Discussion ??? ????

SPECIAL REVIEW EDITION
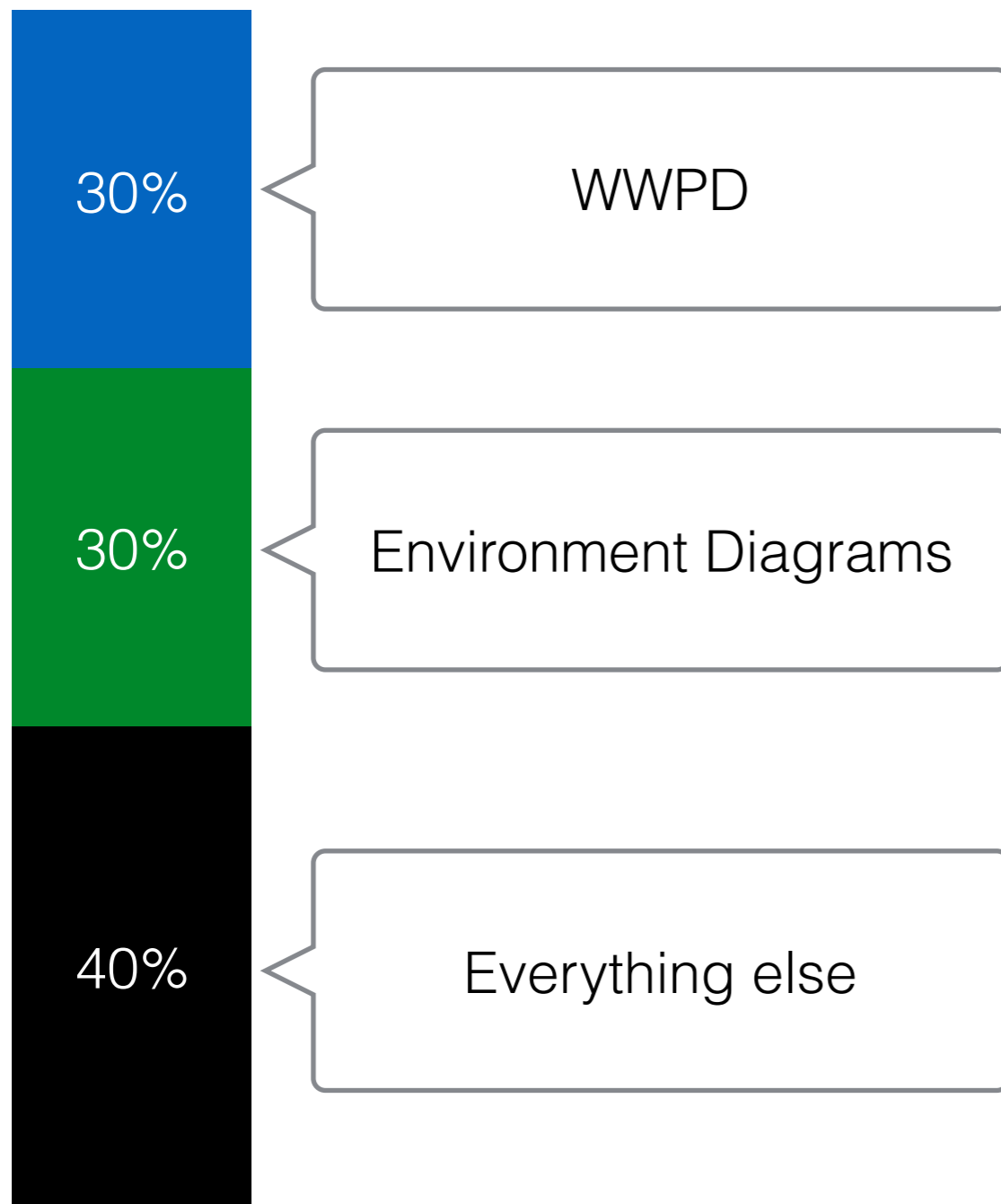
TA: **Jerry Chen**
Email: **jerry.c@berkeley.edu**
TA Website: **jerryjrchen.com/cs61a**
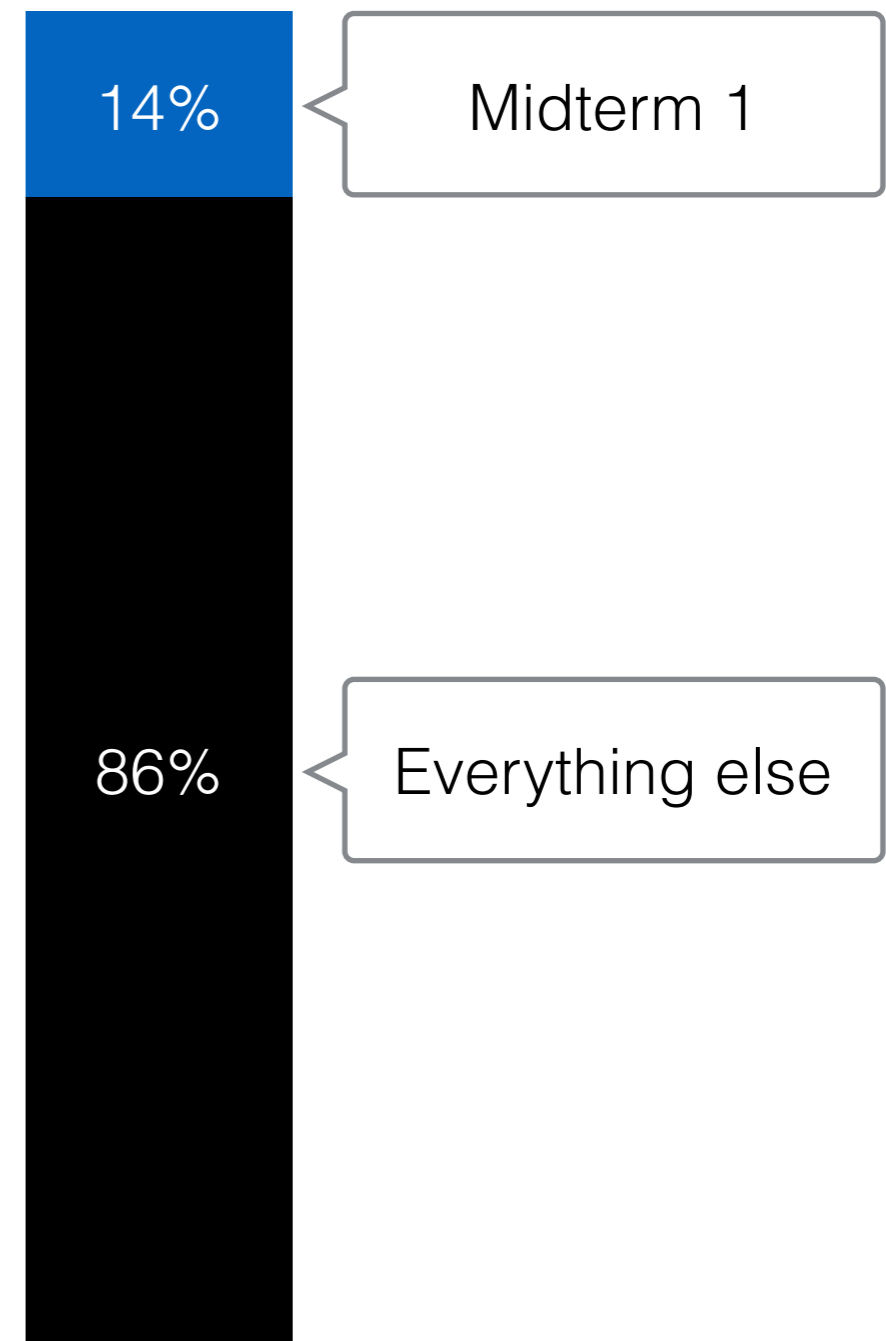
Agenda

1. Points

2. Currying

3. WWPD

4. Environment Diagrams

5. Rest of the Exam

Some Perspective

30% WWPD

30% Environment Diagrams

40% Everything else

From MT1, Fall 2015

14% Midterm 1

86% Everything else

Total Grade

What Would Python Display?

Basic Facts

- Evaluated **in order** in the Python **interpreter**, not run from a **.py** file

- **If it errors, include everything before the error**

- Various other rules are on your exam

```python
def square(x):
    return x * x

def argentina(n):
    print(n)
    if n > 0:
        return lambda k: k(n+1)
    else:
        return 1 / n

def germany(n):
    if n > 1:
        print('hallo')
    if argentina(n-2) >= 0:
        print('bye')
    return argentina(n+2)
```

Expression	Interactive Output
5*5	25
print(5)	5
1/0	ERROR
print(1, print(2))	
argentina(0)	

What Would Python Display?

Strategies

- **Practice!** A lot.

- Draw (hasty) environment diagrams

```python
def square(x):
    return x * x

def argentina(n):
    print(n)
    if n > 0:
        return lambda k: k(n+1)
    else:
        return 1 / n

def germany(n):
    if n > 1:
        print('hallo')
    if argentina(n-2) >= 0:
        print('bye')
    return argentina(n+2)
```

Expression	Interactive Output
5*5	25
print(5)	5
1/0	ERROR
print(1, print(2))	
argentina(0)	

Currying

Currying is like onions:

~~They both make you cry~~ They have layers!

Lambdas and Currying

```
(lambda x, y: x + y * y)(4, 5)
```

Lambda definition

Lambda call

Result (after currying):

```
(lambda x = 4, y = 5: x + y * y)
```

More Currying

```
(lambda m, a: lambda: lambda p: m + a + p)(2, 4)()(4)
```

Lambda definition

Lambda call

```
m = 2, a = 4
>(lambda: lambda p: m + a + p)()(4)
>>(lambda p: m + a + p)(4)
p = 4
>>>m + a + p
```

Practice

```python
def haskell_dream():
    arg = (lambda: lambda s: s + 2)
    return (lambda y: y()(10))(arg)
```

```
>>> haskell_dream()

???

Answer: 12
```

Review

The Return Value

```
(lambda y: y()(10))(lambda: lambda s: s + 2)
```

....

```
(lambda y: y()(10))
```
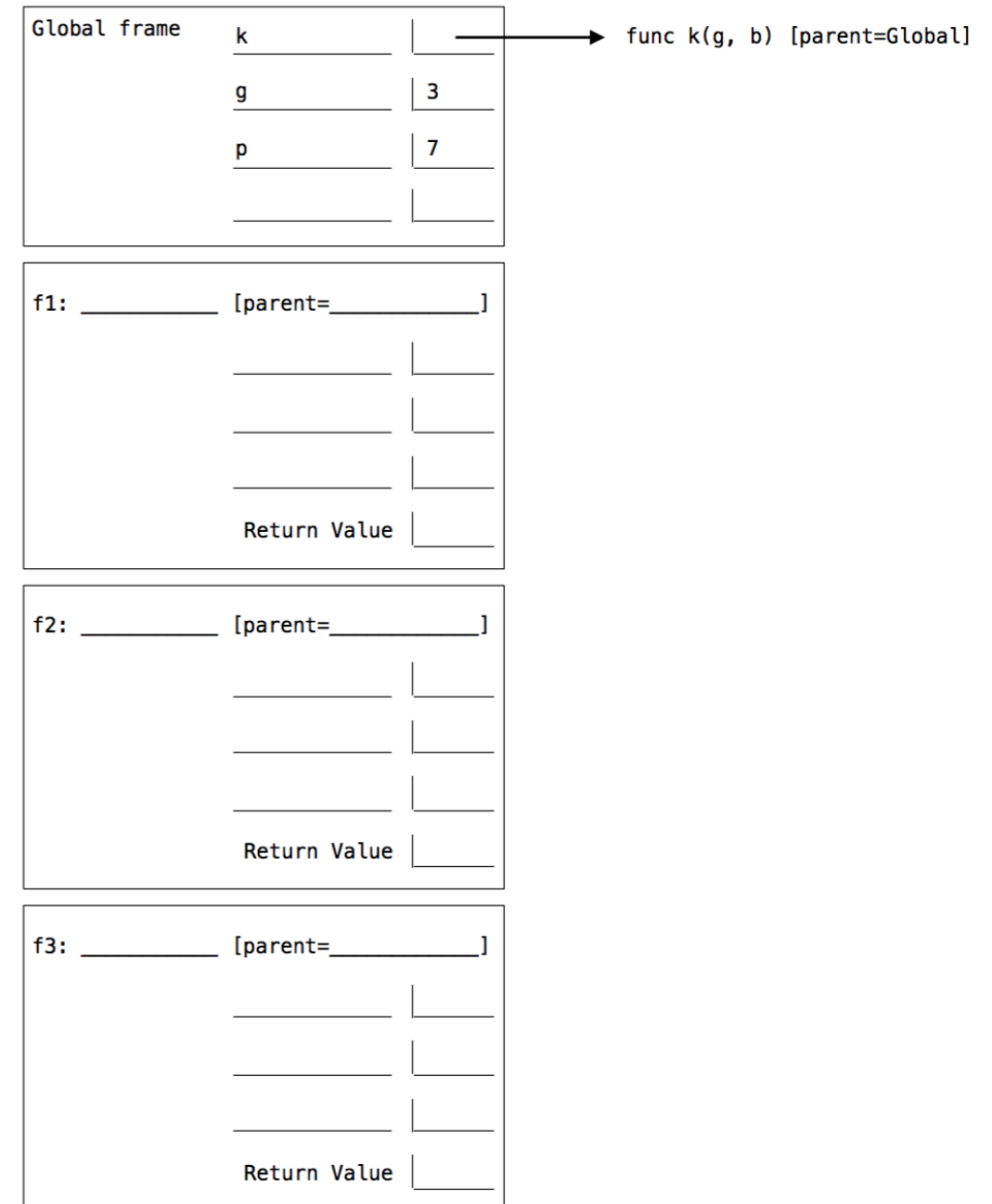
```
y          = lambda: lambda s: s + 2

y()        = lambda s: s + 2

y()(10)  = 12
```

Environment Diagrams

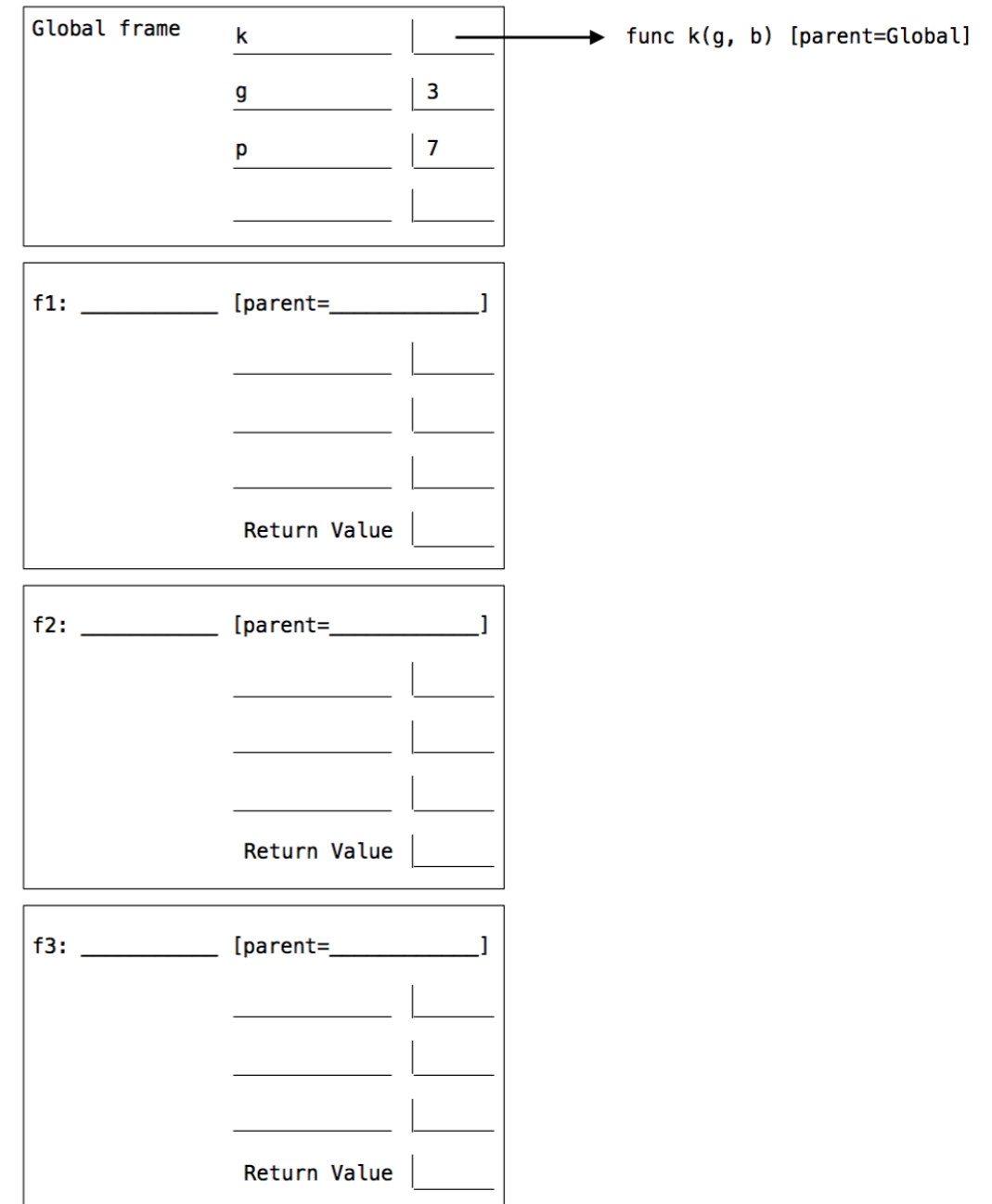**Basic Facts**

- If you practice, it's probably the closest thing to free points

- **If it errors**, you (likely) did something wrong!

```
Global frame        k   _____ |  |————————→  func k(g, b) [parent=Global]
                    g        | 3 |
                    p        | 7 |
                    _____ |  |

f1: _____   [parent=_____]
                    _____ |  |
                    _____ |  |
                    _____ |  |
          Return Value |  |

f2: _____   [parent=_____]
                    _____ |  |
                    _____ |  |
                    _____ |  |
          Return Value |  |

f3: _____   [parent=_____]
                    _____ |  |
                    _____ |  |
                    _____ |  |
          Return Value |  |
```

Environment Diagrams

Strategies

- **Practice!** A lot.

- Don't duplicate names

- Don't forget to label parent frames!

- Fill in **ALL return values**

```
Global frame        k  _____  |___|  ─────────►  func k(g, b) [parent=Global]
                    g  _____  | 3 |
                    p  _____  | 7 |
                       _____  |___|

f1: _____  [parent=_____]
    _____  |___|
    _____  |___|
    _____  |___|
    Return Value      |___|

f2: _____  [parent=_____]
    _____  |___|
    _____  |___|
    _____  |___|
    Return Value      |___|

f3: _____  [parent=_____]
    _____  |___|
    _____  |___|
    _____  |___|
    Return Value      |___|
```

Practice

```python
def haskell_nightmare():
    carrot = 10
    potato = 20
    def onions(onions):
        def carrot(onions, carrot):
            return lambda onions: onions(carrot)
        return carrot
    cook = lambda onions: onions + carrot + potato
    return onions(carrot)(onions, potato)(cook)
```

```
>>> haskell_nightmare()
???
```

Review

```
>>> haskell_nightmare()
50
```

Environment Diagram: http://goo.gl/7eQHKM

Coding Questions

Expect lots of recursion

~Three possibilities:

- **Skeleton code**

- **Cross-outs**

- **Free write** (unlikely?)

Coding Questions

Skeleton Code

- Almost definitely gonna happen

- Iteration and recursion are fair game

- **Look for clues** in the skeleton

- **Test** after you think you are done!

(b) **(5 pt)** Fill in the blanks of the following functions defined together in the same file. **Assume that all arguments to all of these functions are positive integers that do not contain any zero digits.** For example, 1001 contains zero digits (not allowed), but 1221 does not (allowed). You may assume that reverse is correct when implementing remove.

```python
def combine(left, right):
    """Return all of LEFT's digits followed by all of RIGHT's digits."""
    factor = 1
    while factor <= right:
        factor = factor * 10
    return left * factor + right

def reverse(n):
    """Return the digits of N in reverse.

    >>> reverse(122543)
    345221
    """

    if n < 10:

        return n

    else:

        return combine(_____ , _____)

def remove(n, digit):
    """Return all digits of N that are not DIGIT, for DIGIT less than 10.

    >>> remove(243132, 3)
    2412
    >>> remove(243132, 2)
    4313
    >>> remove(remove(243132, 1), 2)
    433
    """

    removed = 0

    while n != 0:


        _____, _____ = _____, _____

        if _____:

            removed = _____

    return reverse(removed)
```

Coding Questions

Cross-outs

- First, get a **general feel of everything**

- Cross out anything that obviously stands out

- **Test** after you think you are done!

```python
class STree(Tree):
    """A smart tree that knows its depth and whether or not it is
    balanced.

    >>> s = STree(6, STree(2, STree(1)), STree(7))
    >>> s.depth
    3
    >>> s.is_balanced
    True
    >>> s.left.right = STree(4, STree(3), STree(5))
    >>> s.depth
    4
    >>> s.is_balanced
    False
    """

    def __init__(self, entry, left=None, right=None):

        Tree.__init__(entry, left, right)

        Tree.__init__(self, entry, left, right)

        self.entry = entry

        self.left = left

        self.right = right

        self.depth = depth(self)

        self.is_balanced = is_balanced(self)

        self.depth = depth

        self.is_balanced = is_balanced

    @property

    def depth(self):

        return depth(self)

    @property

    def is_balanced(self):

        return is_balanced(self)
```

Coding Questions

Free write

- Probably **not** gonna happen

- Treat like a lab/homework problem

- **Test** after you think you are done!

(b) **(4 pt)** The function `equal` takes two differentiable single-argument functions `f` and `g` and returns an `x` for which `f(x)` is equal to `g(x)`. Implement the support function `equal_update` that completes the implementation. You may use `derivative` above, along with `newton_update` from your study guide. You **cannot** use any assignment (=), conditional (`if`), `for`, or `while` statements.

```
def equal(f, g):
    """Return an x for which f(x) == g(x).

    >>> def cube(x):
    ...     return x * x * x
    ...
    >>> def plus_six(x):
    ...     return x + 6
    ...
    >>> equal(cube, plus_six)
    2.0
    """
    def close(x):
        return approx_eq(f(x), g(x))
    return improve(equal_update(f, g), close)

def equal_update(f, g):
    """Return an update function that completes the implementation of equal."""
```

Write It

1. (Don't) panic!

2. Read the doctests and instructions very carefully

3. Before looking at blanks, plan out your own solution

    - More often than not, you can fit your solution to the blanks!

Test It

1. Check **domain** and **range**

   - Do inputs/outputs make sense?

2. Check the **margins**

   - Did I loop one too many/few times?

3. Check the **syntax**

   - Correct number of parens, commas, etc?

4. Check it all again

Efficient Workflow

Shortcut: Assume everything has a **"nice" solution**
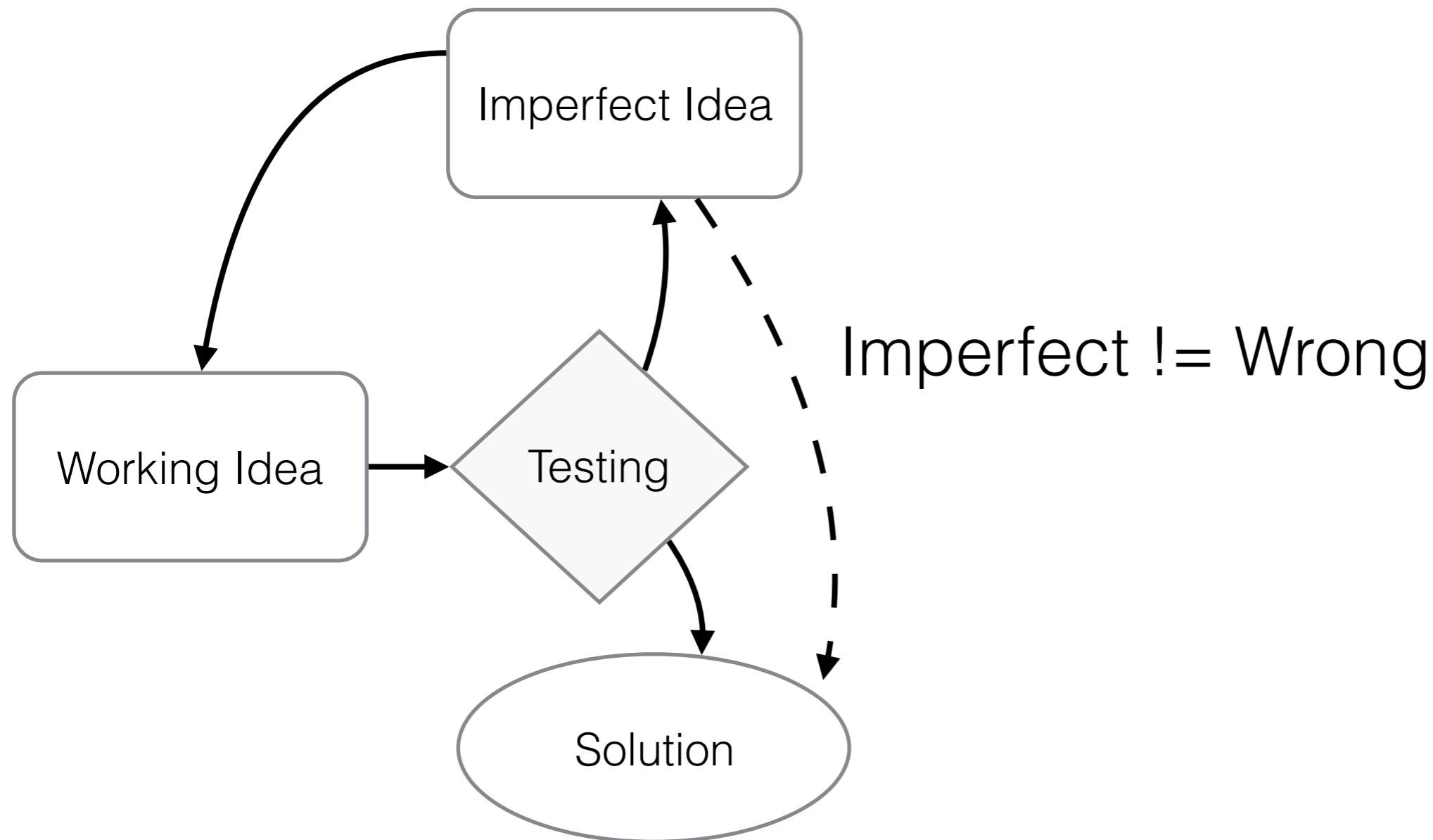
Recursion

Rules of Recursion

1. Define a **base case**

   - If it's not immediately obvious, pick one anyways

2. Break the problem into **smaller but similar subproblems**

3. Actually **solve the problem**

Mutual Recursion

```python
def mouse(n):
    if n >= 10:
        squeak = n // 100
        n = frog(squeak) + n % 10
    return n


def frog(croak):
    if croak == 0:
        return 1
    else:
        return 10 * mouse(croak + 1)
```

```
>>> mouse(21023508479)
???
314159
```

From Andrew Huang's 61A Wiki: https://www.ocf.berkeley.edu/~shidi/cs61a/wiki/Guides#Miscellaneous_2

Review

```python
def mouse(n):
    if n >= 10:
        squeak = n // 100
        n = frog(squeak) + n % 10
    return n
```

"Starting" n gets updated

Get the last digit

All but last two! (what's lost?)

```python
def frog(croak):
    if croak == 0:
        return 1
    else:
        return 10 * mouse(croak + 1)
```

Shift up a digit

Increments last digit

```
>>> mouse(21023508479)
```

Cheat Sheets

"Less is more"

What to write?

1. **Nothing** — study guide is quite comprehensive!

2. **Code examples**

3. Rules — env diagrams, recursion fundamentals

Final Logistics

- Eat a meal or a light snack beforehand!

- I hope you had a good night's sleep

- Arrive on time or early, **not Berkeley time**

- Also, it would be great if you could remember discussion time/TA :)

Final Thoughts

Go back and look at the "perspective" slide

I will repeat this before (and after) every exam —

"One test will not define who you are and whether or not you'll be a successful computer scientist."