NOTE TO STUDENTS: This file contains sample solutions to the term test together with the marking scheme and comments for each question. Please read the solutions and the marking schemes and comments carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes.

Remember that although you may not agree completely with the marking scheme given here it was followed the same way for all students. We will remark your test only if you clearly demonstrate that the marking scheme was not followed correctly.

For all remarking requests, please submit your request **in writing** directly to your instructor. For all other questions, please don't hesitate to ask your instructor during office hours or by e-mail.

Question 1. [15 MARKS]

Each of these subquestions contains a block of code. Treat each block of code independently (*i.e.*, code in one question is not related to code in another), and answer each question in the space provided.

Part (a) Simple Syntax [1 MARK]

```
def add_ints(x, y):
    return x + y

x = 1
y = 2
add_ints()
```

In the block of code to the left, circle every line that would cause the code to fail—there is at least one. Then, in the space below, explain why the line(s) fail.

SAMPLE SOLUTION:

Circle `add_ints()`, because it is missing arguments to the function call.

MARKING SCHEME:

- **Answer** [0.5]
- **Explanation** [0.5]

Part (b) Simple Syntax [1 MARK]

```
def loopy(L)
    for item in L
        print item

my_list = [1, 2, 3]
loopy(my_list)
```

In the block of code to the left, circle every line that would cause the code to fail—there is at least one. Then, in the space below, explain why the line(s) fail.

SAMPLE SOLUTION:

Circle `def loopy(L)` and `for item in L`, because they are missing a colon (`:`) at the end of the line.

MARKING SCHEME:

- **Answer** [0.5]
- **Explanation** [0.5]

Question 1. (continued)

Part (c) Scope [1 mark]

```
def sum_to_n(n):
    total = (n * (n + 1)) / 2.0

sum_to_n(5)
print total
```

In the block of code to the left, circle every line that would cause the code to fail—there is at least one. Then, in the space below, explain why the line(s) fail.

Sample Solution:

Circle `print total`, because it is trying to access a variable local to function `sum_to_n`.

Marking Scheme:

- **Answer** [0.5]
- **Explanation** [0.5]

Part (d) Scope [1 mark]

```
value = 1

def printer(value):
    print value
    value = 42
    print value

print value
printer(value)
print value
```

What is the output of the code to the left?

Sample Solution:

```
1
1
42
1
```

Marking Scheme:

- **First three lines** [0.5]
- **Last line** [0.5]

Part (e) Order of Execution [1 mark]

```
def printer():
    print "Hello"
print "Hi"

printer()
printer()
```

What is the output of the code to the left?

Sample Solution:

```
Hi
Hello
Hello
```

Marking Scheme:

- **Output** [1] (Don't take off marks for including quotes in the output.)

Question 1. (CONTINUED)

Part (f) Order of Execution [1 MARK]

```
var_A = 11
var_B = var_A
var_A = 42
```

After this code is executed, the value of `var_B` is:

`11`

SAMPLE SOLUTION: (See above.)

MARKING SCHEME:

- **Answer** [1]

Part (g) While Loops [1 MARK]

```
def find_o(search_str):
    index = 0
    while index < len(search_str) and \
            search_str[index] != 'o':
        index += 1
    return index

print find_o("eeyore")
print find_o("pooh")
print find_o("tigger")
```

What is the output of the code to the left?

SAMPLE SOLUTION:

```
3
1
6
```

MARKING SCHEME:

- **Answer** [1] (Take off only 0.5 if every answer is off-by-one.)

Part (h) Mutability [1 MARK]

```
def doubler(L):
    for item in L:
        item = item * 2
    print L

my_list = [1, 2, 3]
doubler(my_list)
```

What is the output of the code to the left?

SAMPLE SOLUTION:

```
[1, 2, 3]
```

MARKING SCHEME:

- **Answer** [1]

Question 1. (continued)

Part (i) Aliasing and Mutability [1 mark]

```
def doubler(L):
    dL = L
    for index in range(len(dL)):
        dL[index] = dL[index] * 2

my_list = [1, 2, 3]
doubler(my_list)
print my_list
```

What is the output of the code to the left?

Sample Solution:

```
[2, 4, 6]
```

Marking Scheme:

- **Answer** [1]

Part (j) Conditionals and Booleans [2 marks]

The table to the right shows how an employee's age and experience affects his or her hourly wage. Assume that you have a `boolean` variable `experienced` and an `int` variable `age` that correspond with the labels in the table. Fill in the conditions in the code below to calculate the hourly wage for the employee.

	Experienced?	
Age	Yes	No
under 18	$12.00	$9.50
18 and over	$15.00	$10.50

Sample Solution:

```
if __ experienced ____________________________________________________________:

    if __ age < 18 _____________________________________________________________:
        wage = 12
    else:
        wage = 15
else:
    if __ age < 18 _____________________________________________________________:
        wage = 9.5
    else:
        wage = 10.5  # typo corrected during the test
```

Marking Scheme:

- **Format:** [1] all expressions are boolean (even if incorrect)
- **Idea:** [1] correct expressions (even if not expressed correctly)

Question 1. (continued)

Part (k) Data Types [2 marks]

Fill in the blank so that when this code is run, the user is asked to enter two numbers and then the average of those numbers is printed. The payrates are likely to contain decimal values.

```
num1 = raw_input("Please enter your hourly wage: ")
num2 = raw_input("Please enter your friend's hourly wage: ")
```

Sample Solution:

```
print "Your average wage is", __ ( float(num1) + float(num2) ) / 2 _____________
```

Marking Scheme:

- **Expression** [0.5] correct high-level expression, ignoring any issues of type
- **Conversion** [1.5] correct use of `float` to convert values (give 0.5 for using `int` instead)

Part (l) Calling Functions [2 marks]

Fill in the blank to call `city_elevation` to obtain the elevation (height above sea level) of Monkton.

```
def city_elevation(city):
    '''Return the elevation of the city (given as a string).'''

    ...  (The rest of the code for this function is not shown.)

    return elevation
```

Sample Solution:

```
print "The elevation of Monkton is", __ city_elevation("Monkton") _______________
```

Marking Scheme:

- **Call** [1] correct syntax for calling `city_elevation`
- **Argument** [1] correct syntax for the `str` argument `"Monkton"`

Question 2. [8 marks]

Part (a) [4 marks]

Complete the function `egg_category` which returns a `str` describing an egg's category given its `int` weight in grams. Here is a table specifying the weight ranges—if an egg's weight is on the boundary between two category ranges, it is assigned the smaller category.

Category	Weight
Small	no more than 50 grams
Medium	50–57 grams

Category	Weight
Large	57–64 grams
Jumbo	more than 64 grams

```
def egg_category(weight):
    '''Return a str describing the category of an egg of the specified int weight.
    '''
```

Sample Solution:

```
if weight <= 50:
    category = "Small"
elif weight <= 57:
    category = "Medium"
elif weight <= 64:
    category = "Large"
else:
    category = "Jumbo"

return category
```

Marking Scheme:

- **String** [1] always returning (not printing) a string descriptor
- **Only one** [1] handles only one category correctly
- **All** [1] handles all other category correctly
- **Boundary** [1] handles the boundary cases correctly

Marker's Comments:

- common error [−1]: writing "⩽" instead of "`<=`"

Part (b) [4 marks]

Complete the `main` block below. Your program should use `raw_input` to ask the user for an egg weight and should print the category for that weight, in the form: "`An egg of weight W is a C egg.`", where `W` is the weight the user entered and `C` is the category returned by your function from above. (Note that you can complete this part even if you did not write the function above.)

```
if __name__ == "__main__":
```

Sample Solution:

```
weight = int(raw_input("Please enter the weight of an egg: "))
print "An egg of weight", weight, "is a", egg_category(weight), "egg."
```

Marking Scheme:

- **Call** [1] clearly making a call to `egg_category` to get the category
- **Argument** [1] calling `egg_category` with an `int` argument
- **Input** [1] using `raw_input` correctly to get input from the user
- **Output** [1] formatting the output appropriately

Marker's Comments:

- common error: most students forgot to cast the string returned by `raw_input` into an `int`

Question 3. [8 MARKS]

Part (a) [4 MARKS]

Complete the function below according to its docstring.

```
def print_time(sec):
    '''A day has 86400 = 24 * 60 * 60 seconds.  Given an int in the range 0 to 86399,
    print the current time as hours, minutes, seconds on a 24-hour clock.  For example:
    >>> print_time(70000)
    19 h, 26 m, 40 s
    '''
```

SAMPLE SOLUTION:

```
    hrs = sec / 3600   # integer number of hours
    sec -= hrs * 3600  # remaining number of seconds
    min = sec / 60     # integer number of minutes
    sec -= min * 60    # remaining number of seconds

    print hrs, "h,", min, "m,", sec, "s"
```

MARKING SCHEME:

- **Print** [1] `print`ing rather than `return`ing information
- **Values** [2] correct values computed
- **Format** [1] following the format specified in the docstring

MARKER'S COMMENTS:

- common error [−0.5]: badly formatted output (*e.g.*, missing commas)
- common error: using "`+`" to concatenate a `str` and an `int`
- small arithmetical errors were penalized −0.5 to −1, depending on severity

Part (b) [4 MARKS]

Fill the table below with four **different** test cases for function `print_time` above—do *not* test for invalid inputs. For each test case, indicate clearly the expected outcome and your reason for choosing this case (in column "Explanation"). Note that you can answer this part even if you did not complete the code above.

Test Case	Expected Outcome	Explanation
`0`	`0 h, 0 m, 0 s`	Boundary case: smallest argument
`15`	`0 h, 0 m, 15 s`	Less than 1 minute but not zero
`900`	`0 h, 15 m, 0 s`	Less than 1 hour but more than 1 minute
`70000`	`19 h, 26 m, 40 s`	More than 1 hour
`86399`	`23 h, 59 m, 59 s`	Boundary case: largest argument

SAMPLE SOLUTION: (Any four cases similar to one of those above.)

MARKING SCHEME:

- 1 mark for each test case whose purpose is clearly different from the others (take off 0.5 for each missing outcome and 0.5 for each missing/unclear explanation)

MARKER'S COMMENTS:

- common error: testing for invalid input, including tests for non-`int` input or input out of range (read the question and the docstring carefully)

Question 4. [5 MARKS]

Part (a) [2 MARKS]

Write a suitable docstring for the following function.

```
def func(n):
    '''

    Return the value of 1**2 + 2**2 + ... + n**2
    for any integer n.

    '''
    total = 0
    while n > 0:
        total += n * n
        n -= 1
    return total
```

SAMPLE SOLUTION: (See above.)

MARKING SCHEME:

- **Parameter** [1] `n` is mentioned by name and its type specified
- **Return** [1] clear description of the return value

MARKER'S COMMENTS: Well done.

Part (b) [3 MARKS]

Complete the function below according to its docstring. *(Hint: Look at the string method `.isdigit()`.)*

```
def int_input(prompt):
    '''Repeatedly ask the user for a value, using string prompt, until the user enters
    an integer, then return that integer.
    '''
```

SAMPLE SOLUTION:

```
    value = raw_input(prompt)
    while not value.isdigit():
        value = raw_input(prompt)

    return int(value)
```

Marking Scheme:

- **Input** [1] correct use of `raw_input`
- **Loop** [1] correct loop, including stopping condition
- **Return** [1] function `return`s an `int` (no `print`ing)

Marker's Comments:

- common error [−0.5]: not casting return value to `int`
- common error [−0.5]: misunderstanding the use of `prompt`, *e.g.*, "`prompt = raw_input(...)`"

Question 5. [6 marks]

Write a Python program to control the temperature in a building, using the following functions.

- `current_temp()`: Return the current temperature. The initial temperature is `20`, and the temperature remains constant unless it is changed by one of the other functions.
- `raise_temp(deg)`: Raise the temperature by `deg` degrees.
- `lower_temp(deg)`: Lower the temperature by `deg` degrees.

Your solution will be graded on its design as well as its functionality.

In-Test Announcement:

> You must implement the functions (*i.e.*, write code for them), *not* just "use"them.

Sample Solution:

```
temp = 20  # Current temperature -- global.

def current_temp():
    return temp

def raise_temp(deg):
    global temp
    temp += deg

def lower_temp(deg):
    global temp
    temp -= deg
```

Marking Scheme:

- **Overall** [1] correct overall design (global variable with functions)
- **Initialization** [1] global variable initialized properly
- **Accessor** [1] correct code for `current_temp()`
- **Mutators** [1] correct code for `raise_temp` and `lower_temp`, ignoring local *vs.* global issues
- **Global** [1] correct use of `global` keyword in `raise_temp` and `lower_temp`
- **Syntax** [1] general Python syntax

Marker's Comments:

- Functions were not penalized if they failed to handle both positive and negative inputs.
- The functions should do *exactly* and *only* what is stated, so marks were taken off for printing or returning values in the mutators (`raise_temp` and `lower_temp`).
- Many students forgot the "`def`" keyword! Also, be careful with indentation and capitalization.
- Generally done well.

Question 6. [8 marks]

Complete the function below according to its docstring.

```
def longest_sequence(search_str, ch):
    '''Return the length of the longest consecutive sequence of the character ch in the
    string search_str.  For example:
    >>> longest_sequence("aababbbabb", "b")
    3
    >>> longest_sequence("aababbbabb", "a")
    2
    '''
```

Sample Solution:

```
    max_run = run = 0
    for c in search_str:
        if c == ch:
            run += 1
        else:
            if run > max_run:
                max_run = run
            run = 0
    return max_run
```

Marking Scheme:

- **Loop** [1] loop over entire `str`
- **Max** [2] tracks maximum sequence length and updates it appropriately
- **Current** [2] tracks current sequence length and updates it appropriately
- **Return** [1] returns an `int` (no printing)
- **Value** [1] correct value returned (or printed)
- **Syntax** [1] general Python syntax

Marker's Comments:

- common error: not finding the maximum run length
- common error: returning the number of non-consecutive occurrences