Announcements

- Quiz today!
  - Only worth two points, so don’t worry!

- Hog project
  - Get started early!
  - If you still don’t have a partner (and want one), find one on Piazza
    - Use existing post; don’t make a new one
The Art of the Function
The Art of the Function

- Give each function exactly one job
The Art of the Function

- Give each function exactly one job

- Don’t repeat yourself (DRY).
The Art of the Function

- Give each function exactly one job

- Don’t reapeat yourself (DRY).

- Don’t reapeat yourself (DRY).
The Art of the Function

- Give each function exactly one job
- Don’t reapeat yourself (DRY).
- Don’t reapeat yourself (DRY).
- Define functions generally
Generalizing Patterns with Parameters
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:

- Square
- Circle
- Hexagon
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:
Regular geometric shapes relate length and area.
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:
- Square: \( r \)
- Circle: \( r \)
- Hexagon: \( r \)

Area:
- Square: \( r^2 \)
Regular geometric shapes relate length and area.

**Shape:**
- Square: $r$
- Circle: $r$
- Hexagon: $r$

**Area:**
- Square: $r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: unknown
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:
- Square: $r$
- Circle: $r$
- Hexagon: $r$

Area:
- Square: $r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:

Area: \[ \begin{align*}
\text{Square:} & \quad 1 \cdot r^2 \\
\text{Circle:} & \quad \pi \cdot r^2 \\
\text{Hexagon:} & \quad \frac{3\sqrt{3}}{2} \cdot r^2
\end{align*} \]
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:

- **Square:** $r$ (Area: $r^2$)
- **Circle:** $r$ (Area: $\pi \cdot r^2$)
- **Hexagon:** $r$ (Area: $\frac{3\sqrt{3}}{2} \cdot r^2$)
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:
- Square: \( r \)
- Circle: \( r \)
- Hexagon: \( r \)

Area:
- Square: \( 1 \cdot r^2 \)
- Circle: \( \pi \cdot r^2 \)
- Hexagon: \( \frac{3\sqrt{3}}{2} \cdot r^2 \)
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:

Area:

Square: $r^2$

Circle: $\pi r^2$

Hexagon: $\frac{3\sqrt{3}}{2} r^2$
Generalizing Patterns with Parameters

Regular geometric shapes relate length and area.

Shape:

- Square: $r$
- Circle: $r$
- Hexagon: $r$

Area:

- Square: $1 \cdot r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$

Finding common structure allows for shared implementation.
Generalizing Over Computational Processes
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[ \sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15 \]

\[ \sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225 \]

\[ \sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04 \]
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[ \sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15 \]

\[ \sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225 \]

\[ \sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04 \]
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[ \sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15 \]

\[ \sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225 \]

\[ \sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04 \]
Generalizing Over Computational Processes

The common structure among functions may itself be a computational process, rather than a number.

\[
\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225
\]

\[
\sum_{k=1}^{5} \frac{8}{(4k - 3) \cdot (4k - 1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04
\]
Functions as Arguments
Functions as Arguments

Function values can be passed as arguments
Functions as Arguments

Function values can be passed as arguments

def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence.\n    >>> summation(5, cube)
    225\n    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."""
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
Functions as Arguments

Function values can be passed as arguments

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    # Sum the first n terms of a sequence.
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
    # Sum the first n terms of a sequence.
```

Function of a single argument (not called term)

A formal parameter that will be bound to a function

The function bound to term gets called here
Function values can be passed as arguments

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
```

The cube function is passed as an argument value

The function bound to term gets called here

Function of a single argument (not called term)

A formal parameter that will be bound to a function
Functions as Arguments

Function values can be passed as arguments

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

>>> summation(5, cube)
225
"""
```

A formal parameter that will be bound to a function

```
0 + 1^3 + 2^3 + 3^3 + 4^3 + 5^5
```

Function of a single argument (not called term)

The cube function is passed as an argument value

The function bound to term gets called here
Function Values as Parameters

Example: http://goo.gl/e4YBH
Parameters can be bound to function values

Example: http://goo.gl/e4YBH
Parameters can be bound to function values

```python
1 def cube(k):
2     return pow(k, 3)
3 
4 def summation(n, term):
5     total, k = 0, 1
6     while k <= n:
7         total, k = total + term(k), k + 1
8     return total
9 
10 result = summation(5, cube)
```

Example: [http://goo.gl/e4YBH](http://goo.gl/e4YBH)
Function Values as Parameters

Parameters can be bound to function values

Example: [http://goo.gl/e4YBH](http://goo.gl/e4YBH)
Function Values as Parameters

Parameters can be bound to function values

Example: http://goo.gl/e4YBH
Function Values as Parameters

Parameters can be bound to function values

Example: [link](http://goo.gl/e4YBH)
Parameters can be bound to function values

Example: [http://goo.gl/e4YBH](http://goo.gl/e4YBH)
def make_adder(n):
    """Return a function that adds n to its argument.
    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return add(n, k)
    return adder
Functions as Return Values

Locally defined functions can be returned

def make_adder(n):
    """Return a function that adds n to its argument."""

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """

def adder(k):
    return add(n, k)

return adder
Functions as Return Values

Locally defined functions can be returned
They have access to the frame in which they are defined

```python
def make_adder(n):
    """Return a function that adds n to its argument."

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return add(n, k)
    return adder
```
Locally defined functions can be returned
They have access to the frame in which they are defined

def make_adder(n):
    """Return a function that adds n to its argument."

    def adder(k):
        return add(n, k)
    return adder

>>> add_three = make_adder(3)
>>> add_three(4)
7
"""
Locally defined functions can be returned
They have access to the frame in which they are defined

```python
def make_adder(n):
    '''Return a function that adds n to its argument.'''
    def adder(k):
        return add(n, k)
    return adder
```
Functions as Return Values

Locally defined functions can be returned
They have access to the frame in which they are defined

A function that returns a function

```python
def make_adder(n):
    '''Return a function that adds n to its argument.'''
    def adder(k):
        return add(n, k)
    return adder

>>> add_three = make_adder(3)
>>> add_three(4)
7

A local def statement
The name add_three is bound to a function
Locally defined functions can be returned
They have access to the frame in which they are defined

```
def make_adder(n):
    """Return a function that adds n to its argument."
    def adder(k):
        return add(n, k)
    return adder

>>> add_three = make_adder(3)
>>> add_three(4)
7

A function that returns a function

The name add_three is bound to a function
```

```
def adder(k):
    return add(n, k)
return adder
```

A local def statement

Can refer to names in the enclosing function
Call Expressions as Operators

\[
\text{make}\_\text{adder}(1)(2)
\]

def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder
Call Expressions as Operators

```python
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```

```
make_adder(1) ( 2  )
```
Call Expressions as Operators

```
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```

```
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```
Call Expressions as Operators

```python
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```

```
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```
Call Expressions as Operators

```
def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder
make_adder(1)(2)
```

An expression that evaluates to a function value
Call Expressions as Operators

\[ \text{make\_adder}(1)(2) \]

\[ \begin{align*}
\text{make\_adder}(1) & \quad ( \quad 2 \quad ) \\
\text{Operator} & \quad \text{Operand 0}
\end{align*} \]

An expression that evaluates to a function value

An expression that evaluates to any value

\[
\text{def make\_adder}(n): \\
\text{def adder}(k): \\
\quad \text{return add}(n, k) \\
\text{return adder}
\]
Higher-Order Functions
Higher-Order Functions

Functions are first-class: they can be manipulated as values in Python
Functions are first-class: they can be manipulated as values in Python

Higher-order function: a function that takes a function as an argument value or returns a function as a return value
Higher-Order Functions

Functions are first-class: they can be manipulated as values in Python

Higher-order function: a function that takes a function as an argument value or returns a function as a return value

Higher order functions:
Higher-Order Functions

Functions are first-class: they can be manipulated as values in Python

Higher-order function: a function that takes a function as an argument value or returns a function as a return value

Higher order functions:

- Express general methods of computation
Higher-Order Functions

Functions are first-class: they can be manipulated as values in Python

Higher-order function: a function that takes a function as an argument value or returns a function as a return value

Higher order functions:
- Express general methods of computation
- Remove repetition from programs
Higher-Order Functions

Functions are first-class: they can be manipulated as values in Python

Higher-order function: a function that takes a function as an argument value or returns a function as a return value

Higher order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions