Lesson 3 - Thomas Deneuville

October 3, 2017

This session is going over the problems that you solved on CodingBat as well as some acceptable practices for Boolean expressions.

```
In [1]: def sleep_in(weekday, vacation):
            if weekday == False or vacation == True:
                return True
            else:
                return False

        # Better to do it this way with the expression being
        # a single statement
        def sleep_in2(weekday, vacation):
            return not weekday or vacation

        # Same here
        def monkey_trouble(a_smile, b_smile):
            return a_smile == b_smile

        # Solution for missing_char - try not to use
        # lists to solve this problem
        def missing_char(s, n):
            return s[:n] + s[n+1:]
```

```
In [2]: print(missing_char('kitten', 0))
        print(missing_char('kitten', 1))
        print(missing_char('kitten', 4))
```

```
itten
ktten
kittn
```

```
In [3]: def front_back(s):
            if len(s) == 1:
                return s
            else:
                # s[-1] accesses the last character
                # s[1:-1] gives me the second character to the second last character
                # 1 is the beginning index
                # -1 is the last element, but we exclude this from slicing
                # s[0] is the first character
                return s[-1] + s[1:-1] + s[0]

In [4]: print(front_back('code'))
        print(front_back('a'))
        print(front_back('ab'))

eodc
a
ba


In [5]: def front3(s):
            #str_list = list(s)
            #str3 = str_list[:3]
            #str3 = ''.join(str3)

            # If there are less than three characters, just use the
            # front as is, if not, take the first three characters
            # for the front
            if len(s) < 3:
                str3 = s
            else:
                str3 = s[:3]
            return 3*str3

In [6]: print(front3('Java'))
        print(front3('Chocolate'))
        print(front3('Ray'))
        print(front3('a'))

JavJavJav
ChoChoCho
RayRayRay
aaa


In [7]: def front_times(s,n):
            if len(s) < 3:
                str3 = s
            else:
                str3 = s[:3]
            return n*str3

In [8]: print(front_times('Ray', 2))
        print(front_times('Thomas', 4))
```

```
RayRay
ThoThoThoTho
```

```
In [9]: def string_splosion(s):
            r = '' # Start with the empty string
            # For 0, 1, 2 up to the length of the string - 1
            for i in range(len(s)):
                # i accesses the end of the slice we need, then
                # we need to add 1 because the slicing is exclusive
                # We access more of the string at each iteration and concatenate
                r += s[:i+1]

            return r
            # You can also do the code below
            #return ''.join([s[:i+1] for i in range(len(s))])
```

```
In [10]: print(string_splosion('Code'))
```

```
CCoCodCode
```

```
In [11]: def last2(s):
             if len(s) <= 2:
                 return 0

             start = 0 # Remembers the last position of where we found the substring
             count = 0 # Remembers how many times we have seen the substring
             sub = s[-1:-3:-1][::-1] # Gets the last two characters
             # We don't remove the last two characters - the entire string is searched
             # but we stop when our index is greater than the length of the string
             # minus 2
             while True: # Loop until we don't find any more substrings
                 # find method gives you the starting location of the substring
                 # Returns -1 if we don't find it
                 start = s.find(sub, start)

                 # If we don't find the substring, or if the index of where
                 # we found the string is beyond the length of the string minus
                 # 2, we simply quit
                 if start == -1 or start >= len(s) - 2:
                     break
                 else:
                     # We have found the substring - increase the count by 1
                     count = count + 1

                     # Make the starting position move over by 1 so we can
                     # search for substrings in the next position after
                     # previous starting position
```

```
                start = start + 1

        return count # Returns # of times we see the substring
```

```
In [12]: print(last2('xxxx'))
```

```
2
```