61A Lecture 3

Wednesday, August 29
Life Cycle of a User-Defined Function

Def statement:

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

Call expression: `square(2+2)`

Calling/Applying: `square( x )`
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:

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

**Def statement:**

```python
Name: square( x ):
    return mul(x, x)
```

**Call expression:**  `square(2+2)`

**Calling/Applying:**

```python
square( x ):
```
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:  
\texttt{def square(x):}  
\hspace{2em} \texttt{return mul(x, x)}

Call expression:  \texttt{square(2+2)}

Calling/Applying:  
\texttt{square( x ):}  
\hspace{2em} \texttt{return mul(x, x)}
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

Def statement

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: \( x \)

Return expression: \( \text{mul}(x, x) \)

**Body (return statement):**

**What happens?**

**Call expression:** \( \text{square}(2+2) \)

**Calling/Applying:**

\( \text{square}(x) : \)
Life Cycle of a User-Defined Function

Def statement: 

Formal parameter: $x$

Return expression: $\text{mul}(x, x)$

Body (return statement):

Function created

What happens?

Name

Call expression: $\text{square}(2+2)$

Calling/Applying:

$\text{square}(x)$:
Life Cycle of a User-Defined Function

Def statement:  

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

Calling/Applying:  

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

Def statement: 

Formal parameter

Return expression

Name

square(x):

return mul(x, x)

Body (return statement)

Def statement

What happens?

Function created

Name bound

Call expression: square(2+2)

operand: 2+2

argument: 4

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

Def statement:

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

Calling/Applying:

```
square(2+2)
```

What happens?

Function created
Name bound

Formal parameter
Return expression
Body (return statement)

Operator: square
Function: func square(x)
Operand: 2+2
Argument: 4
Life Cycle of a User-Defined Function

Def statement:

Formal parameter

Name

square(x):

Return expression

return mul(x, x)

Body (return statement)

Op's evaluated

What happens?

Function created

Name bound

Call expression:

operator: square

function: func square(x)

operand: 2+2

argument: 4

Calling/Applying:

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

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

**What happens?**
- Function created
- Name bound

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

**Calling/Applying:**
- \( \text{square}(x) : \)
- \( \text{return mul}(x, x) \)
Life Cycle of a User-Defined Function

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

What happens?
- Function created
- Name bound

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

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

**Def statement:**
- Name: `square(x)`: Formally parameterized function.
- Body: `return mul(x, x)` — return statement.

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

**Calling/Applying:**
- 4 > `square(x)`: Function called with argument(s).

**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)`
- Body (return statement): `return mul(x, x)`

**Call expression:**
- Operator: `square`
- Function: `func square(x)`
- Signature: 4
- Argument: 2+2
- Op's evaluated: 4
- Argument: 4
- Function called with argument(s): 16

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

Def statement:

Formal parameter

Return expression

Name

Body (return statement)

Call expression:

operator: square
function: func square(x)

operand: 2+2
argument: 4

Calling/Applying:

Argument

Signature

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):`
- **Body:** `return mul(x, x)`

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

**Calling/Applying:**
- **Argument:** 4
- **Signature:** `4` → `square(x)`
- **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):  
Body (return statement):  
Return expression: return mul(x, x)

What happens?  
Function created
Name bound

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

Op's evaluated  
Function called with argument(s)

Calling/Applying:  
Argument: 4  
Signature:  
Return value: 16
Life Cycle of a User-Defined Function

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

**What happens?**
- Function created
- Name bound

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

**Calling/Applying:**
- Signature
- Argument: 4
- Return value: 16
- New frame!
- Params bound

**Op's evaluated**
- Function called with argument(s)
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**: square(x):
- **Return**: 16

**What happens?**
- Function created
- Name bound
- Op's evaluated
- Function called with argument(s)
- New frame!
- Params bound
- Body executed
Multiple Environments in One Diagram!

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

(Demo)

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

(Demo)

\texttt{square(square(3))}

Example: \texttt{http://goo.gl/668fU}
Multiple Environments in One Diagram!

(Demo)

square(square(3))
Multiple Environments in One Diagram!

(Demo)

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

(Demo)

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

(Demo)

Example: [http://goo.gl/668fU](http://goo.gl/668fU)
Multiple Environments in One Diagram!

(Demo)

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

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

(Demo)

Example: http://goo.gl/668fU

```python
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```
Multiple Environments in One Diagram!

(Demo)

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

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

Example: http://goo.gl/668fU

An environment is a sequence of frames.
Multiple Environments in One Diagram!

Example: http://goo.gl/668fU

An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

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

Example: http://goo.gl/668fU
Multiple Environments in One Diagram!

An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Example: [http://goo.gl/668fU](http://goo.gl/668fU)
An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Example: [http://goo.gl/668fU](http://goo.gl/668fU)
Multiple Environments in One Diagram!

An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Example: http://goo.gl/668fU
Names Have No Meaning Without Environments

```python
from operator import mul

def square(x):
    return mul(x, x)
square(square(3))
```

Example: http://goo.gl/668fU
Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

```
1 from operator import mul
2 def square(x):
3    return mul(x, x)
4 square(square(3))
```
Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

```python
from operator import mul

def square(x):
    return mul(x, x)
square(square(3))
```

Example: [http://goo.gl/668fU](http://goo.gl/668fU)
Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

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

Example: http://goo.gl/668fU
Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

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

Example: [http://goo.gl/668fU](http://goo.gl/668fU)
Names Have No Meaning Without Environments

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

```
from operator import mul

def square(x):
    return mul(x, x)
square(square(3))
```

Example: http://goo.gl/668fU
Formal Parameters

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

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

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

```python
def square(x):
    return mul(x, x)  # vs
```

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

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

vs

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

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

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

vs

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

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

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

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

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

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

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

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

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

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

(Formal parameters have local scope)

Example: http://goo.gl/OapJa
Python Feature Demonstration

Operators
Multiple Return Values
Docstrings
Doctests
Default Arguments
Statements
A statement is executed by the interpret to perform an action.
Statements

A statement is executed by the interpreter to perform an action.

Compound statements:

- `<header>`:
  `<statement>`
  `<statement>`
  ```
  ...
  ```
- `<separating header>`:
  `<statement>`
  `<statement>`
  ```
  ...
  ```
  ```
  ...
A statement is executed by the interpreter to perform an action.

**Compound statements:**

<header>:
  <statement>
  <statement>
  ...
<separating header>:
  <statement>
  <statement>
  ...
  ...

Statements
A statement is executed by the interpreter to perform an action

Compound statements:

<header>:
  <statement>
  <statement>
  <statement>
  ...
<separating header>:
  <statement>
  <statement>
  <statement>
  ...
  ...

Statement

Clause
Statements

A statement is executed by the interpreter to perform an action

Compound statements:

- **Statement**
- **Clause**
- **Suite**

- `<header>:
  <statement>
  <statement>
  ...
- `<separating header>:
  <statement>
  <statement>
  ...
  ...
  ...
Statements

A statement is executed by the interpret to perform an action

Compound statements:

The first header determines a statement’s type

<header>:
  <statement>
  <statement>
  ...
<separating header>:
  <statement>
  <statement>
  ...
  ...

Statement
Clause
Suite
Statements

A statement
is executed by the interpret
to perform an action

Compound statements:

The first header
determines a
statement’s type

The header of a clause
“controls” the suite
that follows
A statement is executed by the interpreter to perform an action.

Compound statements:

The first header determines a statement’s type.

The header of a clause “controls” the suite that follows.

def statements are compound statements.
Compound statements:

<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
    ...

Suite
Compound Statements

**Compound statements:**

```
<header>:
  <statement>
  <statement>
  ...
<separating header>:
  <statement>
  <statement>
  ...
  ...
```

A suite is a sequence of statements
Compound Statements

Compound statements:

A suite is a sequence of statements

To “execute” a suite means to execute its sequence of statements, in order
Compound Statements

Compound statements:

A suite is a sequence of statements

Execution Rule for a sequence of statements:

• Execute the first

• Unless directed otherwise, execute the rest
Local Assignment

```python
1   def percent_difference(x, y):
2       difference = abs(x-y)
3       return 100 * difference / x
4   diff = percent_difference(40, 50)
```

Example: http://goo.gl/wcF71
Local Assignment

```python
def percent_difference(x, y):
    difference = abs(x - y)
    return 100 * difference / x

diff = percent_difference(40, 50)
```

**Execution rule for assignment statements:**

Example: [http://goo.gl/wcF71](http://goo.gl/wcF71)
Local Assignment

```
1 def percent_difference(x, y):
2    difference = abs(x-y)
3    return 100 * difference / x
4    diff = percent_difference(40, 50)
```

**Execution rule for assignment statements:**

1. Evaluate all expressions right of `=`, from left to right.

2. Bind the names on the left the resulting values in the **first frame** of the current environment.

Example: [http://goo.gl/wcF71](http://goo.gl/wcF71)
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x

Execution rule for conditional statements:
def absolute_value(x):
    
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x

Execution rule for conditional statements:

Each clause is considered in order.

1. Evaluate the header's expression.

2. If it is a true value, execute the suite & skip the remaining clauses.
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x

George Boole
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
def absolute_value(x):
    '''Return the absolute value of x.'''
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x

Two boolean contexts
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
Boolean Contexts

```python
def absolute_value(x):
    '''Return the absolute value of x.'''
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

Two boolean contexts

False values in Python: False, 0, '', None
Boolean Contexts

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

**Two boolean contexts**

George Boole

False values in Python:  False, 0, '', None  *(more to come)*
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x

Two boolean contexts

False values in Python: False, 0, '', None (more to come)

True values in Python: Anything else (True)
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x

Two boolean contexts

False values in Python:    False, 0, '', None    (more to come)

True values in Python:    Anything else (True)

Read Section 1.5.4!
Iteration

\[
i, \text{total} = 0, 0
\]

while \(i < 3\):
  \[i = i + 1\]
  \[\text{total} = \text{total} + i\]

**Execution rule for while statements:**

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/07y0D](http://goo.gl/07y0D)
**Iteration**

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)
Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/07y0D
Iteration

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

**Execution rule for while statements:**

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/07y0D](http://goo.gl/07y0D)
Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/07y0D](http://goo.gl/07y0D)
Iteration

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/07y0D
Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/07y0D](http://goo.gl/07y0D)
Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/07y0D
Iteration

```
>>> i, total = 0, 0
>>> while (i < 3):
    i = i + 1
    total = total + i
```

**Execution rule for while statements:**

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/07y0D](http://goo.gl/07y0D)
**Iteration**

```python
d, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
</tr>
<tr>
<td>total</td>
</tr>
</tbody>
</table>

**Execution rule for while statements:**

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)
Iteration

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/07y0D
Iteration

- i, total = 0, 0
- while (i < 3):
  - i = i + 1
  - total = total + i

Global frame

| i | 2 |
| total | 1 |

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)
Iteration

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/07y0D
Iteration

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)
Iteration

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/O7y0D
**Iteration**

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

**Execution rule for while statements:**

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)
**Iteration**

 Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)
Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: http://goo.gl/O7y0D
Iteration

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value, execute the (whole) suite, then return to step 1.

Example: [http://goo.gl/O7y0D](http://goo.gl/O7y0D)