Practise Quiz
AI - Fall 2017

For each code snippet, find a way to express it in one line. If it is already on one line, find a way to make it shorter. If the variable name indicates a data type, you may assume it is so (eg. `listOfInts`).

A. Same problem as on the prior worksheet: `{*range(0,91,10)}.` The vast majority got this right

Some incorrect answers:

`*range(0,91,10)`	`{*range(0,99,10}`	`{*range(0,10,91)}`	`{* range*0,99,10)}`
`{*range(0,101,10)}`	`{*range(0, 90, 10)}`		

Suboptimal answer:
`{*range(0,100,10)}`

B. Although it doesn't say, we assume that the key is there. `myDct["myKey"]`

Some incorrect answers:

`myDct[myKey]`	forgot the quotes	
`myDct["mykey"]`	Capitalization	
`myDict["myKey"]`	Typo	10% (!)

C. Looking for either one of
```
val1, val2 = val2, val1
val2, val1 = val1, val2
```

D. Looking for some variation of
```
if sn2 in {sn, symNum): continue
if sn2 in (sn,symNum): continue
if sn2 in [sn, symNum]: continue
```
Note that the continue must be included. This wasn't a question about binary representations.

E. Most common was:

```
return myNum != 27
```

Note that parens aren't required. Shorter (one person put this down) is the below. While it doesn't return a Boolean, it should be OK:

```
return myNum - 27
```

The most common incorrect answer was:

```
return myNum == 27
```

F. You were meant to respond with one of:

```
if myKey not in myDct:
if not myKey in myDct:
```

The colon is required.

G. This is the most difficult question on here. Please consult the answer to K first.
An example is `lstOfLstOfSetOfInts = [[{1,2}, {3,4,5}], [{6}, {7,8,9}, {10}]]`
The issue is that you must explicitly copy the lowest level elements because otherwise you'll wind up with a pointer to the element meaning that if you change part of the copy, you'll also be changing the original. For each counterexample, the final test is to show `lstOfLstOfSetOfInts`. If it is altered, the test has failed.

Examples of tries that don't work:

Try:	Counterexample
`myCopy = lstOfLstOfSetOfInts[:]`	`myCopy[1][2] = 13`
`myCopy = [*lstOfLstOfSetOfInts]`	same
`myCopy = [x for x in lstOfLstOfSetOfInts]`	same
`myCopy = [x[:] for x in lstOfLstOfSetOfInts]`	`myCopy[1][2].add(14)`
`myCopy = [[s for s in i] for i in \` `lstOfLstOfSetOfInts]`	same

Correct answers:
```
myCopy = [[*map(set, l)] for l in lstOfLstOfSetOfInts]
myCopy = [[{*s} for s in i] for i in lstOfLstOfSetOfInts]
myCopy = [[set(s) for s in i] for i in lstOfLstOfSetOfInts]
myCopy = [[{*[*s][:]} for s in i] for i in lstOfLstOfSetOfInts]
```
This latter is somewhat overkill on the dereferencing (eg. `[:]` is redundant). And `*s` accomplishes the same thing as `*[*s]`.

H. Since only odd numbers are sought, division by 8 is irrelevant. It also means that 100 is not included. Some correct answers:
```
[i for i in range(9,98,2) if(i%7)*(i%9)]
[j for j in range(9,98,2) if j%7 if j%9]
[j for j in range(9,98,2) if j%7 and j%9]
```

I. Correct answers:
```
binNum1 ^ binNum2                          - exclusive or
(binNum1 | binNum2) - (binNum1 & binNum2)    - any bit on less those with both bits on
```

Some incorrect answers:
```
binNum1 & binNum2        - That ands corresponding bits.
binNum1 | binNum2        - That ors corresponding bits
```

J. This question is a bit ambiguous since we've been working with Sudoku where we subtract the period out. For the purposes of this question, let's assume that we want to find all the symbols that occur in the list or string `pzl`. If your answer included a `-{"."}` then of course that would not count against you, but that is not indicated in the below.

Correct answers:

```
{*pzl}
set(pzl)
{i for i in pzl}
```

Incorrect answers:

`{*puzl}`	- Need to pay closer attention to what is given
`{*myStr}`	- same as first example
`[*{pzl}]`	- This gives a list instead of a set. Also, it appears it untested, since it gives a list of exactly one element
`list(set(pzl))`	- Just want a set and not a list.
`s = {*pzl}`	- Not looking to set a variable
`[*{*pzl}]`	- Overeager to use new knowledge; we're looking for a set

K. It is most important to understand this problem. The obvious answer of `[{}]*30` or the equivalent `[dict()]*30` is incorrect because what happens is that **the same** empty dictionary is replicated. If you have python display it, it looks OK. And if you do:

```
lst = [{}]*30
lst[2]=3
```

and then display `lst`, it will still look OK because the item at that position is being replaced.
However, if you do

```
lst[4][5] = {6, 7}
```

you'll be in for a bit of a surprise when you display `lst`. Since it is the same dictionary that is being replicated, if you change part of the dictionary (as opposed to overwriting it), that change will show up in all the positions (rather than just the dictionary at index position 4) since each position is pointing to the same dictionary. In order to address this problem, you should create a separate empty dictionary at each index position. Thus, some correct answers:

```
[{} for i in range(30)]
[dict() for i in range(30)]
```

This same type of thing goes on with G when you do `[:]` or dereference or use `.copy()`. A shallow copy is made, which means that everything at the next level down is simply pointed to (and not copied). That is why you must manually do the copies with a list comprehension, and you can only do the shallow copy (eg. `set(…), list(…), [:], .copy()`) at the final level.

L. A gift: `2*myInt`

M. Another gift (you may assume myInt is not 0):

```
1
1.0
```

The division actually produces 1.0, but in almost all circumstances, it will be sufficient to write 1.

N. Correct answers:

```
sum(i%7==0 for i in listOfInts)
sum(not i%7 for i in listOfInts)
sum([i%7==0 for i in listOfInts])
sum([not i%7 for i in listOfInts])
[i%7 for i in listOfInts].count(0)
sum(1 for x in listOfInts if x%7==0)
len([i for i in listOfInts if i%7==0])
```

O. This is simply: `myStr[i]`

P. This one has a few approaches. If one allows altering the underlying set, `mySet`, the most common answer was `mySet.pop()`
However, other answers are possible:

```
min(mySet)
max(mySet)
[*mySet][0]
{*mySet}.pop()
list(mySet)[0]
next(iter(mySet))
```

Incorrect answers include:

`mySet[0]`	– `mySet` does not have a 0th element
`[*mySet[0]]`	– `mySet` does not have a 0th element
`mySet.get()`	– no such operator
`mySet.add(mySet.pop())`	– doesn't return a value