Announcements

- Reminder: hw0 due tonight, hw1 due Wed.

- In-class quiz on Friday
  - Covers through Wednesday’s lecture
  - Bring a writing implement

- Hog project out
  - Get started early!
  - More on hog next time
The Elements of Programming
Primitive Expressions and Statements

- The simplest building blocks of a language
The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language

- Means of Combination
  - Compound elements built from simpler ones
The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language

- Means of Combination
  - Compound elements built from simpler ones

- Means of Abstraction
  - Elements can be named and manipulated as units
Environment diagrams visualize the interpreter’s process.

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Code (left):

```
1 from math import pi
2 tau = 2 * pi
```

Frames (right):

```
Global frame
pi  3.1416
```

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Code (left):
```
1 from math import pi
2 tau = 2 * pi
```

Frames (right):
```
Global frame
pi  3.1416
```

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

**Code (left):**
```
1 from math import pi
2 tau = 2 * pi
```

**Frames (right):**
```
Global frame
pi 3.1416
```

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

Next line is highlighted

**Frames (right):**

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

**Code (left):**

```
from math import pi
```

```
tau = 2 * pi
```

**Frames (right):**

```
Global frame
pi 3.1416
```

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

Code (left):

Statements and expressions

Next line is highlighted

Frames (right):

A name is bound to a value

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

Next line is highlighted

**Frames (right):**

A name is bound to a value

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

Code (left):
Statements and expressions
Next line is highlighted

Frames (right):
A name is bound to a value

Example: [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

**Code (left):**
Statements and expressions

**Frames (right):**
A name is bound to a value

**Example:** [http://goo.gl/SK13i](http://goo.gl/SK13i)
Environment diagrams visualize the interpreter’s process.

Code (left):
Statements and expressions
Next line is highlighted

Frames (right):
A name is bound to a value
In a frame, there is at most one binding per name

Example: http://goo.gl/SK13i
User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction
User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

```python
>>> def <name>(<formal parameters>):
    return <return expression>
```
User-Defined Functions

Named values are a simple means of abstraction.

Named computational processes are a more powerful means of abstraction.

Function “signature” indicates how many parameters:

```python
>>> def <name>(<formal parameters>):
    return <return expression>
```
User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function “signature” indicates how many parameters

```python
>>> def <name>(<formal parameters>):
    return <return expression>
```

Function “body” defines a computational process
Named values are a simple means of abstraction.

Named computational processes are a more powerful means of abstraction.

Execution procedure for `def` statements:

```python
>>> def <name>(<formal parameters>):
    return <return expression>
```

Function “signature” indicates how many parameters.

Function “body” defines a computational process.
User-Defined Functions

Named values are a simple means of abstraction.

Named computational processes are a more powerful means of abstraction.

Function “signature” indicates how many parameters:

```python
>>> def <name>(<formal parameters>):
    return <return expression>
```

Function “body” defines a computational process.

Execution procedure for `def` statements:
1. Create a function value with signature:
   `<name>(<formal parameters>)`
Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

>>> def <name>(<formal parameters>):
    return <return expression>

Execution procedure for def statements:
1. Create a function value with signature
   <name>(<formal parameters>)
2. Bind <name> to that value in the current frame
Calling User-Defined Functions

```python
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1 from operator import mul
2 def square(x):
   3    return mul(x, x)
4 square(-2)

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame

Example: http://goo.gl/boCk0
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: http://goo.gl/boCk0
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example: http://goo.gl/boCk0
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

Example:

```python
from operator import mul

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

square(-2)
```

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

A function’s signature has all the information to create a local frame

Example: http://goo.gl/boCk0
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):
1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```

A function’s signature has all the information to create a local frame

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Calling User-Defined Functions

Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the new environment

A function’s signature has all the information to create a local frame

Example: http://goo.gl/boCk0
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

```python
from operator import mul

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

square(-2)
```

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame

Example: http://goo.gl/boCk0
Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error

Example: http://goo.gl/boCk0
General Lookup Procedure
General Lookup Procedure

- Every expression is evaluated in the context of an environment
General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:
Every expression is evaluated in the context of an environment

So far, the current environment is either:

- The global frame alone, or
General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:
  - The global frame alone, or
  - A local frame, followed by the global frame
General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:
  - The global frame alone, or
  - A local frame, followed by the global frame

- Important properties of environments:
General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:
  - The global frame alone, or
  - A local frame, followed by the global frame

- **Important properties of environments:**
  - An environment is a sequence of frames
Every expression is evaluated in the context of an environment.

So far, the current environment is either:
- The global frame alone, or
- A local frame, followed by the global frame

Important properties of environments:
- An environment is a sequence of frames
- The earliest frame that contains a binding for a name determines the value that the name evaluates to
General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:
  - The global frame alone, or
  - A local frame, followed by the global frame

- **Important properties of environments:**
  - An environment is a sequence of frames
  - The earliest frame that contains a binding for a name determines the value that the name evaluates to

- The *scope* of a name is the region of code that has access to it
Multiple Environments in a Diagram

```python
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

Example: [http://goo.gl/hrfnV](http://goo.gl/hrfnV)
Every expression is evaluated in the context of an environment.

Example: http://goo.gl/hrfnV
Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

Example: [link](http://goo.gl/hrfnV)
Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

Example: \texttt{http://goo.gl/hrfnV}
Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment. The earliest frame that contains a binding for a name determines the value that the name evaluates to.

Example: http://goo.gl/hrfnV
Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```

```
1 Global frame
2     | mul
3     | func mul(...)
4     | func square(x)

square

3 Return value
9

x
```

Example: [http://goo.gl/hrfnV](http://goo.gl/hrfnV)

“mul” is not found
Formal Parameters

Example:

```python
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```

Example: [link](http://goo.gl/boCk0)
def square(x):
    return mul(x, x)

Example: [link](http://goo.gl/boCk0)
Formal Parameters

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

Example: http://goo.gl/boCk0
Formal Parameters

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

vs

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

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Formal Parameters

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

vs

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

Example: http://goo.gl/boCk0
Formal Parameters

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

vs

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

Example: [http://goo.gl/boCk0](http://goo.gl/boCk0)
Life Cycle of a User-Defined Function

Def statement:

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

What happens?

Call expression:

`square(2+2)`

Calling/Applying:

```
square(x):
```

```python
square(2+2)
```
Life Cycle of a User-Defined Function

Def statement:

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

What happens?

Call expression: square(2+2)

Calling/Applying:

square(x):
Life Cycle of a User-Defined Function

Def statement:

Name

square( x ):

return mul(x, x)

What happens?

Call expression:

square(2+2)

Calling/Applying:

square( x ):
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square(x):

return mul(x, x)

What happens?

Call expression: square(2+2)

Calling/Applying:

square(x):
Life Cycle of a User-Defined Function

Def statement: square(x):

- **Name**: square(x)
- **Formal parameter**: x
- **Body**: return mul(x, x)

Call expression: square(2+2)

Calling/Applying:

What happens?
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square(x):

return mul(x, x)

Body (return statement)

What happens?

Call expression: square(2+2)

Calling/Applying:

square(x):
Life Cycle of a User-Defined Function

Def statement:

Def statement

Name

Formal parameter

square(x):

return: \text{mul}(x, x)

Return expression

Body (return statement)

Call expression:

square(2+2)

Calling/Applying:

square(x):

What happens?
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square(x):

Return expression

Body (return statement)

Def statement

What happens?

Function created

Call expression:

square(2+2)

Calling/Applying:

square( x ):
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square(x):

Return expression

Body (return statement)

Def statement

What happens?

Function created

Name bound

Call expression:

square(2+2)

Calling/Applying:

square(x):
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square(x):

return: \textit{mul}(x, x)

Body (return statement)

Def statement:

Calling/Applying:

Def statement:

Formal parameter

Name

square(x):

return: \textit{mul}(x, x)

Body (return statement)

Call expression:

square(2+2)

operands: 2+2

argument: 4

What happens?

Function created

Name bound

Calling/Applying:

square(x):
Life Cycle of a User-Defined Function

Def statement:

Formal parameter:

Name: square(x):

Return expression:

Body (return statement):

Def statement

Calling/Applying:

operator: square
function: func square(x)

Call expression:

square(2+2)

operand: 2+2
argument: 4

What happens?
Function created
Name bound

Calling/Applying:

square( x ):
Def statement: square(x):

Formal parameter: x

Return expression: return mul(x, x)

Body (return statement)

Def statement: square(2+2)

Calling/Applying: square(x)

Operator: square
Function: func square(x)

What happens?
Function created
Name bound

Op's evaluated

operator: square
operand: 2+2
argument: 4

square(x):
Life Cycle of a User-Defined Function

Def statement:

- Formal parameter
- Name: `square(x)`
- Return expression: `mul(x, x)`

Def statement:

- Body (return statement):

Call expression:

- Operator: `square`
- Function: `func square(x)`
- Operand: `2+2`
- Argument: `4`

What happens?
- Function created
- Name bound

Op's evaluated
- Function called with argument(s)

Calling/Applying:

```
square(x):
```

```
square(2+2):
```
Life Cycle of a User-Defined Function

Def statement:
- **Formal parameter**
  - Name: `square(x)`: return `mul(x, x)`
  - Body (return statement)

Call expression:
- **Calling/Applying**: `square(2+2)`
  - Operator: `square`
  - Function: `func square(x)`
  - Operand: `2+2`
  - Argument: `4`

What happens?
- Function created
- Name bound
- Op's evaluated
- Function called with argument(s)
Life Cycle of a User-Defined Function

Def statement: square(x):
    return mul(x, x)

What happens?
Function created
Name bound

Call expression: square(2+2)
operator: square
function: func square(x)

Op's evaluated
Function called with argument(s)

Calling/Applying: 4
    square(x):
Life Cycle of a User-Defined Function

Def statement:

- **Name**: square(x):
- **Formal parameter**: x
- **Return expression**: return mul(x, x)
- **Body** (return statement)

What happens?
Function created
Name bound

Call expression:

- **Operator**: square
- **Function**: func square(x)
- **Operand**: 2+2
- **Argument**: 4

Op's evaluated
Function called with argument(s)

Calling/Applying:

- **Signature**: 4 → square(x):
- **Result**: 16
Life Cycle of a User-Defined Function

Def statement: square(x):

Formal parameter: x

Body (return statement):
return mul(x, x)

Def statement: square(x):

Name bound: square

Function created: func square(x)

Calling/Applying: square(2+2)

Operator: square
Function: func square(x)
Operand: 2+2
Argument: 4

What happens?
Function called with argument(s)
Op's evaluated
Argument: 4
Signature: square(x):
Result: 16
Life Cycle of a User-Defined Function

Def statement:
- Name: square(x):
- Return expression: return mul(x, x)
- Body (return statement)

Call expression:
- Operator: square
- Function: func square(x)
- Operand: 2+2
- Argument: 4

Calling/Applying:
- Argument: 4
- Signature
- Return value: 16

What happens?
- Function created
- Name bound

Op's evaluated
- Function called with argument(s)
Life Cycle of a User-Defined Function

Def statement:
- **Name**: square(x)
- **Formal parameter**: x
- **Return expression**: return mul(x, x)
- **Body**: (return statement)

Call expression:
- **Operator**: square
- **Function**: func square(x)
- **Operand**: 2+2
- **Argument**: 4

Calling/Applying:
- Argument: 4
- Signature: 4
- **Return value**: 16

What happens?
- Function created
- Name bound
- Op's evaluated
- Function called with argument(s)
- New frame!
Life Cycle of a User-Defined Function

Def statement: 
- Name: square(x):
- Return expression: return mul(x, x)
- Body (return statement)

Call expression:
- Operator: square
- Function: func square(x)
- Operand: 2+2
- Argument: 4

What happens?
- Function created
- Name bound
- Op's evaluated
- Function called with argument(s)

Calling/Applying:
- Argument: 4
- Signature: square(x):
- Return value: 16

New frame!
- Params bound
Life Cycle of a User-Defined Function

Def statement:
- **Formal parameter**
  - Name: square(x):
  - Return expression: return mul(x, x)

Def statement:
- **Body** (return statement)

Call expression:
- **operator**: square
- **function**: func square(x)

Calling/Applying:
- **Argument**: 4
- **Signature**: square(x):
  - **Operand**: 2+2
    - **Argument**: 4

What happens?
- Function created
- Name bound

Op's evaluated
- Function called with argument(s)

New frame!
- Params bound
- Body executed

Return value 16