Practical Guidance: Choosing Names

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

```python
boolean turn_is_over  # disc 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))
```  

Use names for meaningful parts of compound expressions

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

```python
disc_term = sqrt(square(b) - 4 * a * c)
x = (-b + disc_term) / (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

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

We used `itertools.combinations` to determine the set of matches
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.

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

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

When rolling 2 dice:

A Function with Evolving Behavior

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

Argument: amount to withdraw

Return value: remaining balance

Different return value!

Where’s this balance stored?

Within the function!

Reminder: Local Assignment

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

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.
Non-Local Assignment

```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
```

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.

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.

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 'x' to object 2 in the first frame of the current environment.</td>
</tr>
<tr>
<td>No nonlocal statement</td>
<td>Re-bind name 'x' to object 2 in the first frame of the current environment in which it is bound.</td>
</tr>
<tr>
<td>'x' is bound locally</td>
<td>Re-bind 'x' to 2 in the first non-local frame of the current environment in which it is bound.</td>
</tr>
<tr>
<td>'x' is bound in a non-local frame</td>
<td>SyntaxError: no binding for nonlocal 'x' found</td>
</tr>
<tr>
<td>'x' is not bound in a non-local frame</td>
<td>SyntaxError: name 'x' is parameter and nonlocal</td>
</tr>
</tbody>
</table>

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.

Mutable Values and Persistent State

Mutable values can be changed without a nonlocal statement.

Creating Two Withdraw Functions

Example: [http://goo.gl/cEpmz](http://goo.gl/cEpmz)

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

Example: [http://goo.gl/X2qG9](http://goo.gl/X2qG9)

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></th>
<th>Weasley Account</th>
<th>Potter Account</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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.

- `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.