# Exam 1

Wed 28 Jun 2017

<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>Total:</td>
<td>100</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.

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

(c) (2 points) Give the value of \texttt{x} at the end of the following snippet:
\begin{verbatim}
x = [1,2,3]
y = [x,x,x]
y[2].append(100)
\end{verbatim}

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

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

(f) (2 points) In Python, how do we mark a field of a class as private?
(g) (2 points) At runtime, what does an assert do?

(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.)

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

(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:

```python
Fred: 123   Barney: [10,11]
```
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
(b) f[-1][-15]
(c) len(c)
(d) tuple(c)
(e) c*2+d
(f) 13 in e
(g) 13 in c
(h) sum(c)
(i) keys(e)
(j) (len(a)+len(b)) == len(a+b)
(k) a.split()
(l) " ".join(e.values())
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.
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.

(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 $val^{max\_pow}$. 
(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.

(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.