Quick Point

• Testing and readability are about the very human sides of software development (no proofs!)

• Suggestions here are based on experience in big corps

• Readability and testing are a HUGE HUGE HUGE part of getting things to work in the Real World!
Why care about readability?

• SWE is as much a maintenance problem as an authoring problem

• In the real world, most code is written once, read many times

• Don’t waste your coworkers’ time making them puzzle out what you were trying to do

• Don’t waste your own time, 6 months on!
Big Companies think this is important!

- Companies have released their style guides:
  - Google
  - Facebook
  - Others
Use positive names

```python
list_out_of_order = False
for i in range(len(list)) - 1:
    if list(i+1) < list[i]:
        list_out_of_order = True

list_in_order = True
for i in range(len(list)) - 1:
    if list(i+1) < list[i]:
        list_in_order = False
```
# At pixel (row, col), compute sum
# of surrounding 8 pixels

pixel_sum = 0
for delta_row in [(-1, 0, 1)]:
    for delta_col in [(-0, 0, 1)]:
        if not (delta_row == 0 and delta_col==0)
            and row+delta_row < height
            and row+delta_row >= 0
            and col+delta_col < width
            and col+delta_col >= 0:
                pixel_sum +=
                    pixel(row+delta_row, col+delta_col)
At pixel (row, col), compute sum of surrounding 8 pixel values

```
pixel_sum = 0
for delta_row in [(-1, 0, 1)]:
    for delta_col in [(-0, 0, 1)]:
        if delta_row == 0 and delta_col==0:
            continue
        if row+delta_row >= height: continue
        if row+delta_row <  0     : continue
        if col+delta_col >= width : continue
        if col+delta_col <  0     : continue

        pixel_sum +=
            pixel(row+delta_row, col+delta_col)
```
Use descriptive names

for i in range(vertices):
    for j in range(neighbors[i]):
        print("There is an edge \%d->\%d" %(i,j))

for vertex in range(vertices):
    for neighbor in range(neighbors[vertex]):
        print("There is an edge \%d->\%d" %(vertex,neighbor))
Single source of truth

```python
infile = open("/tmp/experiment3/input.txt")
processed_data = process(f)

outfile = open("/tmp/experiment3/output.txt")
write_data_to_file(processed_data, outfile)

path = "/tmp/experiment3"
infile = open(os.join(path,"input.txt"))
processed_data = process(f)

outfile = open(os.join(path,"output.txt"))
write_data_to_file(processed_data, outfile)
```
def get_prime_factors(n):
    # if prime, done!
    if is_prime(n): return [n]

    still_to_factor = [n]
    prime_factors = []
    while len(still_to_factor) == 0:
        next_factor = still_to_factor.pop()
        if is_prime(next_factor):
            prime_factors.append(next_factor)
            continue

        factors = getTwoFactors(next_factor)
        still_to_factor.push(factors[0])
        still_to_factor.push(factors[1])
Enforce assumptions with assertions

def slope(x1, x2, y1, y2):
    return (y2-y1)/(x2-x1)

def slope(x1, x2, y1, y2):
    assert(abs(x2-x1) > 1e-8)
    return (y2-y1)/(x2-x1)
Testing

- Why do we write tests for software?
  - As a check when authoring it
  - To avoid future bugs ("regressions")
  - To help refactoring
  - As a form of documentation
Unit Testing

- “Unit” = “Smallest testable part”

- Most widely used kind of test
Useful testing-related tools in the real world

- Unit testing framework
  - e.g., pyunit
- Continuous integration frameworks
  - e.g., travis
- Code review frameworks
  - e.g., gerrit
Good properties of test?

• Fast
• Deterministic
• Correct
• Readable
• Hermetic
Test-Driven development

- A software practice where you write the tests first
- No strong opinion from me
Test-Driven bug fixing

- When you find a bug, write a test that would catch it before you fix it
- There is a strong opinion from me: do this!
A few testing patterns
Test the “contract” not the implementation

\[
L = [1,2,3,4,3,2,1]
\]
\[
\text{assert}(\text{min\_idx}(L) == 0)
\]

\[
L = [1,2,3,4,3,2,1]
\]
\[
\text{assert}(\text{min\_idx}(L) \text{ in } [0,6])
\]

\[
L = [1,2,3,4,3,2,1]
\]
\[
\text{assert}(L[\text{min\_idx}(L)] == 1)
\]
def HashTableTests():
    HT = HashTable()

    #test insert
    HT.insert(1, 'a')
    assert(HT.get(1) == 'a')

    #test overwriting
    HT.insert(1, 'b')
    assert(HT.get(1) == 'b')

    #test size
    assert(HT.size() == 1)

    #test remove
    HT.remove(1)
    assert(HT.size() == 0)
    assert(HT.get(1) == None)

    # test multiple inserts…
Separate tests

def HashTableTestInsert(): ...
def HashTableTestSize(): ...
def HashTableTestRemove(): ...
def HashTableTestOverwrite(): ...
Test weird/edge cases!

- Empty lists, empty tuples
- Null pointers (Nones in python)
- Numbers that are positive, negative, zero, infinity, -infinity, NAN, very large, very small
- Inputs somehow wrong
Don’t just repeat the implementation in the test!

```python
def TestSorted():
    L = [1, 3, 2, 9, 4, 8, 2]
    L_sorted = MySort(L)
    while len(L) > 0:
        assert(L_sorted[0] == min(L))
        L = [x for x in L if x != min(L)]

```
Where to start

- One “happy path” test
  - Medium-complex example
- Tests for exceptional/edge cases
- Additional tests if you find bugs later
Let’s work an example

• Toy problem: given a list of 2-D points, find the closest pair

• I have some code and even a test! It passes, woohoo!
Example, part 2

- New strategy:
  - Sort points by x coordinate
  - Ignore points whose x distance from active point is beyond lowest distance so far
  - Much faster (?)!

Best distance so far

Only need to look in this region!