Name:_____

1. A for-loop is useful for building tables. For example, we could calculate n^2 and 2^n for values of n from 0 to 9.

```
print "n","n**2","2**n"
for n in xrange(10):
    print n, n**2, 2**n
```

Actual output

```
n n**2 2**n
0 0 1
1 1 2
2 4 4
3 9 8
4 16 16
5 25 32
6 36 64
7 49 128
8 64 256
9 81 512
```

Desired output

```
 n   n**2   2**n
------------------
 0     0      1
 1     1      2
 2     4      4
 3     9      8
 4    16     16
 5    25     32
 6    36     64
 7    49    128
 8    64    256
 9    81    512
```

Normally, we want the numbers to be *right-justified* in a column with specific *field width*. Formatted text can be achieved using the general syntax of: "<format string>" % (<datum1>, ..., <dataN>) with the format string containing a format operator corresponding to each data item in the tuple. The formats for each type of date are:

Data Type	**General Format Operator**	**Example**	
		Format operator	**String produced**
string	%<field width>s	"%8s" % 'house'	" house"
integer	%<field width>d	"%7d" % 1234	" 1234"
float	%<field width>.<precision>	"%-8.2f" % 11.234	"11.23 "

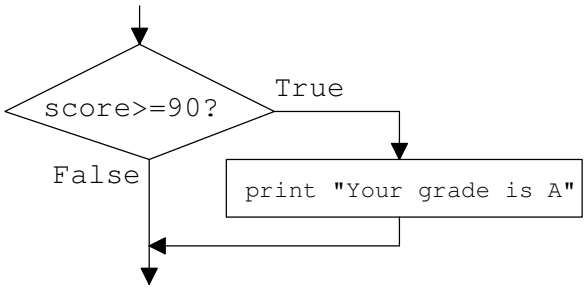
A positive field width specifies right-justification, and a negative specifies left-justification.
Rewrite the above for-loop using formatted text to print the table as shown on the right above.

2. An `if` statement allows code to be executed or not based on the result of a comparison. The syntax of an if statement is:

```
if <condition>:
    statement1
    statement2
    statement3
```

If the condition evaluates to True, then the statements of the indented body is executed. If the condition is False, then the body is skipped.
Typically, the condition involves comparing "stuff" using relational operators (`<, >, ==, <=, >=, !=`). For example, we might want to print "Your grade is A." if the variable score is greater-than or equal to 90.

```
if score >= 90:
    print "Your grade is A."
```



Complex conditions might involve several comparisons combined using Boolean operators: `not, or, and`. For example, we might want to print "Your grade is B." if the variable score is less than 90, but greater than or equal to 80.

```
if score < 90 and score >= 80:
    print "Your grade is B."
```

Name:_____

Using only `if` statements, write Python code to output the appropriate string according to variable temperature's value.

Temperature	String
temperature < 0	"Its bitterly cold!"
0 <= temperature <= 32	"Its freezing outside."
32 < temperature < 68	"Light jacket weather."
68 <= temperature	"Its warm outside."

3. An `if-else` statement allows a block of code to be executed if the result of a comparison is True; otherwise the "else" block of code will be executed. The syntax of an `if-else` statement is:

```
if <condition>:
    # only executed if the condition is True
    statement_T1
    statement_T2
else:
    # only executed if the condition is False
    statement_F1
    statement_F2
```

Using only nested `if-else` statements, write Python code to output the appropriate string according to variable temperature's value.

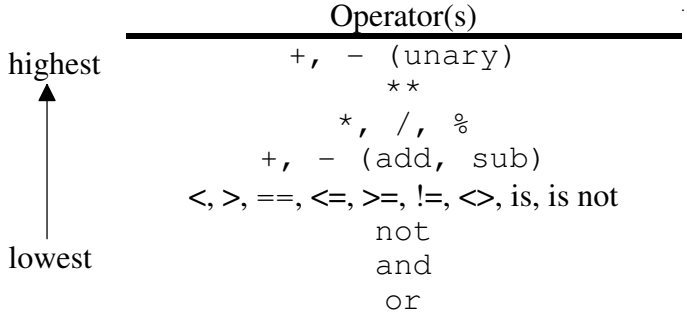
Temperature	String
temperature < 0	"Its bitterly cold!"
0 <= temperature <= 32	"Its freezing outside."
32 < temperature < 68	"Light jacket weather."
68 <= temperature	"Its warm outside."

Name:_____

4. Using an `if-elif-else` statement, write Python code to output the appropriate string according to variable temperature's value.

Temperature	String
temperature < 0	"Its bitterly cold!"
0 <= temperature <= 32	"Its freezing outside."
32 < temperature < 68	"Light jacket weather."
68 <= temperature	"Its warm outside."

5. The precedence for mathematical operators, Boolean operators, and comparisons are given in the table
For each of the expressions, determine if the expression is legal. If it is, determine the order of operations:



a) `x + 5 < y**2 and not y >= 6`

b) `not x < 8 or 3y >= 10`

6. If we "`import random`", some of the useful function are:

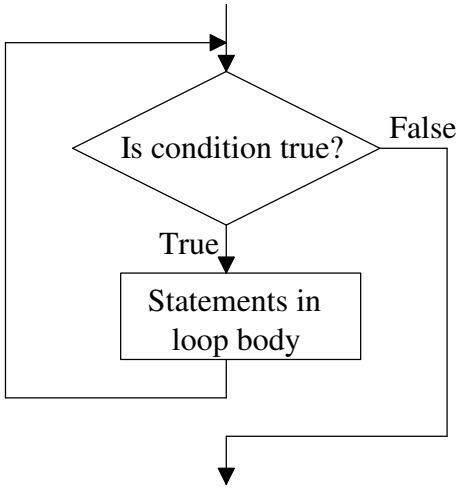
General Syntax	**Description**	**Example**
`random.randrange(start,stop[,step])`	Choose a random item from range(start, stop[, step]).	`random.randrange(0,110,10)`
`random.randint(a, b)`	Return random integer in [a, b], including both ends	`random.randint(a, b)`
`random.random(`	Return a value in the range [0,1).	`random.random(`

Suppose we wanted to randomly roll a 6-sided die to get a value between 1 and 6. How would you use each of the above to generate a die value between 1 and 6?

Name:_____

7. A `while` statement allows code to be executed repeated (zero or more times) as long as the condition evaluates to True. The syntax of a `while` statement is:

```
while <condition>:
    statement1
    statement2
    statement3
```



Consider the following program to print even numbers up to some user specified amount.

```
stopValue = input("Enter a stopping value: ")
counter = 0
while counter != stopValue:
    print counter
    counter = counter + 2
print "Done"
```

a) In the box, show a few lines of the programs output assuming the user enters 8 for a stopping value.

b) What would happen if the user enters 9 for a stopping value?

c) What modification would improve this program?

8. An *infinite loop* is one that would loop forever. (FYI, in a Python shell ctrl-c (^c) can be used to kill the running program.) Most infinite loops are caused by programmer error, but sometimes they are intentional. The following code uses an infinite loop and a *break* statement that immediately causes control to exit the loop.

```
total = 0
counter = 0
while True:     # an infinite loop
    score = input("Enter a score (or -99 to exit): ")
    if score < 0:
        break
    total += score
    counter += 1
print "Average is", float(total)/counter
```

Rewrite the above "*sentinel-controlled*" while loop without using a `break` statement.