CS 149 Final Exam Practice

1. **Terminology**

```
import sys

MIN_CHARGE = 10.0

class Student:

    total_fees = 0

    def __init__(self, name, sid):
        self.name = name
        self.sid = sid
        self.balance = 0

    def apply_charge(self, amount):
        charge = amount
        if charge <= MIN_CHARGE:
            charge = MIN_CHARGE

        self.balance += charge
        Student.total_fees += charge

    def __eq__(self, other):
        print("DEBUGGING")
        return self.name == other.name and self.sid == other.sid


if __name__ == "__main__":
    guido = Student("Guido", 1000)
    alice = Student("Alice", 1001)
    alice.apply_charge(float(sys.argv[1]))
```

Circle and label an example of each of the following in the code sample above.

A. class variable
B. local variable
C. instance variable
D. global variable
E. method call
F. constructor call
G. variable containing a student object
H. command line argument
I. string literal
J. relational operator
K. Boolean operator
L. class definition
M. method definition
N. constructor definition
O. parameter
P. argument

2. **Tracing OO Code**

Determine what will be printed by each of the following code snippets given the `Student` class from the previous page. Each snippet should be considered independently: do not assume that they are executed sequentially, or that the code in the `__name__ == "__main__"` block has been executed. If a snippet will result in an error, write ERROR.

a.

```
s1 = Student("Greta", 1000)
s1.apply_charge(5.0)
print(f"{s1.name} {s1.balance}")
```

b.

```
s1 = Student("Greta", 1000)
s2 = Student("Harry", 1000)
s1.apply_charge(12.0)
s2.apply_charge(6.0)
print(f"{s1.name} {s1.balance}")
print(f"{s2.name} {s2.balance}")
print(f"{Student.total_fees}")
```

c.

```
s1 = Student("Greta", 1000)
s2 = Student("Harry", 1000)
s3 = s2

s3.apply_charge(12.0)
s2.apply_charge(6.0)

print(f"{s1.name} {s1.balance}")
print(f"{s2.name} {s2.balance}")
print(f"{s3.name} {s3.balance}")
```

d.

```
s1 = Student("Greta", 1000)
s2 = Student("Greta", 1000)
s1.apply_charge(100.0)
print(s1 is s2)
print(s1 == s2)
```

e.

```
s1 = Student("Greta", 1000)
print(s1 == "Greta")
```

3. **Nested For Loops**

Determine what will be printed by each of the code snippets below.

a.

```
letters = [["A", "B", "C"], ["X", "Y"]]

result = ""
for var1 in letters:
    for var2 in var1:
        result += var2
    result += "\n"

print(result)
```

b.

```
var1 = "cat bat"
var2 = var1.split()

for first in var2:
    print(first)
    for second in first:
        print(second)
```

c.

```
result = ""
for i in range(2, 5):
    for j in range(i):
        result += "*"
    result += "\n"

print(result)
```

4. **Finding Errors**

For each set of code snippets below, only one of the options will execute without error. Circle the block that will execute successfully.

a.

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
for i in range(0, len(my_dictionary)):
      print(my_dictionary[i])
```

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
for i in my_dictionary:
      print(f"{i} -> { my_dictionary[i] }")
```

b.

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
while i in my_dictionary:
      print(f"{i} -> { my_dictionary[i] }")
      i += 1
```

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
for i in my_dictionary:
      print(i + " -> " + str(my_dictionary[i]))
```

c.

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
if len(my_dictionary) > 7:
      print("this is a large dictionary.")
else:
      print("this is a normal-sized dictionary.")
elif len(my_dictionary) < 3:
      print("this is a small dictionary.")
```

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
if len(my_dictionary) > 7:
      print("this is a large dictionary.")
elif 3 <= len(my_dictionary) <= 7:
      print("this is a normal-sized dictionary.")
else:
      print("this is a small dictionary.")
```

```
my_dictionary = { "a": 1, "b": 2, "c": 3 }
      if len(my_dictionary) > 7:
      print("this is a large dictionary.")
elif 3 < len(my_dictionary) < 7:
      print("this is a normal-sized dictionary.")
else:
      print("this is a small dictionary.")
```

5. **Developing Unit Tests**

Complete the following test file so that it provides 100% statement coverage of the `Student` class defined above.

```
"""Unit tests for the Student class."""
import unittest
from student_accounts import Student

class TestStudent(unittest.TestCase):

    def test_constructor(self):
        student1 = Student("Fiona", 2000)
        self.assertEqual(student1.name, "Fiona")
        self.assertEqual(student1.sid, 2000)
        self.assertEqual(student1.balance, 0)

    # ADDITIONAL TEST METHODS HERE...









if __name__ == "__main__":
      unittest.main()
```

6. **File IO**

```
def add_numbers_in_file(file_name):
    """Return the sum of all numbers that appear in the provided file.

    Each line in the file will contain an arbitrary sequence
    of numbers separated by white-space. This function will read
    the file and return the sum of all numbers. For example, if the file
    contains the following four lines:

    12.0 2.0

    6.0
    10.0 2.5 2.5

    Then the return value will be 35.0

    Args:
        file_name (str): The name of a file. E.g. "numbers.txt"

    Returns:
        float: The sum of all numbers stored in the indicated file.
    """
```