Lecture 3: Control

Marvin Zhang
06/22/2016
Announcements
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm
• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast...)}
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm
• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast…)
  • How should I prepare? Read this Piazza post
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm
• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast...)
  • How should I prepare? Read this Piazza post
• Go to lab today! Each lab is worth two points
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm
• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast…)
  • How should I prepare? Read this Piazza post
• Go to lab today! Each lab is worth two points
• Go to discussion tomorrow! Each discussion is worth two exam recovery points
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm

• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast…)
  • How should I prepare? Read this Piazza post

• Go to lab today! Each lab is worth two points

• Go to discussion tomorrow! Each discussion is worth two exam recovery points
  • If you do poorly (< 20 points) on the midterm or final, exam recovery points can help you make up a portion of the lost points, up to a score of 19.5
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm

• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast…)
  • How should I prepare? Read this Piazza post

• Go to lab today! Each lab is worth two points

• Go to discussion tomorrow! Each discussion is worth two exam recovery points
  • If you do poorly (< 20 points) on the midterm or final, exam recovery points can help you make up a portion of the lost points, up to a score of 19.5

• Details on cs61a.org/articles/about.html#discussion-participation
Announcements

• Do HW0! Due today (Wednesday, 6/22) at 11:59pm
• First quiz is tomorrow at the beginning of lecture (yes, this class moves fast...)
  • How should I prepare? Read this Piazza post
• Go to lab today! Each lab is worth two points
• Go to discussion tomorrow! Each discussion is worth two exam recovery points
  • If you do poorly (< 20 points) on the midterm or final, exam recovery points can help you make up a portion of the lost points, up to a score of 19.5
  • Details on cs61a.org/articles/about.html#discussion-participation
• Ask questions during lecture on Piazza! Read this post
Functions Review
Functions Review

• The operands of a call expression can be any expression
Functions Review

• The operands of a call expression can be any expression

• This includes expressions that evaluate to functions, such as function names!
Functions Review

- The operands of a call expression can be any expression

- This includes expressions that evaluate to functions, such as function names!

Interactive Diagram
Roadmap

- Introduction
- Functions
- Data
- Mutability
- Objects
- Interpretation
- Paradigms
- Applications
Roadmap

Introduction

• This week (Introduction), the goals are:
  • To learn the fundamentals of programming
  • To become comfortable with Python

Functions

Data

Mutability

Objects

Interpretation

Paradigms

Applications
Control
Control

- So far, our programs have included:
Control

• So far, our programs have included:
  • Expressions (call expressions in particular)
Control

- So far, our programs have included:
  - Expressions (call expressions in particular)
  - Assignment and def statements
Control

• So far, our programs have included:
  • Expressions (call expressions in particular)
  • Assignment and def statements

• But this is not enough to (easily) write most useful programs
Control

• So far, our programs have included:
  • Expressions (call expressions in particular)
  • Assignment and def statements

• But this is not enough to (easily) write most useful programs

• For example, how would you write a function that:
Control

• So far, our programs have included:
  • Expressions (call expressions in particular)
  • Assignment and def statements

• But this is not enough to (easily) write most useful programs

• For example, how would you write a function that:
  • Returns the absolute value of a number?
Control

• So far, our programs have included:
  • Expressions (call expressions in particular)
  • Assignment and def statements

• But this is not enough to (easily) write most useful programs

• For example, how would you write a function that:
  • Returns the absolute value of a number?
  • Returns the factorial of a number?
Control

- So far, our programs have included:
  - Expressions (call expressions in particular)
  - Assignment and def statements

- But this is not enough to (easily) write most useful programs

- For example, how would you write a function that:
  - Returns the absolute value of a number?
  - Returns the factorial of a number?

- These functions are easy to write if we introduce control
Control

• So far, our programs have included:
  • Expressions (call expressions in particular)
  • Assignment and def statements

• But this is not enough to (easily) write most useful programs

• For example, how would you write a function that:
  • Returns the absolute value of a number?
  • Returns the factorial of a number?

• These functions are easy to write if we introduce control
  • Special expressions and statements can control how the program is executed by the interpreter
Conditionals

if statements and Boolean operators
Conditional statements
Conditional statements (demo)
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    else:
        return x
def absolute_value(x):
    """Return the absolute value of x.""
    if x < 0:
        return -x
    else:
        return x

Syntax:

- Always starts with if clause.
- Zero or more elif clauses.
- Zero or one else clause, always at the end.
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    else:
        return x

Execution Rule for Conditional Statements:

Syntax:

- Always starts with if clause.
- Zero or more elif clauses.
- Zero or one else clause, always at the end.
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    else:
        return x

Syntax:
- Always starts with `if` clause.
- Zero or more `elif` clauses.
- Zero or one `else` clause, always at the end.

Execution Rule for Conditional Statements:
Each header is considered in order.
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    else:
        return x

Execution Rule for Conditional Statements:

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an else.
def absolute_value(x):
    """Return the absolute value of x.""
    if x < 0:
        return -x
    else:
        return x

Execution Rule for Conditional Statements:
Each header is considered in order.
1. Evaluate the header's expression, if the header is not an else.
2. If the expression is a true value or the header is an else,
def absolute_value(x):
    """Return the absolute value of x.""
    if x < 0:
        return -x
    else:
        return x

Execution Rule for Conditional Statements:

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an else.
2. If the expression is a true value or the header is an else, execute the suite & skip the remaining headers.

Syntax:
- Always starts with if clause.
- Zero or more elif clauses.
- Zero or one else clause, always at the end.
Boolean contexts
Boolean contexts

George Boole
Boolean contexts

George Boole
Boolean contexts

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

George Boole
Boolean contexts

George Boole

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

Execution Rule for Conditional Statements:

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.

2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.
Boolean contexts

George Boole

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

Execution Rule for Conditional Statements:

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.
2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.
Boolean contexts

George Boole

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

Execution Rule for Conditional Statements:
Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.
2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.
Boolean contexts

**George Boole**

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

**Execution Rule for Conditional Statements:**

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.
2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.

**False values in Python:**
Boolean contexts

**George Boole**

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

**Execution Rule for Conditional Statements:**

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.
2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.

**False values in Python:**

`False`, `None`, `0`, `0.0`, `''`, `[]`
**Boolean contexts**

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

**Execution Rule for Conditional Statements:**

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.

2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.

**False values in Python:**

False, None, 0, 0.0, '', []  (more to come)
Boolean contexts

George Boole

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

**Execution Rule for Conditional Statements:**

Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.
2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.

False values in Python: `False, None, 0, 0.0, '', []` (more to come)

True values in Python:
Boolean contexts

George Boole

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

Execution Rule for Conditional Statements:
Each header is considered in order.

1. Evaluate the header's expression, if the header is not an `else`.
2. If the expression is a true value or the header is an `else`, execute the suite & skip the remaining headers.

False values in Python:  `False, None, 0, 0.0, '', []`  (more to come)

True values in Python:  Everything else
Boolean expressions
Boolean expressions
Boolean expressions

- Expressions that contain special operators **and**, **or**, **not**
Boolean expressions

• Expressions that contain special operators \texttt{and, or, not}

• \texttt{not} \ <exp> \ evaluates to \texttt{True} if \ <exp> \ is a false value,
  \texttt{False} if \ <exp> \ is a true value
Boolean expressions

• Expressions that contain special operators \texttt{and}, \texttt{or}, \texttt{not}

• \texttt{not} <exp> evaluates to \texttt{True} if <exp> is a false value, \texttt{False} if <exp> is a true value

• Special \textit{short-circuiting behavior}: 
Boolean expressions

- Expressions that contain special operators \texttt{and}, \texttt{or}, \texttt{not}

- \texttt{not} \texttt{<exp>} evaluates to \texttt{True} if \texttt{<exp>} is a false value, \texttt{False} if \texttt{<exp>} is a true value

- Special \textit{short-circuiting behavior}:
  - \texttt{<left> and <right>} does not evaluate \texttt{<right>} if \texttt{<left>} evaluates to a false value
Boolean expressions

• Expressions that contain special operators **and**, **or**, **not**

• **not** `<exp>` evaluates to **True** if `<exp>` is a false value, **False** if `<exp>` is a true value

• Special *short-circuiting behavior*:
  • `<left> and <right>` does not evaluate `<right>` if `<left>` evaluates to a false value
  • `<left> or <right>` does not evaluate `<right>` if `<left>` evaluates to a true value
Boolean expressions

- Expressions that contain special operators `and`, `or`, `not`

  - `not <exp>` evaluates to True if `<exp>` is a false value, False if `<exp>` is a true value

- Special short-circuiting behavior:
  - `<left> and <right>` does not evaluate `<right>` if `<left>` evaluates to a false value
  - `<left> or <right>` does not evaluate `<right>` if `<left>` evaluates to a true value

- `0 and 1/0` evaluates to 0, `0 or 1/0` gives an error
Boolean expressions

- Expressions that contain special operators \texttt{and}, \texttt{or}, \texttt{not}

- \texttt{not} \texttt{<exp>} evaluates to \texttt{True} if \texttt{<exp>} is a false value, \texttt{False} if \texttt{<exp>} is a true value

- Special \textit{short-circuiting behavior}: 
  - \texttt{<left>} \texttt{and} \texttt{<right>} does not evaluate \texttt{<right>} if \texttt{<left>} evaluates to a false value 
  - \texttt{<left>} \texttt{or} \texttt{<right>} does not evaluate \texttt{<right>} if \texttt{<left>} evaluates to a true value

- \texttt{0 and 1/0} evaluates to \texttt{0}, \texttt{0 or 1/0} gives an error
Iteration

*while* loops, Sequences, and *for* loops
while loops
while loops (demo)
while loops

def factorial(n):
    """Return the factorial of n."""
    i, total = 1, 1
    while i < n:
        i += 1
        total *= i
    return total
while loops

```python
def factorial(n):
    """Return the factorial of n."""
    i, total = 1, 1
    while i < n:
        i += 1
        total *= i
    return total
```

Execution Rule for while Statements:
while loops

def factorial(n):
    """Return the factorial of n."""
    i, total = 1, 1
    while i < n:
        i += 1
        total *= i
    return total

Execution Rule for while Statements:
1. Evaluate the header’s expression.
while loops

def factorial(n):
    """Return the factorial of n."""
    i, total = 1, 1
    while i < n:
        i += 1
        total *= i
    return total

Execution Rule for while Statements:
1. Evaluate the header’s expression.
2. If it is a true value, execute the suite, then return to step 1.
Sequences and **for** loops
Sequences and **for** loops (demo)
Sequences and **for** loops (demo)

```python
def factorial(n):
    """Return the factorial of n."""
    total = 1
    for i in range(1, n+1):
        total *= i
    return total
```
Sequences and **for** loops

```python
def factorial(n):
    """Return the factorial of n.""
    total = 1
    for i in range(1, n+1):
        total *= i
    return total
```

Execution Rule for **for** Statements:
Sequences and **for** loops  

```python
def factorial(n):
    """Return the factorial of n."""
    total = 1
    for i in range(1, n+1):
        total *= i
    return total
```

**Execution Rule for **for** Statements:**

1. Evaluate the **sequence** in the header’s expression.
Sequences and **for** loops

```python
def factorial(n):
    """Return the factorial of n."""
    total = 1
    for i in range(1, n+1):
        total *= i
    return total
```

**Execution Rule for **for** Statements:**

1. Evaluate the **sequence** in the header’s expression.
2. For each value in the sequence, in order:
Sequences and **for** loops

```python
def factorial(n):
    '''Return the factorial of n.'''
    total = 1
    for i in range(1, n+1):
        total *= i
    return total
```

**Execution Rule for **for** Statements:**

1. Evaluate the **sequence** in the header’s expression.
2. For each value in the sequence, in order:
   1. Bind the **name** in the header’s expression to that value.
Sequences and **for** loops

```python
def factorial(n):
    """Return the factorial of n."""
    total = 1
    for i in range(1, n+1):
        total *= i
    return total
```

**Execution Rule for **for** Statements:**

1. Evaluate the **sequence** in the header’s expression.
2. For each value in the sequence, in order:
   1. Bind the **name** in the header’s expression to that value.
   2. Execute the **suite**
Summary
Summary

• *Control* allows the interpreter to selectively or repeatedly execute parts of our program
Summary

- **Control** allows the interpreter to selectively or repeatedly execute parts of our program.

- **Conditionals** allows for different behavior based on the input to and state of the program.
Summary

• *Control* allows the interpreter to selectively or repeatedly execute parts of our program

• *Conditionals* allows for different behavior based on the input to and state of the program
  • Using this, we wrote an absolute value function
Summary

• Control allows the interpreter to selectively or repeatedly execute parts of our program

• Conditionals allows for different behavior based on the input to and state of the program
  • Using this, we wrote an absolute value function

• Iteration allows for parts of our program to be repeatedly executed a specific number of times
Summary

- **Control** allows the interpreter to selectively or repeatedly execute parts of our program

- **Conditionals** allows for different behavior based on the input to and state of the program
  - Using this, we wrote an absolute value function

- **Iteration** allows for parts of our program to be repeatedly executed a specific number of times
  - Using this, we wrote a factorial function
Summary

- **Control** allows the interpreter to selectively or repeatedly execute parts of our program

- **Conditionals** allows for different behavior based on the input to and state of the program
  - Using this, we wrote an absolute value function

- **Iteration** allows for parts of our program to be repeatedly executed a specific number of times
  - Using this, we wrote a factorial function

- Putting it all together: let’s look at one more example
Summary

- Control allows the interpreter to selectively or repeatedly execute parts of our program.

- Conditionals allows for different behavior based on the input to and state of the program.
  - Using this, we wrote an absolute value function.

- Iteration allows for parts of our program to be repeatedly executed a specific number of times.
  - Using this, we wrote a factorial function.

- Putting it all together: let’s look at one more example.