Review

- How can we make some code execute only "sometimes"?
  - What are alternative Python constructs to do that? How do they differ?
- How do we express the condition that two things must be true to satisfy a condition?

Short-circuit Evaluation

- Don't necessarily need to evaluate all expressions in a compound expression
- A `and` B
  - If A is `False`, compound expression is `False`
- A `or` B
  - If A is `True`, compound expression is `True`
- No need to evaluate B
  - Put more important/limiting expression first
  - Example:

```
if count != 0 and sum/count > 10:
   do something
```


SYS MODULE

sys module

- Has useful "system" functions
- Use the `exit([status])` function
  - **Exits the whole program**
  - If status is empty, defaults to 0
  - Status of 0 means success
  - Other values are various failures
- *Another example of changing control flow*

Instead of

```
print "This program determines your birth year"
print "given your age and current year"
print
age = input("Enter your age >> ")

if age > 120:
    print "Don't be ridiculous, you can't be that old."
else:
    currentYear = input("Enter the current year >> ")
    birthyear = currentYear - age
    print
    print "You were either born in", birthyear, "or",
    print birthyear-1
```


Example Use of sys module

```
import sys
print "This program determines your birth year"
print "given your age and current year"
print
age = input("Enter your age >> ")
if age > 120:
   print "Don't be ridiculous.  You can't be that old!"
   sys.exit(1)
```

Ejector seat

```
# input is reasonable …
currentYear = input("Enter the current year >> ")
birthyear = currentYear - age
print
print "You were either born in", birthyear, "or",
print birthyear-1
```

Lab 2 Feedback

- Getting a little tougher in grading
  - Paying more attention to style (e.g., variable names), efficiency, readability, good output
  - More strict on adhering to problem specification
  - Constants
  - Demonstrate program **more than once** if gets input from user or outcome changes when run again
    - Find errors before I do!

Lab 2 Feedback: Common Issues

- Over `str`ing
  - If variable assigned value of raw_input, it is a string already
  - "\\" is a string
  - Result of format operator (%) is a string
- Only need to use `str`
  - In `print` when don't want the space that is added by the comma and have a number
  - In `input` statement because need to pass in *one string* as the prompt parameter

Lab 2 Feedback: Doing Too Much

```
print "This molecule weighs", round(hyd_atm*H+cb_atm*
        +o_atm*O ,3), "g/mol"
```

Instead: 1st: Compute the weight
2nd: Round
3rd: Display

```
cTemp = "%.2f" % ((5.0/9.0)*(fTemp-32.0))
```

Instead: 1st: Compute the Celsius temperature
2nd: Display

Why do I care?
Error prone; More difficult to debug, read

Lab 2 Feedback: Common Issues

- Efficiency

```
i=6
for j in xrange(1, 9):
   result = i % j
   print i, "%", j, "=", result
```

vs

```
for j in xrange(1, 9):
   i=6
   result = i % j
   print i, "%", j, "=", result
```


Lab 2 Feedback: Common Issues

- Format specifiers
  - Use whenever I say "displays X decimal places"
  - Use *width* when need columns
  - Otherwise, just precision is usually enough
    - "%.2f" -- exactly the width of your number, with two decimals of precision
- Problem 2: asked for **built-in function**
  - round

Lab 2 Feedback: Formatting

- Don't mix tabs and spaces and format specifiers
  - Let format specifiers do the work
  - Instead of

```
print "\t    %6.2f" % value
```

  - Just use format specifier

```
print "%13.2f" % value
```

Easier to figure out output

String Formatting

- What does this do?

```
print "%.2f deg F is %.2f deg C" % (degreesF, degreesC)
```


String Formatting: Equivalent Statements

```
print "%.2f deg F is %.2f deg C" % (degreesF, degreesC)
```

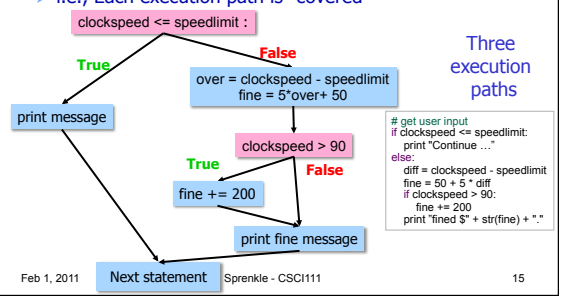
```
print "%.2f" % degreesF, "deg F is %.2f deg C" % degreesC)
```

```
print "%.2f" % degreesF, "deg F is", "%.2f" % degreesC, "deg C"
```

```
formattedFTemp = "%.2f" % degreesF
formattedCTemp = "%.2f" % degreesC
print formattedFTemp, "deg F is", formattedCTemp, "deg C"
```

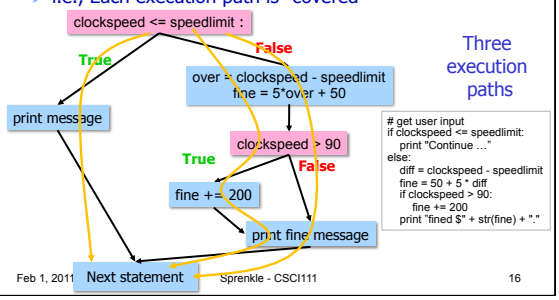

Review: Testing with `if` Statements

- Make sure have test cases that execute each branch in control flow diagram
  - i.e., Each execution path is "covered"



Review: Testing with `if` Statements

- Make sure have test cases that execute each branch in control flow diagram
  - i.e., Each execution path is "covered"



Review: Efficiency of `if` statements

- Which is more efficient?

```
if x < 0:
   print x, "is negative"
if x >= 0:
   print x, "is 0 or positive"
```

```
if x < 0:
   print x, "is negative"
else:
   print x, "is 0 or positive"
```


Review: Efficiency of `if` statements

- Which is more efficient?

```
if x < 0:
   print x, "is negative"
if x >= 0:
   print x, "is 0 or positive"
```

Additional computation

```
if x < 0:
   print x, "is negative"
else:
   print x, "is 0 or positive"
```

Lab 3 Overview

- Practice Python programming
  - Advanced For loops
  - If statements
  - Using random module
- More difficult word problems
  - If and Loops opened up world of new problems
  - Work out as much as possible, then move on and come back to problem later with a fresh mind

Break down problem into smaller pieces
Then, build up to solution!