# Software for Digital Innovation (CIS4044-N)

# Tutorial 7: List Comprehension and Password Mangling

## Before You Start

It is imperative that you complete last week’s tutorial, as you will need those skills to complete you ICA Part 2.

## Introduction

This session aims to familiarise you with list comprehensions and higher-order functions.

## Activity 1: Portfolio Catch-up

Take this opportunity to catch up on your portfolio work. You should now have portfolio 3 complete (review is in this weeks session) and made a start on portfolio 4. If you are unclear on any individual keywords, please speak with you tutor.

## Activity 2: List comprehension

List comprehensions allow you to create a new list based on another in a single line of code. Let’s say we wanted to square a list of numbers:

squared = []

for num in range(1, 6):

    squared.append(num \*\* 2)

print(squared)

We could simplify to:

squared = [num \*\* 2 for num in range(1, 6)]

print(squared)

We can also add conditions to the end of our expression. The pseudocode would be:

my\_list = [calculation for variable in iterable if condition]

Try to create list comprehensions for the following:

1. Given a list of numbers, return a new list but add 3 to each item.
2. For the following list, return a new list of all positive numbers, but as integers.

nums = [33.6, -210.1, 55.3, 28.4, -10.2, 77.9, 22.7]

1. A list of all numbers divisible by 7 in the range 1 – 100.
2. For the string “An Apple a Day Keeps The Doctor Away”, return a list of all capital letters.
3. Given a list of strings, create a new list which contains all the palindromes (a word, phrase, or sequence that reads the same backwards as forwards). You can use the list following list to test:

["civic", "motor", "level", "madam", "hello", "racecar", "python", "rotor"]

1. Given a string, count the number of spaces.
2. Given a string, return a new string but remove all the vowels.

## Activity 3: P@55word Mangling

In this week’s lecture we discussed possible methods for mangling text to form many permutations of mangled passwords.

# Small example of applying mangling rules to "password"

# James Fairbairn 02/03/2020

def capitalise(base):

    return [base.capitalize()]

def up(base):

    return [base.upper()]

def replace\_a(base):

    return [base.replace('a', '@')]

def append\_num(base, start=0, end=100):

    return [base + str(num) for num in range(start,end)]

base = "password"

test\_rules = [replace\_a, up, append\_num]

permutations = [base]

for rule in test\_rules:

    new\_permutations = []

    for perm in permutations:

        new\_permutations.extend(rule(perm))

    permutations.extend(new\_permutations)

print(permutations)

print(len(permutations))

1. Download and run the example above from Blackboard.
2. Try adding different mangling rules (this will require *independent* research) as functions and extend the test\_rules list. How many possible permutations can you produce (you should aim for 10000+).
3. When you are satisfied with the number of permutations for a mangled “password”, update the program to take an intput() string. Is your program affected by string length?
4. Are there opportunities for you to use map(), filter(), or reduce()?

## Document History

Revision 0 (08-Nov-20): This is the initial version of the 2020/21 exercise.