Name:_____

1. Draw a box around each scope (function definition) and list the user-defined names (functions and variables) that can be accessed inside of the scope. For names that occur multiple places, be sure that you are clear which is accessable (e.g., write subscripts on them).

```
# Program to demonstrate scope
def main():
    value = 99   # local variable
    print 'In main before call: value is', value
    change_me(value)
    print 'Back in main after change_me: value is', value
    value = more_change(value, value)
    print 'Back in main after more_change: value is', value

def change_me(arg):
    def more_change(arg2, arg3):
        print 'In local more_change: arg2 is', arg2, 'and arg3 is', arg3
        arg = value + arg2
        print 'In local more_change: value is', value
        return arg3 + 100

    value = 10   # local variable
    print 'In change_me: I am changing the formal param.'
    arg = 0
    print 'In change_me: arg is', arg, 'and value is', value
    value = more_change(value, arg)
    print 'Back in change_me: arg is', arg, 'and value is', value

def more_change(arg, arg2):
    print 'In more_change: value is', value
    print 'In more_change: arg is', arg, 'and arg2 is', arg2
    arg = arg2
    print 'In more_change: arg is', arg, 'and arg2 is', arg2
    return arg2 * 5

#Global variable
value = 5
# Call the main  function.
main()
print 'after main call: value is', value
```

2. Use the run-time stack to trace the following program and predict the output

Predicted Output	

Name:_____

3. 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."
```

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."

Name:_____

4. 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
    statement_T3
else:
    # only executed if the condition is False
    statement_F1
    statement_F2
    statement_F3
```

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:_____

5. 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."