Announcements

- HW6 due next Thursday

- Trends project due on Tuesday
  - Partners are required; find one in lab or on Piazza
  - Will not work in IDLE
  - New bug submission policy; see Piazza
Practical Guidance: Choosing Names

Names typically don’t matter for correctness, but they matter tremendously for legibility
Names typically don’t matter for correctness, but they matter tremendously for legibility

```java
boolean d = play_helper;
```
Names typically don’t matter for correctness, but they matter tremendously for legibility.

boolean turn_is_over  d  dice  play_helper  take_turn
Practical Guidance: Choosing Names

Names typically don’t matter for correctness, but they matter tremendously for legibility

`boolean turn_is_over d dice play_helper take_turn`

Use names for repeated compound expressions
Practical Guidance: Choosing Names

Names typically don’t matter for correctness, but they matter tremendously for legibility

boolean turn_is_over  d dice  play_helper take_turn

Use names for repeated compound expressions

\[
\text{if sqrt(square(a) + square(b)) > 1:} \\
\quad x = x + \text{sqrt(square(a) + square(b))}
\]
Practical Guidance: Choosing Names

Names typically don’t matter for correctness, but they matter tremendously for legibility.

boolean turn_is_over d dice play_helper take_turn

Use names for repeated compound expressions:

```python
if sqrt(square(a) + square(b)) > 1:
    x = x + sqrt(square(a) + square(b))
    h = sqrt(square(a) + square(b))
    if h > 1:
        x = x + h
```
Names typically don’t matter for correctness, but they matter tremendously for legibility.

```python
boolean turn_is_over
dice play_helper take_turn
```

Use names for repeated compound expressions:

```python
if sqrt(square(a) + square(b)) > 1:
    x = x + sqrt(square(a) + square(b))
    h = sqrt(square(a) + square(b))
    if h > 1:
        x = x + h
```

Use names for meaningful parts of compound expressions.
Practical Guidance: Choosing Names

Names typically don’t matter for correctness, but they matter tremendously for legibility

```python
boolean turn_is_over   d  dice       play_helper  take_turn
```

Use names for repeated compound expressions

```python
if sqrt(square(a) + square(b)) > 1:
    x = x + sqrt(square(a) + square(b))
```

```python
    h = sqrt(square(a) + square(b))
    if h > 1:
        x = x + h
```

Use names for meaningful parts of compound expressions

```python
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```
Practical Guidance: Choosing Names

Names typically don’t matter for correctness, but they matter tremendously for legibility

boolean  turn_is_over    d  dice    play_helper  take_turn

Use names for repeated compound expressions

\[
\text{if } \sqrt{\text{square}(a) + \text{square}(b)} > 1:
\]
\[
x = x + \sqrt{\text{square}(a) + \text{square}(b)}
\]
\[
\text{if } h > 1:
\]
\[
x = x + h
\]

Use names for meaningful parts of compound expressions

\[
x = \left(-b + \sqrt{\text{square}(b) - 4 \times a \times c}\right) / \left(2 \times a\right)
\]
\[
\text{disc_term} = \sqrt{\text{square}(b) - 4 \times a \times c}
\]
\[
x = \left(-b + \text{disc_term}\right) / \left(2 \times a\right)
\]
Practical Guidance: DRY
Sometimes, removing repetition requires restructuring the code.
Sometimes, removing repetition requires restructuring the code

def find_quadratic_root(a, b, c, plus=True):
    """Applies the quadratic formula to the polynomial
    ax^2 + bx + c.""
    if plus:
        return (-b + \sqrt{\text{square}(b) - 4 \times a \times c}) / (2 \times a)
    else:
        return (-b - \sqrt{\text{square}(b) - 4 \times a \times c}) / (2 \times a)
Sometimes, removing repetition requires restructuring the code

```python
def find_quadratic_root(a, b, c, plus=True):
    """Applies the quadratic formula to the polynomial
    ax^2 + bx + c.""
    if plus:
        return (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
    else:
        return (-b - sqrt(square(b) - 4 * a * c)) / (2 * a)
```

Practical Guidance: DRY
Sometimes, removing repetition requires restructuring the code.

```python
def find_quadratic_root(a, b, c, plus=True):
    """Applies the quadratic formula to the polynomial ax^2 + bx + c.""

    if plus:
        return (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
    else:
        return (-b - sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```python
def find_quadratic_root(a, b, c, plus=True):
    """Applies the quadratic formula to the polynomial ax^2 + bx + c.""
    disc_term = sqrt(square(b) - 4 * a * c)
    if not plus:
        disc_term *= -1
    return (-b + disc_term) / (2 * a)
```
Test-Driven Development
Test-Driven Development

Write the test of a function before you write a function
Test-Driven Development

Write the test of a function before you write a function

A test will clarify the (one) job of the function
Write the test of a function before you write a function

A test will clarify the (one) job of the function

Your tests can help identify tricky edge cases
Test-Driven Development

Write the test of a function before you write a function

A test will clarify the (one) job of the function
Your tests can help identify tricky edge cases

Develop incrementally and test each piece before moving on
Test-Driven Development

Write the test of a function before you write a function

- A test will clarify the (one) job of the function
- Your tests can help identify tricky edge cases

Develop incrementally and test each piece before moving on

- You can’t depend upon code that hasn’t been tested
Test-Driven Development

Write the test of a function before you write a function

A test will clarify the (one) job of the function
Your tests can help identify tricky edge cases

Develop incrementally and test each piece before moving on

You can’t depend upon code that hasn’t been tested
Run your old tests again after you make new changes
Hog Contest
Contest rules:

- All entries run against every other entry
- An entry wins a match if its true win rate is > 0.5
- All strategies must be deterministic, pure functions and must not use pre-computed data
- Extra credit for entries with the most wins or the highest cumulative win rate
- Total of 54 valid submissions
Hog Contest

Contest rules:

- All entries run against every other entry
- An entry wins a match if its true win rate is > 0.5
- All strategies must be deterministic, pure functions and must not use pre-computed data
- Extra credit for entries with the most wins or the highest cumulative win rate
- Total of 54 valid submissions

We used `itertools.combinations` to determine the set of matches
<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>1:05</td>
</tr>
<tr>
<td>2</td>
<td>Sarah</td>
<td>1:08</td>
</tr>
<tr>
<td>3</td>
<td>Mike</td>
<td>1:10</td>
</tr>
<tr>
<td>4</td>
<td>Emily</td>
<td>1:12</td>
</tr>
<tr>
<td>5</td>
<td>David</td>
<td>1:15</td>
</tr>
</tbody>
</table>

Top Finishers
Top Finishers

Congratulations to the team of Colin Lockard and Sherry Xu, who achieved a perfect 53-0 record and the highest win rate (28.77)!
Top Finishers

Congratulations to the team of Colin Lockard and Sherry Xu, who achieved a perfect 53-0 record and the highest win rate (28.77)!

Second-most wins (51-2): Eric Holt and Anna Carey
Top Finishers

Congratulations to the team of Colin Lockard and Sherry Xu, who achieved a perfect 53-0 record and the highest win rate (28.77)!

Second-most wins (51-2): Eric Holt and Anna Carey

Second-highest win rate (28.70): Don Mai and Jeechee Chen
Top Finishers

Congratulations to the team of Colin Lockard and Sherry Xu, who achieved a perfect 53-0 record and the highest win rate (28.77)!

Second-most wins (51-2): Eric Holt and Anna Carey

Second-highest win rate (28.70): Don Mai and Jeechee Chen

Third-highest in both (50-3, 28.67): Sean Scofield and Frank Lu
Top Finishers

Congratulations to the team of Colin Lockard and Sherry Xu, who achieved a perfect 53-0 record and the highest win rate (28.77)!

Second-most wins (51-2): Eric Holt and Anna Carey

Second-highest win rate (28.70): Don Mai and Jeechee Chen

Third-highest in both (50-3, 28.67): Sean Scofield and Frank Lu

Complete rankings will be posted on the website
Computing Win Rates Exactly
Computing Win Rates Exactly

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table
(me,0,0): 5

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
   (who rolls next?, player score, opponent score)

A strategy is a table

   (me,0,0): 5
   (me,0,70): 9
   ...
   (me,96,99): 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

Each state has a chance to win

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table
Each state has a chance to win

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10
(me,99,100+)

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

Each state has a chance to win

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

(me,99,100+): 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0)</td>
<td>5</td>
</tr>
<tr>
<td>(me,0,70)</td>
<td>9</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>(me,96,99)</td>
<td>0</td>
</tr>
<tr>
<td>(me,99,99)</td>
<td>10</td>
</tr>
</tbody>
</table>

Each state has a chance to win

<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(you,100+,99)</td>
<td>0</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

Each state has a chance to win

(you,100+,99) 1
(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
 who rolls next?, player score, opponent score

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0)</td>
<td>5</td>
</tr>
<tr>
<td>(me,0,70)</td>
<td>9</td>
</tr>
<tr>
<td>(me,96,99)</td>
<td>0</td>
</tr>
<tr>
<td>(me,99,99)</td>
<td>10</td>
</tr>
<tr>
<td>(you,98,99)</td>
<td></td>
</tr>
<tr>
<td>(you,100+,99)</td>
<td>1</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

Each state has a chance to win

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

(you,98,99)
(you,100+,99) 1
(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table
Each state has a chance to win

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

(you,98,99) 0
(you,100+,99) 1
(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0)</td>
<td>5</td>
</tr>
<tr>
<td>(me,0,70)</td>
<td>9</td>
</tr>
<tr>
<td>(me,96,99)</td>
<td>0</td>
</tr>
<tr>
<td>(me,99,99)</td>
<td>10</td>
</tr>
<tr>
<td>(you,98,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,100+,99)</td>
<td>1</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
A state in the game:
  (who rolls next?, player score, opponent score)

A strategy is a table

Each state has a chance to win

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

(you,98,99) 0
(you,100+,99) 1
(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

Each state has a chance to win

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

(you,90,99)
...
(you,98,99) 0
(you,100+,99) 1
(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

\[
\begin{align*}
(me,0,0) & : 5 \\
(me,0,70) & : 9 \\
... & \\
(me,96,99) & : 0 \\
... & \\
(me,99,99) & : 10 \\
\end{align*}
\]

Each state has a chance to win

\[
\begin{align*}
(you,90,99) & : 0 \\
... & \\
(you,98,99) & : 0 \\
(you,100+,99) & : 1 \\
(me,99,100+) & : 0 \\
\end{align*}
\]

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Win Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0)</td>
<td>5</td>
</tr>
<tr>
<td>(me,0,70)</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>(me,96,99)</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>(me,99,99)</td>
<td>10</td>
</tr>
</tbody>
</table>

Each state has a chance to win

<table>
<thead>
<tr>
<th>State</th>
<th>Win Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(you,88,99)</td>
<td></td>
</tr>
<tr>
<td>(you,90,99)</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>(you,98,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,100+,99)</td>
<td>1</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0): 5</td>
<td></td>
</tr>
<tr>
<td>(me,0,70): 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>(me,96,99): 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>(me,99,99): 10</td>
<td></td>
</tr>
</tbody>
</table>

Each state has a chance to win

<table>
<thead>
<tr>
<th>State</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>(you,88,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,90,99)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>(you,98,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,100+,99)</td>
<td>1</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0)</td>
<td>5</td>
</tr>
<tr>
<td>(me,0,70)</td>
<td>9</td>
</tr>
<tr>
<td>(me,96,99)</td>
<td>0</td>
</tr>
<tr>
<td>(me,99,99)</td>
<td>10</td>
</tr>
</tbody>
</table>

Each state has a chance to win

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,87,99)</td>
<td></td>
</tr>
<tr>
<td>(you,88,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,90,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,98,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,100+,99)</td>
<td>1</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

Each state has a chance to win
...
(me,87,99)
...
(you,88,99) 0
(you,90,99) 0
...
(you,98,99) 0
(you,100+,99) 1
(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(\text{me},0,0): 5
(\text{me},0,70): 9
...
(\text{me},96,99): 0
...
(\text{me},99,99): 10

Each state has a chance to win

... 
...
(\text{me},87,99)
...
(\text{you},88,99): 0
(\text{you},90,99): 0
...
...
(\text{you},98,99): 0
(\text{you},100+,99): 1
(\text{me},99,100+): 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,0,0)</td>
<td>5</td>
</tr>
<tr>
<td>(me,0,70)</td>
<td>9</td>
</tr>
<tr>
<td>(me,96,99)</td>
<td>0</td>
</tr>
<tr>
<td>(me,99,99)</td>
<td>10</td>
</tr>
</tbody>
</table>

Each state has a chance to win

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(me,87,99)</td>
<td>...</td>
</tr>
<tr>
<td>(you,88,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,90,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,98,99)</td>
<td>0</td>
</tr>
<tr>
<td>(you,100+,99)</td>
<td>1</td>
</tr>
<tr>
<td>(me,99,100+)</td>
<td>0</td>
</tr>
</tbody>
</table>

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9

... (me,96,99): 0

... (me,99,99): 10

Each state has a chance to win

... (me,87,99)

... (you,88,99) 0
(you,90,99) 0

... (you,98,99) 0
(you,100+,99) 1

(me,99,100+) 0

Requires access to both strategies, which must be deterministic
Computing Win Rates Exactly

A state in the game:
(who rolls next?, player score, opponent score)

A strategy is a table

(me,0,0): 5
(me,0,70): 9
...
(me,96,99): 0
...
(me,99,99): 10

Each state has a chance to win

When rolling 2 dice:

1/36 * 1 + 35/36 * 0

Requires access to both strategies, which must be deterministic
Achieving the Perfect Strategy
Achieving the Perfect Strategy

Optimal strategy given an opponent:
Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability
Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!
Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!

- Initial guess: always roll 5
Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!

- Initial guess: always roll 5
- Update to: optimal opponent of current strategy
Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!

- Initial guess: always roll 5
- Update to: optimal opponent of current strategy
- Done when: 0.5 win rate against optimal opponent
Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!

- Initial guess: always roll 5
- Update to: optimal opponent of current strategy
- Done when: 0.5 win rate against optimal opponent

Takes only 16 steps to converge!
Achieving the Perfect Strategy

Optimal strategy given an opponent:

- At each state, compute probability of winning for each allowed number of dice
- Choose the number of dice that maximizes the probability

The perfect strategy: use iterative improvement!

- Initial guess: always roll 5
- Update to: optimal opponent of current strategy
- Done when: 0.5 win rate against optimal opponent

Takes only 16 steps to converge!

Can also compute perfect strategy directly using table
A Function with Evolving Behavior
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
```
Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75
```
Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75

>>> withdraw(25)
```
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75

>>> withdraw(25)
50
```
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
```
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

```python
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
```
Let's model a bank account that has a balance of $100.

Return value: remaining balance

```python
>>> withdraw(25)
75
```

Different return value!

```python
>>> withdraw(25)
50
```

Second withdrawal of the same amount

```python
>>> withdraw(60)
'Insufficient funds'
```

```python
>>> withdraw(15)
35
```
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

Return value: remaining balance

```python
>>> withdraw(25)
75
```

```python
>>> withdraw(25)
50
```

```python
>>> withdraw(60)
'Insufficient funds'
```

```python
>>> withdraw(15)
35
```

Argument: amount to withdraw

Second withdrawal of the same amount

Where's this balance stored?
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

Return value: remaining balance

>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35

>>> withdraw = make_withdraw(100)

Argument: amount to withdraw

Second withdrawal of the same amount

Where's this balance stored?
A Function with Evolving Behavior

Let's model a bank account that has a balance of $100

Return value: remaining balance

>>> withdraw(25)
75

Argument: amount to withdraw

Second withdrawal of the same amount

>>> withdraw(25)
50

where's this balance stored?

>>> withdraw(60)
'Insufficient funds'

Different return value!

Within the function!

>>> withdraw = make_withdraw(100)

>>> withdraw(15)
35
Persistent Local State

Example: http://goo.gl/5LZ6F
Persistent Local State

A function with a parent frame

Example: [http://goo.gl/5LZ6F](http://goo.gl/5LZ6F)
Persistent Local State

A function with a parent frame

The parent contains local state

Example: http://goo.gl/5LZ6F
Persistent Local State

A function with a parent frame

The parent contains local state

Every call changes the balance

Example: [http://goo.gl/5LZ6F](http://goo.gl/5LZ6F)
def percent_difference(x, y):
    difference = abs(x-y)
    return 100 * difference / x

diff = percent_difference(40, 50)
Reminder: Local Assignment

Assignment binds name(s) to value(s) in the first frame of the current environment
Reminder: Local Assignment

Assignment binds name(s) to value(s) in the first frame of the current environment

Example: [http://goo.gl/xkYgN](http://goo.gl/xkYgN)
Reminder: Local Assignment

Execution rule for assignment statements:
1. Evaluate all expressions right of =, from left to right.
2. Bind the names on the left the resulting values in the first frame of the current environment.

Example: [http://goo.gl/xkYgN](http://goo.gl/xkYgN)
Non-Local Assignment
Non-Local Assignment

def make_withdraw(balance):

Non-Local Assignment

def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):
        pass
def make_withdraw(balance):

    """Return a withdraw function with a starting balance."""

def withdraw(amount):

    if amount > balance:
def make_withdraw(balance):
    
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):

        if amount > balance:

            return 'Insufficient funds'
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

def withdraw(amount):
    if amount > balance:
        return 'Insufficient funds'
    balance = balance - amount
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance

    return withdraw
def make_withdraw(balance):
    
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):

        if amount > balance:

            return 'Insufficient funds'

        balance = balance - amount

        return balance

    return withdraw
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):
        nonlocal balance

        if amount > balance:
            return 'Insufficient funds'

        balance = balance - amount

        return balance

    return withdraw
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance

    return withdraw
```python
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""

def withdraw(amount):
    nonlocal balance
    if amount > balance:
        return 'Insufficient funds'
    balance = balance - amount
    return balance

return withdraw
```

- Declare the name "balance" nonlocal
- Re-bind balance where it was bound previously
nonlocal <name>, <name 2>, ...
The Effect of Nonlocal Statements

```
nonlocal <name>, <name 2>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the first non-local frame of the current environment in which that name is bound.
The Effect of Nonlocal Statements

```python
nonlocal <name>, <name 2>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"
nonlocal <name>, <name 2>, ...

Effect: Future assignments to that name change its pre-existing binding in the first non-local frame of the current environment in which that name is bound.

From the Python 3 language reference:
The Effect of Nonlocal Statements

```
nonlocal <name>, <name 2>, ...
```

Effect: Future assignments to that name change its pre-existing binding in the first non-local frame of the current environment in which that name is bound.

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope. Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.
The Effect of Nonlocal Statements

nonlocal <name>, <name 2>, ...

Effect: Future assignments to that name change its pre-existing binding in the first non-local frame of the current environment in which that name is bound.

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope. Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

http://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement
http://www.python.org/dev/peps/pep-3104/
Effects of Assignment Statements
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
x = 2
\]
# Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>• &quot;x&quot; is not bound locally</td>
</tr>
</tbody>
</table>

\[
x = 2
\]
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>- No nonlocal statement</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>- &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
</tbody>
</table>

\[ x = 2 \]
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>

$x = 2$
### Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current env.</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
</tbody>
</table>
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current env.</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
  • "x" is not bound locally                                             | Create a new binding from name "x" to object 2 in the first frame of the current environment.                                      |
| • No nonlocal statement  
  • "x" is bound locally                                                 | Re-bind name "x" to object 2 in the first frame of the current env.                                                                   |
| • nonlocal x  
  • "x" is bound in a non-local frame                                   | Re-bind "x" to 2 in the first non-local frame of the current environment in which it is bound.                                      |
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
• "x" is not bound locally | Create a new binding from name "x" to object 2 in the first frame of the current environment. |
| • No nonlocal statement  
• "x" is bound locally | Re-bind name "x" to object 2 in the first frame of the current env. |
| • nonlocal x  
• "x" is bound in a non-local frame | Re-bind "x" to 2 in the first non-local frame of the current environment in which it is bound. |
| • nonlocal x  
• "x" is not bound in a non-local frame | |

\[
x = 2
\]
# Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement &lt;br/&gt;• &quot;x&quot; is not bound locally</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>• No nonlocal statement &lt;br/&gt;• &quot;x&quot; is bound locally</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current env.</td>
</tr>
<tr>
<td>• nonlocal x &lt;br/&gt;• &quot;x&quot; is bound in a non-local frame</td>
<td>Re-bind &quot;x&quot; to 2 in the first non-local frame of the current environment in which it is bound.</td>
</tr>
<tr>
<td>• nonlocal x &lt;br/&gt;• &quot;x&quot; is not bound in a non-local frame</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
</tbody>
</table>
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No nonlocal statement</td>
<td>Create a new binding from name &quot;x&quot; to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound locally</td>
<td></td>
</tr>
<tr>
<td>• No nonlocal statement</td>
<td>Re-bind name &quot;x&quot; to object 2 in the first frame of the current env.</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound locally</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>Re-bind &quot;x&quot; to 2 in the first non-local frame of the current environment in which it is bound.</td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>• &quot;x&quot; is not bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• nonlocal x</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; is bound in a non-local frame</td>
<td></td>
</tr>
<tr>
<td>• &quot;x&quot; also bound locally</td>
<td></td>
</tr>
</tbody>
</table>

\[ x = 2 \]
## Effects of Assignment Statements

<table>
<thead>
<tr>
<th>Status</th>
<th>Effect</th>
</tr>
</thead>
</table>
| • No nonlocal statement  
  • "x" is not bound locally                                             | Create a new binding from name "x" to object 2 in the first frame of the current environment. |
| • No nonlocal statement  
  • "x" is bound locally                                                   | Re-bind name "x" to object 2 in the first frame of the current env.    |
| • nonlocal x  
  • "x" is bound in a non-local frame                                    | Re-bind "x" to 2 in the first non-local frame of the current environment in which it is bound. |
| • nonlocal x  
  • "x" is not bound in a non-local frame                                | SyntaxError: no binding for nonlocal 'x' found                         |
| • nonlocal x  
  • "x" is bound in a non-local frame  
  • "x" also bound locally                                               | SyntaxError: name 'x' is parameter and nonlocal                        |

\[ x = 2 \]
Python Particulars
Python pre-computes which frame contains each name before executing the body of a function.
Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Therefore, within the body of a function, all instances of a name must refer to the same frame.
Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Therefore, within the body of a function, all instances of a name must refer to the same frame.

```python
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

wd = make_withdraw(20)
wd(5)
```
Python pre-computes which frame contains each name before executing the body of a function.

Therefore, within the body of a function, all instances of a name must refer to the same frame.

```python
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

wd = make_withdraw(20)
wd(5)
```
Python Particulars

Python pre-computes which frame contains each name before executing the body of a function.

Therefore, within the body of a function, all instances of a name must refer to the same frame.

```python
def make_withdraw(balance):
    def withdraw(amount):
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw

wd = make_withdraw(20)
wd(5)
```

UnboundLocalError: local variable 'balance' referenced before assignment
Mutable Values and Persistent State
Mutable Values and Persistent State

Mutable values can be changed without a nonlocal statement.
Mutable values and Persistent State

Mutable values can be changed without a nonlocal statement.

Example: [link](http://goo.gl/cEpmz)
Mutable values can be changed without a nonlocal statement.

Example: [http://goo.gl/cEpmz](http://goo.gl/cEpmz)
Mutable Values and Persistent State

Mutable values can be changed without a nonlocal statement.

Example: [http://goo.gl/cEpmz](http://goo.gl/cEpmz)
Creating Two Withdraw Functions

Example:

```python
1  def make_withdraw(balance):
2      def withdraw(amount):
3          nonlocal balance
4          if amount > balance:
5              return 'Insufficient funds'
6          balance = balance - amount
7          return balance
8      return withdraw

9  wd = make_withdraw(100)
10  wd2 = make_withdraw(100)
11  wd(25)
12  wd2(15)
```

Example: [http://goo.gl/glTyB](http://goo.gl/glTyB)
Multiple References to a Withdraw Function

Example: http://goo.gl/X2qG9
The Benefits of Non-Local Assignment
The Benefits of Non-Local Assignment

Ability to maintain some state that is local to a function, but evolves over successive calls to that function.
The Benefits of Non-Local Assignment

10 Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

10 The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.
The Benefits of Non-Local Assignment

10 Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

10 The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

10 An abstraction of a bank account that manages its own internal state.
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

- An abstraction of a bank account that manages its own internal state.

| Weasley Account | $10 |
The Benefits of Non-Local Assignment

- Ability to maintain some state that is local to a function, but evolves over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is inaccessible to the rest of the program.

- An abstraction of a bank account that manages its own internal state.

<table>
<thead>
<tr>
<th>Weasley Account</th>
<th>Potter Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10</td>
<td>$1,000,000</td>
</tr>
</tbody>
</table>
Referential Transparency
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3)
\]
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3) \\
\text{mul}(\text{add}(2, 24), 3)
\]
Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3) \\
\text{mul}(\text{add}(2, 24), 3) \\
\text{mul}(26, 3)
\]
Referential Transparency

Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3)
\]

\[
\text{mul}(\text{add}(2, 24), 3)
\]

\[
\text{mul}(26, 3)
\]

Mutation is a \textit{side effect} (like printing)
Expresssions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul}(\text{add}(2, \text{mul}(4, 6)), 3) \\
\text{mul}(\text{add}(2, 24), 3) \\
\text{mul}(26, 3)
\]

Mutation is a *side effect* (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.
Referential Transparency

Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

mul(add(2, mul(4, 6)), 3)

mul(add(2, 24), 3)

mul(26, 3)

Mutation is a side effect (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.
Referential Transparency

Expressions are referentially transparent if substituting an expression with its value does not change the meaning of a program.

\[
\text{mul(add(2, mul(4, 6)), 3)}
\]

\[
\text{mul(add(2, 24 ), 3)}
\]

\[
\text{mul( 26 , 3)}
\]

Mutation is a *side effect* (like printing)

Side effects violate the condition of referential transparency because they do more than just return a value; they change the state of the computer.