CSCI160 Computer Science I
Lab Assignment
Decision Structures

Pro Tip 1: Read this entire document before beginning coding
Pro Tip 2: Read this entire document before beginning coding
Pro Tip 3: Read this entire document before beginning coding

Chained conditionals

The **elif** statements only execute if the previous tests are **False.** The **else** clause only executes if **ALL** previous tests are **False**

```
if(x < y):
    print("x is less than y")
elif(x > y):
    print("x is greater than y")
else:
    print("x and y are equal")
```

Nested Conditionals

One conditional can also be "nested" within another. Essentially this means that you can define an if statement that will only execute if the previous if statement is **True.** This is the opposite of the **if . . . elif, elif** paradigm where an elif test executes only if the previous test is **False**

We could have written the above example like this . . . notice the indentation:

```
if(x == y):
    print("x and y are equal")
else:
    if(x < y):
        print("x is less than y")
    else:
        print("x is greater than y")
```

Compound Conditional Statements

The logical operators **(and, or, not)** often provide a way to simplify nested conditional statements. For example, we can rewrite the following code using a single conditional.

```
if(x > 0):
    if(x < 10):
        print("x is a positive single-digit number.")
'''
The print statement is executed only if we make it past both conditionals,
so we can get the same effect with the and operator:
'''

if(x > 0 and x < 10):
    print("x is a positive single-digit number.")
```

Problem Statement: Write a Python program to calculate the cost of a phone call.

The phone company applies the following rules to a phone call to calculate the charge:

- The minimum **before-tax** charge of 59.40 cents applies to all calls to any destination up to 50m away and 89.00 cents for any destination further than 50m away.
    - Define variables at the module level to store these charge amounts. **Name them descriptively**
- Calls are charged on a **per-second** basis at
    - 0.759 cents per second (when distance <= 50m) and
    - 1.761 cents per second (when distance > 50m)
    - Define variables at the module level to store these charge amounts. **Name them descriptively**
- **Off-peak calls** (from 19:00:00 to 06:59:59 the next day) are given a discount of
    - 40% off (distance <= 50m) and
    - 50% off (distance > 50m) off the above rate

- **Peak calls** (from 07:00:00 to 18:59:59) are not discounted
    - You must handle the scenario where the call **begins in off-peak time** and **ends in peak-time** and compile the appropriate charges
- If the call was a conference-call AND the destination is more than 50m away, there is a discount of 50% off **after** any off-peak discount (minimum charge still applies).
- Conference-calls over shorter distances are not discounted.
- Finally, VAT of 14% is added to give the final cost.

Your program should ask for the following input:

1. The starting time of the call in HH:MM:SS (I have provided code that will split the input into hours, minutes and seconds)
2. The duration of the call in MM:SS (I have provided code that will split the input into minutes and seconds)
3. The distance of the call's destination
4. Whether the call was a conference-call. This should be entered as "Y" for yes and "N" for no. You can easily test string equivalency using ==

    if(response == "Y"):

Note: You may assume that the user will enter valid input, and that no call will exceed 59 minutes and 59 seconds.

Your program should output the following information:

- The basic cost
- The off-peak discount
- The conference-call discount
- The net cost
- The VAT
- The total cost

Program Structure

You have been provided with starter files that contains the function definitions that you will need to complete and the input and split operations for the times. You may not change the names or parameters of these functions. The functions come with a descriptive documentation string (**docstring**). Leave these in . . . but pay close attention to what they are saying. Your code goes after these strings

Testing:

I want you to also submit the test cases that you used to determine that your code runs correctly. Here is an example of a collection of test cases that should all spit out the same result. You should be able to predict what your code will return with a test case.

These tests fall within off-peak hours, < 50 m distance, not a conference call

("23:59:59", "10:00", 39, “N")
("00:00:00", "10:00", 12, "N")
("00:00:01", "10:00",   9, "N")
("05:59:59", "10:00", 27, "N")
("06:00:00", "10:00", 42, "N")
("06:00:01", "10:00", 32, "N")
("19:00:00", “10:00",   1, "N")
("19:00:01", "10:00", 21, “N")

What are the other conditions that you need to determine tests for? Attach a text file with a collection of test cases for each condition. This will be graded.

Submission:

Attach **call_charge_app.py**, **call_charge_functions.py** and your **test cases**