Functional Abstraction
Announcements

Sign up for tutorials [here](#)!

Hog, Homework 1, and Lab 1 have been released.

Hog checkpoint is due this Friday, 6/30. The entire project is due next Thursday 7/6, you can submit 1 day early, Wednesday 7/5, for a bonus point.

If you see a 0/1 for lab, don’t panic, it takes time to update. Please see [this Ed post](#)

Regular OH schedule this week

Instructor OH starts this week

[Advising OH](#) starts this week

Sections are finalized on 6/30. No section switches after this point
Decorators
(Demo)

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

is identical to

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

Why not just use this?
Return
Return Statements

A return statement completes the evaluation of a call expression and provides its value:

\[ f(x) \text{ for user-defined function } f : \text{ switch to a new environment; execute } f's \text{ body} \]

**return** statement within \( f \): switch back to the previous environment; \( f(x) \) now has a value

Only one return statement is ever executed while executing the body of a function

```python
def end(n, d):
    """Print the final digits of N in reverse order until D is found.
    Assume N is non-negative"

    >>> end(34567, 5)
    7
    6
    5
    while n > 0:
        last, n = n % 10, n // 10
        print(last)
    if d == last:
        return None  # (Demo)
```
Abstraction
Functional Abstractions

• Square takes one argument.
• Square has the intrinsic name square.
• Square computes the square of a number.
• Square computes the square by calling mul.

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

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*

ey 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>play_helper</td>
<td>take_turn</td>
</tr>
<tr>
<td>my_int</td>
<td>num_rolls</td>
</tr>
<tr>
<td>l, I, O</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 (print), their behavior (triple), or the value returned (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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n, k, i</td>
<td>Usually integers</td>
</tr>
<tr>
<td>x, y, z</td>
<td>Usually real numbers</td>
</tr>
<tr>
<td>f, g, h</td>
<td>Usually functions</td>
</tr>
</tbody>
</table>
Break
Errors & Tracebacks
## Taxonomy of Errors

<table>
<thead>
<tr>
<th>Errors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax Errors</td>
<td>Detected by the Python interpreter (or editor) before the program executes</td>
</tr>
<tr>
<td>Runtime Errors</td>
<td>Detected by the Python interpreter while the program executes</td>
</tr>
<tr>
<td>Logic &amp; Behavior Errors</td>
<td>Not detected by the Python interpreter; what tests are for</td>
</tr>
</tbody>
</table>

*(Demo)*
Common Bugs

NameError
Hello != hello != helo

SyntaxError
Missing parenthesis, Missing close quotes (EOL)

Logic & Behavior Errors
= vs ==
Infinite loops
Off by 1 errors

```python
i = 0
while i < 10:
    print(i)
```
Common Bugs

IndentationError
Improper indentation

```python
def f(x):
    print(x)
    return(x)
```

TypeError
Invalid types for an operator
Using non-function objects in a function call
Passing an incorrect number of arguments to a function

IndexError
Index a sequence with a number that exceeds the size of the sequence (preview to next week)
Debugging

(Demo)
Debugging Strategies and Techniques

Traceback messages

Running Doctests + writing your own tests

Using print statements (DEBUG: for okpy)

Interactive debugging

PythonTutor

Assert statements
Implementing Functions
Implementing a Function

def remove(n, digit):
    """Return all digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10."

>>> remove(231, 3)  # 1 1
21
>>> remove(243132, 2)  # + 20 + 30
4313

kept, digits = 0, 0

while ________________________________:
    n, last = n // 10, n % 10
    if _______________________________
        kept = _______________________
    digits = _____________________

return _______________________________

Read the description

Verify the examples & pick a simple one

Read the template

Implement without the template, then change your implementation to match the template. **OR**
If the template is helpful, use it.

Annotate names with values from your chosen example

Write code to compute the result

Did you really return the right thing?

Check your solution with the other examples
Implementing a Function

```python
def remove(n, digit):
    """Return all digits of non-negative N that are not DIGIT, for some non-negative DIGIT less than 10."

>>> remove(231, 3)
21
>>> remove(243132, 2)
4313

kept, digits = 0, 0

while n > 0:
    n, last = n // 10, n % 10
    if last != digit:
        kept = kept/10 + last
    digits = digits + 1
    return round(kept * 10 ** (digits-1))
```

Read the description
Verify the examples & pick a simple one
Read the template
Implement without the template, then change your implementation to match the template.
OR
If the template is helpful, use it.

Annotate names with values from your chosen example

Write code to compute the result

Did you really return the right thing?

Check your solution with the other examples