Name & Recitation Section:

Due **Wednesday, Jan 5 at 2:10 PM in 34-101**. Please print out your code files (`homework_1.py`, `rps.py`, `loops.py`, and any code you wrote for optional problems), staple them to the back of these exercises, and turn them in at the beginning of Wednesday's lecture.

Exercise 1.9 – Variable Names

The Python interpreter has strict rules for variable names. Which of the following are legal Python names? If the name is not legal, state the reason.

1. `and`

2. `_and`

3. `var`

4. `var1`

5. `1var`

6. `my-name`

7. `your_name`

8. `COLOR`

Exercise 1.10 – Types

It is important that we know the type of the values stored in a variable so that we can use the correct operators (as we have already seen!). Python automatically infers the type from the value you assign to the variable. Write down the type of the values stored in each of the variables below. Pay special attention to punctuation: values are not always the type they seem!

1. `a = False`

2. `b = 3.7`

3. `c = 'Alex'`

4. `d = 7`

5. `e = 'True'`

6. `f = 17`

7. `g = '17'`

8. `h = True`

9. `i = '3.14159'`

To verify your answers, you can use the interactive Python shell, but first try to do the exercise without help.

```
>>> x = 100
>>> type(x)
<type 'int'>
>>>
```

Exercise 1.11 – Natural Language Processing

Consider the following sentence:
Alice saw the boy on the hill with the telescope.

1. Draw a sketch of what's described in this sentence.

2. Draw a different sketch that could also be described by this sentence.

3. Write the sentence in two different ways, that clarifies the meaning of each of your sketches, next to your above sketches (hint: rewrite the sentence using extra words, commas, etc).
4. The ambiguity illustrated by this sentence is known as "prepositional phrase attachment." Think about this as you continue to learn how to program, and consider how programming languages are designed to avoid the ambiguity illustrated by this example!

Exercise 1.12 – Boolean operators

Boolean operators can seem tricky at first, and it takes practice to evaluate them correctly. Write the value (`True` or `False`) produced by each expression below, using the assigned values of the variables a, b, and c. Try to do this without using your interpreter, but you should check yourself when you think you've got it. **Hint**: Work from the inside out, starting with the inner-most expressions, like in arithmetic.

```
a = False
b = True
c = False
```

1. `b and c`

2. `b or c`

3. `not a and b`

4. `(a and b) or not c`

5. `not b and not (a or c)`

Exercise 1.13 – Conditionals

The purpose of this exercise is to understand conditionals. Tiberius is looking for his dream job, but has some restrictions. He loves California and would take a job there if it paid over 40,000 a year. He hates Massachusetts and demands at least 100,000 to work there. Any other place he's content to work for 60,000 a year, unless he can work in space in which case he would work for free. The following code shows his basic strategy for evaluating a job offer.

```
pay = _____
location = _____

if location == "U.S.S. Enterprise":"
    print "So long, suckers! I'll take it!"
elif location == "Massachusetts":
    if pay < 100000:
       print "No way"
    else:
       print "I'll take it!"
elif location == "California" and pay > 40000:
    print "I'll take it!"
elif pay > 60000:
    print "I'll take it!"
else:
    print "No thanks, I can find something better."
```

For each of the following job offers, write down the output that would be generated. Do this without running the code. It is an important skill to be able to understand what a piece of code does without running it.

1. ```
   location = "Massachusetts"
   pay = 50000
   ```

2. ```
   location = "Iowa"
   pay = 50000
   ```

3. ```
   location = "California"
   pay = 50000
   ```

4. ```
   location = "U.S.S. Enterprise"
   pay = 1
   ```

5. ```
   location = "California"
   pay = 25000
   ```
   ```

Exercise 1.14 – Understanding loops

For each of the following fragments of code, write what the output would be. Again, do this without running the code (although feel free to check yourself when you're done).

1.
```
num = 10
while num > 3:
   print num
   num = num - 1
```

2.
```
divisor = 2
for i in range(0, 10, 2):
   print i/divisor
```

3.
```
num = 10
while True:
   if num < 7:
      break
   print num
   num -= 1
```

4.
```
count = 0
for letter in 'Snow!':
   print 'Letter #', count, 'is', letter
   count += 1
```

Exercise 1.15 – Buggy loop (aka Find The Bug!)

Consider the following program that Ben Bitdiddle handed in to the course staff (again, try to do this exercise without running the code in IDLE!):

```
n = 10
i = 10

while i > 0:
   print i
   if i % 2 == 0:
      i = i / 2
   else:
      i = i + 1
```

What do you think this code is doing? Without comments it is hard to guess what Ben's intention was (*cough* this is why the staff *loves* to look at commented code!!), so read through it and make a sensible guess as to what it is doing. There's a lot of mistakes in the code so your guess is as good as ours!

1. Draw a table that shows the value of the variables `n` and `i` during the execution of the program. Your table should contain two columns (one for each variable) and one row for each iteration. For each row in the table, write down the values of the variables as they would be at the line containing the print statement.

2. Ben made a lot of mistakes. State what you think Ben was trying to do and suggest one or more ways he could fix his code (there's a few good answers for this depending on what you think the code should be doing).

6.189 A Gentle Introduction to Programming

January IAP 2011