Quiz Today

Covers:

- Concepts from lab 2
- Material from lecture 2
- Reading from:
  - Model Chap 2 & 3, pages 21-23, 47-71

Time:
Programming for Biologists and Biochemists

Lecture 3:
Control Flow

Brian Kidd
April 12, 2010
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
From Last Time

Get on the Class email List:

  Currently 57 / 57 signed up, thanks!

Grades:

  We will post answer keys to the labs and quizzes, which will include a point break down

  Read the key before asking questions about your grade
Lab 2 Feedback

Overall people did well!

*Drop box versus assignment*

*Follow instructions, you have the tools*

*No need to do things that aren't asked*

*Parsing with split()*

*Slicing*

*Outline your code before*
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
Simple Statements

So far we've discussed simple statements

Examples:
- **expressions**: +, -, *, /, ***, etc.
- **assignments**: =, +=, *=, etc.
- **print**
- **function & method calls**: len(), .count()

Limited control over commands, why?
Limited Control

Given your current knowledge of Python, how do you deal with the following?

- Testing conditions
- Repeating an operation, statement or function
- Going through all elements in a collection
- Checking for errors and handling exceptions
Compound Statements*

Control the order (or number of times) in which statements in your code are executed

This concept of determining the order of statements is called **control flow** or **flow of control** and compound statements are often called **control statements**

*http://docs.python.org/reference/compound_stmts.html*
Compound Statements

- **keyword1:**
  - statement1a
  - statement1b

- **keyword2:**
  - statement2a
  - statement2b
  - statement2c

- **suite (block)**
- **clause**
- **compound statement**
Examples

Conditionals (decision points)
  if

Loops (cycles)
  while

Iterations (specified cycles)
  for

Exception Handlers (anticipate problems)
  try
Python Syntax Rules I

End of line is end of statement

x = 3.4
print "Hello World"

End of indentation is end of block

keyword1:
    print "Inside block"
    y = 3.14159

print "Outside of block"
Python Syntax Rules II

Special Cases

More than one statement per line with ;

```
a = 1; b = 2; c = 3; print a + b
```

Multiline statements with enclosure, i.e. (), [], \ 

```
mlist = [111,
        222]
x = a + b + \
c
```

Single block line

```
if b > a: print b
```
Python Syntax Rules III

Statements execute one after another, until you say otherwise

Block and statement boundaries are automatically detected

Compound statements follow prototype

\[
\text{header:} \\
\text{indented statements}
\]

Blank lines*, spaces**, and comments are usually ignored

Docstrings are ignored but saved and displayed by tools

*Terminate compound statements at prompt
**Strings and indentation.
Suggested Syntax Guidelines

Code is read much more often than it is written*

Adopt a coding style that:

- **is consistent**
  (within modules > within a project)

- **is easy to read**

- **follows some basic guidelines****

*Guido van Rossum

**http://www.python.org/dev/peps/pep-0008/
Suggested Syntax Guidelines

#!/usr/bin/env python

# Name: Brian Kidd (bkidd)
# Group: None
#
# Desc: python coding style examples
# Inputs: none
# Outputs: none

x = 1
y = (1, 2, 3)
if x > 0:
    if y[1] > 1:
        print "Double positive"
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
Conditional Statements

Evaluate the truth of an expression

They ask a question (or a series of questions) to which the answer is either yes or no

Based on the truth of the expression, certain statements are executed
Simple Conditional

Evaluate the truth of a single statement

Basic format of an if statement*

if expression:
    do something if and only if expression is true

Test syntax

equal: ==
not equal: !=
inequalities: >, >=, <, <=

if a == b:
if a != b:
if a > b:

*Code block based on indentation.
Simple Conditional

Define two numeric variables \( x \) and \( y \)

\[
\begin{align*}
x &= 1 \\
y &= 2
\end{align*}
\]

if \( x == y \): \( \times \)
if \( x != y \): \( \checkmark \)
if \( x > y \): \( \times \)

Define two string variables \( a \) and \( b \)

\[
\begin{align*}
a &= "Brian"; \\
b &= "brian";
\end{align*}
\]

if \( a == b \): \( \times \)
if \( a != b \): \( \checkmark \)
if \( a < b \): \( \checkmark \)
One-Alternative Conditional

When you need an alternative, use the if-else blocks

Basic format of an IF-ELSE statement

```python
if expression:
    do something if expression is true
else:
    do something if expression is NOT true
```
One-Alternative Conditional

Define two numeric variables \( x \) and \( y \)

\[
\begin{align*}
x &= 1 \\
y &= 2
\end{align*}
\]

```python
if x == y:
    print "match found"  # ✗
else:
    print "match not found"  # ✔
```

Define two string variables \( a \) and \( b \)

\[
\begin{align*}
a &= "GAATTC"; \\
b &= "GAATTC"
\end{align*}
\]

```python
if a == b:
    print "match found"  # ✔
else:
    print "no match"  # ✗
```
Multi-Test Conditional

When you need multiple alternatives, use the *if-elif-...-else* set of blocks (clauses)

Basic format of an IF-ELIF-ELSE set of blocks

```python
if expression1:
    do something if *expression1* is true
elif expression2:
    do something if *expression2* is true
else:
    do something if both are NOT true
```
Multi-Test Conditional

Define two numeric variables $x$ and $y$

```
x = 1
y = 2
if x == y:
    print "values equal"
elif x > y:
    print "%s > %s" % (x, y)
else:
    print "%s < %s" % (x, y)
```
Graphical View of Control Flow

Control jumps around blocks

if expression1:
    do something if expression1 is true
elif expression2:
    do something if expression2 is true
else:
    do something if both are NOT true
Complex Conditionals

You can create any series of logical comparisons you want, with arbitrary complexity and nesting

```python
if expression1:
    statements1
if expression2:
    statements2
elif expression3:
    statements3
if expression4:
    statements4
```

- ✔ exp1 true
- ✔ exp1 and exp2 true
- ✔ exp1 and exp3 true
- ✔ exp1 and exp3 and exp4 true
Logical Operators

<table>
<thead>
<tr>
<th>and</th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Python operator syntax

and: `and`    
```
a = 1; b = 2; c = 2;
if a == b and b == c:    False
```

or: `or`     
```
if a == b or b == c:    True
```

not: `not`    
```
if not a and b > 0:    False
```
Conditional Exercises
Quick Exercise I

Read in a word from the command line and use conditional statements to:

determine whether the word is 'dna', 'rna', 'protein', or 'other'

Example:
  ask person for word
  read in word
  determine if word is a biological molecule or not
  print result to screen
#!/usr/bin/env python
#
# Name: Brian Kidd (bkidd)
# Group: everyone in class
# Desc: determine whether or not inputed word is one of three words: 'dna', 'rna', or 'protein'
#
# Input: string
# Output: the word that was found (dna, rna, protein, or other)

word = raw_input("enter a word: ")

if word == 'dna':
    print "found dna"
elif word == 'rna':
    print "found rna"
elif word == 'protein':
    print "found protein"
else:
    print "found something that's not recognized"
Quick Exercise II

Read in a sequence from the command line and use conditional statements to:

determine whether the sequence is 'dna', 'rna', or 'other'

Example:

ask person for sequence
read in sequence
determine what type of biological molecule the sequence comes from
print result to screen
#!/usr/bin/env python

# Name: Brian Kidd (bkidd)
# Group: everyone in class
# Desc: Determine whether an input sequence is of type dna or rna or neither

# Input: a string of only letters
# Output: print out what type of sequence this is: dna, rna or neither
# Assumptions:
#   only letters coming in and only contain (A,T,C,G,U)

# grab input sequence and set to lowercase
sequence = set(raw_input("Enter a sequence: ").lower())

#
if 't' in sequence:
    print 'found dna'
elif 'u' in sequence:
    print 'found rna'
else:
    print "can't tell what type of sequence this is"
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
Loops

A loop is a block of statements that are executed as long as some condition is true

Repeat the execution within the indentation block

Types:

while: prototypic loop
while-else: loop with a final clause
while-break: loop with interruption
Basic Scheme of Loop

Loops are blocks of code that execute again and again and again...*

1. Often require initialization

2. Often require updating a variable for evaluation**

*Loops that never end are called infinite loops.

**Forgetting to update can lead to infinite loops!
Simple Loop

Use a while loop to repeat a task as long as a condition is true

```python
while expression:
    statements and update
```

Example:

```python
password = "open sesame"
guess = raw_input("enter password: ")
while guess != password:
    guess = raw_input("wrong password, try again: ")
print "access granted, welcome"
```
Loop with Final Clause

Add an `else` clause that is executed after while loop evaluates to false

```
while expression:
    statements1 and update
else:
    statements2
```

Not a common construct, but useful if you want a block of code to be run after the while is false, but will be skipped because of a `break` or an `exception`
Loop Interruption I

Occasionally it is useful to interrupt* the execution of a loop's statement

while expression:
    statements
    conditionals
    break or return

*Interrupt statements can be used in iteration too!
Loop Interruption II

The execution of a loop will stop in one of three ways:

1. **Normally**: test evaluates to false

2. **Abnormally**: an error occurs in the evaluation of the test or the body of the loop

3. **Prematurely**: the body of the loop executes a `return` or `break` statement

When you write a loop, plan for 1 or 3

make sure condition is updated during loop execution

ctrl-c aborts the program
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
Types of Collections

Sequences – ordered collection

- **strings** e.g. "VLSPADKTN", **lists** e.g. ["EcoR1", "BamHI", "Xbal"],
- **ranges** e.g. range(0, 10), **tuples** e.g. (x, y, z)

Sets – unordered, unique collection

bases of DNA or RNA, collection of IDs, etc.

Mappings – unordered collection of key/value pairs

- codon tables, 3to1 letter tables, restriction enzyme maps

Collections contain objects
Collections Contain Objects

Some built-in functions work for any type of collection

\[ \text{min, max, any, all, in, not in, len} \]

The way these functions tend to work is they consider every element in the collection

**Iteration** is the process of doing something to each element of collection
Iteration

Use a `for` statement to iterate through every element in a collection

Basic format of a `for` statement*

```python
for item in collection:
    do something with item
```

*Simple, yet powerful statement.
Remember that break statements work with iterations too.
Iteration Example

Use a for statement to iterate through every element in a collection

Example:

```python
aaSeq = list('VLSPADKTNVKAAGWGVGA')
for aa in aaSeq:
    print aa,
```
Iteration Templates I

Iterations are used extensively in Python*

Examples:

```python
# Dictionary Iterations
for key in dictionary.keys():
    do something with key
for key, value in dictionary.items():
    do something with key and value

# Numbering Iterations
# Numbering Iterations
for n, value in enumerate(iterable):
    do something with n and value
```

*Chapter 4, pages 112-131.
Iteration Templates II

Examples:

```python
# Repeat
for count in range(n):
    statements
# Collect
result = []
for item in collection:
    statements using item
    result.append()
# search
for item in collection:
    if test item:
        do something
```

*Chapter 4, pages 112-131.*
Iterations Exercises
Quick Exercise III

Read in a DNA sequence and determine the position of each T in the sequence

Example:

ask person for a dna sequence
read in sequence
iterate each position and keep track of Ts
print results to screen
#!/usr/bin/env python
#
# Name: Brian Kidd (bkidd)
# Group: everyone in class
# Desc: find position of Ts in a DNA sequence
#
# Input: dna sequence
# Output: locations of Ts

sequence = raw_input("enter sequence: ").upper()

# don't store positions
for i, letter in enumerate(sequence):
    if letter == 'T':
        print i + 1

i = 0
while i < len(sequence):
    if sequence[i] == 'T':
        print i + 1
    i += 1

for i in range(len(sequence)-1):
    if sequence[i] == 'T':
        print i + 1
Full Problem Example
Assignment Description

# protein sequence analysis program:
# this exercise reads in a protein sequence string
# and counts the number of lysine residues (K),
# making sure to keep track of the location
# (position) within the sequence.
#
# the first letter (amino acid) of the sequence
# is position number 1 (no assumptions about case)

HBA_seq.txt

VLSPADKTNVKAAWGKVGAHAGEYGAEEALERMFLSFPPTTKTYPFPFHDLSH
GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKL
LSHCLLVTLAHLPAEFTPAVHASLKDKFLASVSTVLTSDKYR

test_seq.txt

VLSPADKTNVKAAW
Assignment Specifications

# read in a protein sequence file
# count the number of lysine residues (K)
# track position of lysine residues in sequence
# print positions of lysines in sequence
# print total number of lysines in sequence
#
# first letter (amino acid) of the sequence is position 1
# no assumptions about case

Information flow: input(s) ➔ process ➔ output(s)

Input(s):
  protein sequence, sequence starts at 1

Processing:
  count lysines, track position, counter, step through seq

Output(s):
  total count of lysines, position
Specifications to Pseudocode

# read in a protein sequence file
# count the number of lysine residues (K)
# track position of lysine residues in sequence
# print positions of lysines in sequence
# print total number of lysines in sequence
#
# first letter (amino acid) of the sequence is position 1
# sequence will be all uppercase letters

Read in Data
- Get sequence data
- Read into variable

Process Step 1
- Change case
- Container for something (depends on design)

Process Step 2
- Sequence iteration
- Compare each letter to "K"
- Add to count

Print results
- Print out positions
- Print total count

input(s) ➔ process ➔ output(s)
Pseudocode to Code
Verify Code

1. Eliminate syntax and definition errors*
2. Use test cases to check code logic**
3. Write down expected results
4. Run your program and check

<table>
<thead>
<tr>
<th>test_seq.txt</th>
<th>HBA_seq.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLSPADKTNVKAAW</td>
<td>VLSADKTNVKAAGKVGAVHAGEYGAEEALERMFLSFPPTTKTYFPFHDLSH</td>
</tr>
<tr>
<td>K @ 7, 11</td>
<td>K @ 7, 11, 16, 40, 56, 60, 61, 90, 99, 127, 139</td>
</tr>
<tr>
<td>total = 2</td>
<td>total = 11</td>
</tr>
</tbody>
</table>

*Python interpreter.
**IDLE debugger.
What to Remember

1. Convert description to specifications
2. Convert specifications to pseudocode
3. Convert pseudocode to perl code
4. Debug code (easy and hard way)
5. Verify code against test cases

Express Test Cycle
Overview

1. Housekeeping Items and Questions
2. Statement Review and Control Flow
3. Conditionals
4. Loops
5. Iterations
6. Exception Handlers
Python Errors

In the process of executing Python code, everyone is going to run into some sort of error

Tracebacks: "trace back" through the code to display the preceding outputs and functions leading up to the error

Runtime errors: problem that occurs during the execution of the code
Examples of Runtime Errors

Wrong inputs for your code

Comparison of incompatible values

Wrong outputs from your code

NameError - name 'x' not defined
ValueError - list.index(x): x not in list
KeyError - 'b'
AttributeError - 'y' object has no attribute 'x'
TypeError - 'x' expected 1 arguments, got 0
IndexError - string index out of range
IOError - No such file or directory: ''
Exception Handling Statements

Adding statements to your code to handle "errors" (exceptions) to what is expected

```python
try:
    statements
except ErrorClass:
    except-statements
```

Exception handling allows you to anticipate and recover from expected errors
try:
    statements
except ErrorClass:
    except-statements

inSeq = raw_input("Please enter a sequence: ")
try:
    inSeq.index('M')
except ValueError:
    print 'M not found in %s' % inSeq
Homework

Reading:
  Model:
    Chapters 2, 3, & 6 pages 24-46, 72-79, 226-237

Quiz:
  Quiz 3 next Tues (4/19)
  Covers readings, lectures, and lab
Lab 3 Preview

Control of flow in python (6 programs)

1. count start codons

2. calculate protein properties (multi-part)

3. transcribe and translate

Username/password: bme60/bme60

*remember to read the notes and hints document!*