## 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

Your name could be here FOREVER!

**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**

## Abstraction

## 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.

## Choosing Names

Names typically don't matter for correctness

***but***

they matter a lot for composition

| **From:** | **To:** |
|---|---|
| true_false | rolled_a_one |
| d | dice |
| helper | take_turn |
| my_int | num_rolls |
| l, I, O | k, i, m |

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 (**print**), their behavior (**triple**), or the value returned (**abs**).

## Which Values Deserve a Name

**Reasons to add a new name**

*Repeated compound expressions:*

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

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

PRACTICAL GUIDELINES

*Meaningful parts of complex expressions:*

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

```
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:

```
average_age = average(age, students)
```

is preferable to

```
# 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.*

(Demo)

## Currying

## Function Currying

```
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

(Demo)

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

## Decorators

## Function Decorators

(Demo)

Function decorator

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

Decorated function

*is identical to*

Why not just use this?

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

## Review

## What Would Python Display?

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

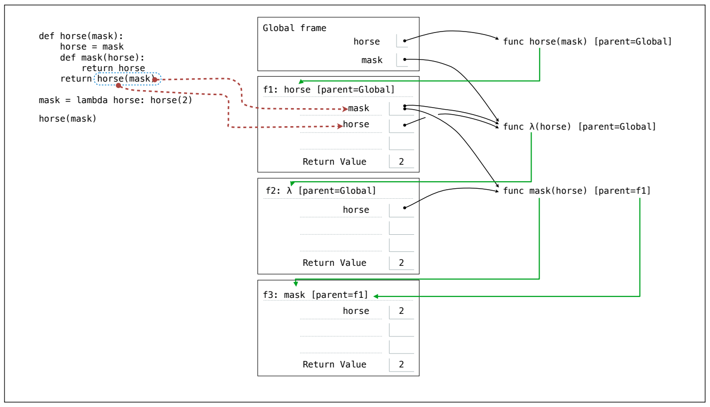
```
from operator import add, mul
def square(x):
    return mul(x, x)
```

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

| This expression | Evaluates to | Interactive Output |
|---|---|---|
| 5 | 5 | 5 |
| print(5) | None | 5 |
| print(print(5)) None | None | 5<br>None |
| delay(delay)()(6)() | 6 | delayed<br>delayed<br>6 |
| print(delay(print)()(4)) | None | delayed<br>4<br>None |



```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```