They call that an alien?

Spock *mind-melds* three-eyed aliens!

Provably.

http://www.youtube.com/watch?v=iapcKvn7DdY
http://www.youtube.com/watch?v=yuEZeYDdmvQ
They call that an alien? Spock mind-melds three-eyed aliens! Provably.

http://www.youtube.com/watch?v=iapcKv7DdY
http://www.youtube.com/watch?v=yuEZEyDdmvQ
import random

print("Welcome to rock/paper/scissors, human!\n")

comp = random.choice(["rock","paper","scissors"])
user = input(" +++ Choose wisely: ")

print(" You chose", user)
print(" I chose", comp)

if user == 'rock':
    if comp == 'paper':
        print(" paper defeats rock - I win!")
Your Quest!

Create a short text-adventure in Python...

Use at least five control structures with decisions: (if/elif/else)

We look forward to adventuring!
Welcome back to CS 5!

**Homework 0**

- **Problem 0:** Reading + response... *due last Monday!*
- **Problem 1:** Four-fours program: Can be done for lab... *due this Sunday...*
- **Problem 2:** Rock-paper-scissors + Adventure *due this Sunday...*
- **Problems 3-4:** Picobot! empty room (3) maze (4) *Next week!*
Welcome back to CS 5!

Homework 1

Problem 1a: Slicing and indexing: Do these during "lab"...

Problem 2: Functions! In lab or beyond...

hw2pr2: PythonBat functions (not due this week - but addictive!)
Problem 0?

Typically an article on CS or an application...

Submit a one-paragraph response

\[ \text{Use VSCode to create it!} \]

Small part (5 pts)

\[ \begin{align*}
5 & \text{ – insightful, careful} \\
4 & \text{ – thoughtful} \\
3 & \text{ – complete, on topic} \\
0-2 & \text{ – less than complete}
\end{align*} \]

A few sentences that raise or address questions, using the article as a guide.

This week's article might not seem like CS at first...
Seventy years ago, in 1940, a popular science magazine published a short article that set in motion one of the trendiest intellectual fads of the 20th century. At first glance, there seemed little about the article to augur its subsequent celebrity. Neither the title, "Science and Linguistics," nor the magazine, M.I.T.’s Technology Review, was most people’s idea of glamour. And the author, a chemical engineer who worked for an insurance company and moonlighted as an anthropology lecturer at Yale University, was an unlikely candidate for international superstardom. And yet Benjamin Lee Whorf let loose an alluring idea about language’s power over the mind, and his stirring prose seduced a whole generation into believing that our mother tongue restricts what we are able to think.

and I thought my language was alien!
But then a remote Australian aboriginal tongue, Guugu Yimithirr, from north Queensland, turned up, and with it came the astounding realization that not all languages conform to what we have always taken as simply “natural.” In fact, Guugu Yimithirr doesn’t make any use of egocentric coordinates at all. The anthropologist John Haviland and later the linguist Stephen Levinson have shown that Guugu Yimithirr does not use words like “left” or “right,” “in front of” or “behind,” to describe the position of objects. Whenever we would use the egocentric system, the Guugu Yimithirr rely on cardinal directions. If they want you to move over on the car seat to make room, they’ll say “move a bit to the east.” To tell you where exactly they left something in your house, they’ll say, “I left it on the southern edge of the western table.” Or they would warn you to “look out for that big ant just north of your foot.” Even when shown a film on television, they gave descriptions of it based on the orientation of the screen. If the television was facing north, and a man on the screen was approaching, they said that he was “coming northward.”
Readings' key:

CS  !=  Programming
Readings' key:

CS != Programming

not equal to
What *is* programming?

Programming as recipe-writing...
What *is* programming?

Programming as recipe-writing

vs.

Programming as learning a foreign language

1) Expect it to be different!

2) Don't memorize anything!

3) Immerse == Experiment!
What about the *Python* programming language?
One possible relationship...
One possible relationship...

Happy co-existence...

*It can even be comfy!*
The *foreign language* of python...

- **syntax**: How it looks
- **semantics**: What it does
- **intent**: What it should do
The *foreign language* of python...

<table>
<thead>
<tr>
<th>syntax</th>
<th>semantics</th>
<th>intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>How it looks</td>
<td>What it does</td>
<td>What it should do</td>
</tr>
</tbody>
</table>

```python
name = raw_input('Hi... what is your name? ')  
print  # prints
if name == 'Eliot' or name == 'Ran':  
  print 'I \ '  
  print 'm "offline." Try later.'
elif name == 'Zach':  
  print 'Zach Quinto...? No? \ Oh.'
else:  
  print 'Welcome', name, '!
my_choice = random.choice( [ 'R','P','S' ] )
print 'My favorite object is', my_choice, '!
```

This program should greet its user appropriately.
The *foreign language* of python...

**syntax**
How it looks

**semantics**
What it does

**intent**
What it should do

```python
name = raw_input('Hi... what is your name? ')  # prints a blank line
if name == 'Eliot' or name == 'Ran':
    print 'I \n    "offline." Try later.'
elif name == 'Zach':  # is it Zach?
    print 'Zach Quinto...? No?
    Oh.'
else:  # in all other cases...
    print 'Welcome', name, '!
    my_choice = random.choice( [ 'R','P','S' ] )
    print 'My favorite object is', my_choice, '!'  
```

This program should greet its user appropriately.
The *foreign language* of python...

**syntax**
- How it looks

**semantics**
- What it does

**intent**
- What it should do

```python
name = raw_input('Hi... what is your name? ')
print # prints a blank line

if name == 'Eliot' or name == 'Ran':
    print 'I\'m "offline." Try later.'

elif name == 'Zach':    # is it Zach?
    print 'Zach Quinto...?', 'No?', 'Oh.'

else:                   # in all other cases...
    print 'Welcome', name, '!!'
    my_choice = random.choice( [ 'R','P','S' ] )
    print 'My favorite object is', my_choice, '!!'
```
The **foreign language** of python...

**syntax**
- How it looks

**semantics**
- What it does

**intent**
- What it should do

```python
name = raw_input('Hi... what is your name? ')
print # prints a blank line

if name == 'Eliot' or name == 'Ran':
    print 'I\'m "offline." Try later.'

elif name == 'Zach':    # is it Zach?
    print 'Zach Quinto...?', 'No?', 'Oh.'

else:                   # in all other cases...
    print 'Welcome', name, '!!'
    my_choice = random.choice( [ 'R','P','S' ] )
    print 'My favorite object is', my_choice, "!
```

The **foreign language** of python...

**syntax**
- How it looks

**semantics**
- What it does

**intent**
- What it should do

```python
name = raw_input('Hi... what is your name? ')
print # prints a blank line

if name == 'Eliot' or name == 'Ran':
    print 'I\'m "offline." Try later.'

elif name == 'Zach':    # is it Zach?
    print 'Zach Quinto...?', 'No?', 'Oh.'

else:                   # in all other cases...
    print 'Welcome', name, '!!'
    my_choice = random.choice( [ 'R','P','S' ] )
    print 'My favorite object is', my_choice, "!
```
Flowchart...

a graphical view of a program's semantics

prompt and user input

name

decision

Is name equal to 'Eliot' or 'Ran'

machine-produced output

Is name equal to 'Zach'

print message for Eliot or Ran

print message for Zach

print message for everyone else
The *foreign language* of python...

- **Syntax**: How it looks
- **Semantics**: What it does
- **Intent**: What it should do

- How Python *looks*!
  - how punctuation is used
  - the language *keywords* used
  - use of whitespace

- peculiarities of formatting
- how behavior is affected ...
The *foreign language* of python...

- **syntax**
  - How it looks

- **semantics**
  - What it does

- **intent**
  - What it should do

- **human-typed input**

---

**How Python looks!**

- how punctuation is used
- the language *keywords* used
- use of whitespace

- peculiarities of formatting
- how behavior is affected ...
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**
(1) Find and correct as many errors as you can in this code:

```python
import random

user = input( "Choose your weapon! " )
comp = random.choice( [ 'rock', 'paper', 'scissors' ] )
print('user (you) chose:', 'user')
print('comp (me!) chose:' comp)

if user == rock and comp = 'paper'
    print('The result is, YOU LOSE.'
    print('unless you're a CS5 grader, then YOU WIN :)')
```

(2) This one line does three things... what are they?

(3) Extra! Can you find 7 punctuation marks used in more than one way here?
Syntax challenge!

(1) Find and correct as many errors as you can in this code:

```python
import random

user = input( "Choose your weapon! " )
comp = random.choice( [ 'rock', 'paper', 'scissors' ])

print('user (you) chose:'

print('comp (me!) chose:'

if user == 'rock' and comp == 'paper' :
    print('The result is, YOU LOSE.'
    print('unless you're a CS5 grader, then YOU WIN :))'

(2) This one line does **three** things... what are they?

(3) Extra! Can you find 7 punctuation marks used in **more than one way** here?

You'll hand this in as a screenshot under day1quiz1

Name(s):
(1) Find and correct as many errors as you can in this code:

```python
import random

user = input( "Choose your weapon! " )

comp = random.choice( [ 'rock', 'paper', 'scissors' ] )

print('user (you) chose:', user)

print('comp (me!) chose:' comp)

if user == rock and comp = 'paper'
    print('The result is, YOU LOSE.'
    print('unless you're a CS5 grader, then YOU WIN :)' )
```

(2) This one line does **three** things... what are they?

(3) Extra! Can you find 7 punctuation marks used in **more than one way** here?
The *challenge* of programming...

This seems somehow familiar...?!

**syntax**
- How it looks

**semantics**
- What it does

**intent**
- What it should do

---

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

?!
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

Look deep into my eyes...
(1) Find and correct as many errors as you can in this code:

```python
import random

user = input( "Choose your weapon! " )
comp = random.choice( [ 'rock','paper','scissors' ] )
print('user (you) chose:', 'user')
print('comp (me!) chose:', comp)

if user == rock and comp == 'paper'
    print('The result is, YOU LOSE.'
    print('unless you're a CS5 grader, then YOU WIN :))')
```

(2) This one line does **three** things... what are they?

(3) Extra! Can you find 7 punctuation marks used in *more than one way* here?
import random

user = input( "Choose your weapon! " )
comp = random.choice( [ 'rock', 'paper', 'scissors' ] )

print('user (you) chose:', user)
print('comp (me!) chose:', comp)

if user == 'rock' and comp == 'paper':
    print('The result is, YOU LOSE. ')
    print('unless you\'re a CS5 grader, then YOU WIN! : )')

(1) Find and correct as many errors as you can here...

(2) This line is doing **three** things... what are they?

(3) Punctuation used in more than one way: () . ' = , :
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

<table>
<thead>
<tr>
<th>Box</th>
<th>Name</th>
<th>Type</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>x</td>
<td>int</td>
<td>312</td>
</tr>
<tr>
<td>42</td>
<td>y</td>
<td>int</td>
<td>324</td>
</tr>
<tr>
<td>83</td>
<td>z</td>
<td>int</td>
<td>336</td>
</tr>
</tbody>
</table>

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

- **512 MB of memory**

- **bit** = smallest amount 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>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 spell!  

"Boolean values"  

"Boolean operators"  

George Boole
Operate!

( )

**

- 

* / % //

+ -

> === <

=
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 modulo operator `%` is used to find the remainder when one number is divided by another. For example:

- $7 \% 3 = 1$
- $8 \% 3 = 2$
- $9 \% 3 = 0$
- $30 \% 7 = 3$

$x \% y$ is 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

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

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

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

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

Decomposition of 30 into 7's:

Why?

30 == (4) * 7 + (2)

Decomposition of x into y's:

\[ x == \left( \frac{x}{y} \right) \times y + (x \% y) \]

# 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 ===!
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):
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)

- **x**: 41
  - name: x
  - type: int
  - LOC: 312

- **y**: 42
  - name: y
  - type: int
  - LOC: 324

- **z**: 83
  - name: z
  - type: int
  - LOC: 312

- **x**: 83
  - name: x
  - type: int
  - LOC: 324

id, del
Thought experiment...

- 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!

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

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)**
  - **41**
    - name: \(x\)
    - type: int
    - LOC: 312
  - **42**
    - name: \(y\)
    - type: int
    - LOC: 324
  - **83**
    - name: \(z\)
    - type: int
    - LOC: 312
  - **83**
    - name: \(x\)
    - type: int
    - LOC: 324

id, del
how = works

Thought experiment...

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?

- try these on the back page first...
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.

among many 42 references...
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*

```python
strings
s = 'scripps'
c = 'college'

type...
type(s)

len
len(s)

add!
s + c

multiply!!
2*s + 3*c
```
strings: *textual data*

\[
\begin{aligned}
\text{Given} & \quad \begin{cases}
    s_1 = 'ha' \\
    s_2 = 't'
\end{cases} \\
\text{What are} & \quad s_1 + s_2 \\
2s_1 + s_2 + 2(s_1+s_2)
\end{aligned}
\]
strings: textual data

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

What are \( s1 + s2 \) hat

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

What did you say!??
s = 'hahahahahah'
Data, data everywhere...
Data, data everywhere...

1 Yottabyte
1 Zettabyte
1 Exabyte
1 Petabyte
1 Terabyte, TB == 1000 Gigabytes, GB
1 Petabyte, PB == 1000 Terabytes, TB

Data produced each year

- 2002: 5 EB
- 2006: 161 EB
- 2009: 800 EB
- 2011: 1.8 ZB
- 2015: 8.0 ZB
- 2020: 44 ZB
- 2025: 175 ZB

Human brain’s capacity
14 PB
60 PB

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?
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.

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

<table>
<thead>
<tr>
<th>( \text{len}(M) )</th>
<th>( M[0] )</th>
<th>( M[0:3] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>top-level length</td>
<td>indexing</td>
<td>slicing</td>
</tr>
</tbody>
</table>
Lists ~ collections of *any* data

$L = \begin{bmatrix} 3.14, [2,40], 'third', 42 \end{bmatrix}$

- `len(L)`
- `L[0]`
- `L[0:1]`

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

\[ s = 'harvey mudd college' \]

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

- Freundschaftsbezeigungen.
- Dilettantenaufdringlichkeiten.
- Stadtverordnetenversammlungen.

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

- Mark Twain
Indexing uses [ ]

\[
s = 'harvey mudd college'
\]

index
\[
\begin{align*}
\text{s[0]} & \quad \text{is} & \quad 'h' \\
\text{s[17]} & \quad \text{is} & \quad \text{(omitted)} \\
\text{s[6]} & \quad \text{is} & \quad 'e' \\
\text{s[ ]} & \quad \text{is} & \quad 'e'
\end{align*}
\]

Read as "s-of-zero" or "s-zero" for strings, too
**Negative indices...**

\[ s = 'harvey mudd college' \]

<table>
<thead>
<tr>
<th></th>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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

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

- \( s[\ : \ ] \) 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'

What's going on here?
Slicing

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

\( s[\ : \ ] \) \textit{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?

- `s[15:-1]` is 'mud'
- `s[::2]` is 'e'

and these?

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

\[ s = 'harvey mudd college' \]

\[ s[2:11:2] \]
\[ s[17:12] \]
\[ s[17:12:-1] \]
\[ s[:::6] \]

\[ the \ third \ index \ is \ the \ stride \ length \]
\[ default \ is \ +1 \]
\[ \pi = [3,1,4,1,5,9] \]
\[ L = [ '\pi', '', 'isn't', [4,2] ] \]
\[ M = 'You need parentheses for chemistry!' \]

<table>
<thead>
<tr>
<th>Part 1</th>
<th>Part 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is ( \text{len}(\pi) )</td>
<td>6</td>
</tr>
<tr>
<td>What is ( \text{len}(L) )</td>
<td></td>
</tr>
<tr>
<td>What is ( \text{len}(L[1]) )</td>
<td></td>
</tr>
<tr>
<td>What is ( \pi[2:4] )</td>
<td></td>
</tr>
<tr>
<td>What slice of ( \pi ) is [3,1,4]</td>
<td>[ \pi[0:3] ]</td>
</tr>
<tr>
<td>What slice of ( \pi ) is [3,4,5]</td>
<td></td>
</tr>
<tr>
<td>What is ( L[0] )</td>
<td>'\pi'</td>
</tr>
<tr>
<td>What is ( L[0][1] )</td>
<td></td>
</tr>
<tr>
<td>What is ( L[0:1] )</td>
<td></td>
</tr>
<tr>
<td>What slice of ( M ) is 'try'?</td>
<td></td>
</tr>
<tr>
<td>What slice of ( M ) is 'shoe'?</td>
<td></td>
</tr>
<tr>
<td>What is ( M[9:15] )</td>
<td></td>
</tr>
<tr>
<td>What is ( M[:5] )</td>
<td></td>
</tr>
</tbody>
</table>

These two are different:

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

- **What is** `len(pi)`
  - 6

- **What is** `len(L)`

- **What is** `pi[0:3]`
  - `['p', 'i', ' ']'`

- **What slice of** `pi` **is** `[3, 4, 5]`

- **What slice of** `pi` **is** `[3, 1, 4]`

- **What is** `pi[0]*(pi[1]+pi[2])`

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

### Part 2

- **What is** `L[0]`
  - `'pi'`

- **What slice of** `M` **is** `'shoe'`?

- **What is** `M[9:15]`

- **What is** `M[:,5]`

### Extra! Mind Muddler

These two are different!

- **What are** `pi[0]*(pi[1]+pi[2])` and `pi[0]*(pi[1:2]+pi[2:3])`?
\[ \pi = [3, 1, 4, 1, 5, 9] \]
\[ L = [ '\pi', 'isn't', [4, 2] ] \]
\[ M = 'You need parentheses for chemistry!' \]

---

**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**

- These two are different!
  - What are \( \pi[0]*(\pi[1]+\pi[2]) \) and \( \pi[0]*(\pi[1:2]+\pi[2:3]) \)?
\textbf{Part 1}

What is \texttt{len(pi)} \hspace{2cm} 6

What is \texttt{len(L)}

What is \texttt{len(L[1])}

What is \texttt{pi[2:4]}

What slice of \texttt{pi} is [3,1,4] \hspace{2cm} \texttt{pi[0:3]}

What slice of \texttt{pi} is [3,4,5]

\textbf{Part 2}

What is \texttt{L[0]} \hspace{2cm} \texttt{'pi'}

What is \texttt{L[0][1]}

What is \texttt{L[0:1]}

What slice of \texttt{M} is \texttt{'try'}?\hspace{2cm} These three are all different

What slice of \texttt{M} is \texttt{'shoe'}?

What is \texttt{M[9:15]}

What is \texttt{M[::5]}
\( \pi = [3, 1, 4, 1, 5, 9] \)

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

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

**Part 1**

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

- What is \( \text{len}(L) \)?
  - 3

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

- What is \( \pi[2:4] \)?
  - \([4, 1]\)

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

- What slice of \( \pi \) is \([3, 4, 5]\)?
  - \(\pi[:2]\)

**Part 2**

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

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

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

- What slice of \( M \) is \('try'\)?
  - \(M[31:34]\) or \(M[-5:-2]\)

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

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

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

**Extra! Mind Muddlers**

- What are \( \pi[0]*(\pi[1]+\pi[2]) \) and \( \pi[0]*(\pi[1:2]+\pi[2:3]) \)?
  - These two are different!
  - \(15\) and \([1,4,1,4,1,4]\)
Python slices - it dices...

(data, at least)

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

(data, at least)

Python functions

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

Still broken...!
Functioning in Python

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

documentation string for all users

comment for other coders

Some of Python's baggage...

Python's keywords
Function *Fun*!

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

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

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

*strings, lists, numbers ... all data are fair game*
Have a dedecaf-ternoon!

morning + evening, too

Just undo it!

Homework 1

Problem 0: Reading + response...  due 5/19 (today!)

Problem 1a: Slicing and indexing: Do these during "lab"...

Problem 1b: Functions! In lab or beyond...  due 5/20 (tomorrow!)
CodingBat code practice

Welcome to Codingbat. See help for the latest.

<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
</table>
| **Warmup-1** 🌟🌟🌟🌟
  Simple warmup problems to get started, no loops (solutions available) |
| **Warmup-2** 🌟🌟🌟
  Medium warmup string/list problems with loops (solutions available) |
| **String-1** 🌟🌟🌟
  Basic python string problems -- no loops |
| **List-1** 🌟🌟🌟
  Basic python list problems -- no loops. |
| **Logic-1** 🌟🌟
  Basic boolean logic puzzles -- if else and or not |
| **Logic-2** 🌟🌟
  Medium boolean logic puzzles -- if else and or not |
| **String-2** 🌟
  Medium python string problems -- 1 loop. |
| **List-2** 🌟
  Medium python list problems -- 1 loop. |
Use your email here – and it’s certainly ok to use one of your "not-so-secure passwords..."

[2] Click on prefs and submit the "teacher share" address (see next screenshot)
In this textbox, enter dodds@cs.hmc.edu then hit Share
To start Week 1’s lab, you will want to setup your screen, perhaps with this browser on one side and VSCode (split into editor on top and terminal at bottom) on the other side:

Sane Settings for VSCode...

VSCode’s default Python settings allow - in my opinion - way too many pop-up messages and windows as “help”:

How to disable documentation popups? Example shown in screenshot.