1 Python for Data Analysis 101

1.1 Homework

1.1.1 Instructor: Evelyn J. Boettcher, DiDacTex, LLC

1.1.2 Week 1: Lecture 3

1.2 HW

- Write a conditional expression statement with, “if, elif and else.” to check if my˙string = “Hello World” is a string
- Write a list comprehension that returns all the keys in a dictionary whose associated values are greater than zero.
  - The dictionary: {`'aa': 11, 'cc': 33, 'LESS Than': -55, 'bb': 22`}
  - Output should be a list
- Write a list comprehension that produces even integers from 0 to 10. Use a `for` statement to iterate over those values and print results to screen.
- Write a list comprehension that iterates over two lists and produces all the combinations of items from the lists.
- Using `break`, write a `while` statement that prints integers from zero to 5.
- Using `continue` , write a `while` statement that processes only even integers from 0 to 10. Note: `%` is the modulo operator. Solution:

1.3 Solutions

HW Exercise:

- Write a list comprehension that returns all the keys in a dictionary whose associated values are greater than zero.
  - The dictionary: {`'aa': 11, 'cc': 33, 'LESS Than': -55, 'bb': 22`}

Solution:

```
my_dict = {'aa': 11, 'cc': 33, 'LESS Than': -55, 'bb': 22}
my_list = [x[0] for x in my_dict.items() if x[1] > 0]
```

HW Exercise: - Write a list comprehension that produces even integers from 0 to 10. Use a `for` statement to iterate over those values and print results to screen.

Solution

```
for x in [y for y in range(10) if y % 2 == 0]:
    print( 'x: %s' % x)
```

HW Exercise: - Write a list comprehension that iterates over two lists and produces all the combinations of items from the lists.

Solution:

```
a = range(4)
b = [11, 22, 33]
print(a)  # Out[21]: [0, 1, 2, 3]
print(b)   # [11, 22, 33]
c = [(x, y) for x in a for y in b]
print(c)  # [(0, 11), (0, 22), (0, 33), (1, 11), (1, 22), (1, 33), (2, 11), (2, 22), (2, 33), (3, 11),
```

But, note that in the previous exercise, a generator expression would often be better. A generator expression is like a list comprehension, except that, instead of creating the entire list, it produces a generator that can be used to produce each of the elements.

The `break` and `continue` statements are often useful in a `for` statement.

HW Exercise:

- Write a `while` statement that prints integers from zero to 5.

Solution:

```
count = 0
while count < 5:
    count += 1
    print(count)
```

Exercise:

- Using `break` , write a `while` statement that prints integers from zero to 5.

Solution:

`python count = 0 while True: count += 1 if count > 5: break print(count)`

Notes:

- A `for` statement that uses `range()` would be better than a `while` statement for this use.

HW Exercise: - Using `continue` , write a `while` statement that processes only even integers from 0 to 10. Note: `%` is the modulo operator. Solution:

Solution:

```
count = 0
while count < 10:
    count += 1
    if count % 2 == 0:
        continue
    print(count)
```