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 “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. “No worries,” Dodds mused – in fact, this might help me tomorrow…”

see three-eyed alien attack, p. 42
CS5 Favorites!
In-person help: *Tutoring hours...*

<table>
<thead>
<tr>
<th>Time</th>
<th>CS5 at HMC's LAC (up to 5)</th>
<th>CS5 away from HMC (up to 5)</th>
<th>CS 42 (up to 2)</th>
<th>CS 60 (up to 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday 8pm-10pm</td>
<td>Alicia Pentico</td>
<td>Frankie Konner (Pitzer Grovehouse)</td>
<td>Christina Lau (<a href="mailto:clau@g.hmc.edu">clau@g.hmc.edu</a>)</td>
<td>Elena Ehrlich (<a href="mailto:eehrlich@g.hmc.edu">eehrlich@g.hmc.edu</a>)</td>
</tr>
<tr>
<td></td>
<td>(<a href="mailto:apentico@g.hmc.edu">apentico@g.hmc.edu</a>, green)</td>
<td><a href="mailto:fkonner@students.pitzer.edu">fkonner@students.pitzer.edu</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aely Aronoff (<a href="mailto:aaronoff@hmc.edu">aaronoff@hmc.edu</a>)</td>
<td>Graham Brady (pitzer grove house)</td>
<td>David Mindlin</td>
<td>Harris McCullers (<a href="mailto:harrismccullers@gmail.com">harrismccullers@gmail.com</a>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(<a href="mailto:gbrady@students.pitzer.edu">gbrady@students.pitzer.edu</a>)</td>
<td></td>
<td>“I am willing to switch, email me</td>
</tr>
<tr>
<td>8-11pm</td>
<td>Jenna Kahn (Green)</td>
<td>Kate Emery (New Hall Kitchen)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(<a href="mailto:jmkahn@hmc.edu">jmkahn@hmc.edu</a>)</td>
<td>(<a href="mailto:kemery5437@scrippscollege.edu">kemery5437@scrippscollege.edu</a>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jerry Liang (<a href="mailto:jyliang@hmc.edu">jyliang@hmc.edu</a>)</td>
<td>Jacob Adolphe (Chall lounge probably somewhere by the TV)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="mailto:jadolphe21@cmc.edu">jadolphe21@cmc.edu</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Isaiah Fujii Bresnihan</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I don’t know how many Scripps CS5 grutors there are this semester, but it has come to my attention that there are not enough CS5 tutoring opportunities on Scripps right now. I had my first Scripps hours yesterday, which at the time were the only tutoring hours available on Scripps, and I had a lot of people show up (especially considering the fact that it was 3 or 4 days before the first deadline of the...
In-person help: Tutoring hours...

Monday 8pm-10pm or 11pm

Grutors: if you want to stay until 11pm, great! But, please don’t stay later than that...!
In-person help: *Tutoring hours...*

Join us Fri. aft.!
Ready for Picobot!
Picobot tutoring gets real!
Homework 0...  The adventure begins!

Lab!

Poptarts!
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**

- ?
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

name: x
type: int
LOC: 312
memory location 312

name: y
type: int
LOC: 324
memory location 324

id, del
Memory!

512 MB of memory

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Location</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>336</td>
</tr>
<tr>
<td></td>
<td></td>
<td>348</td>
</tr>
</tbody>
</table>

**Bit** = smallest amt of info.: 0 or 1

**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><strong>float</strong></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><strong>int</strong></td>
<td>42 or 10<strong>100</strong></td>
<td>integers – Python has <em>infinite precision ints!</em></td>
</tr>
<tr>
<td><strong>bool</strong></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>
Operate!

higher precedence

\[
\begin{array}{c}
( ) \\
** \\
- \\
* / \% // \\
+ - \\
> == < \\
=
\end{array}
\]
O-per-ate!

`**` higher precedence

`( )`

`**`

`-`

`* / % //`

`+ -`

`> == <`

`=`
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

\[
\begin{align*}
7 \mod 3 &= 1 \\
8 \mod 3 &= 2 \\
9 \mod 3 &= 0 \\
30 \mod 7 &= 3
\end{align*}
\]

\(x \mod y\) is the \textit{remainder} when \(x\) is divided by \(y\)

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

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

If \(x\) is a year, what happens on these years!? If \(x\) is a year, what happens on these years, football-wise!?
// integer division

7 // 3
8 // 3
9 // 3
30 // 7

\(x\div y\) is \(x/y\), rounded-down to an integer
integer division

7 // 3
8 // 3
9 // 3
30 // 7

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

Decomposition of 30 into 7's:

Why?

\[ 30 = (4) \times 7 + (2) \]

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

\[ x = (x \div y) \times y + (x \% y) \]

# 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!*
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 === !

Difference between == and === in JavaScript
how = works

x = 41
y = x + 1
z = x + y

Run these lines

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?

"Quiz"

name(s)

a = 11//2
b = a%3
c = b** a+b *a

Extra!

What are the values of a, b, and c after the 3 lines, at left, run?
Inside the machine...

What's happening in Python:

```python
x = 41
y = x + 1
z = x + y
x = x + y
```

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

- **Computation**
  - **x**: 41
    - Name: x
    - Type: int
    - LOC: 312
  - **y**: 42
    - Name: y
    - Type: int
    - LOC: 324
  - **z**: 83
    - Name: z
    - Type: int
    - LOC: 312

- **Memory (Data Storage)**
  - **x**: 41
    - Name: x
    - Type: int
    - LOC: 324
  - **y**: 42
    - Name: y
    - Type: int
    - LOC: 312
  - **z**: 83
    - Name: z
    - Type: int
    - LOC: 324
how = works

"Quiz"

x = 41
y = x + 1
z = x + y

What are x, y, and z at this time?

x = x + y

What are x, y, and z at this time?

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?

Extra!
how = works

"Quiz"

-x = 41
-y = x + 1
-z = x + y

Run these lines

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?
The number 42 is, in *The Hitchhiker's Guide to the Galaxy* by Douglas Adams, the "Answer to the Ultimate Question of Life, the Universe, and Everything", calculated by an enormous supercomputer named Deep Thought over a period of 7.5 million years. Unfortunately, no one knows what the question is. Thus, to calculate the Ultimate Question, a special computer the size of a small planet was built from organic components and named "Earth". The Ultimate Question "What do you get when you multiply six by nine"[17] was found by Arthur Dent and Ford Prefect in the second book of the series, *The Restaurant at the End of the Universe*. This appeared first in the radio play and later in the novelization of *The Hitchhiker's Guide to the Galaxy*. The fact that Adams named the episodes of the radio play "fits", the same archaic title for a chapter or section used by Lewis Carroll in "The Hunting of the Snark", suggests that Adams was influenced by Carroll's fascination with and frequent use of the number. The fourth book in the series, the novel *So Long, and Thanks for All the Fish*, contains 42 chapters. According to the novel *Mostly Harmless*, 42 is the street address of Stavromula Beta. In 1994 Adams created the 42 Puzzle, a game based on the number 42.
A free, teacher-less university in France is schooling thousands of future-proof programmers among many 42 references... mostly in cs5...!
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*

\[
\begin{align*}
\text{strings} & \quad s = \text{'}scripps\text{'}, \\
\text{c} & \quad = \text{'}college\text{'}, \\
\text{type...} & \quad \text{type}(s), \\
\text{len} & \quad \text{len}(s), \\
\text{add!} & \quad s + c, \\
\text{multiply!!} & \quad 2*s + 3*c
\end{align*}
\]
strings: textual data

Given \[
\begin{align*}
\text{s1} &= \text{'ha'} \\
\text{s2} &= \text{'t'}
\end{align*}
\]

What are \( \text{s1} + \text{s2} \)

\[
2*\text{s1} + \text{s2} + 2*(\text{s1}+\text{s2})
\]
Strings: textual data

Given

\[
\begin{align*}
  s_1 &= \text{'ha'} \\
  s_2 &= \text{'t'}
\end{align*}
\]

What are

\[
\begin{align*}
  s_1 + s_2 &= \text{hat} \\
  2s_1 + s_2 + 2(s_1+s_2) &= \text{haha that that that that}
\end{align*}
\]

What did you say?!
hahahahah
Pass those to the East!
Data, data everywhere...
1 Petabyte, PB == 1000 Terabytes, TB
1 Terabyte, TB == 1000 Gigabytes, GB

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?
Data's elevation?

G. Garcia Marquez

Google's users

Google's users

Wisdom

Knowledge

Information

Data
Lists ~ collections of any data

\[ M = [ 4, 7, 100, 42, 5, 47 ] \]
Lists ~ collections 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!
Indexing uses [ ]

Strings

s = 'harvey mudd college'

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

$s = 'harvey mudd college'\$

Some **German words** are so long that they have a perspective. For example,

- Freundschaftsbezeigungen.
- Dilettantenaufringlichkeiten.
- Stadtverordnetenversammlungen.

*These things are not words, they are alphabetical processions.*

- Mark Twain
Indexing uses [ ]

\[ s = 'harvey mudd college' \]

\[\begin{align*}
\text{index} & & \text{Read as} \\
\text{s}[0] & \text{is} & \text{"h"} & \text{"s-of-zero"} \\
\text{s}[17] & \text{is} & \text{"e"} & \text{or "s-zero"} \\
\text{s}[6] & \text{is} & & \\
\text{s}[ ] & \text{is} & \text{"e"} & \\
\end{align*}\]
Negative indices...

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

Negative indices count \textit{backwards} from the end!

\[
\begin{align*}
  s[\neg{-1}] & \quad \text{is} & & \quad 'e' \\
  s[\neg{-18}] & \quad \text{is} \\
  s[\neg{-7}] & \quad \text{is} \\
  s[\neg{0}] & \quad \text{is}
\end{align*}
\]
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'
```

```
s[start:end+1] 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'`

A missing index means that `end` of the string.
Slicing

`s = 'harvey mudd college'

What are these slices?

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

and these?

\[
\text{s[:2] is 'e'}
\]
Skip-Slicing

```
the third index is
the stride length
default is +1
```

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

```
s[2:11:2] is 're ud'
s[17:12] is 'doe'
s[17:12:-1] is 'doe'
s[: : -1] is 'doe'
s[1: : 6] is 'harvey mudd'
```

- G. Garcia Marquez

I love this one.
\[ \pi = [3,1,4,1,5,9] \]
\[ L = [ '\pi', 'isn't', [4,2] ] \]
\[ M = 'You need parentheses for chemistry !' \]

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

**Part 2**
- What is \( L[0] \) \( '\pi' \)
- What is \( L[0][1] \)
- What is \( L[0:1] \)
- What slice of \( M \) is \'try'?\)
- What slice of \( M \) is \'shoe'?\)
- What is \( M[9:15] \)
- What is \( M[::5] \)

**Extra! Mind Muddler**
- What are \( \pi[0]*(\pi[1]+\pi[2]) \) and \( \pi[0]*(\pi[1:2]+\pi[2:3]) \)?

These two are different!
\( \pi = [3,1,4,1,5,9] \)

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

\( M = 'You need parentheses for chemistry !' \)

**Part 1**

- What is \( \text{len}(\pi) \)?
  - 6

- What is \( \text{len}(L) \)?

- What is \( \text{len}(L[1]) \)?

- What is \( \pi[2:4] \)?

- What slice of \( \pi \) is \([3,1,4]\)?

- What slice of \( \pi \) is \([3,4,5]\)?

**Part 2**

- What is \( L[0] \)?
  - 'pi'

- What is \( L[0][1] \)?

- What is \( L[0:1] \)?

- What slice of \( M \) is 'try'?  

- What slice of \( M \) is 'shoe'?  

- What is \( M[9:15] \)?

- What is \( M[::5] \)?

**Extra! Mind Muddler**

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

*These two are different!*
\[ \text{pi} = [3,1,4,1,5,9] \]

\[ \text{L} = [ 'pi', "isn't", [4,2] ] \]

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

**Part 1**

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

**Part 2**

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

**Extra! Mind Muddlers**

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

These two are different! \( \rightarrow 15 \)
Python slices - it dices...

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

... but wait, there's more!

( data, at least )

Python functions

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

Still broken...!
# my own function!

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

comment for other coders

documentation string for all users

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
Have a dedecaf-ternoon!
morning + evening, too

Just undo it!

This week's lab ~
first two hw problems