Function Examples

Hog Contest Rules
- Up to two people submit one entry;
- Max of one entry per person
- Small 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

Hog Contest Winners

Fall 2011 Winners
Kaylee Mann
Yan Duan & Zining Li
Brian Price & Zhenghai Qian
Parker Schuh & Robert Chatham

Fall 2012 Winners
Chenyang Yuan
Joseph Mui

Fall 2013 Winners
Paul Bransen
San Kumar & Kangsik Lee
Kevin Chen

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

Fall 2015 Winners
Sinho Chewi & Alexander Nguyen Tran
Zhaoxi Li
Stella Tao and Yan Ge

Fall 2016 Winners
Micah Carroll & Vasilis Oikonomou
Matthew Wu
Anthony Young and Alexander Dai

Spring 2017 Winners
Cindy Jin and Sunjun Lee
Amy Patino and Christian Vasquez
Avane Choudhury and Jenna Xion
Michelle Lee and Nicholas Chow

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

Spring 2018 Winners
Eric James Michaud
Ziyu Dong

Spring 2019 Winners
Cindy Jin and Sunjun Lee
Stella Tao and Yao Ge

Spring 2020 Winners
Chenyang Yuan
Joseph Hui

Choosing Names

Names typically don’t matter for correctness but they matter a lot for composition

| From: true_false d helper my_int 1, 0 | To: rolled_a_one dice take_turn num_rolls k, i, x |

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).

Abstraction

Functional Abstractions

```
def square(x):
    return x**2
```

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

```
def hypotenuse(a, b):
    return hypotenuse = sqrt(square(a) + square(b))
```

```
def is_even(n):
    return n % 2 == 0
```

```
def is_prime(n):
    return all(n % i for i in range(2, int(sqrt(n)) + 1))
```

Choosing Names

Names typically don’t matter for correctness but they matter a lot for composition

| From: true_false d helper my_int 1, 0 | To: rolled_a_one dice take_turn num_rolls k, i, x |

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 += sqrt(square(a) + square(b))
- Meaningful parts of complex expressions:
  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)
- Names can be short if they represent generic quantities: counts, arguments to mathematical operations, etc.
  n, k, i – Usually integers
  x, y, z – Usually real numbers
  f, g, h – Usually functions

Abstraction

```
def square(x):
    return x**2
```

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

```
def hypotenuse(a, b):
    return hypotenuse = sqrt(square(a) + square(b))
```

```
def is_even(n):
    return n % 2 == 0
```

```
def is_prime(n):
    return all(n % i for i in range(2, int(sqrt(n)) + 1))
```

Choosing Names

Names typically don’t matter for correctness but they matter a lot for composition

| From: true_false d helper my_int 1, 0 | To: rolled_a_one dice take_turn num_rolls k, i, x |

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).
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.

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

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

Function Decorators

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

is identical to

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

Why not just use this?

Function decorator

Decorated function

Interactive Output

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)

Names in nested def statements can refer to their enclosing scope
def horse(mask):
    horse = mask
    return horse

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