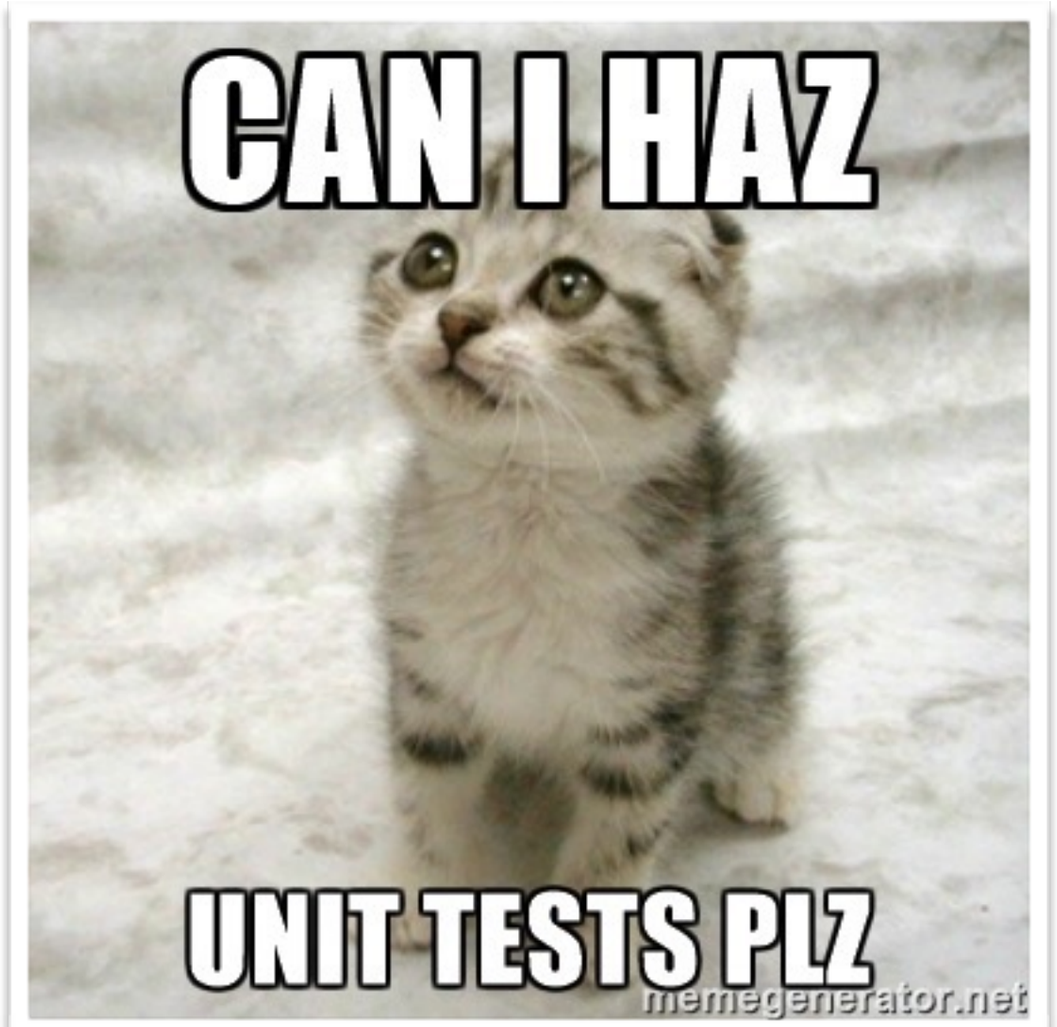
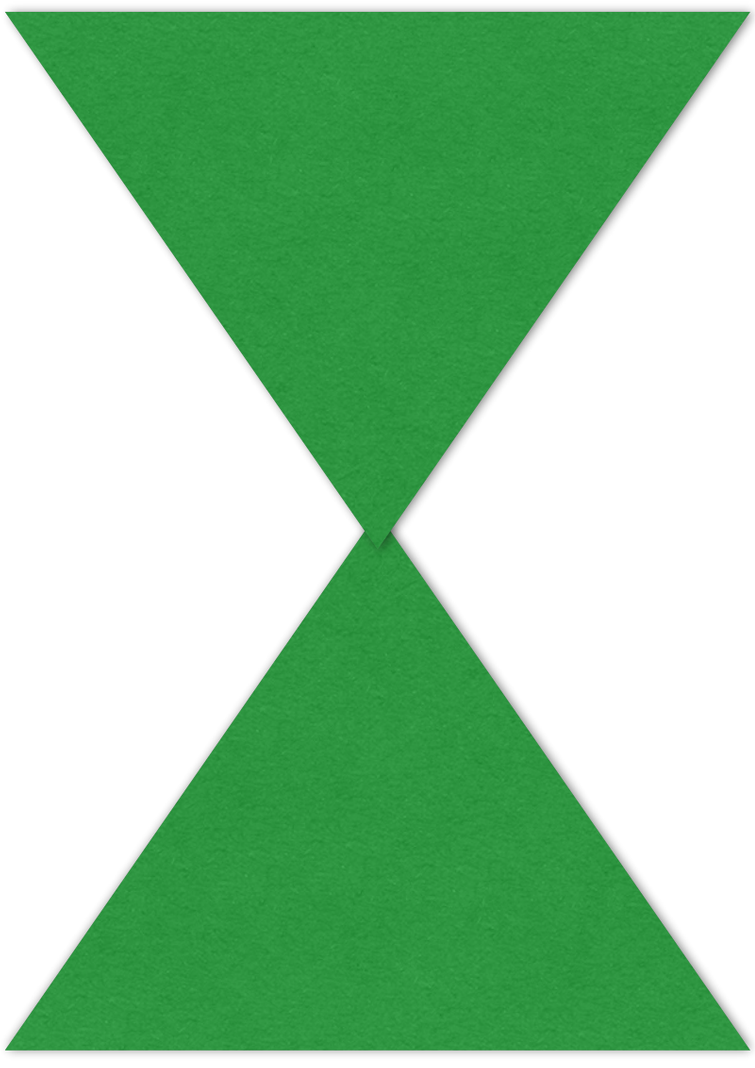
Testing and Object-Oriented Design

Matt Dickenson

March 16, 2016


CAN I HAZ
UNIT TESTS PLZ
memegenerator.net

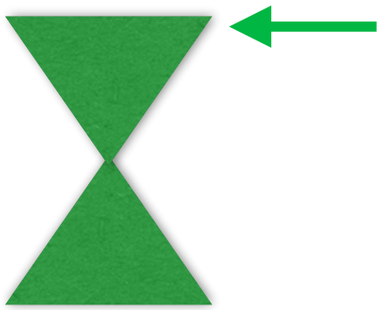
Outline



Overall recommendations

Specific examples

General guidelines



Why Test?

working examples

regressions

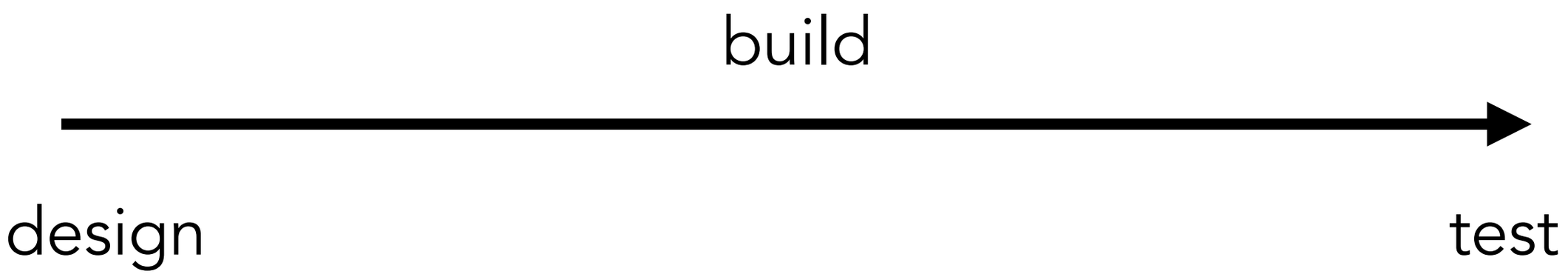
confidence

edge cases

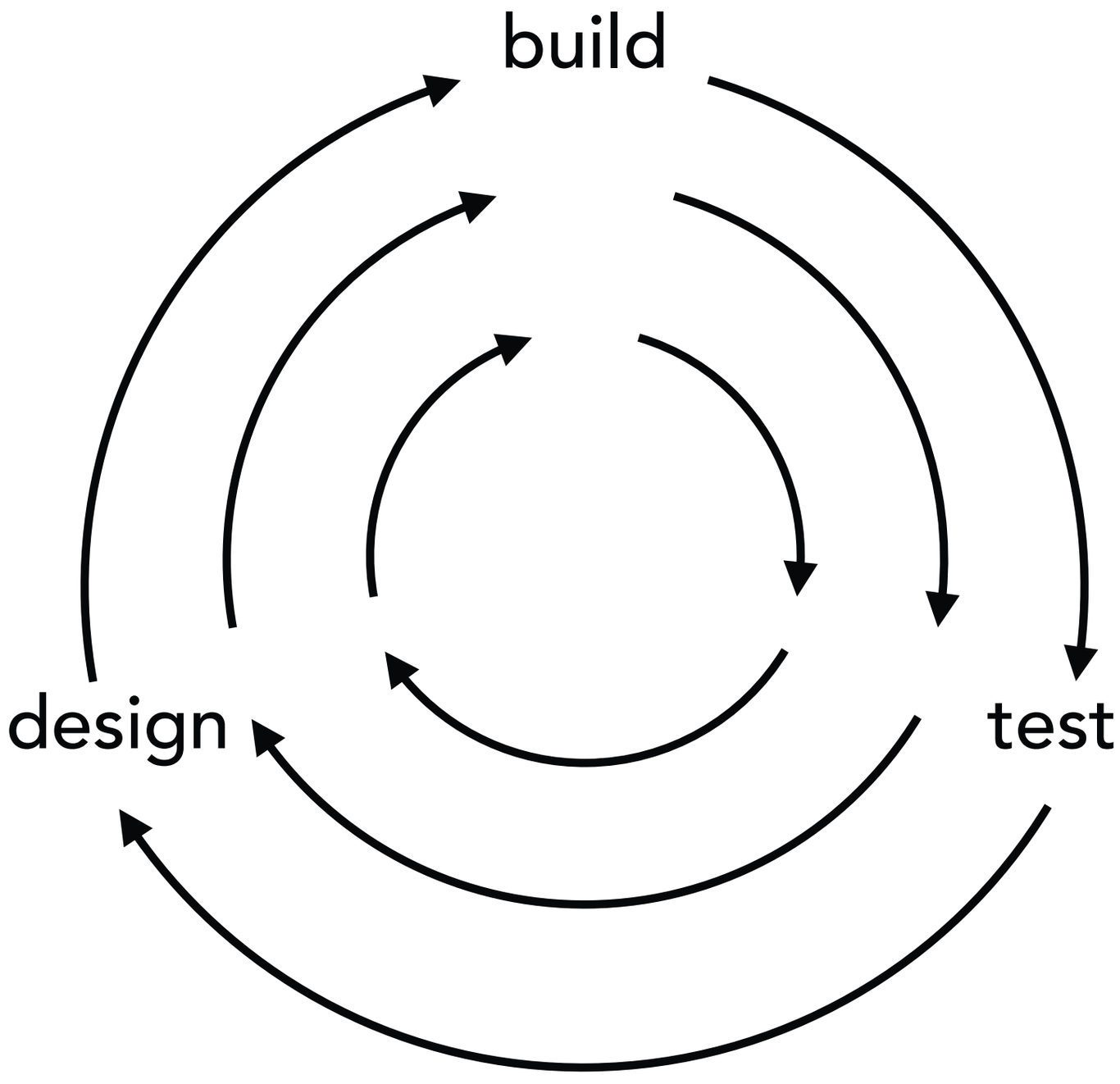
easier refactoring

inform design

Model 1: Linear



Model 2: Cyclical



Design Happens Every Day

determining class boundaries

organizing files

naming variables

choosing function parameters

structuring projects

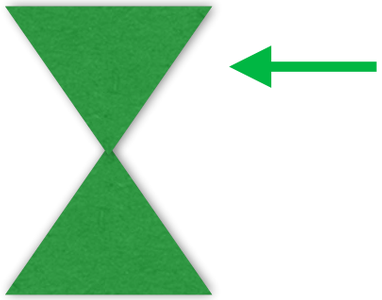
establishing service interfaces

Questions for Design

🌍 Does this convey knowledge about the world?

🙀 Would this decision surprise another developer?

🗓️ Will I remember in 6 months why I did this?



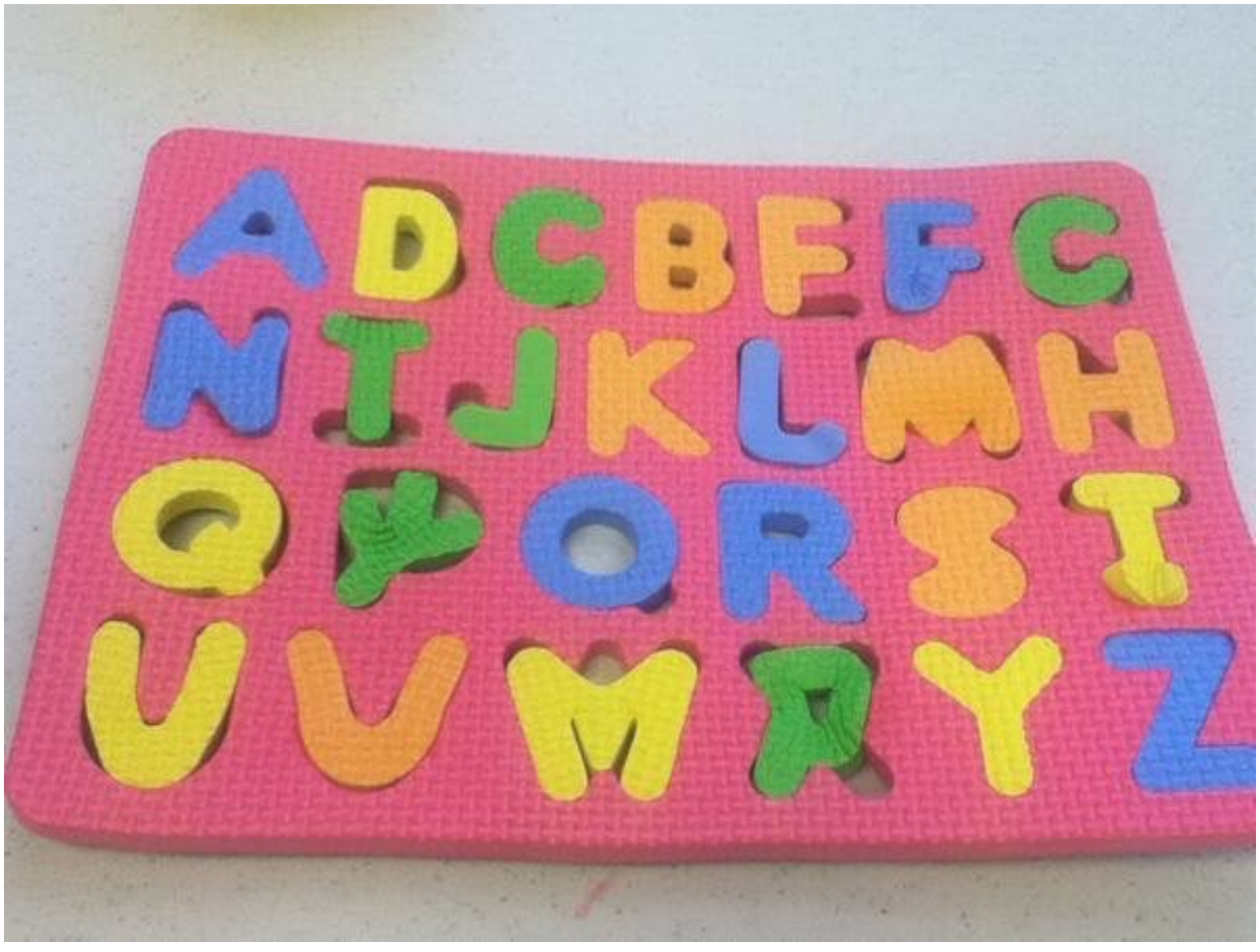
Tests are an essential part of your development toolkit.

especially when working in a dynamically typed, interpreted language

(but not exclusively)

Do Static Types Make Tests Unnecessary?

```
public class CollectionTest {
    public static int add(int num1, int num2){
        return num1 * num2;
    }
}

int expected = 4;
int result = add(2, 2);
assertEquals(result, expected);
```



"All tests passing."

Types of Testing

✔️ Unit

📲 Integration

👍🏼 Acceptance

📈 Functional

🔥 Failure

Anatomy of a Test

Given

When

Then

✓ Unit Test

Given input `1` and `2`

When `sum()` is called

Then output should be `3`

📲 Integration Test

Given a client and server

When the client parses the server payload

Then the app state should be updated

"Two unit tests, no integration tests."

🔥 Failure Test

Given a system deployed in SJC and DCA

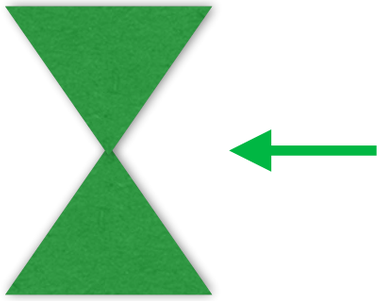
When SJC goes down

Then the system should remain reachable

🔥 Failure Test

Do	Don’t
Inject dependencies	Create flaky tests
Check edge cases	Leave side effects
Use mocks & stubs	Mock object under test


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION



Refactoring for Testability

```
x = time.time()
random.seed(x)
y = random.random()
sys.stdout.write(str(y))
```

```
x = time.time()
random.seed(x)
y = random.random()
sys.stdout.write(str(y))
```

What does this code do?

How would you test it?

How would you refactor it?

```
x = time.time()
random.seed(x)
y = random.random()
sys.stdout.write(str(y))
```

uninformative names

can't be imported

What does this code do?

How would you test it?

How would you refactor it?

```
def main():
    now = time.time()
    random.seed(now)
    rand = random.random()
    sys.stdout.write(str(rand))

if __name__ == '__main__':
    main()
```

```
def main():
    now = time.time()
    random.seed(now)
    rand = random.random()
    sys.stdout.write(str(rand))

if __name__ == '__main__':
    main()
```

better names

can now be imported

```
def main(args):
    now = time.time()
    random.seed(now)
    rand = random.random()
    with open(args.output, 'w') as f:
        f.write(str(rand))

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('--output',
      required=True, help="Output file")
    args = parser.parse_args()

    main(args)
```

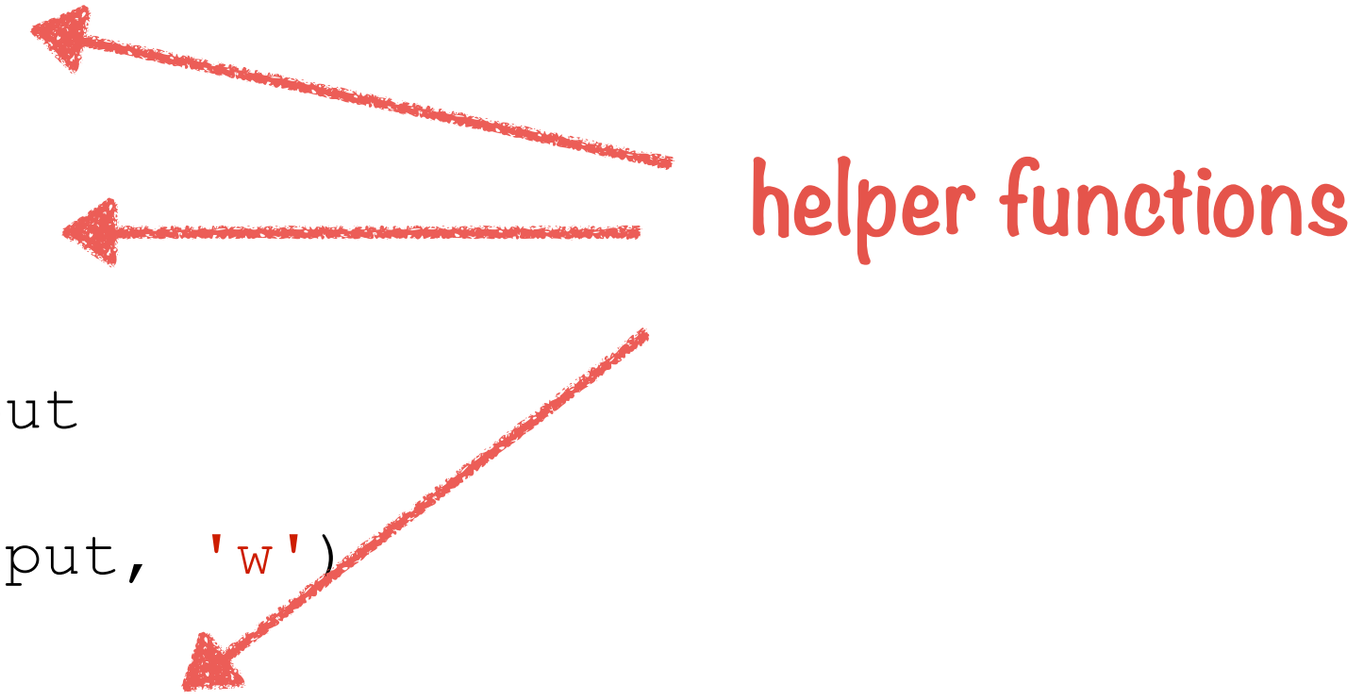
```
def main(args):
    now = time.time()
    random.seed(now)
    rand = random.random()
    with open(args.output, 'w') as f:
        f.write(str(rand))

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument('--output',
      required=True, help="Output file")
    args = parser.parse_args()

    main(args)
```

can't control this in tests

CLI arguments

```
def get_seed():
    return time.time()

def get_writer(output):
    if output is None:
        writer = sys.stdout
    else:
        writer = open(output, 'w')
    return writer

def write_random_to_file(output=None, seed=None):
    seed = get_seed() if seed is None else seed
    random.seed(seed)
    writer = get_writer(output)
    rand = random.random()
    with writer as w:
        w.write(str(rand))

if __name__ == '__main__':
    …
    parser.add_argument('--seed', required=False, help="Random
seed")
    args = parser.parse_args()

    write_random_to_file(args.output, args.seed)
```

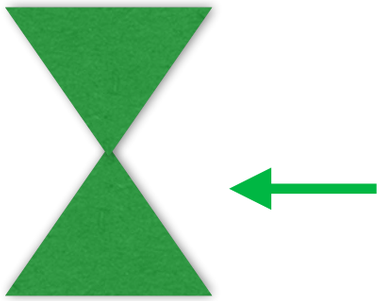
```
def get_seed():
    return time.time()

def get_writer(output):
    if output is None:
        writer = sys.stdout
    else:
        writer = open(output, 'w')
    return writer

def write_random_to_file(output=None, seed=None):
    seed = get_seed() if seed is None else seed
    random.seed(seed)
    writer = get_writer(output)
    rand = random.random()
    with writer as w:
        w.write(str(rand))

if __name__ == '__main__':
    …
    parser.add_argument('--seed', required=False, help="Random
seed")
    args = parser.parse_args()

    write_random_to_file(args.output, args.seed)
```



```
def get_seed():
    return time.time()

def get_writer(output):
    if output is None:
        writer = sys.stdout
    else:
        writer = open(output, 'w')
    return writer

def write_random_to_file(randomizer, writer):
    rand = randomizer.random()
    with writer as w:
        w.write(str(rand))

if __name__ == '__main__':
    …
    # Set up default writer and randomizer
    writer = get_writer(args.output)
    randomizer = random
    seed = get_seed() if args.seed is None else args.seed
    randomizer.seed(seed)

    write_random_to_file(randomizer, writer)
```

```
def get_seed():
    return time.time()

def get_writer(output):
    if output is None:
        writer = sys.stdout
    else:
        writer = open(output, 'w')
    return writer

def write_random_to_file(randomizer, writer):
    rand = randomizer.random()
    with writer as w:
        w.write(str(rand))

if __name__ == '__main__':
    …
    # Set up default writer and randomizer
    writer = get_writer(args.output)
    randomizer = random
    seed = get_seed() if args.seed is None else args.seed
    randomizer.seed(seed)

    write_random_to_file(randomizer, writer)
```

dependency injection



Object-Oriented Design

Single-Responsibility

```
class InputHandler:
  def __init__(input_data):
    fmt = self.check_format(
      input_data
    )
    if fmt == 'CSV':
      ...
    else if fmt == 'JSON':
      ...
    else:
      raise TypeError
```

```
class CsvHandler:
  ...

class JsonHandler:
  ...
```

SOLID

Open vs. Closed

```
if x%3 == 0 and x%5 == 0:
  print("FizzBuzz")
elif x%5 == 0:
  print("Buzz")
elif x%3 == 0:
  print("Fizz")
else:
  print(x)
```

```
class FizzBuzz():
  def __init__(cls, rules):
    cls.rules = rules


def say(self,x):
  for rule in self.rules:
    if rule.is_handle(x):
      return rule.say(x)
```

SOLID

Liskov Substitution

```
class Rectangle(Shape):
  def __init__(width, height):
    self.width = width
    self.height = height
    ...

class Square(Rectangle):
  ...
```

```
class Rectangle(Shape):
  def __init__(width,
height):
    self.shape =
      Shape(width, height)


class Square(Shape):
  def __init__(width):
    height = width
    self.shape =
      Shape(width, height)
```

SO**L**ID

Interface Segregation

```
class ProcessHandler:
  def step1(start_data, case):
    if case == 1:
        ...
    else:
      ...

  def step2(step1_output):
    if case == 1:
        ...
    else:
      ...

  def step3(step2_output):
    if case == 1:
        ...
    else:
      ...
```

```
class BaseCaseHandler:
  def process(data):
    ...

class ExceptionHandler:
  def process(data):
    ...
```

SOLID

Dependency Injection

```
def split_data_sets(
    data,
    percent_train
  ):
  for datum in data:
    if random.random() <
      percent_train:
      ...
    else:
      ...
```

```
def split_data_sets(
    data,
    randomizer
  ):
  for datum in data:
    if randomizer.set ==
      'TRAIN':
      ...
    else:
      ...
```

SOLI**D**

SOLID is good for describing systems, but what about prescribing day-to-day improvements?

STABLE Design

S

T

A

B

L

E

STABLE Design

Smell your code

T

A

B

L

E

STABLE Design

Smell your code

T

A

B

L

E

STABLE Design

Smell your code

Tiny problems first

A

B

L

E

STABLE Design

Smell your code

Tiny problems first

Augment Your Tests

B

L

E

STABLE Design

Smell your code

Tiny problems first

Augment your tests

Back up (sunk costs)

L

E

STABLE Design

Smell your code

Tiny problems first

Augment your tests

Back up (sunk costs)

Leave it better than you found it

E

STABLE Design

Smell your code

Tiny problems first

Augment your tests

Back up (sunk costs)

Leave it better than you found it

Expect good reasons

References

- github.com/CaitieM20/TheVerificationOfDistributedSystem
- techblog.netflix.com/2011/07/netflix-simian-army.html
- ai.stanford.edu/~zayd/why-is-machine-learning-hard.html
- speakerdeck.com/sarahmei/is-your-code-too-solid-with-transcript