61A Lecture 2
Announcements
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:

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

**Primitive expressions:**

- 2
- `add`
- `'hello'`

  - **Number or Numeral**
  - **Name**
  - **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
- `add`
- `'hello'`

**Operator:**
- Number or Numeral
- Name
- String

**Call expressions:**
- `max ( 2, 3 )`
- `Operator`
Types of Expressions

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

**Call expressions:**
- max
- (2, 3)
Types of Expressions

**Primitive expressions:**

<table>
<thead>
<tr>
<th>Number or Numeral</th>
<th>Operator</th>
<th>Name</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>add</td>
<td>'hello'</td>
<td></td>
</tr>
</tbody>
</table>

**Call expressions:**

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

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

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

An operand can also be a call expression:

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

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

**Call expressions:**
- \( \text{max}(\text{min}(\text{pow}(3, 5), -4), \text{min}(1, -2)) \)
  - **Operator**
  - \( 2 \)
  - \( 3 \)
  - **Operand**
  - **Operand**

An operand can also be a call expression.

\( \text{max}(\text{min}(\text{pow}(3, 5), -4), \text{min}(1, -2)) \)
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
```

5
Discussion Question 1

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

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

5
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
```
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)
5
```
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)
>>>?
```
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.

Interactive Diagram
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.

<table>
<thead>
<tr>
<th>Code (left):</th>
<th>Frames (right):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 from math import pi</td>
<td>Global frame</td>
</tr>
<tr>
<td>2 tau = 2 * pi</td>
<td>pi 3.1416</td>
</tr>
</tbody>
</table>

Interactive Diagram
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

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

**Code (left):**

**Frames (right):**

Statements and expressions

**Interactive Diagram**
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Import statement

1 from math import pi

2 tau = 2 * pi

Global frame

| pi | 3.1416 |

Code (left):

Statements and expressions

Frames (right):

Interactive Diagram
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

**Frames (right):**

Global frame

pi 3.1416

---

Interactive Diagram
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Code (left):

Statements and expressions

Arrows indicate evaluation order

Frames (right):

Interactive Diagram
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

![Environment Diagram]

**Code (left):**

Statements and expressions

**Arrows indicate evaluation order**

**Frames (right):**

Global frame

pi | 3.1416
Environment Diagrams

Environment diagrams visualize the interpreter’s process.

**Code (left):**

Statements and expressions

Arrows indicate evaluation order

**Frames (right):**

Global frame

| pi  | 3.1416 |

Interactive Diagram
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

Interactive Diagram
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

---

Interactive Diagram
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

Interactive Diagram
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)
Assignment Statements

Interactive Diagram
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

Interactive Diagram

Global frame

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

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

Interactive Diagram

Global frame

- $a \rightarrow 1$
- $b \rightarrow 2$
Assignment Statements

Just executed

1  a = 1
2  b = 2

Next to execute

3  b, a = a + b, b

Global frame

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

Execution rule for assignment statements:

Interactive Diagram
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.

Interactive Diagram
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.
Discussion Question 1 Solution

(Demo)

Interactive Diagram
Discussion Question 1 Solution

(Demo)

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

Interactive Diagram
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

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)

Interactive Diagram
Discussion Question 1 Solution

```python
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)

Interactive Diagram
Discussion Question 1 Solution

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

Interactive Diagram
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)

Interactive Diagram
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)

---

Interactive Diagram
Discussion Question 1 Solution

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

**Interactive Diagram**
Discussion Question 1 Solution

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

Interactive Diagram
func min(...)  
\[ f(2, g(h(1, 5), 3)) \]

func max(...)  
\[ g(h(1, 5), 3) \]

func min(...)  
\[ h(1, 5) \]

func max(...)  
\[ 1 \]

func max(...)  
\[ 2 \]

func min(...)  
\[ 5 \]

func max(...)  
\[ \text{Global frame} \]

Interactive Diagram
Discussion Question 1 Solution

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

(Demo)

Interactive Diagram
Discussion Question 1 Solution

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

## Interactive Diagram
Discussion Question 1 Solution

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

Interactive Diagram
Discussion Question 1 Solution

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

(Demo)

---

Interactive Diagram
Discussion Question 1 Solution

```python
func min(...) 4

f(2, g(h(1, 5), 3))

func max(...) 3

g(h(1, 5), 3)

func min(...) 5

h(1, 5)

func max(...) 1 5
```

(Demo)

Global frame

- `f`
- `h`
- `g`
- `max`

Interactive Diagram
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
f  h  g
max
```

Interactive Diagram
Discussion Question 1 Solution

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

Interactive Diagram
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

Function signature indicates how many arguments a function takes

```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.

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

>>> 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>)
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

```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>)`
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

Interactive Diagram
Calling User-Defined Functions

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

Interactive Diagram
Calling User-Defined Functions

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

1. Add a local frame, forming a new environment
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)
```

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

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

```
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

```python
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

Interactive Diagram
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
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)