Name:  

Person to your left:  Person to your right:  

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Answer</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Expressions</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Code Snippets</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. For each question below, give a short answer - a few words or symbols, maybe a sentence or two.

(a) (2 points) Using the proper Python syntax, give an example of a list with at least four elements.

**Solution:** Solutions will vary.

(b) (2 points) Suppose that we have a list named `blue`. Give one line of code which will add the element 123 to the end of the list.

**Solution:** `blue.append(123)`

(c) (2 points) Give the value of `x` at the end of the following snippet:

```python
x = [1,2,3]
y = [x,x,x]
y[2].append(100)
```

**Solution:** `[1,2,3,100]`

(d) (2 points) What does it mean that a variable is immutable?

**Solution:** It cannot be changed.

**Instructor’s Note:** Technically, it’s imprecise (in Python) to say that a **variable** is immutable, since variables just hold references to things. `x` might refer to a tuple today (immutable) but a list tomorrow (mutable).

Technically, we should say that some **objects** are immutable - and that a variable might (for now) be pointed at an immutable object. But we often get a little imprecise with our terminology when talking about variables and the objects they refer to.

(e) (2 points) What is the difference between a class and an object?

**Solution:** A class is the pattern (gives the code and data fields), an object is one instance of that class.

(f) (2 points) In Python, how do we mark a field of a class as private?

**Solution:** The name should start with a single underscore.
(g) (2 points) At runtime, what does an assert do?

**Solution:** It checks to see if a condition is true. If it is not, it kills the program.

**Instructor’s Note:** Technically, it throws an exception. Either an informal or formal description will be accepted.

(h) (2 points) Assume that we have a list, tuple, or string named `thing`. Use a slice to make a copy of it - removing the first and last elements. Store the result into a variable named `smaller`. (You may assume that the `thing` has at least two elements.)

**Solution:** `smaller = thing[1:-1]`

**Instructor Got This Wrong The First Time!**
The ‘end’ index for a slice is **exclusive** - meaning that specifying an end index of -1 (the last element) actually excludes that element.

In other words, to read the next-to-last element, you use this:

```
thing[-2]
```

But to slice off the last element (but not the second-to-last), you use this:

```
thing[:-1]
```

(i) (2 points) What is multiple assignment in Python?

**Solution:** Multiple assignment allows you to take two or more values, and assign them to an equal number of variables, in one step.

(j) (2 points) Assume that you have two variables, named `fred` and `barney`. Create a string (using the pattern below) which includes the value of both; store that string into the variable `message`. For instance, if `fred=123` and `barney=[10,11]`, then the string should be:

Fred: 123   Barney: [10,11]

**Solution:**

```
message = "Fred: {}   Barney: {}".format(fred,barney)
```
2. (20 points) Assume that we have the following variables defined:

```python
a = "The quick brown 
 b = "fox jumps over the"
 c = [10,13,-17]
 d = []
 e = {3:"foo", -15:"bar"}
 f = (a,c,e)
```

For each part below, I have given you an expression using the variables above. Give the value of that expression.

In some cases, the order of the values in a list may not be predictable; make a note of all such cases. (You will only get partial credit if you do not.)

(a) \(a+b\)

**Solution:** "The quick brown fox jumps over the"

(b) \(f[-1][-15]\)

**Solution:** "bar"

(c) \(\text{len}(c)\)

**Solution:** 3

(d) \(\text{tuple}(c)\)

**Solution:** (10,13,-17)

(e) \(c*2+d\)

**Solution:** [10,13,-17,10,13,-17]

(f) 13 in e

**Solution:** False

(g) 13 in c

**Solution:** True
(h) \text{sum}(c)

\text{Solution: 6}

(i) \text{keys}(e)

\text{Solution: [3,-15]}

(j) (\text{len}(a)+\text{len}(b)) == \text{len}(a+b)

\text{Solution: True}

(k) a.split()

\text{Solution: ["The","quick","brown"]}

(l) " ".join(e.values())

\text{Solution: "foo bar" (or "bar foo")}
3. (20 points) Write a class named `Exam1`. It should have three properties (you may choose their names):

- One should be passed as a parameter to `__init__()`, but should be immutable - that is, it should be impossible to change it.
- One should be passed as a parameter to `__init__()`, and it should have a setter method.
- One should not be passed as a parameter to `__init__()`; instead give it a default value. Also include a setter for this.

Don’t write any getters, to save time.

**Solution:**

```python
class Exam1:
    def __init__(self, alpha, bravo):
        self._alpha = alpha
        self._bravo = bravo
        self._charlie = 13  # default value

    def set_bravo(self, new_bravo):
        self._bravo = new_bravo

    def set_charlie(self, new_charlie):
        self._charlie = new_charlie
```

Page 6
4. (a) (10 points) Write a snippet of code which will print all of the integers from 1 to 100, inclusive (but in reverse order). Print one per line.

Solution:
```python
for i in range(100, 0, -1):
    print(i)
```

(b) (10 points) Write a function named `powers(val, max_pow)`. It should return a list of values, which are the powers of `val`, from $val^0 = 1$ to $val^{max_pow}$.

Solution:
```python
def powers(val, max_pow):
    retval = []
    cur_val = 1
    for p in range(max_pow + 1):
        retval.append(cur_val)
        cur_val *= val
    return retval
```

Instructor’s More Advanced Solution (list comprehensions)
```python
def powers(val, max_pow):
    return [val**i for i in range(max_pow+1)]
```
(c) (10 points) Write a snippet of code that will 1) read a line of input from the user; 2) split it into words (separated by whitespace); 3) convert each word to an integer (you don’t have to check for exceptions here); and 4) print out the sum of all of those integers.

Solution:
words = input().split()
total = 0
for word in words:
    total += int(word)
print(total)

Instructor’s More Advanced Solution

words = input().split()
vals = map(int, words)  # map applies a funct to all elements, returns a list
print(sum(vals))

Even denser:
print(sum(map(int, input().split())))

(d) (10 points) Write a function add_pairs(vals) which, when given a list, will return a new list, where each pair of adjacent values have been added together. Thus, the new list is half the length; for instance, add_pairs([1,10,23,47]) should return [11,70].

Use an assert to ensure that the length of the parameter is an even number of elements.

Solution:
def add_pairs(vals):
    assert len(vals) % 2 == 0
    retval = []
    for i in range(0,len(vals), 2):
        retval.append(vals[i]+vals[i+1])
    return retval

Alternate: Slicing
def add_pairs(vals):
    assert len(vals) % 2 == 0
    retval = []
    while len(vals) > 0:
        retval.append(vals[0]+vals[1])
        vals = vals[2:]
    return retval

More Advanced Solution (list comprehensions)
def add_pairs(vals):
    assert len(vals) % 2 == 0
    return [vals[i]+vals[i+1] for i in range(0,len(vals), 2)]