Announcements

- Review Chapter 4 (functions)
- Read Chapter 5 (conditionals)
- Work on Project 1
Python Boot Camp

- Booleans
  - Logic Operators
- While loops
- If statements
- Else clauses
Boolean Values

- True, False
- Result of comparison, e.g. $x < y$
- Boolean operations:
  - And: $(x \geq 5)$ and $(x \leq 10)$
  - Or: $(x < 5)$ or $(x > 10)$
  - Not: $\neg (3 == 4)$
<table>
<thead>
<tr>
<th>and</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Truth Tables: or

<table>
<thead>
<tr>
<th>or</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Truth Tables: not

<table>
<thead>
<tr>
<th></th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison Tests

- We also need a set of test operators to build useful conditions:
  - `==` Equality
  - `!=` Inequality
  - `>` Greater Than
  - `>=` Greater Than or Equal
  - `<` Less Than
  - `<=` Less Than or Equal
Precedence

- Multiplication and division take precedence over addition and subtraction

- Consider \(a*b+c*d\); which of the three is it equal to?
  A. \((a*b) + (c*d)\)
  B. \(a*(b+c) *d\)
  C. \(((a*b) +c) *d\)

- What about comparison (<, <=, ==, >=, >) vs. Boolean ops (and, or, not)?

- Comparisons take precedence …
Example

- $x \geq 5$ and $y \leq 10$ is equivalent to $(x \geq 5)$ and $(y \leq 10)$

- Python allows an unusual comparison chain:
  - $x \leq y \leq z$ is understood as $(x \leq y)$ and $(y \leq z)$
Loops

- Loops allow for code to be run multiple times
- There are many kinds of loops
  the most basic kind is a while loop

```python
count = 0
while (count < 8):
    print 'The count is: ', count
    count = count + 1
```
count = 0
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
print ( 'The count is: ', count )
count = count + 1
Loops

- Note: The count variable is incremented after the last print statement.
  - Is this the case with the loop code?
  - Verify this!

- In addition to saving you from copying code over and over, it allows the condition to be a variable
  - Suppose the number of times we want to print depends on some other code.
Example Revisited: motivation behind loops

- Intuition: we can leverage functions + arguments to get input, we can use the input to *parameterize* the loop!

```python
def printCountNTimes(n):
    count = 0
    while (count < n):
        print ( 'The count is: ', count )
        count = count + 1
```
Loop Types

- The two common types of loops you will use in python are the
  - While loop
  - For loop

- Which type you use depends on the particular situation
While Loop

- The while loop checks whether a condition is true or not.

- In this example “is count < 9”
  - If this is true it executes the body of the loop
  - If it is false it skips past the loop body to the next line of code

```python
count = 0
while (count < 9):
    print ('The count is: ', count )
    count = count + 1
```
while count < 9:
    print('The count is: ', count)
    count = count + 1
CQ: Are these programs equivalent?

```
def printCountNTimes(n):
    count = 0
    while (count < n):
        print ('The count is: ', count)
        count = count + 1
```

1. `printCountNTimes(8)`
2. `printCountNTimes(4)`

A: yes  B: no
While Loop Dangers

• While loops do not require that the loop ever stop.

```python
count = 0
while (count < 9):
    print ('The count is: ', count )
    count = count - 1
```

• In this example, count starts at 0 and gets smaller. Thus it will always be < 9
Example

- Print every other character of a string

```python
stg = "hello there"
pos = 0
while pos < len(stg):
    print(stg[pos])
pos = pos+2
```
IF Statement

- For the conditional expression <exp>, evaluating to True or False, the simple IF statement is

```python
if <exp>:
    <body>
```

```python
x = 7
if x > 10:
    print (x)
x = x + 1
```

```python
x = True
if x:
    print (x)
```
Conditionals

- It is also possible to specify what to do if the condition is False. Contrast these two examples.

```python
x = 7
if x > 10:
    print(x)
x = x + 1
```

```python
x = 7
if x > 10:
    print(x)
else:
    x = x + 1
```
```python
x = 7
if x > 10:
    print(x)
    x = x + 1
```
x = 7
if x > 10:
    print (x)
else:
    x = x + 1
Conditions

- At the heart of a conditional is a condition evaluation.
- The code after the `if` but before the `:` should evaluate to either `True` or `False`.
  - These are special keywords in Python and are case sensitive.
Beyond True and False

- Python provides True and False, even for general values
  - Numbers: zero \equiv False, not zero \equiv True
  - Lists: [ ] \equiv False, any other list \equiv True
  - Strings: "" \equiv False, any other string \equiv True
- Experiment with it!

```python
x = 0
if 1:
    print(x)

x = 0
if 0:
    print(x)
```
CQ: Do these programs print the same things?

1

\begin{verbatim}
x = 7
if x > 10:
    print (x)
x = x + 1
print (x)
\end{verbatim}

2

\begin{verbatim}
x = 7
if x > 10:
    print (x)
else:
    x = x + 1
print (x)
\end{verbatim}

A: yes
B: no
CQ: Do these programs print the same things?

1

\begin{align*}
\text{x} &= 12 \\
\text{if } x > 10: & \quad \text{print}(x) \\
  & \quad x = x + 1 \\
\text{print}(x)
\end{align*}

2

\begin{align*}
\text{x} &= 12 \\
\text{if } x > 10: & \quad \text{print}(x) \\
\text{else:} & \quad x = x + 1 \\
\text{print}(x)
\end{align*}

A: yes
B: no
Therefore

- Always ask what “equivalent” means…

- For us from now on:
  - Two programs are equivalent if they give the same output for all inputs
  - Any other meaning should be made clear by explicit wording
Some Interesting Observations

- Comparisons can be chained
  - $x < y \leq z$ is equivalent to: $x < y$ and $y \leq z$

- The not operator has a lower order of precedence than comparisons
  - $\neg a == b$ is equivalent to: $\neg(a == b)$
Clicker Question

- Now we can start building useful conditions

```python
if x and y > 0:
    print(x, y)
```

- Does this check if \( x > 0 \)?

A: yes
B: no
Tests Continued

- We could write such a condition as follows:

```python
if x > 0 and y > 0:
    print (x + y)
```
Clicker Question: Are these programs equivalent?

A: yes
B: no
Nested Conditionals

```python
if x < 10:
    print(x+y)
    if y < 100:
        print(x)
    else:
        print(y)
```
Why nested conditionals?

- We could just write many complex conditionals and avoid the nested conditionals

- This is inefficient (many redundant checks) and may lead to code duplication

- But is the nesting always easy to understand?
Example Revisited

if x < 10:
    print(x+y)
    if y < 100:
        print(x)
    else:
        print(y)

if x < 10 and y < 100:
    print(x+y)
    print(x)
if x < 10 and y >= 100:
    print(x+y)
    print(y)
Defensive Coding

- Comment your code
- Use conditionals to check/verify assumptions
- Use variable names that have meaning
- Avoid reusing variable names
- Use parentheses when order matters
- Keep the code simple
def minOfThree(a,b,c):
    if a<=b and a<=c:
        return a
    if b<=a and b<=c:
        return b
    if c<=a and c<=b:
        return c
def minOfThree(a,b,c):
    if a<=b:
        if a<=c:
            return a
        else:
            return c
    else:
        if b<=c:
            return b
        else:
            return c
def minOfThree(a, b, c):
    if a <= b:
        if a <= c:
            return a
        else:
            return c
    else:
        if b <= c:
            return b
        else:
            return c
def minOfThree(a, b, c):
    if a <= b:
        if a <= c:
            return a
        else:
            return c
    else:
        if b <= c:
            return b
        else:
            return c
Clicker Question

1
if “False”:
    print(“hi”)

2
if False:
    print(“hi”)

A: 1 and 2 both print
B: only 1 prints
C: only 2 prints
D: neither 1 nor 2 print
Clicker Question

3

if eval("False"):
    print("hi")

2

if False:
    print("hi")

A: 3 and 2 both print
B: only 3 prints
C: only 2 prints
D: neither 3 nor 2 print
# This function expects two positive integers
# Returns -1 as an error condition

def sumOfTwo(a, b):
    if type(a) == int and type(b) == int:
        if a > 0 and b > 0:
            return a + b
    return -1
Error Checking Revisited

#This function expects two positive integers
# Returns -1 as an error condition

def sumOfTwo(a,b):
    if type(a) == int and type(b) == int :
        if a > 0 and b > 0:
            return a+b
        else:
            return -1
Python Boot Camp!

- Complex control flow
- Examples of nested if
- Examples of if and else clauses
- elif
import math

def main():
    print("This program finds the real solutions to a quadratic")
    a, b, c = eval(input("\nPlease enter coefficients (a, b, c): "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nThe solutions are: ", root1, root2)

main()
Two-Way Decisions

- As per the comment, when $b^2-4ac < 0$, the program crashes.

This program finds the real solutions to a quadratic

Please enter coefficients (a, b, c): 1,1,2

Traceback (most recent call last):

File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS 120\Textbook\code\chapter3\quadratic.py", line 21, in -toplevel-
    main()

File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS 120\Textbook\code\chapter3\quadratic.py", line 14, in main
    discRoot = math.sqrt(b * b - 4 * a * c)

ValueError: math domain error
Two-Way Decisions

- We can check for this situation. Here’s our first attempt.

```python
# quadratic2.py
# A program that computes the real roots of a quadratic equation.
# Bad version using a simple if to avoid program crash

import math

def main():
    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))
    discrim = b * b - 4 * a * c
    if discrim >= 0:
        discRoot = math.sqrt(discrim)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
```
Two-Way Decisions

- We first calculate the discriminant \( b^2 - 4ac \) and then check to make sure it is not negative. If it is, the program proceeds and we calculate the roots.

- Look carefully at the program. What’s wrong with it? Hint: What happens when there are no real roots?
Two-Way Decisions

- This program finds the real solutions to a quadratic

Please enter the coefficients \((a, b, c)\): 1,1,1

- This is almost worse than the version that crashes, because we don’t know what went wrong!
Two-Way Decisions

- We could add another `if` to the end:
  ```python
  if discrim < 0:
      print("The equation has no real roots!")
  ```

- This works, but feels wrong. We have two decisions, with *mutually exclusive* outcomes (if `discrim >= 0` then `discrim < 0` must be false, and vice versa).
Two-Way Decisions

- Discrim < 0?
  - Yes: Print "no roots"
  - No: Calculate roots
Two-Way Decisions

# quadratic3.py
# A program that computes the real roots of a quadratic equation.
# Illustrates use of a two-way decision

import math

def main():
    print "This program finds the real solutions to a quadratic\n"

    a,b,c = eval(input("Please enter the coefficients (a, b, c): "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!"")
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are: ", root1, root2)

main()
Two-Way Decisions

>>> This program finds the real solutions to a quadratic
Please enter the coefficients (a, b, c): 1,1,2
The equation has no real roots!

>>> This program finds the real solutions to a quadratic
Please enter the coefficients (a, b, c): 2, 5, 2
The solutions are: -0.5 -2.0
Multi-Way Decisions

- The newest program is great, but it still has some quirks!

- This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1,2,1

The solutions are: -1.0 -1.0
Multi-Way Decisions

- While correct, this method might be confusing for some people. It looks like it has mistakenly printed the same number twice!

- Double roots occur when the discriminant is exactly 0, and then the roots are $-b/2a$.

- It looks like we need a three-way decision!
Multi-Way Decisions

- Check the value of discrim
  - when < 0: handle the case of no roots
  - when = 0: handle the case of a double root
  - when > 0: handle the case of two distinct roots

- We can do this with two if-else statements, one inside the other.

- Putting one compound statement inside of another is called *nesting*.
Multi-Way Decisions

if discrim < 0:
    print("Equation has no real roots")
else:
    if discrim == 0:
        root = -b / (2 * a)
        print("There is a double root at", root)
    else:
        # Do stuff for two roots
Multi-Way Decisions
Multi-Way Decisions

- Imagine if we needed to make a five-way decision using nesting. The if-else statements would be nested four levels deep!
- There is a construct in Python that achieves this, combining an else followed immediately by an if into a single elif.
Multi-Way Decisions

- if <condition1>:
  <case1 statements>
elif <condition2>:
  <case2 statements>
elif <condition3>:
  <case3 statements>
...
else:
  <default statements>
Multi-Way Decisions

- This form sets of any number of mutually exclusive code blocks.

- Python evaluates each condition in turn looking for the first one that is true. If a true condition is found, the statements indented under that condition are executed, and control passes to the next statement after the entire `if-elif-else`.

- If none are true, the statements under `else` are performed.

- The else is optional. If there is no `else`, it is possible no indented block would be executed.
Multi-Way Decisions

- Conditions need not be mutually exclusive. Therefore, changing the sequence may give different results!

- Consider the following two versions when a==0 and b==2

```python
if b<0:
    x=1
elif b>0 and a<3:
    x=2
elif b>0:
    x=3
else:
    x=4
```

```python
if b<0:
    x=1
elif b>0:
    x=3
elif b>0 and a<3:
    x=2
else:
    x=4
```
def main():
    a, b, c = eval(input("Please enter the
coefficients (a, b, c): "))
    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!"")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThe equation has a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
Which version is simpler?

```python
def sort3(a, b, c):
    if a <= b:
        if b <= c:
            return [a, b, c]
        elif a <= c:
            return [a, c, b]
        else:
            return [c, a, b]
    else:
        if a <= c:
            return [b, a, c]
        elif b <= c:
            return [b, c, a]
        else:
            return [c, b, a]
```

```python
def sort3(a, b, c):
    if a <= b:
        if b <= c:
            return [a, b, c]
        else:
            if a <= c:
                return [a, c, b]
            else:
                return [c, a, b]
    else:
        if a <= c:
            return [b, a, c]
        else:
            if b <= c:
                return [b, c, a]
            else:
                return [c, b, a]
```