Blue Belt Exam

Your name: _____ Email ID: _____

Congratulations for qualifying to take the Blue Belt exam!

This is a *pass/retry* exam — if you pass, you advance to the Blue Belt level (and are essentially guaranteed at least a B- in the class); if you do not pass it, you will have other opportunities to earn the Blue Belt, and there is no penalty to your grade (so long as you eventually reach the desired level before the end of the course).

Closed Resources, No Help. For this exam you should **work on your own** and are not permitted to use any resources other than your own mind and body, a simple writing implement, and **one page** of notes that you created.

Not like other exams. Because of the point-based grading used in most exams, they are usually designed to have some "easy" questions (to separate the total failures from the have-some-clue failures), mostly "moderate" questions (so most people can get around 70 points, depending on the grading scheme), a few harder questions (to allow a small group of students to get up to 90 points), and a few really challenging questions (to make sure no one gets a perfect score, since that would send a message that they have nothing left to learn).

The purpose of this exam is to determine if you deserve to be promoted to the Blue Belt level (and if not, to figure out what you need to learn to be promoted). So, instead of having questions over a range of difficulty levels on the same content, all the questions are roughly at the same level of difficulty to cover everything you should be able to do to deserve a Blue Belt promotion. All questions are at the level I would expect a Blue Belt to be able to answer (but that mostly wouldn't be answered well by a Green Belt). Hence, you should not be off-put if the early questions seem more difficult than you expect, but should realize that it is necessary to pass or nearly pass every (non-bonus) question to demonstrate that you should be promoted.

Answer well. Answer all 8 questions (and the optional ninth question). You do not necessarily need to answer every question correctly to pass the test, but do need to demonstrate good understanding of all of the concepts expected for the Blue Belt.

Your answers need to be clear and convince a reader that you understand well the essentials of the question. You will not be penalized for minor syntactic or notation problems, or if your code is not exactly correct, so long as it is close enough to convince us that you understand enough that you would have been able to eventually figure out the correct code on your own (using an interpreter).

Entering the Dojang

Master Kang is having trouble keeping all the belt levels in order, so has written a program to help keep track of them. The program is almost finished, except for the last line.

```
def belt_level(bcolor, levels):
    """
    Returns a number that is the level corresponding to the input belt color,
    given in the input levels.  If bcolor is not a belt color, returns -1.
    """
    if levels == []:
        return -1
    else:
        if levels[0] == bcolor:
            return 0
        else:
            return ____________________ # Question 1
```

It should have the behavoir shown below:

```
BELT_LEVELS = ["none", "white", "yellow", "green", "blue", "purple", "red", "black"]

def test_belts():
    assert belt_level("white", BELT_LEVELS) == 1
    assert belt_level("blue", BELT_LEVELS) == 4
    assert belt_level("black", BELT_LEVELS) == 7
    assert belt_level("orange", BELT_LEVELS) == -1
```

1. Complete the last line of the program above.

2. What is the (worst case) running time of the completed `belt_level` function? A good answer will state the asymptotic running time of the `belt_level` function, and be clear about the meaning of all variables you use. You may assume all operations are constant time (including list indexing, string equality testing, `range` and `len`).

Board Breaking

3. Write Python code to define a function, `next_belt(bcolor, levels)` that takes as input a belt color and returns the color of the next level belt. Once someone has reached the highest belt color, there is no higher belt color, so the result of `next_belt` should be the input color. (You don't need to worry about what your function does if `bcolor` is not a valid belt color, though.)

For example,

```
def test_next_belt():
    assert next_belt("none", BELT_LEVELS) == "white"
    assert next_belt("green", BELT_LEVELS) == "blue"
    assert next_belt("red", BELT_LEVELS) == "black"
    assert next_belt("black", BELT_LEVELS) == "black"
```

Master Kang also needs a way to keep track of all her students. She keeps a list of students, and represents each student with a list of three elements: name (a string), age (a number), and belt level (a string that is a belt color).

For example, here is a class with four students:

```
students = [["Anirod", 3, "yellow"],
            ["Naya", 4, "yellow"],
            ["Mingood", 1, "none"],
            ["Ana", 29, "red"]]
```

So, the belt level of the second student in the list ("Naya") is `students[1][2]`.

4. Define a function `promote_student` that takes three inputs: a student (a string that is the name of a student), a list of students (represented as a list of three-element lists as above), and a list of belt levels (as in the previous two questions). It should *modify* the value of the list passed as the second input to promote the student whose name matches the first input to the next level.

For example,

```
def test_promotion():
    students = [["Anirod", 3, "yellow"],
                ["Ayna", 4, "yellow"],
                ["Mingood", 1, "none"],
                ["Ana", 29, "red"]]
    promote_student("Anirod", students, BELT_LEVELS)
    assert students[0][2] == "orange"
    promote_student("Anirod", students, BELT_LEVELS)
    assert students[0][2] == "green"
    promote_student("Ana", students, BELT_LEVELS)
    assert students[3][2] == "black"
```

Hint: a good definition only needs three lines of code.

Sparring

Master Kang also needs a way to match up students in the class for sparring. She defines the function below, that takes as inputs a list of students (represented as in the previous problem) and a `better_sparring_matchup` function. The function passed as `better_sparing_match` takes as inputs three students, and returns `True` if the second input is a better sparring match for the first input than the third input would be, and `False` otherwise.

```
def find_sparring_partners(students, better_sparring_matchup):
    partners = []
    unmatched = students
    while len(unmatched) >= 2:
        student = unmatched.pop()
        best_match = 0
        for partner in range(1, len(unmatched)):
            if better_sparring_matchup(student, unmatched[partner],
                                       unmatched[best_match]):
                best_match = partner
        partners.append([student[0], unmatched[best_match][0]])
        unmatched.remove(unmatched[best_match])
    return partners
```

5. Which of these *correctly* describe the cost of `find_sparring_partners`? Circle *all* answers that are correct, and none of the incorrect answers. You should assume the cost of `better_sparing_matchup` is constant, as are the costs of all the operations used including list `remove` (which is certainly not constant time in reality) and `append`. It is not necessary to provide a justification for your answers if you are confident they are all correct, but if you are worried any of them are incorrect it is a good idea to explain your reasoning so we know what you are thinking and can potentially pass you on this question even if some of the answers are wrong.

a. The running time is *linear* in the size of `students`.

b. The running time is *quadratic* in the size of `students`.

c. The running time is *exponential* in the size of `students`.

d. The running time is in $O(N)$ where N is the number of students.

e. The running time is in $O(N^2)$ where N is the number of students.

f. The running time is in $O(2^N)$ where N is the number of students.

g. The running time is in $\Theta(N)$ where N is the number of students.

h. The running time is in $\Theta(N^2)$ where N is the number of students.

i. The running time is in $\Theta(2^N)$ where N is the number of students.

j. The running time is in $\Omega(N)$ where N is the number of students.

k. The running time is in $\Omega(N^2)$ where N is the number of students.

l. The running time is in $\Omega(2^N)$ where N is the number of students.

This repeats the code from the previous page (just so you don't need to flip back-and-forth to read it):

```
def find_sparring_partners(students, better_sparring_matchup):
    partners = []
    unmatched = students
    while len(unmatched) >= 2:
        student = unmatched.pop()
        best_match = 0
        for partner in range(1, len(unmatched)):
            if better_sparring_matchup(student, unmatched[partner],
                                       unmatched[best_match]):
                best_match = partner
        partners.append([student[0], unmatched[best_match][0]])
        unmatched.remove(unmatched[best_match])
    return partners
```

6. The `find_sparring_partners` function produces a reasonable output (at least the first time it is called), but has at least one serious flaw (hint: Master Kang keeps losing all her students every time they do sparring!) Describe it, and suggest a simple way to fix it.

Questions

7. What is a *computer*?

8. In what ways are the Colossus computer built to break the Lorenz cipher and one of the 200 computer cores in a typical modern cellphone similar and different? (A good answer will include at least 2 similarities, and at least 2 differences.)

Similarities:

Differences:

9. (Optional, not required to pass Blue Belt) Once the whole class has passed the Blue Belt level, we should:

a. Decide that its a bad idea to have no C's or below in the class and change the grading standards.

b. Have pizza in class.

c. Have cupcakes in class.

d. Move the next Friday class to Jump Trampoline Park.

e. Other idea:

End of Exam. Feel free to use the back for any additional comments.

