CPSC 301: Computing in the Life Sciences
Lecture Notes 6:
Control Flow Statements

2017-01-31

Jessica Dawson
jqdawson@cs.ubc.ca
http://www.cs.ubc.ca/~jqdawson

University of British Columbia
Department of Computer Science

2016 W2
Admin

• Midterm date – confirmed for Feb 28 (in class)
  – Will cover everything up to and including Feb 16
  – Note that some topics we won’t have done labs on yet
    • But textbook / lecture content will be fair game
  – More details / practice materials in the coming weeks

• Lab 2 marks
  – Will be posted by Wednesday

• Lab 5 switching partners

• Jessica office hours rescheduled for this Thursday
  – 3-4pm instead of usual 11:30am – 12:30pm
Objectives

When you complete this section you should be able to:

• Read, trace, modify and develop programs that use relational operators, Boolean values and Boolean operators.
• Read, trace, modify and develop programs that make choices using `if`/`elif`/`else` conditional statements
• Use the `assert` statement to infer or enforce properties about data (and fail early if those properties are not true)
• Read, trace, modify and develop programs that use simple types of repetition: `for` and `while` loops
• Read, trace, modify and develop programs that use `break` and `continue` statements
• Read, trace, modify and develop programs that use combinations of all of the above, including nested loops and conditionals.
Python uses the key words **True** and **False** to represent the two Boolean values.

Can compare numbers and strings with the following relational operators (like in Scratch). Each comparison returns True or False:

- $>, <, \geq, \leq, ==$ (equal), $!=$ (not equal)
- $5 < 3$ is False, $5.5 \leq 20$ is True, 'hello' $\leq$ 'world' is True,
  'hello' $!=$ 'world' is True, $5 == (2+3)$ is True
Boolean Operators

- Boolean operators: **and**, **or**, and **not**
  - a **and** b : True if a and b are both True; False otherwise
  - a **or** b : False if a and b are both False; True otherwise
  - **not** a : True if a is False and False if a is True

- Can combine Boolean operators with relational operators (like Scratch) Example:
  - if x is 5 and y is 10 then  x >= y or y == 10 is True
  - if x is 1 and y is -1 then  x >= y and y == 10 is False
Number and Strings as Boolean Values

• In Python we can apply boolean operators to numbers and strings
• Python converts other values to Booleans as following:
  – 0, 0.0, the empty string, and None are treated as False
  – any other value is treated as True
• Examples:
  – 0 and True is 0
  – 0 and 3 is 0
  – 0 or 3 is 3
• Nevertheless, in order to make programs easy to understand we should avoid mixing different types of values that don't make sense.
  – We generally recommend you avoid using statements like the above.
More on Boolean Values

- We can check if a string is a substring of another string using the `in` operator:
  - "gc" in "aaggagacgccccat" is True
  - "ggcc" in "aaggagacgccccat" is False

- We can assign Boolean values to variables
  - `speed = 100`
  - `high = speed > 100` (then high is False)

- We can define functions that return Boolean values:
  ```python
def between(value, low, high):
    """ returns True if value is strictly between low and high""
    return low < value < high
```

```
val = 15
between(val, -20, 20)    # returns True
between(val, -20, 15)    # returns False
```
Boolean Operation Gotchas

- Test for equality \(==\) is different from assignment \(=\)
  - \(==\) is a *relational operator* forming part of an *expression* that evaluates to *True* or *False*
  - \(=\) is an *assignment operator* forming part of a *statement* which assigns a value to a variable; statements have no value of their own

- Be careful about *operation precedence*
  - If in doubt, just use parentheses!

- Recommend you avoid combining comparisons
  - eg: use \(0 < x \text{ and } x <= 42\), avoid \(0 < x <= 42\)

- Be careful about comparing strings
  - eg: 'XYZ' < 'abc', 'a*c' < 'abc', 1 < 'a'
  - String comparison is obvious as long as the strings are all of a single case (upper or lower) and contain no numbers or symbols
Control Flow Statements

- Python has similar control structures (and several more) like those we have seen in Scratch
  - Choice statements like `if/elif/else`
  - Loops like `for–loop, while-loop`
- Each of these statements always ends with a `:` (colon) followed by a block of statements (also called compound block)
- Statements within the compound block are indented
  - All statements in the same block use the same indentation (typically 4 spaces) unless there are sub blocks
  - Good style: use the same amount of indentation throughout a file, and indent comments to match the statements
  - Compound block ends when indentation ends
- Indentation provides a visual cue similar to Scratch’s clamp shaped control blocks
  - In Python, correct indentation is mandatory
  - In most other programming languages, compound statements are explicitly delimited and indentation is a style choice
Making Choices

• Form of an if-statement:
  
  ```python
  if condition:
      block
  
  or
  ```

  ```python
  if condition:
      if-block
  else:
      else-block
  ```

  where each block is a sequence of statements which are indented

• `condition` can be a boolean or any other value. In the second case:
  – a value 0, 0.0, None and '""' (the empty string) are treated as False
  – any other value is treated as True

• Example:
  
  ```python
  if total_purchase >= 1000:
      print("You are eligible for 20% discount")
  else:
      print("You are eligible for 10% discount")
  ```
Nested if-Statements

Use `elif` to combine `else` and `if` statements

Instead of:

```python
def standing(grade):
    if grade >= 80:
        return "1st class"
    else:
        if grade >= 65:
            return "2nd class"
        else:
            if grade >= 50:
                return "pass"
            else:
                return "fail"
```

We write:

```python
def standing(grade):
    if grade >= 80:
        return "1st class"
    elif grade >= 65:
        return "2nd class"
    elif grade >= 50:
        return "pass"
    else:
        return "fail"
```