Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza
Boolean Operators
Boolean Operators

And, Or, Not
True and False are Python values!

Boolean Operators
And, Or, Not
if expression1 and expression2:
  statement
  statement
  statement
  statement
else:
  statement
  statement
  statement
  statement
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement

True
True
if `expression1` and `expression2`:
    statement
    statement
    statement
else:
    statement
    statement
    statement

True

True

True
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
    statement
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
    statement
if `expression1 and expression2`:
    statement
    statement
    statement
else:
    statement
    statement
    statement
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
    statement
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement

True  True
True  False
False  True
False  False
if expression1 or expression2:
  statement
  statement
  statement
else:
  statement
  statement
  statement
if not expression:
    statement
    statement
    statement
else:
    statement
    statement
    statement

False

True
else is not always used...

```python
if no20s == 1:
    print( no20s, "\$20-bill" )
else:
    print( no20s, "\$20-bills" )
```
else is not always used...

caption = "\$20-bill"
if no20s != 1:
    caption = caption + "s"

print( no20s, caption )
Outline

Boolean Operators

Exercises

**If Statements and Graphics**

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza
If-Statements and Graphics
Where are Graphic Objects Defined?

Zelle's Graphics.py for Python 3

--D. Thiebaut (talk) 11:12, 8 March 2015 (EDT)

The file below, copyrighted by John Zelle, was downloaded from http://mcsp.wartburg.edu/zelle/python/graphics.py on 3/8/15, and mirrored here for convenience.

```
# graphics.py
"""Simple object oriented graphics library

The library is designed to make it very easy for novice programmers to experiment with computer graphics in an object oriented fashion. It is written by John Zelle for use with the book "Python Programming: An Introduction to Computer Science" (Franklin, Beedle & Associates).

LICENSE: This is open-source software released under the terms of the GPL (http://www.gnu.org/licenses/gpl.html).

PLATFORMS: The package is a wrapper around Tkinter and should run on any platform where Tkinter is available.

INSTALLATION: Put this file somewhere where Python can see it.

OVERVIEW: There are two kinds of objects in the library. The GraphWin class implements a window where drawing can be done and various GraphicsObjects are provided that can be drawn into a GraphWin. As a simple example, here is a complete program to draw a circle of radius 10 centered in a 100x100 window:
```

http://cs.smith.edu/dftwiki/index.php/Zelle%27s_Graphics.py_for_Python_3
Every element is an OBJECT

Examples of If-Statements in Graphics

Organization of a graphic program

Something completely different...
Examples of If-Statements in Graphics
p = Point( 50, 150 )

point1 = Point( 50, 150 )

x = point1.getX()
y = point1.getY()
if x <= 0 or y <= 0:
    # the point is outside the window
    ...

p.getX()
point1 = Point( ..., ... )
circ1 = Circle( point1, ... )

center1 = circ1.getCenter()
x = center1.getX()
y = center1.getY()
if x <= 0 or y <= 0:
    # the center is outside the window
    ...

Circle Object

getCenter()

Circle

50 number

Point Object

ggetX()
point1 = Point( 50, 150 )
circ1 = Circle( point1, 30 )

center1 = circ1.getCenter()
x = center1.getX()
y = center1.getY()
if x <= 0 or y <= 0:
    # the center is outside the window
    ...

x = circ1.getCenter().getX()
y = circ1.getCenter().getY()
if x <= 0 or y <= 0:
    #
Rectangle Object

\[
\begin{align*}
&\text{P1} \quad \text{P2} \\
&\text{getP1()} \\
&\text{getP2()}
\end{align*}
\]

Point Object

\[x, y\]

\[50\] number

\[
\begin{align*}
&\text{r = Rectangle( Point( 50, 150 ),} \\
&\quad \text{Point( 150, 150 ) )} \\
&r.\text{move( dx, dy )} \\
&x_1 = r.\text{getP1().getX()} \\
&y_1 = r.\text{getP1().getY()} \\
&x_2 = r.\text{getP2().getX()} \\
&y_2 = r.\text{getP2().getY()} \\
&\text{mouseP = win.checkMouse()} \\
&\text{if mouseP != None:} \\
&\quad x = \text{mouseP.getX()} \\
&\quad y = \text{mouseP.getY()} \\
&\quad \text{if } x_1 \leq x \leq x_2 \text{ and } \ldots
\end{align*}
\]
Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza
Organization of a Graphic Program
def main():
    # open the graphic window

    # define and initialize the graphic objects

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed
        # and direction

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects

    # start animation loop. Stop on specific user interaction

    # move/update each object according to its speed
    # and direction

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    
    # move/update each object according to its speed
    # and direction

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed
        # and direction

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed and direction
        circ.move( dx, dy )

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user interaction
    while win.checkMouse() == None:
        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )

    # Loop is over.
    # close the graphic window
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill('red')
    circ.draw(win)
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move(dx, dy)

    # Loop is over.
    # close the graphic window
    win.close()
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop. Stop on specific user interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )

    # Loop is over.
    # close the graphic window
    win.close()
Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza
Measuring Distances
\[ d^2 = a^2 + b^2 \]
\[ d^2 = a^2 + b^2 \]
\[ d^2 = a^2 + b^2 \]
\[ d^2 = a^2 + b^2 \]
\[ d = \sqrt{ (y_2-y_1)^2 + (x_2-x_1)^2 } \]
```python
from math import *

def distance(x1, y1, x2, y2):
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2))
```
```python
from math import *

def distance(x1, y1, x2, y2):
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2))

def distanceP(p1, p2):
    x1, y1 = p1.getX(), p1.getY()
    x2, y2 = p2.getX(), p2.getY()
    return distance(x1, y1, x2, y2)
```
Testing
```python
# distanceDemo.py
# D. Thiebaut

from math import *
from graphics import *

def distance( x1, y1, x2, y2 ):
    return sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) )

def distanceP( p1, p2 ):
    x1, y1 = p1.getX(), p1.getY()
    x2, y2 = p2.getX(), p2.getY()
    return distance( x1, y1, x2, y2 )

def main():
    point1 = Point( 3, 5 )
    point2 = Point( 7, 8 )
    d = distanceP( point1, point2 )
    print( "distance =", d )

main()
```
Graphics: Detecting Obstacles
Exercise: Obstacle

Take the graphic program moving a circle around, and create an obstacle.
We stopped here last time...
CSC111: Amount of Work

Level of Difficulty vs. Semester

Your level
Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza
Concepts to Explore
With the Bouncing Ball

• Do this on your own (not in the lab… The lab is dense this week!)

• Multiple balls

• Balls lose energy every time they hit a wall

• Balls lose energy as they move around
Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza
Eliza
The Turing Test

Turing 1912-1954

Dr Suilin Lavelle
University of Edinburgh
Turing

The Imitation Game
Blade Runner
Eliza

- Example of Natural Language Processing (NLP)
- MIT, 1964, Joseph Weizenbaum
- One of the first chat-bots (Amazon Alexa)
- Emulates a Rogerian psychotherapist
- Example of dialogs: https://web.stanford.edu/group/SHR/4-2/text/dialogues.html
# Eliza1.py
# D. Thiebaut
# A very short beginning program for Eliza
# just print the string to the console
# will be transformed to something better later...
def myprint( string ):
    print( string )

def sayGoodBye( name ):
    myprint( "Good bye " + name )

def isGoodBye( userAnswer ):
    if userAnswer.lower().strip() in [ "bye", "goodbye", "ciao" ]:
        return True
    else:
        return False

def greetings():
    myprint( "Hello there!" )
    myprint( "What is your name?" )
    name = input( "> " )
    myprint( "Welcome " + name )
    return name

# greet user and get her name
userName = greetings()

# conversation: get user input, and respond
for i in range( 1000 ):
    # get user's statement
    userAnswer = input( "> " )

    # if it is a goodbye statement, exit the loop
    if isGoodBye( userAnswer ) == True:
        break

    # tell the user to continue speaking
    myprint( "Please tell me more..." )

# if we're here, it's because the loop stopped.
# say goodbye to the user
sayGoodBye( userName )
main()
• Use the random library

```python
from random import *
...

prompts = ["Please go on...", "Please tell me more...",
            "Interesting... Go on, please!",
            "Yes? Really? Go on",
            "Weird... I'm not sure what to think of that..."
            ]
...

myprint( choice( prompts ) )
```
Looking for String Patterns
The Problem

User types: "I had a HUGE fight with my brother"

Program knows: ["mother", "father", "brother", "sister"]
"I had a HUGE fight with my brother"

```python
split()
```

```python
["I", "had", "a", "huge", "fight", "with", "my", "brother"]
```

```python
in
```

```python
["mother", "father", "brother", "sister"]
```
"I had a HUGE fight with my brother"

```
[ "I", "had", "a", "huge", "fight", "with", "my", "brother" ]
```

Option 1
"I had a HUGE fight with my brother"

Option 1

[ "I", "had", "a", "huge", "fight", "with", "my", "brother" ]

in

[ "mother", "father", "brother", "sister" ]
"I had a HUGE fight with my brother"
"I had a HUGE fight with my brother"

family = ["mother", "father", "brother", "sister"]

userInput = input("> ")
words = userInput.lower().split()

familyMatter = False
for word in words:
    if word in family:
        familyMatter = True

if familyMatter == True:
    doSomething()
"I had a HUGE fight with my brother"

[ "mother",
  "father",
  "brother",
  "sister" ]
"I had a HUGE fight with my brother".find

["mother",
"father",
"brother",
"sister"]
"I had a HUGE fight with my brother"

```
[ "mother",
  "father",
  "brother",
  "sister"
]
```
"I had a HUGE fight with my brother"

```javascript
.find(['mother', 'father', 'brother', 'sister'])
```
"I had a HUGE fight with my brother"

family = ["mother", "father", "brother", "sister"]

userInput = input( "> " ).lower()

familyMatter = False
for word in family:
    if userInput.find( word ) != -1:
        familyMatter = True

if familyMatter == True:
    doSomething()
Ways to Make Eliza Program Smarter

- Respond to "No", "Never", "Nope" with a different answer
- Detect "I xxx you" and respond with "You xxx me?"
- Add generated "You xxx me?" to canned answers
We stopped here last time...
Indefinite Loops
(Chapter 8)
Reviewing For-Loops

Applications

While Loops for Robustness

Break & Continue
For-Loops

Items: [ dog, cat, horse, hen, pig ]
For-Loops

Items: [ dog, cat, horse, hen, pig ]

```python
list = [ dog, cat, horse, hen, pig ]
for x in list:
    process( x )
```
For-Loops

Items: [ dog, cat, horse, hen, pig ]

Index Generator
For-Loops

Items: [ dog, cat, horse, hen, pig ]
For-Loops

Items: [ dog, cat, horse, hen, pig ]

```
list = [ dog, cat, horse, hen, pig ]
for i in range( 1, len(list), 3):
    x = list[i]
    process( x )
```
For loops in context

Applications

While Loops for Robustness

Break & Continue
Applications
Count Matching Items

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, cat, pigeon, hen, sheep ]
Count Matching Items

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, cat, pig, hen, sheep ]

Exact Place Matching
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

items1 = [ dog, cat, horse, hen, pig ]
items2 = [ dog, cat, pig, hen, sheep ]

count = 0
for i in range( len( items1 ) ):
    if items1[i] == items2[i]:
        count += 1
What if the lists do not have the same length?

```python
#           0    1    2      3    4
items1 = [ dog, cat, horse, hen, pig ]
items2 = [ dog, cat, pig ]

count = 0
for ???:
    if items1[i]==items2[i]:
        count += 1
```
What is the risk?
What could go wrong?
What kind of error?
What is the risk?
What could go wrong?
What kind of error?
What if the lists do not have the same length?

```python
#   0  1  2  3  4
items1 = [ dog, cat, horse, hen, pig ]
items2 = [ dog, cat, pig ]

count = 0
for i in range( len( items1 ) ):
    if items1[i]==items2[i]:
        count += 1
```
items1 may not be the longest list

```python
#           0    1    2    3    4
items1 = [ dog, cat, horse]
items2 = [ dog, cat, pig, hen, pig ]

count = 0
for i in range( len( items1 ) ):
    if items1[i]==items2[i]:
        count += 1
```
2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here:

<table>
<thead>
<tr>
<th>abs()</th>
<th>dict()</th>
<th>help()</th>
<th>min()</th>
<th>setattr()</th>
</tr>
</thead>
<tbody>
<tr>
<td>all()</td>
<td>dir()</td>
<td>hex()</td>
<td>next()</td>
<td>slice()</td>
</tr>
<tr>
<td>any()</td>
<td>divmod()</td>
<td>id()</td>
<td>object()</td>
<td>sorted()</td>
</tr>
<tr>
<td>ascii()</td>
<td>enumerate()</td>
<td>input()</td>
<td>oct()</td>
<td>staticmethod()</td>
</tr>
<tr>
<td>bin()</td>
<td>eval()</td>
<td>int()</td>
<td>open()</td>
<td>str()</td>
</tr>
<tr>
<td>bool()</td>
<td>exec()</td>
<td>instanceof()</td>
<td>ord()</td>
<td>sum()</td>
</tr>
<tr>
<td>bytearray()</td>
<td>filter()</td>
<td>issubclass()</td>
<td>pow()</td>
<td>super()</td>
</tr>
<tr>
<td>bytes()</td>
<td>float()</td>
<td>iter()</td>
<td>print()</td>
<td>tuple()</td>
</tr>
<tr>
<td>callable()</td>
<td>format()</td>
<td>len()</td>
<td>property()</td>
<td>type()</td>
</tr>
<tr>
<td>chr()</td>
<td>frozenset()</td>
<td>list()</td>
<td>range()</td>
<td>vars()</td>
</tr>
<tr>
<td>classmethod()</td>
<td>getattr()</td>
<td>locals()</td>
<td>repr()</td>
<td>zip()</td>
</tr>
<tr>
<td>compile()</td>
<td>globals()</td>
<td>map()</td>
<td>reversed()</td>
<td><strong>import</strong>()</td>
</tr>
<tr>
<td>complex()</td>
<td>hasattr()</td>
<td>max()</td>
<td>round()</td>
<td></td>
</tr>
<tr>
<td>delattr()</td>
<td>hash()</td>
<td>memoryview()</td>
<td>set()</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
abs(x)
```

https://docs.python.org/3.4/library/functions.html
# 0 1 2 3 4

```python
items1 = ['dog', 'cat', 'horse']
items2 = ['dog', 'cat', 'pig', 'hen', 'pig']

count = 0
len1 = len(items1)
len2 = len(items2)
for i in range(min(len1, len2)):
    if items1[i] == items2[i]:
        count += 1
```
Applications, #2
Count Matching
Misplaced Items

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ cat, pig, pigeon, hen, dog ]
Count Matching Misplaced Items

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]
Algorithm

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items1:</td>
<td>dog, cat, horse, hen, pig</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Items2:</td>
<td>dog, pig, pigeon, hen, cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]
Algorithm

Items1: [dog, cat, horse, hen, pig]

Items2: [dog, pig, pigeon, hen, cat]

count: 1
Algorithm

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]

count: 1
Algorithm

\[
\begin{align*}
\text{Items1: } & \{\text{dog, cat, horse, hen, pig}\} \\
\text{Items2: } & \{\text{dog, pig, pigeon, hen, cat}\}
\end{align*}
\]

\text{count: 1}
Algorithm

$i$

0  1  2  3  4
Items1: [ dog, cat, horse, hen, pig ]

0  1  2  3  4
Items2: [ dog, pig, pigeon, hen, cat ]

count: 1
Algorithm

Items1: [dog, cat, horse, hen, pig]

Items2: [dog, pig, pigeon, hen, cat]

count: 1
Algorithm

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]

count: 1
Algorithm

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]

count: 1
Algorithm

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]

count: 1
Algorithm

Items1: [ dog, cat, horse, hen, pig ]

Items2: [ dog, pig, pigeon, hen, cat ]

count: 1
Algorithm

count: 1
Algorithm

Items1: [dog, cat, horse, hen, pig]

Items2: [dog, pig, pigeon, hen, cat]

count: $\not/2$
items1 = [ dog, cat, horse, hen, pig]
items2 = [ dog, pig, pigeon, hen, cat]

count = 0

for i in range( len( items1 ) ):
    for j in range( len( items2 ) ):
        if items1[i]==items2[j]:
            count += 1
For loops in context

Applications

**While Loops for Robustness**

Break & Continue
# get a positive number from user

```python
while boolean_expression:
    # code block
```

While Loop

D. Thiebaut, Computer Science, Smith College
# get a positive number from user

```python
while boolean_expression:
    # code block
```

While Loop
While Loop

# get a positive number from user
while boolean_expression:
    # code block
    # Is expression True?
While Loop

# get a positive number from user
while boolean_expression:
    # execute body of While Loop

If True, then execute body of While Loop
While Loop

# get a positive number from user
while boolean_expression:
    # code block
    # code block
    # code block
While Loop

# get a positive number from user
while boolean_expression:
    Expression is False
Robust Input
With While Loops

• Example 1: While quantity not valid

• Example 2: While item not in list
# get a positive number from user
x = eval(input(" > ") )

...
What if user enters negative number?

```python
# get a positive number from user
x = eval(input("> " ))

...
Solution: keep on asking until input is ok

```python
# get a positive number from user
x = eval(input( "> " ))
while x < 0:
    x = eval(input("Invalid number\n> "))
```
Write Robust Functions That Prompt for Quantities
# get a positive number from user

def getPositiveInt():
    x = int(input("> " ))
    while x < 0:
        x = int(input("Invalid number
> " ))
    return x

x = getPositiveInt()
Robust Input With While Loops

- **Example 1:** While quantity not valid
- **Example 2:** While item not in list
# get a YES/NO answer from user

def getAnswerYesNo():
    x = input("Continue (Yes/No)? ")

    while ???:
        print("Invalid input, must be YES or NO")
        x = input("Continue (Yes/No)? ")

    return x

ans = getAnswerYesNo()
# get a YES/NO answer from user

def getAnswerYesNo():
    x = input( "Continue (Yes/No)? " ).upper()

    while ( x in [ "YES", "NO" ] ) == False:
        print( "Invalid input, must be YES or NO" )
        x = input( "Continue (Yes/No)? " ).upper()

    return x

ans = getAnswerYesNo()
# get a YES/NO answer from user

def getAnswerYesNo():
    x = input("Continue (Yes/No)? ").upper()

    while not (x in ["YES", "NO"]):
        print("Invalid input, must be YES or NO")
        x = input("Continue (Yes/No)? ").upper()

    return x

ans = getAnswerYesNo()
For loops in context

Applications

While Loops for Robustness

Break & Continue
Break and Continue
Monopoly = loop
break = ?
continue = ?

http://en.wikipedia.org/wiki/File:German_Monopoly_board_in_the_middle_of_a_game.jpg
Monopoly = loop
break = ?
continue = ?
Break

Monopoly = loop
break = ?
continue = ?
Monopoly = loop
break = ?
continue = ?
Monopoly = loop
break = ?
continue = ?
Applying *Break* and *Continue* to Eliza