Announcements

• Homework 1 is due next Tuesday at 5pm (no email when you submit).
  • Homework is graded for effort.
Announcements

- Homework 1 is due next Tuesday at 5pm (no email when you submit).
  - Homework is graded for effort.

- Take-home quiz released next Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  - 3 points, graded for correctness.
  - Similar in format to a homework assignment.
  - If you receive 0/3, you will need to talk to the course staff or be dropped.
Announcements

- Homework 1 is due next Tuesday at 5pm (no email when you submit).
  - Homework is graded for effort.

- Take-home quiz released next Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  - 3 points, graded for correctness.
  - Similar in format to a homework assignment.
  - If you receive 0/3, you will need to talk to the course staff or be dropped.

- Open-computer: You can use the Python interpreter, watch course videos, and read the online text (http://composingprograms.com).

- No external resources: Please don't search for answers, talk to your classmates, etc.
Announcements

• Homework 1 is due next Tuesday at 5pm (no email when you submit).
  ▪ Homework is graded for effort.

• Take-home quiz released next Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  ▪ 3 points, graded for correctness.
  ▪ Similar in format to a homework assignment.
  ▪ If you receive 0/3, you will need to talk to the course staff or be dropped.
  ▪ Open-computer: You can use the Python interpreter, watch course videos, and read the online text (http://composingprograms.com).
  ▪ No external resources: Please don't search for answers, talk to your classmates, etc.

• Project 1 posted this Friday, due Thursday 9/19 at 11:59pm.
  ▪ Demo during next lecture
Multiple Environments
Life Cycle of a User-Defined Function

Def statement:

Call expression:

Calling/Applying:

What happens?
Life Cycle of a User-Defined Function

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

Call expression:

Calling/Applying:
Life Cycle of a User-Defined Function

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

Call expression:

Calling/Applying:
Life Cycle of a User-Defined Function

Def statement:

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

What happens?

Call expression:

Calling/Applying:
Life Cycle of a User-Defined Function

**Def statement:**

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

**Call expression:**

**Calling/Applying:**
Life Cycle of a User-Defined Function

Def statement:

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

What happens?

Call expression:

Calling/Applying:
Life Cycle of a User-Defined Function

Def statement:

Formal parameter
Name

Def statement

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

Body (return statement)

What happens?

Call expression:

Calling/Applying:
Life Cycle of a User-Defined Function

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

Call expression:

Calling/Applying:
Life Cycle of a User-Defined Function

**Def statement:**

Name: `square(x):`

Formal parameter: `x`

Body (return statement):

Return expression: `mul(x, x)`

What happens?

A new function is created!

**Call expression:**

**Calling/Applying:**
Life Cycle of a User-Defined Function

**Def statement:**

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

**Body (return statement)**

**What happens?**

- A new function is created!
- Name bound to that function in the current frame

**Call expression:**

**Calling/Applying:**
Life Cycle of a User-Defined Function

Def statement:  
\[ \text{\texttt{square}(\texttt{x}): \texttt{return mul(x, x)}} \]

What happens?

- A new function is created!
- Name bound to that function in the current frame

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

Calling/Applying:
Life Cycle of a User-Defined Function

Def statement:

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

What happens?
- A new function is created!
- Name bound to that function in the current frame

Call expression:

- Operator: square
- function: func square(x)

Calling/Applying:
Life Cycle of a User-Defined Function

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

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

**Calling/Applying:**

**What happens?**

- A new function is created!
- Name bound to that function in the current frame
Life Cycle of a User-Defined Function

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

Call expression: `square(2+2)`

Calling/Applying:

What happens?

A new function is created!

Name bound to that function in the current frame

Operator & operands evaluated
Life Cycle of a User-Defined Function

Def statement:

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

A new function is created!
Name bound to that function in the current frame

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

Operator & operands evaluated
Function (value of operator) called on arguments (values of operands)

Call expression:

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

What happens?

Calling/Applying:
Life Cycle of a User-Defined Function

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

Call expression:
- operand: 2+2
- argument: 4
- operator: square
- function: func square(x)

What happens?
A new function is created!
Name bound to that function in the current frame
Operator & operands evaluated
Function (value of operator) called on arguments (values of operands)
Life Cycle of a User-Defined Function

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

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

**Calling/Applying:**
- **Signature**: square(x):

---

**What happens?**

A new function is created!

Name bound to that function in the current frame

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)
Life Cycle of a User-Defined Function

Def statement:

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

What happens?

A new function is created!
Name bound to that function in the current frame

Call expression:

- **Operator & operands evaluated**: `operand: 2+2`, `argument: 4`
- **Function (value of operator) called on arguments (values of operands)**

Calling/Applying:

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

**Def statement:**

- **Name:** `square(x):`
- **Body (return statement):** `return mul(x, x)`
- **Formal parameter:** `x`
- **Return expression:** `x ** 2`

**Calling/Applying:**

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

**What happens?**

- **A new function is created!**
- **Name bound to that function in the current frame**
- **Operator & operands evaluated**
- **Function (value of operator called on arguments (values of operands)**
Life Cycle of a User-Defined Function

**Def statement:**

- Name: `square(x):`
- Body (return statement): `return mul(x, x)`

**Call expression:**

- Operator: `square(2+2)`
- Argument: `4`

**Calling/Applying:**

- Argument: `4`
- Signature: `16`

**What happens?**

- A new function is created!
- Name bound to that function in the current frame
- Operator & operands evaluated
- Function (value of operator) called on arguments (values of operands)
Life Cycle of a User-Defined Function

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

**Call expression:**
- `square(2+2)`
- **Operand:** `2+2`
- **Argument:** `4`

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

**What happens?**
- A new function is created!
- Name bound to that function in the current frame
- Operator & operands evaluated
- Function (value of operator) called on arguments (values of operands)
Life Cycle of a User-Defined Function

Def statement:

**Name**

square(x):

**Formal parameter**

return mul(x, x)

**Body (return statement)**

A new function is created!

Name bound to that function in the current frame

Call expression:

**operator**: square

**function**: func square(x)

**Operand**: 2+2

**Argument**: 4

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

Calling/Applying:

**Argument**: 4

**Signature**

**Return value**: 16

what happens?

A new frame is created!
Life Cycle of a User-Defined Function

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

What happens?

A new function is created!
Name bound to that function in the current frame

Call expression: square(2+2)

Operator & operands evaluated
Function (value of operator) called on arguments (values of operands)

What happens?

A new frame is created!
Parameters bound to arguments
Life Cycle of a User-Defined Function

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

What happens?
- A new function is created!
- Name bound to that function in the current frame

Call expression:
- Operator & operands evaluated
- Function (value of operator) called on arguments (values of operands)

Calling/Applying:
- Arguments bound to arguments
- Body is executed in that new environment
Multiple Environments in One Diagram!

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

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

Example: http://goo.gl/XVtEms

```python
from operator import mul

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

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

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

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

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

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

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

square(square(3))
```

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

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

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

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

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

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

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

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

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

```python
from operator import mul

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

square(square(3))
```

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

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

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

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

An environment is a sequence of frames.

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

A function call: `square(square(3))`

Example:
```python
from operator import mul

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

An environment is a sequence of frames.

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

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

```python
from operator import mul

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

square(square(3))
```

An environment is a sequence of frames.

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

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

```python
from operator import mul

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

An **environment** is a *sequence of frames*.

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

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

Every expression is evaluated in the context of an environment.

An environment is a sequence of frames.

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

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

```python
from operator import mul

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

square(square(3))
```

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.

An environment is a sequence of frames.

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

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

An environment is a sequence of frames.

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

Example: http://goo.gl/XVtEms
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.

An environment is a sequence of frames.

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

Example: http://goo.gl/XVtEms
Miscellaneous Python Features

Operators
Multiple Return Values
Docstrings
Doctests
Default Arguments

(Demo)
Conditional Statements
Statements
A *statement* is executed by the interpreter to perform an action.
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>
  ...
  ...
  ...
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>
    ...
    ...

A statement is executed by the interpreter to perform an action

Compound statements:

The first header determines a statement’s type
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
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

**Compound statements:**

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

Compound statements:

A suite is a sequence of statements

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

Compound statements:

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

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

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

Execution Rule for a sequence of statements:

• Execute the first statement

• Unless directed otherwise, execute the rest
Conditional Statements

(Demo)
Conditional Statements

(Demo)

```python
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
Conditional Statements

(Demo)

```python
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:
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.
**Conditional Statements**

(Demo)

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

1 statement, 3 clauses, 3 headers, 3 suites

**Execution rule for conditional statements:**

1. Evaluate the header's expression.

2. If it is a true value, execute the suite & skip the remaining clauses.

**Syntax Tips**
Conditional Statements

 Deng 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.

Syntax Tips

1. Always starts with "if" clause.
2. Zero or more "elif" clauses.
3. Zero or one "else" clause, always at the end.
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

Boolean Contexts

False values in Python: False, 0, '', None
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
```

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

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

George Boole

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

True values in Python: Anything else (True)

Read Section 1.5.4!

Reading: http://composingprograms.com/pages/15-control.html#conditional-statements
Iteration
While Statements

(Demo)

Example: http://goo.gl/0d2cjF
While Statements

(Demo)

1  i, total = 0, 0
2  while i < 3:
3       i = i + 1
4  total = total + i

Example: http://goo.gl/0d2cjF
While Statements

(Demo)

1 i, total = 0, 0
2 while i < 3:
3 i = i + 1
4 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.
While Statements

(Demo)

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

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```
While Statements

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/0d2cjF
While Statements

(Demo)

1  i, total = 0, 0
2  while \( i < 3 \):
3     i = i + 1
4     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/0d2cjF
While Statements

George Boole

(Demo)

1 i, total = 0, 0
2 while i < 3:
3 i = i + 1
4 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/0d2cjF
While Statements

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/0d2cJF
While Statements

George Boole

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/0d2cjF
While Statements

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/0d2cjF](http://goo.gl/0d2cjF)
While Statements

George Boole

(Demo)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>i, total = 0, 0</td>
</tr>
<tr>
<td>2</td>
<td>while i &lt; 3:</td>
</tr>
<tr>
<td>3</td>
<td>i = i + 1</td>
</tr>
<tr>
<td>4</td>
<td>total = total + i</td>
</tr>
</tbody>
</table>

Global frame

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>0</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/0d2cjF
While Statements

George Boole

(Demo)

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/0d2cjF
While Statements

(Demo)

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/0d2cjF
While Statements

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/0d2cjF
While Statements

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/0d2cjF
While Statements

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: [link to example](http://goo.gl/0d2cjF)
While Statements

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/0d2cjF
While Statements

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/0d2cjF](http://goo.gl/0d2cjF)
While Statements

(Demo)

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:

Example: [http://goo.gl/0d2cJF](http://goo.gl/0d2cJF)
While Statements

George Boole

(Demo)

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/0d2cjF
While Statements

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/0d2cjF](http://goo.gl/0d2cjF)
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/0d2cjF
While Statements

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: [link](http://goo.gl/0d2cjF)
Discussion Question

Complete the following definition by placing an expression in ____________________.

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

def choose(total, selection):

Example: http://goo.gl/38ch3o
**Discussion Question**

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL."
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    """
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ____________________.

def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    \[
    \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 2 \cdot 1}
    \]

Example: http://goo.gl/38ch3o
**Discussion Question**

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10

    \[
    \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 2 \cdot 1}
    \]
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    ""

>>> choose(5, 2)
10
>>> choose(20, 6)
38760

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ____________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
```

Example: http://goo.gl/38ch3o
**Discussion Question**

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL."

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    ""

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * ________________, total - 1
    return ways
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.
    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example:  http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    """

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: http://goo.gl/38ch3o
**Discussion Question**

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.
    
    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * (total // selected), total - 1
    return ways
```

Example: http://goo.gl/38ch3o