

61A Lecture 7

Monday, September 10

Hog Contest Rules

- Two people submit one entry; Max of one entry per person
- The score for an entry is the sum of win rates against every other entry.
- All strategies must be deterministic, pure functions of the current player scores! *Non-deterministic strategies will be disqualified.*
- To enter: *submit projlcontest* with a file *hog.py* that defines a *final_strategy* function by **Monday 9/24 @ 11:59pm**
- All winning entries will receive 2 points of extra credit
- The real prize: honor and glory

Fall 2011 Winners

Keegan Mann,
Yan Duan & Ziming Li,
Brian Prike & Zhenghao Qian,
Parker Schuh & Robert Chatham

Choosing Names

Practical
guidance

Names typically *don't* matter for correctness
but
they matter tremendously for legibility

From:	To:
boolean	turn_is_over
d	dice
play_helper	take_turn

```
>>> from operator import mul
>>> def square(let):
    return mul(let, let)
```



Not stylish

Which Values Deserve a Name

Practical
guidance

Repeated compound expressions:

```
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
```

However, not every value needs a name (demo)



Meaningful parts of complex expressions:

```
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```
d = sqrt(square(b) - 4 * a * c)
x = (-b + d) / (2 * a)
```

Test-Driven Development

Practical
guidance

Write the test of a function before you write the function

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

Your tests can help identify tricky edge cases

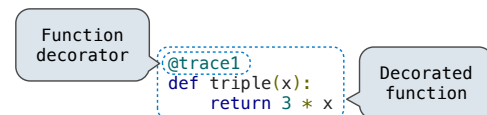
Develop incrementally and test each piece before moving on

You can't depend upon code that hasn't been tested

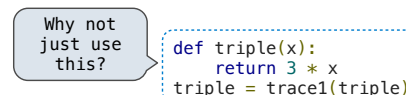
Run your old tests again after you make new changes

Function Decorators

(demo)



is identical to



Functional Abstractions

```
def square(x):           def sum_squares(x, y):
    return mul(x, x)      return square(x) + square(y)
```

What does `sum_squares` need to know about `square`?

- `square` takes one argument. **Yes**
- `square` has the intrinsic name `square`. **No**
- `square` computes the square of a number. **Yes**
- `square` computes the square by calling `mul`. **No**

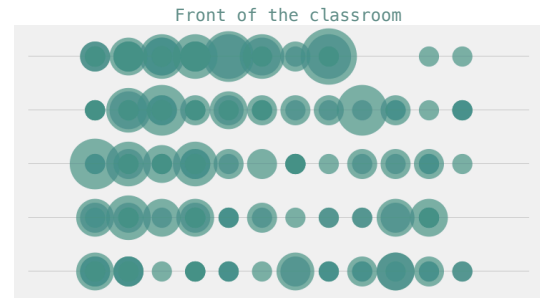
```
def square(x):           def square(x):
    return pow(x, 2)     return mul(x, x-1) + x
```

If the name "square" were bound to a built-in function, `sum_squares` would still work identically

7

Data

Student seating preferences at MIT



<http://www.skyrill.com/seatinghabits/>

8

Objects

- Representations of information
- Data and behavior, bundled together to create...

Abstractions

- Objects represent properties, interactions, & processes
- Object-oriented programming:
 - A metaphor for organizing large programs
 - Special syntax for implementing classic ideas

(Demo)

9

Python Objects

In Python, every value is an object.

- All objects have attributes
- A lot of data manipulation happens through methods
- Functions do one thing; objects do many related things

The next four weeks:

- Use built-in objects to introduce classic ideas
- Create our own objects using the built-in object system
- Implement an object system using built-in objects

10

Native Data Types

In Python, every object has a type.

```
>>> type(today)
<class 'datetime.date'>
```

Properties of native data types:

1. There are primitive expressions that evaluate to native objects of these types.
2. There are built-in functions, operators, and methods to manipulate these objects.

11

Numeric Data Types

Numeric types in Python:

```
>>> type(2)
<class 'int'>
```

Represents integers exactly

```
>>> type(1.5)
<class 'float'>
```

Represents real numbers approximately

```
>>> type(1+1j)
<class 'complex'>
```

(demo)

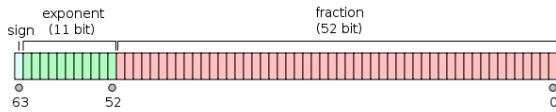
12

Working with Real Numbers

Bonus
Material

Care must be taken when computing with real numbers!
(Demo)

Representing real numbers:



$1/3 = 0011\ 1111\ 1101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101$

False in a Boolean contexts:

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

http://en.wikipedia.org/wiki/File:IEEE_754_Double_Floating_Point_Format.svg

13

Working with Real Numbers

Bonus
Material

```
>>> def approx_eq_1(x, y, tolerance=1e-18):
    return abs(x - y) <= tolerance
```

```
>>> def approx_eq_2(x, y, tolerance=1e-7):
    return abs(x - y) <= abs(x) * tolerance
```

```
>>> def approx_eq(x, y):
    if x == y:
        return True
    return approx_eq_1(x, y) or approx_eq_2(x, y)
```

```
>>> def near(x, f, g):
    return approx_eq(f(x), g(x))
```

or approx_eq_2(y,x)

14

Moral of the Story

Life was better when numbers were just numbers!

Having to know the details of an abstraction:

- Makes programming harder and more knowledge-intensive
- Creates opportunities to make mistakes
- Introduces dependencies that prevent future changes

Coming Soon: Data Abstraction

15