23.5. Zip

One more common pattern with lists, besides accumulation, is to step through a pair of lists (or several lists), doing something with all of the first items, then something with all of the second items, and so on. For example, given two lists of numbers, you might like to add them up pairwise, taking [3, 4, 5] and [1, 2, 3] to yield [4, 6, 8].

One way to do that with a for loop is to loop through the possible index values.

Save & Run Original - 1 of 1 Show in CodeLens

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L3 = []

for i in range(len(L1)):
    L3.append(L1[i] + L2[i])

print(L3)

```

```
[4, 6, 8]
```

Activity: 1 -- ActiveCode (ac21_5_1)

You have seen this idea previously for iterating through the items in a single list. In many other programming languages that's really the only way to iterate through the items in a list. In Python, however, we have gotten used to the for loop where the iteration variable is bound successively to each item in the list, rather than just to a number that's used as a position or index into the list.

Can't we do something similar with pairs of lists? It turns out we can.

The `zip` function takes multiple lists and turns them into a list of tuples (actually, an iterator, but they work like lists for most practical purposes), pairing up all the first items as one tuple, all the second items as a tuple, and so on. Then we can iterate through those tuples, and perform some operation on all the first items, all the second items, and so on.

Save & Run Original - 1 of 1 Show in CodeLens

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L4 = list(zip(L1, L2))
print(L4)

```

```
[(3, 1), (4, 2), (5, 3)]
```

Activity: 2 -- ActiveCode (ac21_5_2)

Here's what happens when you loop through the tuples.

Save & Run Original - 1 of 1 Show in CodeLens

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L3 = []
L4 = list(zip(L1, L2))

for (x1, x2) in L4:
    L3.append(x1+x2)

print(L3)

```

```
[4, 6, 8]
```

Activity: 3 -- ActiveCode (ac21_5_3)

Or, simplifying and using a list comprehension:

Save & Run | Original - 1 of 1 | Show in CodeLens

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L3 = [x1 + x2 for (x1, x2) in list(zip(L1, L2))]
print(L3)

```

```
[4, 6, 8]
```

Activity: 4 -- ActiveCode (ac21_5_4)

Or, using `map` and not unpacking the tuple (our online environment has trouble with unpacking the tuple in a lambda expression):

Save & Run | Original - 1 of 1 | Show in CodeLens

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L3 = map(lambda x: x[0] + x[1], zip(L1, L2))
print(L3)

```

```
[4, 6, 8]
```

Activity: 5 -- ActiveCode (ac21_5_5)

Consider a function called possible, which determines whether a word is still possible to play in a game of hangman, given the guesses that have been made and the current state of the blanked word.

Below we provide function that fulfills that purpose.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
def possible(word, blanked, guesses_made):
    if len(word) != len(blanked):
        return False
    for i in range(len(word)):
        bc = blanked[i]
        wc = word[i]
        if bc == '_' and wc in guesses_made:
            return False
        elif bc != '_' and bc != wc:
            return False
    return True

print(possible("wonderwall", "_on__r__ll", "otnqurl"))
print(possible("wonderwall", "_on__r__ll", "wotnqurl"))

```

```
True
False
```

Activity: 6 -- ActiveCode (ac21_5_6)

Result	Actual Value	Expected Value	Notes
Pass	True	True	Testing whether possible has been correctly defined.

Pass	False	False	Testing whether possible has been correctly defined.
Pass	False	False	Testing whether possible has been correctly defined.

You passed: 100.0% of the tests

However, we can rewrite that using `zip`, to be a little more comprehensible.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
def possible(word, blanked, guesses_made):
    if len(word) != len(blanked):
        return False
    for (bc, wc) in zip(blanked, word):
        if bc == '_' and wc in guesses_made:
            return False
        elif bc != '_' and bc != wc:
            return False
    return True

print(possible("wonderwall", "_on__r__ll", "otnqurl"))
print(possible("wonderwall", "_on__r__ll", "wotnqurl"))
```

```
True
False
```

Activity: 7 -- ActiveCode (ac21_5_7)

Result	Actual Value	Expected Value	Notes
Pass	True	True	Testing whether possible has been correctly defined.
Pass	False	False	Testing whether possible has been correctly defined.
Pass	False	False	Testing whether possible has been correctly defined.

You passed: 100.0% of the tests

Check Your Understanding

1. Below we have provided two lists of numbers, `L1` and `L2`. Using zip and list comprehension, create a new list, `L3`, that sums the two numbers if the number from `L1` is greater than 10 and the number from `L2` is less than 5. This can be accomplished in one line of code.

Save & Run | 5/15/2021, 12:15:15 AM - 2 of 2 | Show in CodeLens

```

L1 = [1, 5, 2, 16, 32, 3, 54, 8, 100]
L2 = [1, 3, 10, 2, 42, 2, 3, 4, 3]

L3 = [x1 + x2 for (x1, x2) in zip(L1, L2) if x1 > 10 and x2 < 5]
```

Activity: 8 -- ActiveCode (ac21_5_8)

Result	Actual Value	Expected Value	Notes	
Pass	[18, 57, 103]	[18, 57, 103]	Testing that L3 is assigned to correct values	
Pass	'map('	'\nL1 =... < 5]'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'filter('	'\nL1 =... < 5]'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'sum('	'\nL1 =... < 5]'	Testing your code (Don't worry about actual and expected values).	Expand Differences
Pass	'zip('	'\nL1 =... < 5]'	Testing your code (Don't worry about actual and expected values).	Expand Differences

You passed: 100.0% of the tests

You have attempted 9 of 8 activities on this page

23.4. List Comprehensions">
st Comprehensions">
23.6. Exercises">
23.6. Exercises">Next Section - 23.6. Exercises

✔ Completed. Well Done!

| Back to top