functioning in Python

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

documentation string for all users

custom comment for other coders

Some of Python's baggage...
Function Fun!

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
Have a decafe-ternoon!

morning + evening, too

Just undo it!

Lab today ~ first hw problem