Revisit word guessing game

- There is a secret word.
- The user has 10 chances to guess letters until the word has been spelled out.
- We implemented a class SecretWord to keep track of both the word being guessed and what the user sees / has guessed so far.

Play the game.

How does the game go?

word_list = [ ... candidate words for user to guess ... ]
N_CHANCES = 10
Set the secret word

User guesses until no more guesses or secret is solved
Reveal the word

Setting up a while-loop

0. Situation is to do something until an event happens
1. Write the continuation condition
   * Create var names as necessary to express condition
   * May be easier to negate stop condition to get continuation condition
2. Initialize loop vars (vars in loop condition) as necessary
3. In loop body: update loop vars to possibly change loop condition from True to False
4. Write the rest of the loop body

Get and check user input with while-loop

- User may not enter appropriate input
- Can use assert and error out if user provides inappropriate input—not friendly
- Can re-prompt user for appropriate input
- Re-prompt how many times? Can re-prompt until user does the right thing

Start next video:
Use while-loop get and check user input
Other changes to word guessing game?

- Allow 6 strikes instead of 10 guesses
  - Change in game module
- Accommodate space and hyphen
  Eg., “ice cream” displayed as ___ _____
  “high-rise” displayed as ____-____
  - Change in class SecretWord
- Change instance attribute display_word from a string to a list of letters. How about secret_word?

Search Algorithms

- Search for a target x in a list v
  - Start at index 0, keep checking until you find it
  - Start at index 0, keep checking until no more elements to check

Linear search

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>k</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>12</td>
<td>35</td>
<td>33</td>
<td>15</td>
</tr>
<tr>
<td>x</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Repeated halving of “search window”

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>12</td>
<td>35</td>
<td>33</td>
<td>15</td>
</tr>
<tr>
<td>x</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How do you search for a word in a dictionary? (NOT linear search)

To find the word “tanto” in my Spanish dictionary…

while dictionary is longer than 1 page:
  Open to the middle page
  if first entry comes before “tanto”:
    Rip* and throw away the 1st half
  else:
    Rip* and throw away the 2nd half

* For dramatic effect only—don’t actually rip your dictionary! Just pretend that the part is gone.

Start next video:
Search algorithms (linear search, binary search)
Binary Search

- Repeatedly halve the “search window”
- An item in a sorted list of length $n$ can be located with just $\log_2 n$ comparisons.
- “Savings” is significant!

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log_2(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>10000</td>
<td>13</td>
</tr>
</tbody>
</table>

---

**Binary Search: target $x = 70$**

0 1 2 3 4 5 6 7 8 9 10 11

$\text{v: 12 15 33 35 42 45 51 62 73 75 86 98}$

- $i$: 6
- $\text{v[mid] is not x}$
- $\text{x < v[mid]}$
- $\text{j: 11}$
- $\text{So throw away the right half...}$

---

**Binary Search: target $x = 70$**

0 1 2 3 4 5 6 7 8 9 10 11

$\text{v: 12 15 33 35 42 45 51 62 73 75 86 98}$

- $i$: 6
- $\text{v[mid] is not x}$
- $\text{v[mid] < x}$
- $\text{j: 7}$
- $\text{So throw away the left half...}$

---

**Binary Search: target $x = 70$**

0 1 2 3 4 5 6 7 8 9 10 11

$\text{v: 12 15 33 35 42 45 51 62 73 75 86 98}$

- $i$: 7
- $\text{v[mid] is not x}$
- $\text{v[mid] < x}$
- $\text{j: 7}$
- $\text{So throw away the left half...}$

---

**Binary Search: target $x = 70$**

0 1 2 3 4 5 6 7 8 9 10 11

$\text{v: 12 15 33 35 42 45 51 62 73 75 86 98}$

- $i$: 8
- $\text{v[mid] is not x}$
- $\text{v[mid] < x}$
- $\text{j: 7}$
- $\text{DONE because}$
- $\text{i no longer less than j}$
- $\text{no valid search window}$

---