Function Examples
Announcements
Hog Contest Rules

• Up to two people submit one entry;
  Max of one entry per person
• Slight rule changes
• Your score is the number of entries
  against which you win more than
  50.00001% of the time
• Strategies are time-limited
• All strategies must be deterministic,
  pure functions of the players' scores
• All winning entries will receive
  extra credit
• The real prize: honor and glory
• See website for detailed rules

Fall 2011 Winners
Kaylee Mann
Yan Duan & Ziming Li
Brian Prike & Zhenghao Qian
Parker Schuh & Robert Chatham

Fall 2012 Winners
Chenyang Yuan
Joseph Hui

Fall 2013 Winners
Paul Bramsen
Sam Kumar & Kangsik Lee
Kevin Chen

Fall 2014 Winners
Alan Tong & Elaine Zhao
Zhenyang Zhang
Adam Robert Villaflor & Joany Gao
Zhen Qin & Dian Chen
Zizheng Tai & Yihe Li

cs61a.org/proj/hog_contest
Hog Contest Winners

**Spring 2015 Winners**
Sinho Chewi & Alexander Nguyen Tran
Zhaoxi Li
Stella Tao and Yao Ge

**Fall 2015 Winners**
Micah Carroll & Vasilis Oikonomou
Matthew Wu
Anthony Yeung and Alexander Dai

**Spring 2016 Winners**
Michael McDonald and Tianrui Chen
Andrei Kassiantchouk
Benjamin Krieges

**Spring 2017 Winners**
Cindy Jin and Sunjoon Lee
Anny Patino and Christian Vasquez
Asana Choudhury and Jenna Wen
Michelle Lee and Nicholas Chew

**Fall 2017 Winners**
Alex Yu and Tanmay Khattar
James Li
Justin Yokota

**Spring 2018 Winners**
Eric James Michaud
Ziyu Dong
Xuhui Zhou

**Fall 2018 Winners**
Your name could be here FOREVER!
Abstraction
**Functional Abstractions**

```
def square(x):
    return mul(x, x)
```

```
def sum_squares(x, y):
    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):
    return pow(x, 2)
```

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

If the name “square” were bound to a built-in function, `sum_squares` would still work identically.
Choosing Names

Names typically don’t matter for correctness

\textbf{but}

they matter a lot for composition

<table>
<thead>
<tr>
<th>From:</th>
<th>To:</th>
</tr>
</thead>
<tbody>
<tr>
<td>true_false</td>
<td>rolled_a_one</td>
</tr>
<tr>
<td>d</td>
<td>dice</td>
</tr>
<tr>
<td>helper</td>
<td>take_turn</td>
</tr>
<tr>
<td>my_int</td>
<td>num_rolls</td>
</tr>
<tr>
<td>l, l, l,</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>k, i, m</td>
</tr>
</tbody>
</table>

Names should convey the meaning or purpose of the values to which they are bound.

The type of value bound to the name is best documented in a function's docstring.

Function names typically convey their effect (\texttt{print}), their behavior (\texttt{triple}), or the value returned (\texttt{abs}).
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

```python
x1 = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```python
discriminant = square(b) - 4 * a * c
x1 = (-b + sqrt(discriminant)) / (2 * a)
```

More Naming Tips

- Names can be long if they help document your code:
  ```python
  average_age = average(age, students)
  ```
  is preferable to
  ```python
  # Compute average age of students
  aa = avg(a, st)
  ```

- Names can be short if they represent generic quantities: counts, arbitrary functions, arguments to mathematical operations, etc.
  - `n, k, i` - Usually integers
  - `x, y, z` - Usually real numbers
  - `f, g, h` - Usually functions
Testing
Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

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.

Bonus idea: Run your code interactively.

Don't be afraid to experiment with a function after you write it.

Interactive sessions can become doctests. Just copy and paste.
Currying
Function Currying

```python
def make_adder(n):
    return lambda k: n + k

>>> make_adder(2)(3)
5
>>> add(2, 3)
5
```

There's a general relationship between these functions

Curry: Transform a multi-argument function into a single-argument, higher-order function
Decorators
Function Decorators

(Demo)

@trace1
def triple(x):
    return 3 * x

is identical to

def triple(x):
    return 3 * x
triple = trace1(triple)

is identical to

Why not just use this?
Review
What Would Python Display?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul

def square(x):
    return mul(x, x)

A function that takes any argument and returns a function that returns that arg

```def delay(arg):
    print('delayed')
    def g():
        return arg
    return g

Names in nested def statements can refer to their enclosing scope

```
def g():
    return arg

This expression | Evaluates to | Interactive Output
---|---|---
5 | 5 | 5
print(5) | None | 5 None
print(print(5)) | None | 5 None
delay(delay)()(6)() | 6 | delayed 6
delay(delay)(print)()(4)) | None | delayed 4 None
def horse(mask):
    horse = mask
def mask(horse):
    return horse
return horse(mask)

mask = lambda horse: horse(2)
horse(mask)