Functions
Announcements
Expressions
Types of expressions
Types of expressions

An expression describes a computation and evaluates to a value.
Types of expressions

An expression describes a computation and evaluates to a value

18 + 69
Types of expressions

An expression describes a computation and evaluates to a value

$$18 + 69 \quad \frac{6}{23}$$
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \]

\[ \frac{6}{23} \]

\[ \sqrt{3493161} \]
Types of expressions

An expression describes a computation and evaluates to a value

\[
18 + 69 \quad \frac{6}{23} \quad \sin \pi \quad \sqrt{3493161}
\]
Types of expressions

An expression describes a computation and evaluates to a value

$$18 + 69 \quad \frac{6}{23} \quad \sin \pi \quad \sqrt{3493161} \quad | - 1869|$$
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \]
\[ \frac{6}{23} \]
\[ \sin \pi \]
\[ \sqrt{3493161} \]
\[ \sum_{i=1}^{100} i \]
\[ | - 1869| \]
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \]
\[ \frac{6}{23} \]
\[ \sin \pi \]
\[ \sqrt{3493161} \]
\[ \sum_{i=1}^{100} i \]
\[ | - 1869| \]
\[ 69 \]
\[ 18 \]
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \]
\[ \frac{6}{23} \]
\[ \sin \pi \]
\[ f(x) \]
\[ \sqrt{3493161} \]
\[ \sum_{i=1}^{100} i \]
\[ | - 1869 | \]
\[ (69) \]
\[ (18) \]
An expression describes a computation and evaluates to a value

\[ 18 + 69 \]
\[ \frac{6}{23} \]
\[ \sin \pi \]
\[ 2^{100} \]
\[ f(x) \]
\[ \sum_{i=1}^{100} i \]
\[ | - 1869 | \]
\[ \sqrt{3493161} \]
\[ \binom{69}{18} \]
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \quad \frac{6}{23} \quad \sin \pi \quad \log_2 1024 \]

\[ 2^{100} \]

\[ f(x) \quad \sqrt{3493161} \]

\[ \sum_{i=1}^{100} i \]

\[ -1869 \]

\[ \binom{69}{18} \]
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \]
\[ \frac{6}{23} \]
\[ \sin \pi \]
\[ \log_2 1024 \]
\[ 2^{100} \]
\[ f(x) \]
\[ 7 \mod 2 \]
\[ | - 1869| \]
\[ \sum_{i=1}^{100} i \]
\[ \sqrt{3493161} \]
\[ (69) \]
\[ (18) \]
Types of expressions

An expression describes a computation and evaluates to a value

\[
\begin{align*}
18 + 69 & \quad 6 \quad \sin \pi \quad \log_2 1024 \\
\frac{2^{100}}{23} & \quad f(x) \quad \sqrt{3493161} \\
7 \mod 2 & \quad \sum_{i=1}^{100} i \quad \lim_{x \to \infty} \frac{1}{x} \\
| -1869| & \quad \binom{69}{18}
\end{align*}
\]
Types of expressions

An expression describes a computation and evaluates to a value

\[ 18 + 69 \]
\[ \frac{6}{23} \]
\[ \sin \pi \]
\[ \log_2 1024 \]
\[ 2^{100} \]
\[ f(x) \]
\[ 7 \mod 2 \]
\[ | - 1869| \]
\[ \sum_{i=1}^{100} i \]
\[ \sqrt{3493161} \]
\[ \lim_{x \to \infty} \frac{1}{x} \]
\[ \binom{69}{18} \]
All expressions can use function call notation

(Demo)
Anatomy of a Call Expression
Anatomy of a Call Expression

\[ \text{add} \ ( \ 2 \ , \ 3 \ ) \]
Anatomy of a Call Expression

```
add ( 2 , 3 )
```
Anatomy of a Call Expression

\[
\text{add} \quad ( \quad 2 \quad , \quad 3 \quad )
\]

Operator
Anatomy of a Call Expression

\[
\text{add} \quad ( \quad 2 \quad , \quad 3 \quad )
\]

Operator \hspace{1cm} Operand \hspace{1cm} Operand
Anatomy of a Call Expression

Operators and operands are also expressions

\[
\text{add} \ ( \ 2 \ , \ 3 \ )
\]

Operator \hspace{0.2cm} \textit{Operand} \hspace{0.2cm} \textit{Operand}
Anatomy of a Call Expression

![Diagram showing the anatomy of a call expression]

Operators and operands are also expressions

So they evaluate to values
Anatomy of a Call Expression

Evaluation procedure for call expressions:

Operators and operands are also expressions

So they evaluate to values
Anatomy of a Call Expression

Operators and operands are also expressions

So they evaluate to values

Evaluation procedure for call expressions:

1. Evaluate the operator and then the operand subexpressions
Anatomy of a Call Expression

Evaluation procedure for call expressions:

1. Evaluate the operator and then the operand subexpressions

2. **Apply** the **function** that is the value of the operator to the **arguments** that are the values of the operands
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[ \text{mul(add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6))) \]
Evaluating Nested Expressions
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]

\[ \text{mul} \]

\[ \text{add}(4, \text{mul}(4, 6)) \]

\[ \text{add} \]

\[ 4 \]

\[ \text{mul}(4, 6) \]
Evaluating Nested Expressions

mul(add(4, mul(4, 6)), add(3, 5))
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[
mul(add(4, mul(4, 6)), add(3, 5))
\]

- \(mul\)
- \(add\)
- \(4\)
- \(24\)
- \(mul(4, 6)\)
- \(mul\)
- \(4\)
- \(6\)
Evaluating Nested Expressions
Evaluating Nested Expressions

```
mul(add(4, mul(4, 6)), add(3, 5))
```

```
mul
 mul
  4
  mul
    mul
      4
      6
```
Evaluating Nested Expressions

mul(add(4, mul(4, 6)), add(3, 5))

mul

add(4, mul(4, 6))

add

4

24

mul

mul(4, 6)

mul

4

6

add

3

5
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]

\[ \text{add}(4, \text{mul}(4, 6)) \]

\[ \text{mul}(4, 6) \]

\[ \text{mul}(4, 6) \]

\[ \text{add}(3, 5) \]

\[ \text{add}(3, 5) \]
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]

\[ \text{mul}(\text{add}(4, 24), 8) \]

\[ \text{mul}(4, 6) \]
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

\[ \text{mul}\left(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)\right) \]

Diagram:

- \( \text{mul}\)
- \( \text{add}\)
- \( 4 \)
- \( 24 \)
- \( \text{mul}(4, 6) \)
- \( 8 \)
- \( \text{add}(3, 5) \)
- \( \text{mul}(4, 6) \)
- \( 28 \)
- \( 224 \)
Evaluating Nested Expressions

\[ \text{mul}(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5)) \]
Evaluating Nested Expressions

Expression tree:

```
mul(add(4, mul(4, 6)), add(3, 5))
```

```
mul
```

```
add
```

```
4
```

```
mul(4, 6)
```

```
mul
```

```
24
```

```
add
```

```
3
```

```
5
```

```
224
```

```
Expression tree
```
Evaluating Nested Expressions

\[
mul(\text{add}(4, \text{mul}(4, 6)), \text{add}(3, 5))
\]
Evaluating Nested Expressions
Evaluating Nested Expressions

Operand subexpression

Value of subexpression

1st argument to mul

Expression tree
Evaluating Nested Expressions

Expression tree

Operand subexpression

Value of subexpression

1st argument to mul

Value of the whole expression

mul(addd(4, mul(4, 6)), add(3, 5))
Names, Assignment, and User-Defined Functions

(Demo)
Types of Expressions
Types of Expressions

Primitive expressions:
Types of Expressions

Primitive expressions:

2

Number or Numeral
Types of Expressions

**Primitive expressions:**

- **2**
- **add**

- **Number or Numeral**
- **Name**
Types of Expressions

Primitive expressions:

- Number or Numeral: 2
- Name: add
- String: 'hello'
Types of Expressions

**Primitive expressions:**

- 2 (Number or Numeral)
- add (Name)
- 'hello' (String)

**Call expressions:**
Types of Expressions

**Primitive expressions:**

- 2
- add
- 'hello'

- Number or Numeral
- Name
- String

**Call expressions:**

- max
- ( 2 , 3 )
Types of Expressions

**Primitive expressions:**

- 2 (Number or Numeral)
- add (Name)
- 'hello' (String)

**Call expressions:**

\[
\text{max} (2, 3)
\]

*Operator*
Types of Expressions

**Primitive expressions:**
- 2
- `add`
- `'hello'`

- Number or Numeral
- Name
- String

**Call expressions:**
- `max`
- `(2, 3)`

- Operator
- Operand
- Operand
Types of Expressions

**Primitive expressions:**
- 2
- add
- 'hello'
  - Number or Numeral
  - Name
  - String

**Call expressions:**
- max
  - Operator
- (2, 3)
  - Operand
  - Operand

\[
\text{max}(\min(\text{pow}(3, 5), -4), \min(1, -2))
\]
Types of Expressions

**Primitive expressions:**
- 2
- `add`
- `'hello'`

- Number or Numeral
- Name
- String

**Call expressions:**
- `max(min(pow(3, 5), -4), min(1, -2))`

- An operand can also be a call expression.
Types of Expressions

**Primitive expressions:**
- Number or Numeral: 2
- Name: `add`
- String: 'hello'

**Call expressions:**
- Operator: `max`
  - (Operand: 2, Operand: 3)
- `max(min(pow(3, 5), -4), min(1, -2))`

An operand can also be a call expression.
Discussion Question 1
Discussion Question 1

What is the value of the final expression in this sequence?
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min
```
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min

>>> f = max
```
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min

>>> f = max

>>> g, h = min, max
```

11
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
```

11
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

11
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

```python
>>> max(f(2, g(h(1, 5), 3)), 4)
```

```plaintext
11
```
Discussion Question 1

What is the value of the final expression in this sequence?

```python
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

???
Environment Diagrams
Environment Diagrams

Environment diagrams visualize the interpreter’s process.
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

1. from math import pi
2. tau = 2 * pi
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

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

Global frame

pi | 3.1416
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

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

Global frame:
- pi: 3.1416

Code (left):   Frames (right):
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 |

Statements and expressions
Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

**Frames (right):**

Global frame

| pi   | 3.1416 |

---

Import statement

1. `from math import pi`
2. `tau = 2 * pi`
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

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

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

**Frames (right):**

Global frame

- `pi`: 3.1416
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

**Frames (right):**

Arrows indicate evaluation order
Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

**Frames (right):**

Arrows indicate evaluation order
Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

Arrows indicate evaluation order

**Frames (right):**

Global frame

| pi   | 3.1416 |

1. Import statement

   ```python
   from math import pi
   ```

2. Assignment statement

   ```python
   tau = 2 * pi
   ```
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Code (left):

Statements and expressions

Arrows indicate evaluation order

Frames (right):

Each name is bound to a value
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Code (left):
Statements and expressions
Arrows indicate evaluation order

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

Just executed
Import statement
1 from math import pi
2 tau = 2 * pi

Next to execute
Assignment statement

Global frame
Name
pi 3.1416
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

**Code (left):**
Statements and expressions
Arrows indicate evaluation order

**Frames (right):**
Each name is bound to a value
Environment diagrams visualize the interpreter’s process.

**Code (left):**
- Statements and expressions
- Arrows indicate evaluation order

**Frames (right):**
- Each name is bound to a value
- Within a frame, a name cannot be repeated

---

```
from math import pi

tau = 2 * pi
```

[Environment Diagrams](https://pythontutor.com/composingprograms.html)
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

Arrows indicate evaluation order

**Frames (right):**

Each name is bound to a value

Within a frame, a name cannot be repeated

(Demo)

http://pythontutor.com/composingprograms.html?code=from\ math\ import\ pi\n\ntau = 2 * pi&cumulative=false&curInstr=1&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D
Assignment Statements
Assignment Statements

1  a = 1
2  b = 2
3  b, a = a + b, b
Assignment Statements

1  a = 1
2  b = 2
3  b, a = a + b, b

Global frame

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Assignment Statements

1. \( a = 1 \)
2. \( b = 2 \)
3. \( b, a = a + b, b \)

Global frame

| \( a \) | 1 |
| \( b \) | 2 |
Assignment Statements

```
1 a = 1
2 b = 2
3 b, a = a + b, b
```

Global frame

```
  a  1
  b  2
```
Assignment Statements

Execution rule for assignment statements:
Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.
Assignment Statements

Execution rule for assignment statements:

1. Evaluate all expressions to the right of $=$ from left to right.
2. Bind all names to the left of $=$ to those resulting values in the current frame.
Assignment Statements

Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.

2. Bind all names to the left of = to those resulting values in the current frame.

```
Just executed
1  a = 1
2  b = 2
3  b, a = a + b, b

Next to execute

Global frame
a  |  1
b  |  2

Just executed
1  a = 1
2  b = 2
3  b, a = a + b, b
```
Assignment Statements

Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.
2. Bind all names to the left of = to those resulting values in the current frame.
Discussion Question 1 Solution

(Demo)
Discussion Question 1 Solution

(Demo)

1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( \text{g, h} = \text{min, max} \)
4. \( \text{max} = \text{g} \)
5. \( \text{max}(f(2, \text{g(h(1, 5), 3))), 4) \)

(Demo)
Discussion Question 1 Solution

1. \( f = \min \)
2. \( f = \max \)
3. \( g, h = \min, \max \)
4. \( \max = g \)
5. \( \max(f(2, g(h(1, 5), 3)), 4) \)
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min}, \text{max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min}, \text{max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min, max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)

Global frame

- \( f \)
- \( h \)
- \( g \)
- \( \text{max} \)

func min(...)

func max(...)
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min}, \text{max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)
### Discussion Question 1 Solution

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>f = min</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>f = max</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>g, h = min, max</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>max = g</code></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><code>max(f(2, g(h(1, 5), 3)), 4)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- **Global frame**
  - `func max(...)`
  - `func min(...)`
  - `func max(...)`
  - `func min(...)`

- **Code Diagram**
  - `f(2, g(h(1, 5), 3))`
  - `g(h(1, 5), 3)`
  - `2`
```python
func min(...) ➔ 1
f = min

func max(...) ➔ 2
f = max

func min(...) ➔ 3
g, h = min, max

max = g ➔ 4

func min(...) ➔ 5
max(f(2, g(h(1, 5), 3)), 4)
```

(Demo)

```
Global frame

class func max(...)
  f ➔
  h ➔
  g ➔
  max ➔
```
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min}, \text{max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)
Discussion Question 1 Solution

```
1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)
```

(Demo)

Global frame
- `func max(...)`
- `func min(...)`
- `f`
- `g`
- `h`
- `max`

```
func min(...)
  f(2, g(h(1, 5), 3))
  max(f(2, g(h(1, 5), 3)), 4)
  func max(...)
```

```
func max(...)
  g(h(1, 5), 3)
  func min(...)
    h(1, 5)
    func max(...)
      1 5
```
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min}, \text{max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)
Discussion Question 1 Solution

```python
func min(...)
1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)

func min(...)
func max(...)
func max(...)
func min(...)
func max(...)
```
Discussion Question 1 Solution

```python
func min(...)
1 f = min
2 f = max
3 g, h = min, max
4 max = g
5 max(f(2, g(h(1, 5), 3)), 4)
```

(Demo)

```
func max(...)
  f
  h
  g
func min(...)
  max
Global frame
```

```python
func min(...)
  f
  g(h(1, 5), 3)
  max
func max(...)
  2
func min(...)
  1
  5
```
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min, max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)
Discussion Question 1 Solution

1. \[ f = \text{min} \]
2. \[ f = \text{max} \]
3. \[ g, h = \text{min}, \text{max} \]
4. \[ \text{max} = g \]
5. \[ \text{max}(f(2, g(h(1, 5), 3)), 4) \]

(Demo)
Discussion Question 1 Solution

1. \( f = \text{min} \)
2. \( f = \text{max} \)
3. \( g, h = \text{min, max} \)
4. \( \text{max} = g \)
5. \( \text{max}(f(2, g(h(1, 5), 3)), 4) \)

(Demo)
Defining Functions
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

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

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

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

Function **signature** indicates how many arguments a function takes
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function **signature** indicates how many arguments a function takes

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

Function **body** defines the computation performed when the function is applied
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function signature indicates how many arguments a function takes

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

Function body defines the computation performed when the function is applied

Execution procedure for def statements:
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function signature indicates how many arguments a function takes

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

Function body defines the computation performed when the function is applied

Execution procedure for def statements:

1. Create a function with signature `<name>(<formal parameters>)`
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

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

Function *signature* indicates how many arguments a function takes

Function *body* defines the computation performed when the function is applied

Execution procedure for *def* statements:

1. Create a function with signature `<name>(<formal parameters>)`
2. Set the body of that function to be everything indented after the first line
Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function signature indicates how many arguments a function takes

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

Function body defines the computation performed when the function is applied

Execution procedure for def statements:

1. Create a function with signature <name>(<formal parameters>)
2. Set the body of that function to be everything indented after the first line
3. Bind <name> to that function in the current frame
Calling User-Defined Functions

```python
from operator import mul

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

square(-2)
```
Procedure for calling/applying user-defined functions (version 1):
Calling User-Defined Functions

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

1. Add a local frame, forming a new environment
Calling User-Defined Functions

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

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
Calling User-Defined Functions

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

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment
Calling User-Defined Functions

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

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

```python
from operator import mul

def square(x):
    return mul(x, x)
square(-2)
```
Calling User-Defined Functions

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

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

```python
from operator import mul

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

square(-2)
```
Calling User-Defined Functions

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

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

```python
from operator import mul

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

square(-2)
```
### Calling User-Defined Functions

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

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

```python
from operator import mul

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

square(-2)
```
Calling User-Defined Functions

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

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

```python
from operator import mul

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

square(-2)
```
**Calling User-Defined Functions**

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

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

```python
from operator import mul

def square(x):
    return mul(x, x)
square(-2)
```

![Diagram showing the process of calling a user-defined function with parameters and returns.]
Calling User-Defined Functions

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

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

```python
from operator import mul

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

square(-2)
```
Calling User-Defined Functions

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

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

```python
from operator import mul

def square(x):
    return mul(x, x)
square(-2)
```
Calling User-Defined Functions

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

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

```python
from operator import mul

def square(x):
    return mul(x, x)
square(-2)
```

A function’s signature has all the information needed to create a local frame
Calling User-Defined Functions

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

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that 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 needed to create a local frame
Calling User-Defined Functions

Procedure for calling/applyng user-defined functions (version 1):

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

```
from operator import mul

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

square(-2)
```

A function's signature has all the information needed to create a local frame.
Looking Up Names In Environments
Looking Up Names In Environments

Every expression is evaluated in the context of an environment.
Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:
Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:
  • The global frame alone, or
Looking Up Names In Environments

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.
Looking Up Names In Environments

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.

*Most important two things I’ll say all day:*
Looking Up Names In Environments

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.

Most important two things I’ll say all day:
An environment is a sequence of frames.
Looking Up Names In Environments

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.

*Most important two things I’ll say all day:*

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
Looking Up Names In Environments

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.

*Most important two things I’ll say all day:*

An environment is a sequence of frames.

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

E.g., to look up some name in the body of the square function:
Looking Up Names In Environments

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.

Most important two things I’ll say all day:

An environment is a sequence of frames.

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

E.g., to look up some name in the body of the square function:

- Look for that name in the local frame.
Looking Up Names In Environments

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.

**Most important two things I’ll say all day:**

An environment is a sequence of frames.

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

E.g., to look up some name in the body of the square function:

- Look for that name in the local frame.
- If not found, look for it in the global frame.
  (Built-in names like “max” are in the global frame too, but we don’t draw them in environment diagrams.)
Looking Up Names In Environments

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.

Most important two things I’ll say all day:

An environment is a sequence of frames.

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

E.g., to look up some name in the body of the square function:
  •  Look for that name in the local frame.
  •  If not found, look for it in the global frame.
(Built-in names like “max” are in the global frame too, but we don’t draw them in environment diagrams.)

(Demo)