

Name: _____ Partner: _____

Python Activity 48: List Comprehensions

Very common actions, like making a simple list, have some shortcuts in python.

Learning Objectives

Students will be able to:

Content:

- Define a **list comprehension**
- Describe the key pieces of constructing a list comprehension

Process:

- Write code to construct lists using list comprehensions with **mapping** and **filtering**
- Convert multi-line list construction loops into one-line list comprehensions.

Prior Knowledge

- Python concepts: lists, for-each loops, conditionals, range(), file reading

Critical Thinking Questions:

1. Examine the sample code that converts a list of US Dollar amounts to British pound.




Sample Code

```
0 monies = [1.22, 5.50, 3]
1 gbp = []
2 for usd in monies:
3     gbp += [usd*0.9]
```

- What is the accumulator variable? _____
What is the looping variable? _____
What is the sequence we're looping over? _____
- What part of the code converts the values of `monies` from USD to GBP? _____
- What is being added to the accumulator variable? _____
- What are the elements of the list, `gbp`, at the end of this code? _____

2. The following code below results in *identical outcomes* as the above Sample Code:

```
0 monies = [1.22, 5.50, 3]
1 gbp = [ usd*0.9 for usd in monies ]
```

-  What is the accumulator variable? _____
What is the looping variable? _____
What is the sequence we're looping over? _____
-  What part of the code converts the values of `monies` from USD to GBP? _____
-  What is being added to the accumulator variable? _____

- d. What are the elements of the list, `gbp`, at the end of this code?
-
- e. How do we know that `gbp` is a list? (Hint: What punctuation typically indicates lists?)
-

FYI: *List Comprehensions* provide a concise way to create & manipulate lists and are particularly useful for two of the common patterns we see when using lists and loops. One of these patterns is **mapping** in which we iterate over a list and return a new list that results from *performing an operation on each element* of the original list, as in the example above.

3. Examine the sample code below which also uses a list comprehension:

Sample Code

```
0 words = ["short", "petite", "loooooong", "puny"]
1 longer = [wd for wd in words if len(wd) > 6 ]
```

- a. What differs in this list comprehension that we did not have in the previous USD/GBP example?
-
- b. What does the variable `wd` represent in this code?
-
- c. What does the code `if len(wd) > 6` do?
-
- d. Why is this line of code enclosed in square brackets?
-
- e. When this code completes execution, `['loooooong']` is stored in the `longer` variable. Why might this be?:
-



- f. Write code to create a list that contains only words that begin with the letter 'p'. Use a list comprehension:
-
-

FYI: A second common pattern that we often use a list comprehension for is **filtering** in which we iterate over a list and return a new list that results from *keeping only elements of the original list that satisfy some condition*, as in the example above.

FYI: You can imagine visually breaking down the syntax of a list comprehension as follows:

```
result_list = [ <transform> <iteration> <boolean conditional> ]
```

The Boolean conditional works as a filter and may be omitted. Likewise, the transformation may not actually change the value.

4. Examine the following code:

```
0 test_str = "Hello 12345 World"
1 new_lst = []
2 for x in test_str:
3     if x in "1234567890":
4         new_lst += [x]
```

- a. What does the code on line 3 do?

- b. What will `new_lst` contain when this code completes execution?

- c. Is this an example of **mapping** or **filtering**? _____



- d. Construct a list comprehension that accomplishes the same tasks as this example code:

5. Examine the following code from an interactive Python session:

```
0 >>> def has_sub(word, substring):
1 ...     return substring in word
2 >>> names = ['pixel', 'sally', 'wally', 'artie', 'jerry']
3 >>> similar = [ dog for dog in names if has_sub(dog, 'lly') ]
4 >>> similar
5 ['sally', 'wally']
```

- a. When we call `has_sub(dog, 'lly')` on line 3, what does the function return?



- b. Construct a list comprehension that accomplishes the same tasks as this example code, but without the function `has_sub(..)`:

Application Questions: Use the Python Interpreter to check your work

1. Write a list comprehension to make a copy of the list, `my_lst`:

2. Write a list comprehension to create a list of all numbers between 0 and 10 (*Hint: `range(..)`*):

3. Write a function that capitalizes a list of strings into a new list, using list comprehensions. Return the new list. Do not modify the given list!

```
def capitalize(string_lst):
```

4. Write a list comprehension to generate a list, words, where each element is a line from a file, /usr/share/dict/words, stripped of leading and trailing whitespaces:

```
words = _____
```

5. Write a function that returns a list containing the values of num_lst squared. Use a list comprehension. Do not modify the given list, num_lst!

```
def squared(num_lst):
```

6. Using a list comprehension, write a function that returns a list containing the values of num_lst squared, but only of the prime numbers in numList. You can use the function is_prime(..) to determine if a given number is prime. Return the new list. Do not modify the given list!

```
def square_primes(num_lst):
```

```
def is_prime(num):
```

```
    # returns True if num is a prime number, False if it isn't.
```