# Introduction

This lesson will help you review Python functions by providing some challenge exercises involving lists.

As a refresher, function syntax looks like this:

def some\_function(some\_input1, some\_input2):

… do something with the inputs …

return output

For example, a function that returns the sum of the first and last elements of a given list might look like this:

def first\_plus\_last(lst):

return lst[0] + lst[-1]

And this would produce output like:

>>> first\_plus\_last([1, 2, 3, 4])

5

>>> first\_plus\_last([8, 2, 5, -8])

0

>>> first\_plus\_last([-10, 2, 3, -4])

-14

Instructions

When you're ready to do this series of short function challenges, continue on to the rest of the lesson!

# Lists - Code Challenge

About 1 minute

Now you have learned about the basics of Python syntax, control flow, lists and how to create functions that run repeatable blocks of code.

Let's reinforce these concepts with a series of practice problems.

# Double Index

double\_index(lst, index)

**1.**

Create a function named double\_index that has two parameters named lst and index.

The function should double the value of the element at index of lst and return the new list with the doubled value.

If index is not a valid index, the function should return the original list.

After writing your function, un-comment the call to the function that we've provided for you to test your results.

Hint: Use len(list) and index in an if statement to check to see if index is a valid index.

Solution:

#Write your function here

def double\_index(lst, index):

if index < len(lst):

lst[index] = lst[index] \* 2

return lst

#Uncomment the line below when your function is done

print(double\_index([3, 8, -10, 12], 2))

# Remove Middle

remove\_middle(lst, start, end)

**1.**

Create a function named remove\_middle which has three parameters named lst, start, and end.

The function should return a sub-list of lst with all elements with index between start and end removed (inclusive).

For example, the following code should return [4, 23, 42] because elements at indices 1, 2, and 3 have been removed:

removeMiddle([4, 8 , 15, 16, 23, 42], 1, 3)

Hint:

Slice the list twice to get the first half and the last half of the list. Then combine them using +.

The first half of the list should be lst[:start]

Solution:

#Write your function here

def remove\_middle(lst, start, end):

return lst[:start] + lst[end+1:]

#Uncomment the line below when your function is done

print(remove\_middle([4, 8, 15, 16, 23, 42], 1, 3))

Note: ([…], 1, 3)) the 1, and 3 says to omit index 1 through 3 so 8, 15, and 16 will not show thus, leaving you with 4, 23, and 42).

# More Than N

more\_than\_n(lst, item, n)

**1.**

Create a function named more\_than\_n that has three parameters named lst, item, and n.

The function should return True if item appears in the list more than n times. The function should return False otherwise.

Hint: Use lst.count(item) in an if statement

Solution:

#Write your function here

def more\_than\_n(lst, item, n):

if lst.count(item) > n:

return True

else:

return False

#Uncomment the line below when your function is done

print(more\_than\_n([2, 4, 6, 2, 3, 2, 1, 2], 2, 3))

def more\_frequent\_item (lst, item1, item2):

if lst.count(item1) >= lst.count(item2):

return item1

else:

return item2

#Uncomment the line below when your function is done

print(more\_frequent\_item([2, 3, 3, 2, 3, 2, 3, 2, 3], 2, 3))

# Middle Item

middle\_element(lst)

**1.**

Create a function called middle\_element that has one parameter named lst.

If there are an odd number of elements in lst, the function should return the middle element. If there are an even number of elements, the function should return the average of the middle two elements.

**Hint:**

The index of the middle element can be found by using len(lst)/2. However, division results in a float, and indices must be integers. You can cast that number to be an integer using int(len(lst)/2).

For lists with an even number of indices, you will want the element at the index found above and also the element at index int(len(lst)/2) - 1

**Solution:**

#Write your function here

def middle\_element(lst):

if len(lst) % 2 == 0:

sum = lst[int(len(lst)/2)] + lst[int(len(lst)/2) - 1]

return sum / 2

else:

return lst[int(len(lst)/2)]

#Uncomment the line below when your function is done

print(middle\_element([5, 2, -10, -4, 4, 5]))

# Append Sum

append\_sum(lst)

**1.**

Write a function named append\_sum that has one parameter named lst.

The function should add the last two elements of lst together and append the result to lst. It should do this process three times and then return lst.

For example, if lst started as [1, 1, 2], the final result should be [1, 1, 2, 3, 5, 8].

**Hint:**

The last two elements of lst can be found using lst[-1] and lst[-2]

**Solution:**

#Write your function here

def append\_sum(lst):

lst.append(lst[-1] + lst[-2])

lst.append(lst[-1] + lst[-2])

lst.append(lst[-1] + lst[-2])

return lst

#Uncomment the line below when your function is done

print(append\_sum([1, 1, 2]))

# Larger List

larger\_list(lst1, lst2)

**1.**

Write a function named larger\_list that has two parameters named lst1 and lst2.

The function should return the last element of the list that contains more elements. If both lists are the same size, then return the last element of lst1.

**Hint:**

Compare the size of each list by using len(lst1) and len(lst2) in an if statement.

def larger\_list(lst1, lst2):

if len(lst1) >= len(lst2):

return lst1[-1]

else:

return lst2[-1]

#Uncomment the line below when your function is done

print(larger\_list([4, 10, 2, 5], [-10, 2, 5, 10]))

# Combine Sort

combine\_sort(lst1, lst2)

**1.**

Write a function named combine\_sort that has two parameters named lst1 and lst2.

The function should combine these two lists into one new list and sort the result. Return the new sorted list.

**Hint:** First combine lst1 and lst2 using + and then sort the list using .sort() or the sorted function.

**Solution:**

#Write your function here

def combine\_sort(lst1, lst2):

unsorted = lst1 + lst2

sortedList = sorted(unsorted)

return sortedList

#Uncomment the line below when your function is done

print(combine\_sort([4, 10, 2, 5], [-10, 2, 5, 10]))

# Append Size

append\_size(lst)

**1.**

Create a function called append\_size that has one parameter named lst.

The function should add all of the numbers between 1 and the size of lst (inclusive) to the end of lst. The function should then return this new list.

For example, if lst was [23, 42, 108], the function should return [23, 42, 108, 1, 2, 3] because the size of lst was originally 3.

**Hint:**

Use the range function to create a list of the numbers that you want to add to lst. Remember the range function can take two parameters. The first is the number you want the range to start at (we want 1). The second is the number you want the range to end at (exclusive). That number should relate to the length of lst.

Add every number in that range to lst by using append.

**Solution**:

#Write your function here

def append\_size(lst):

to\_append = list(range(1, len(lst)+1))

lst = lst + to\_append

return lst

#Uncomment the line below when your function is done

print(append\_size([23, 42, 108]))

# Every Three Nums

every\_three\_nums(start)

**1.**

Create a function called every\_three\_nums that has one parameter named start.

The function should return a list of every third number between start and 100 (inclusive). For example, every\_three\_nums(91) should return the list [91, 94, 97, 100]. If start is greater than 100, the function should return an empty list.

**Hint:**

Use the range function with three parameters to generate your list. Remember, the first parameter is the starting point (inclusive). The second parameter is the ending point (exclusive). The third parameter is the step.

Range returns a range object, so make sure to cast it as a list by using list(range(...)).

**Solution:**

#Write your function here

def every\_three\_nums(start):

return list(range(start, 101, 3))

#Uncomment the line below when your function is done

print(every\_three\_nums(91))

# Python Loops

About 50 minutes

Welcome to your next course — Loops in Python!

This lesson will teach you how to iterate through items in a list by using for loops and while loops. You will also learn how to create lists from loops! By the end of the lesson, you will know how to set up a loop to best solve your iteration problem of choice. You will get lots of practice with making different loops, so that you won't forget the syntax of this very common coding paradigm.

# Introduction

Suppose we want to print() each item from a list of dog\_breeds. We would need to use the following code snippet:

dog\_breeds = ['french\_bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

print(dog\_breeds[0])

print(dog\_breeds[1])

print(dog\_breeds[2])

print(dog\_breeds[3])

print(dog\_breeds[4])

This seems inefficient. Luckily, Python (and most other programming languages) gives us an easier way of using, or iterating through, every item in a list. We can use loops! A loop is a way of repeating a set of code many times.

In this lesson, we'll be learning about:

* **For Loops**: Loops that let us move through each item in a list, called for loops
* **While Loops**: Loops that keep going until we tell them to stop, called while loops
* **List comprehensions**: Loops that create new lists, called list comprehensions

**1.**

Paste the following code into **script.py**:

for breed in dog\_breeds:

print(breed)

This will print each breed in dog\_breeds.

# Create a For Loop

In the previous exercise, we saw that we can print each item in a list using a for loop. A for loop lets us perform an action on each item in a list. Using each element of a list is known as iterating.

This loop prints each breed in dog\_breeds:

dog\_breeds = ['french\_bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

for breed in dog\_breeds:

print(breed)

The general way of writing a for loop is:

**for <temporary variable> in <list variable>:**

**<action>**

In our dog breeds example, breed was the temporary variable, dog\_breeds was the list variable, and print(breed) was the action performed on every item in the list.

Our temporary variable can be named whatever we want and does not need to be defined beforehand. Each of the following code snippets does the exact same thing as our example:

for i in dog\_breeds:

print(i)

for dog in dog\_breeds:

print(dog)

Notice that in all of these examples the print statement is indented. Everything in the same level of indentation after the for loop declaration is included in the for loop, and run every iteration.

If we forget to indent, we'll get an IndentationError.

**1.**

Run the code. You should get an IndentationError because the print(game) line is not indented.

**2.**

Indent line 6 so that you don't get an IndentationError when you run the code.

**3.**

Write a loop that prints each sport in sport\_games.

**Solution:**

board\_games = ['Settlers of Catan', 'Carcassone', 'Power Grid', 'Agricola', 'Scrabble']

sport\_games = ['football', 'football - American', 'hockey', 'baseball', 'cricket']

for game in board\_games:

print(game)

for sport in sport\_games:

print(sport)

# Using Range in Loops

Previously, we iterated through an existing list.

Often we won't be iterating through a specific list, we'll just want to do a certain action multiple times. For example, if we wanted to print out a "WARNING!" message three times, we would want to say something like:

for i in <a list of length 3>:

print("WARNING!")

Notice that we need to iterate through a list of length 3, but we don't care what's in the list. To create these lists of length n, we can use the range function. range takes in a number n as input, and returns a list from 0 to n-1. For example:

zero\_thru\_five = range(6)

# zero\_thru\_five is now [0, 1, 2, 3, 4, 5]

zero\_thru\_one = range(2)

# zero\_thru\_one is now [0, 1]

So, an easy way to accomplish our "WARNING!" example would be:

for i in range(3):

print("WARNING!")

Instructions

**1.**

Use the range function in a for loop to print out promise 5 times.

**Solution:**

promise = "I will not chew gum in class"

for promise in range(5):

print("I will not chew gum in class")

# Infinite Loops

We've iterated through lists that have a discrete beginning and end. However, let's consider this example:

my\_favorite\_numbers = [4, 8, 15, 16, 42]

for number in my\_favorite\_numbers:

my\_favorite\_numbers.append(1)

What happens here? Every time we enter the loop, we add a 1 to the end of the list that we are iterating through. As a result, we never make it to the end of the list! It keeps growing!

A loop that never terminates is called an infinite loop. These are very dangerous for your code!

A program that hits an infinite loop often becomes completely unusable. The best course of action is to never write an infinite loop. But if you accidentally stumble into one, you can end the loop by using control + c to terminate the program.

**1.**

Suppose we have two lists of students, students\_period\_A and students\_period\_B. We want to combine all students into students\_period\_B.

Write a for loop that goes through each student in students\_period\_A and adds it to the end of students\_period\_B.

**Hint:**

To add an item to the end of a list, remember that you can use the syntax:

my\_list.append(new\_item)

**2.**

Inside the for loop, after appending student to students\_period\_B, print student.

**3.**

Let's suppose you made a typo in the body of your for loop. Oh no!

Inside the for loop, change the object of the append statement from students\_period\_B to students\_period\_A.

Uh oh! Now you're adding each student in students\_period\_A to students\_period\_A! This will be infinite!

Refresh the page (or type something into your editor) escape this infinite loop!

**Hint:**

control + c would escape this infinite loop and terminate the program if you are running the script locally in your computer/laptop's terminal.

In this checkpoint, you just need to refresh the page (or type something into your editor).

**4.**

Having the fear of infinite loops instilled in you, change the inner students\_period\_A back to students\_period\_B and revel in your safety and cleverness.

**Solution:**

students\_period\_A = ["Alex", "Briana", "Cheri", "Daniele"]

students\_period\_B = ["Dora", "Minerva", "Alexa", "Obie"]

for student in students\_period\_A:

#students\_period\_A.append(student)

students\_period\_B.append(student)

print(student)

# Break

We often want to use a for loop to search through a list for some value:

items\_on\_sale = ["blue\_shirt", "striped\_socks", "knit\_dress", "red\_headband", "dinosaur\_onesie"]

# we want to check if the item with ID "knit\_dress" is on sale:

for item in items\_on\_sale:

if item == "knit\_dress":

print("Knit Dress is on sale!")

This code goes through each item in items\_on\_sale and checks for a match. After we find that "knit\_dress" is in the list items\_on\_sale, we don't need to go through the rest of the items\_on\_sale list. Since it's only 5 elements long, iterating through the entire list is not a big deal in this case. But what if items\_on\_sale had 1000 items after "knit\_dress"? What if it had 100,000 items after "knit\_dress"?

You can stop a for loop from inside the loop by using break. When the program hits a break statement, control returns to the code outside of the for loop. For example:

items\_on\_sale = ["blue\_shirt", "striped\_socks", "knit\_dress", "red\_headband", "dinosaur\_onesie"]

print("Checking the sale list!")

for item in items\_on\_sale:

print(item)

if item == "knit\_dress":

break

print("End of search!")

This would produce the output:

Checking the sale list!

blue\_shirt

striped\_socks

knit\_dress

End of search!

We didn't need to check "red\_headband" or "dinosaur\_onesie" at all!

Instructions

**1.**

You have a list of dog breeds you can adopt, dog\_breeds\_available\_for\_adoption. Using a for loop, iterate through the dog\_breeds\_available\_for\_adoption list and print out each dog breed.

**2.**

Inside your for loop, after you print each dog breed, check if it is equal to dog\_breed\_I\_want. If so, print "They have the dog I want!"

**3.**

Add a break statement when your loop has found dog\_breed\_I\_want, so that the rest of the list does not need to be checked.

**Solution:**

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'dalmation'

for dog in dog\_breeds\_available\_for\_adoption:

print(dog)

if dog == dog\_breed\_I\_want:

print("They have the dog I want!")

break

# Continue

When we're iterating through lists, we may want to skip some values. Let's say we want to print out all of the numbers in a list, unless they're negative. We can use continue to move to the next i in the list:

big\_number\_list = [1, 2, -1, 4, -5, 5, 2, -9]

for i in big\_number\_list:

if i < 0:

continue

print(i)

This would produce the output:

1

2

4

5

2

Every time there was a negative number, the continue keyword moved the index to the next value in the list, without executing the code in the rest of the for loop.

**1.**

Your computer is the doorman at a bar in a country where the drinking age is 21.

Loop through the ages list. If an entry is less than 21, skip it and move to the next entry. Otherwise, print the age.

**Solution:**

ages = [12, 38, 34, 26, 21, 19, 67, 41, 17]

for age in ages:

if age < 21:

continue

print(age)

# While Loops

We now have seen and used a lot of examples of for loops. There is another type of loop we can also use, called a while loop. The while loop performs a set of code until some condition is reached.

While loops can be used to iterate through lists, just like for loops:

dog\_breeds = ['bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

index = 0

while index < len(dog\_breeds):

print(dog\_breeds[index])

index += 1

Every time the condition of the while loop (in this case, index < len(dog\_breeds)) is satisfied, the code inside the while loop runs.

While loops can be useful when you don't know how many iterations it will take to satisfy a condition.

**1.**

You are adding students to a Poetry class, the size of which is capped at 6. While the length of the students\_in\_poetry list is less than 6, use .pop() to take a student off the all\_students list and add it to the students\_in\_poetry list.

**Hint:**

You can add an element to a list using the .append() method:

my\_list.append(element\_to\_add)

The .pop() method will take an item off of the end of a list:

my\_list = [1, 4, 10, 15]

number = my\_list.pop()

#number is now 15

**2.**

Print the students\_in\_poetry list .

**Solution:**

all\_students = ["Alex", "Briana", "Cheri", "Daniele", "Dora", "Minerva", "Alexa", "Obie", "Arius", "Loki"]

students\_in\_poetry = []

while len(students\_in\_poetry) < 6:

student = all\_students.pop()

students\_in\_poetry.append(student)

print(students\_in\_poetry)

# Nested Loops

We have seen how we can go through the elements of a list. What if we have a list made up of multiple lists? How can we loop through all of the individual elements?

Suppose we are in charge of a science class, that is split into three project teams:

project\_teams = [["Ava", "Samantha", "James"], ["Lucille", "Zed"], ["Edgar", "Gabriel"]]

If we want to go through each student, we have to put one loop inside another:

for team in project\_teams:

for student in team:

print(student)

This results in:

Ava

Samantha

James

Lucille

Zed

Edgar

Gabriel

**1.**

We have provided the list sales\_data that shows the numbers of different flavors of ice cream sold at three different locations of the fictional shop, Gilbert and Ilbert's Scoop Shop. We want to sum up the total number of scoops sold. Start by defining a variable scoops\_sold and set it to zero.

**2.**

Go through the sales\_data list. Call each inner list location, and print out each location list.

**3.**

Within the sales\_data loop, go through each location list and add the element to your scoops\_sold variable.

By the end, you should have the sum of every number in the sales\_data nested list.

**4.**

Print out the value of scoops\_sold.

**Solution:**

sales\_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]

scoops\_sold = 0

for location in sales\_data:

print(location)

for element in location:

scoops\_sold += element

print(scoops\_sold)

# List Comprehensions

Let's say we have scraped a certain website and gotten these words:

words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]

We want to make a new list, called usernames, that has all of the strings in words with an '@' as the first character. We know we can do this with a for loop:

words = ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]

usernames = []

for word in words:

if word[0] == '@':

usernames.append(word)

First, we created a new empty list, usernames, and as we looped through the words list, we added every word that matched our criterion. Now, the usernames list looks like this:

>>> print(usernames)

["@coolguy35", "@kewldawg54", "@matchamom"]

Python has a convenient shorthand to create lists like this with one line:

usernames = [word for word in words if word[0] == '@']

**This is called a list comprehension**. It will produce the same output as the for loop did:

["@coolguy35", "@kewldawg54", "@matchamom"]

This list comprehension:

1. Takes an element in words
2. Assigns that element to a variable called word
3. Checks if word[0] == '@', and if so, it adds word to the new list, usernames. If not, nothing happens.
4. Repeats steps 1-3 for all of the strings in words

**Note:** if we hadn’t done any checking (let's say we had omitted if word[0] == '@'), the new list would be just a copy of words:

usernames = [word for word in words]

#usernames is now ["@coolguy35", "#nofilter", "@kewldawg54", "reply", "timestamp", "@matchamom", "follow", "#updog"]

**1.**

We have defined a list heights of visitors to a theme park. In order to ride the Topsy Turvy Tumbletron roller coaster, you need to be above 161 centimeters. Using a list comprehension, create a new list called can\_ride\_coaster that has every element from heights that is greater than 161.

**Hint:**

Remember that you can create a list that satisfies a condition (for example, each element is not equal to 0) by using the syntax:

new\_list = [elem for elem in old\_list if elem != 0]

**2.**

Print can\_ride\_coaster.

**Solution:**

heights = [161, 164, 156, 144, 158, 170, 163, 163, 157]

can\_ride\_coaster = [height for height in heights if height > 161]

print(can\_ride\_coaster)

# More List Comprehensions

Let's say we're working with the usernames list from the last exercise:

>>> print(usernames)

["@coolguy35", "@kewldawg54", "@matchamom"]

We want to create a new list with the string " please follow me!" added to the end of each username. We want to call this new list messages. We can use a list comprehension to make this list with one line:

messages = [user + " please follow me!" for user in usernames]

This list comprehension:

1. Takes a string in usernames
2. Assigns that number to a variable called user
3. Adds " please follow me!" to user
4. Appends that concatenation to the new list called messages
5. Repeats steps 1-4 for all of the strings in usernames

Now, messages contains these values:

["@coolguy35 please follow me!", "@kewldawg54 please follow me!", "@matchamom please follow me!"]

Being able to create lists with modified values is especially useful when working with numbers. Let's say we have this list:

my\_upvotes = [192, 34, 22, 175, 75, 101, 97]

We want to add 100 to each value. We can accomplish this goal in one line:

updated\_upvotes = [vote\_value + 100 for vote\_value in my\_upvotes]

This list comprehension:

1. Takes a number in my\_upvotes
2. Assigns that number to a variable called vote\_value
3. Adds 100 to vote\_value
4. Appends that sum to the new list updated\_upvotes
5. Repeats steps 1-4 for all of the numbers in my\_upvotes

Now, updated\_upvotes contains these values:

[292, 134, 122, 275, 175, 201, 197]

**1.**

We have provided a list of temperatures in celsius. Using a list comprehension, create a new list called fahrenheit that converts each element in the celsius list to fahrenheit.

**Note:** To convert, use the formula:

temperature\_in\_fahrenheit = temperature\_in\_celsius \* 9/5 + 32

**2.**

Print fahrenheit.

**Solution:**

celsius = [0, 10, 15, 32, -5, 27, 3]

fahrenheit = [temp\*(9/5) + 32 for temp in celsius]

print(fahrenheit)

# Review

Good job! In this lesson, you learned

* how to write a for loop
* how to use range in a loop
* what infinite loops are and how to avoid them
* how to skip values in a loop
* how to write a while loop
* how to make lists with one line

Let's get some more practice with these concepts!

**1.**

Create a list called single\_digits that consists of the numbers 0-9 (inclusive).

**Hint:**

You can use range(n) to make a list from 0 through n-1. The command:

print(range(5))

yields:

[0, 1, 2, 3, 4]

**2.**

Create a for loop that goes through single\_digits and prints out each one.

You can write a for loop through a list using this syntax:

for element in list\_to\_iterate\_through:

print(element)

**3.**

Create a list called squares. Assign it to be an empty list to begin with.

**4.**

Inside the loop that iterates through single\_digits, append the squared value of each element of single\_digits to the list squares. You can do this before or after printing the element.

You can square a number by either using:

number\_to\_square\*\*2

(which takes it to the second power), or using:

number\_to\_square\*number\_to\_square

which multiplies it by itself.

To append an element to a list, you can use the .append() method:

my\_list.append(number\_to\_add)

**5.**

After the for loop, print out squares.

**6.**

Create the list cubes using a list comprehension on the single\_digits list. Each element of cubes should be an element of single\_digits taken to the third power.

You can cube a number by either using:

number\_to\_cube\*\*3

(which takes it to the third power), or using:

number\_to\_cube\*number\_to\_cube\*number\_to\_cube

which multiplies it by itself twice.

Remember that a list comprehension looks like:

new\_list = [element <OPERATED ON IN SOME WAY> for element in old\_list]

**7.**

Print cubes.

Good job!

Solution:

single\_digits = range(10)

squares = []

for item in single\_digits:

print(item)

squares.append(item\*\*2)

cubes = [item\*\*3 for item in single\_digits]

print(cubes)

# Week 5 Project

## Carly's Clippers

About 40 minutes

In this project, you will be working for a hair salon called Carly's Clippers. You will be using what you have learned about loops to help process some of Carly's data. To complete the project, you will need to understand how to iterate through lists using for, and how to use list comprehensions to create lists. As an added bonus, you'll refresh your knowledge of basic Python syntax and list operations. If you get stuck or confused, remember that your Slack community is there to help!

This project is not graded, and you do not need to submit it anywhere. If you would like to check your results, the [solution code can be found here.](https://s3.amazonaws.com/codecademy-content/programs/programming-with-python/On-platform+solutions/carlys_clippers.py)

# Carly's Clippers

You are the data analyst at Carly's Clippers, the newest hair salon on the block. Your job is to go through the lists of data that have been collected in the past couple of weeks. You will be calculating some important metrics that Carly can use to plan out the operation of the business for the rest of the month.

You have been provided with three lists:

* hairstyles: the names of the cuts offered at Carly's Clippers
* prices: the price of each hairstyle in the hairstyles list
* last\_week: the number of each hairstyle in hairstyles that was purchased last week

### Prices and Cuts

1.

Carly wants to be able to market her low prices. We want to find out what the average price of a cut is.

First, let's sum up all the prices of haircuts. Create a variable total\_price, and set it to 0.

2.

Iterate through the prices list and add each price to the variable total\_price.

3.

After your loop, create a variable called average\_price that is the total\_price divided by the number of haircuts.

You can get the number of haircuts by using the len() function.

4.

Print the value of average\_price, in a string like:

"Average Price: <average\_price>"

Hint:

To print a string with a variable, you can use syntax like:

>>> my\_age = 101

>>> print("My age: " + str(my\_age))

"My age: 101"

5.

That average price is more expensive than Carly thought it would be! She wants to cut all prices by 5 dollars.

Use a list comprehension to make a list called new\_prices, which has each element in prices minus 5.

6.

Print new\_prices.

### Revenue

7.

Carly really wants to make sure that Carly's Clippers is a profitable endeavor. She first wants to know how much revenue was brought in last week.

Create a variable called total\_revenue and set it to 0.

8.

Use a for loop to create a variable i that goes from 0 to len(hairstyles)-1 (you can use range() to do this!)

9.

Add the product of prices[i] (the price of the haircut at position i) and last\_week[i] (the number of people who got the haircut at position i) to total\_revenue at each step.

10.

After your loop, print the value of total\_revenue, in a string like:

"Total Revenue: <total\_revenue>"

11.

Find the average daily revenue by dividing total\_revenue by 7. Call this number average\_daily\_revenue and print it out.

12.

Carly thinks she can bring in more customers by advertising all of the haircuts she has that are under 30 dollars.

Use a list comprehension to create a list called cuts\_under\_30 that has the entry hairstyles[i] for each i for which new\_prices[i] is less than 30.

You can use range() in your list comprehension to make i go from 0 to len(new\_prices) - 1.

Hint:

Syntax you can use for your list comprehension might look like:

new\_list = [old\_list[i] for i in range(old\_list) if different\_list[i] < 0]

This makes a new list of every entry in old\_list for which the index i satisfies the condition different\_list[i] < 0.

13.

Print cuts\_under\_30.

14.

Click here for a video walkthrough from our experts to help you check your work!

Solution

hairstyles = ["bouffant", "pixie", "dreadlocks", "crew", "bowl", "bob", "mohawk", "flattop"]

prices = [30, 25, 40, 20, 20, 35, 50, 35]

last\_week = [2, 3, 5, 8, 4, 4, 6, 2, 1]

total\_price = 0

for price in prices:

total\_price += price

average\_price = total\_price/len(prices)

print("Average Haircut Price: " + str(average\_price))

new\_prices = [price - 5 for price in prices]

print(new\_prices)

total\_revenue = 0

for i in range(len(hairstyles)):

total\_revenue += prices[i]\*last\_week[i]

print("Total Revenue: " + str(total\_revenue))

avg\_daily\_revenue = total\_revenue/7

print("Average Daily Revenue: " + str(avg\_daily\_revenue))

cuts\_under\_30 = [hairstyles[i] for i in range(len(new\_prices)) if new\_prices[i] < 30]

print(cuts\_under\_30)

Lesson

# Loops - Code Challenge

Now you have learned about the basics of Python syntax, control flow, lists, loops and how to create functions that run repeatable blocks of code.

Let's reinforce these concepts with a series of practice problems.

Afterwards, if you want a bigger challenge, you can also do [this set of problems, specifically on List Comprehensions](https://www.codecademy.com/courses/learn-python-list-comprehension/lessons/lists/exercises/introduction)!

# Introduction

This lesson will help you review Python functions by providing some challenge exercises involving loops.

As a refresher, function syntax looks like this:

def some\_function(some\_input1, some\_input2):

… do something with the inputs …

return output

For example, a function that prints all odd numbers in a list would look like this:

def odd\_nums(lst):

for item in lst:

if item % 2 == 1:

print(item)

And this would produce output like:

>>> odd\_nums([4, 7, 9, 10, 13])

7

9

13

When you're ready to do this series of short function challenges, continue on to the rest of the lesson!

# Divisible by Ten

divisible\_by\_ten(nums)

Instructions

**1.**

Create a function named divisible\_by\_ten that takes a list of numbers named nums as a parameter. Return the amount of numbers in that list that are divisible by 10.

Solution:

#Write your function here

def divisible\_by\_ten(nums):

count = 0

for number in nums:

if (number % 10 == 0):

count += 1

return count

#Uncomment the line below when your function is done

print(divisible\_by\_ten([20, 25, 30, 35, 40]))

# Greetings

add\_greetings(names)

**1.**

Create a function named add\_greetings which takes a list of strings named names as a parameter.

In the function, create an empty list that will contain each greeting. Add the string "Hello, " in front of each name in names and append the greeting to the list.

Return the new list containing the greetings.

Use + to concatenate "Hello, " with every name in names. Don't forget to add the comma and the space to the greeting!

**Solution:**

#Write your function here

def add\_greetings(names):

new\_list = []

for name in names:

new\_list.append("Hello, " + name)

return new\_list

#Uncomment the line below when your function is done

print(add\_greetings(["Owen", "Max", "Sophie"]))

# Delete Starting Even Numbers

delete\_starting\_evens(lst)

Instructions

**1.**

Write a function called delete\_starting\_evens that has a parameter named lst. The function should remove element from the front of lst until the front of the list is not even. The function should then return lst.

For example if lst started as [4, 8, 10, 11, 12, 15], then delete\_starting\_evens(lst) should return [11, 12, 15].

Make sure your function works even if every element in the list is even!

Hint

Use a while loop to check two things. First, check if the list has at least one element, using len(lst). Second, check to see if the first element is odd using mod (%). If both of those are True, slice off the first element of the list using lst = lst[1:].

**Solution**:

#Write your function here

def delete\_starting\_evens(lst):

while (len(lst) > 0 and lst[0] % 2 == 0):

lst = lst[1:]

return lst

#Uncomment the lines below when your function is done

print(delete\_starting\_evens([4, 8, 10, 11, 12, 15]))

print(delete\_starting\_evens([4, 8, 10]))

# Odd Indices

odd\_indices(lst)

Instructions

**1.**

Create a function named odd\_indices that has one parameter named lst. The function should create a new empty list and add every element from lst that has an odd index. The function should then return this new list.

For example, odd\_indices([4, 3, 7, 10, 11, -2]) should return the list [3, 10, -2].

Hint:

There are a few ways to do this. range(1, len(lst), 2) will create a list of the indices you're interested in. So you could loop through that list like this:

for index in range(1, len(lst), 2):

new\_list.append(lst[index])

**Solution:**

#Write your function here

def odd\_indices(lst):

new\_lst = []

for i in range(1, len(lst), 2):

new\_lst.append(lst[i])

return new\_lst

#Uncomment the line below when your function is done

print(odd\_indices([4, 3, 7, 10, 11, -2]))

# Exponents

exponents(bases, powers)

Instructions

**1.**

Create a function named exponents that takes two lists as parameters named bases and powers. Return a new list containing every number in bases raised to every number in powers.

For example, consider the following code.

exponents([2, 3, 4], [1, 2, 3])

the result would be the list [2, 4, 8, 3, 9, 27, 4, 16, 64]. It would first add two to the first. Then two to the second. Then two to the third, and so on.

Hint:

Use nested for loops. The outer for loop should loop through all of the bases and the inner for loop should loop through all of the powers.

Remember a \*\* b is a to the power of b

**Solution:**

#Write your function here

def exponents(bases, powers):

new\_lst = []

for base in bases:

for power in powers:

new\_lst.append(base \*\* power)

return new\_lst

#Uncomment the line below when your function is done

print(exponents([2, 3, 4], [1, 2, 3]))

# Larger Sum

larger\_sum(lst1, lst2)

**Instructions**

**1.**

Create a function named larger\_sum that takes two lists of numbers as parameters named lst1 and lst2. The function return the list whose elements sum to the greater number. If the sum of the elements of each list are equal, return lst1.

Hint:

Create variables named sum1 and sum2 and set them to be 0. Loop through each list separately and add to the appropriate variable. After looping through each list, compare the two sums in an if statement and return the correct list.

**Solution:**

#Write your function here

def larger\_sum(lst1, lst2):

sum1 = 0

sum2 = 0

for number in lst1:

sum1 += number

for number in lst2:

sum2 += number

if sum1 >= sum2:

return lst1

else:

return lst2

#Uncomment the line below when your function is done

print(larger\_sum([1, 9, 5], [2, 3, 7]))

# Over 9000

over\_nine\_thousand(lst)

Instructions

**1.**

Create a function named over\_nine\_thousand that takes a list of numbers named lst as a parameter. The function should sum the elements of the list until the sum is greater than 9000. When this happens, the function should return the sum. If the sum of all of the elements is never greater than 9000, the function should return total sum of all the elements. If the list is empty, the function should return 0.

For example, if lst was [8000, 900, 120, 5000], then the function should return 9020.

Hint:

Create a variable named sum that begins at 0. Loop through all of the elements of lst and use a break when the sum is greater than 9000. Return sum after the loop.

**Solution:**

#Write your function here

def over\_nine\_thousand(lst):

sum = 0

for number in lst:

sum += number

if (sum > 9000):

break

return sum

#Uncomment the line below when your function is done

print(over\_nine\_thousand([8000, 900, 120, 5000]))

# Max Num

max\_num(nums)

Instructions

**1.**

Create a function named max\_num that takes a list of numbers named nums as a parameter. The function should return the largest number in nums

**Hint:**

Create a variable called maximum to track the max number, and have it start as the first element in the list. Loop through all of the numbers in the list, and if a number is ever greater than the current max number, the max number should be re-set to that number.

**Solution:**

#Write your function here

def max\_num(nums):

maximum = nums[0]

for number in nums:

if number > maximum:

maximum = number

return maximum

#Uncomment the line below when your function is done

print(max\_num([50, -10, 0, 75, 20]))

# Same Values

same\_values(lst1, lst2)

Instructions

**1.**

Write a function named same\_values that takes two lists of numbers of equal size as parameters. The function should return a list of the indices where the values were equal in lst1 and lst2.

For example, the following code should return [0, 2, 3]

same\_values([5, 1, -10, 3, 3], [5, 10, -10, 3, 5])

Hint:

Loop through all of the indices of each list using for index in range(len(lst1)) and compare lst1[index] to lst2[index]. Append index to a new list if those two items are equal.

**Solution:**

#Write your function here

def same\_values(lst1, lst2):

new\_lst = []

for index in range(len(lst1)):

if lst1[index] == lst2[index]:

new\_lst.append(index)

return new\_lst

#Uncomment the line below when your function is done

print(same\_values([5, 1, -10, 3, 3], [5, 10, -10, 3, 5]))

# Reversed List

reversed\_list(lst1, lst2)

Instructions

**1.**

Create a function named reversed\_list that takes two lists of the same size as parameters named lst1 and lst2. The function should return True if lst1 is the same as lst2 reversed. The function should return False otherwise.

For example, reversed\_list([1, 2, 3], [3, 2, 1]) should return True.

Let's say the list are of size 5. You want to compare lst1[0] with lst2[4], lst1[1] with lst2[3] and so on.

Loop through the numbers created by range(len(lst1)) using a variable named index

Hint:

Compare lst1[index] to lst2[len(lst2) - 1 - index]. If those two items are not equal, return False. If you loop through the entrie list and you never return False, that means that every item was equal, and you should return True.

**Solution:**

#Write your function here

def reversed\_list(lst1, lst2):

for index in range(len(lst1)):

if lst1[index] != lst2[len(lst2) - 1 - index]:

return False

return True

#Uncomment the lines below when your function is done

print(reversed\_list([1, 2, 3], [3, 2, 1]))

print(reversed\_list([1, 5, 3], [3, 2, 1]))

## Reggie's Linear Regression

About 2 minutes

Project: Reggie's Linear Regression

In this project, you’ll combine your knowledge of lists, loops, and syntax to help a mad scientist perform some calculations on his data.

**This project is totally optional; if you’re busy this week, you can skip it!**

If you decide to do this project, you complete this project on your own computer. To do this, you’ll need to install Python. Follow the directions below, if you’re interested.

#### Working on Your Computer

1. If you’ve never used the command line, we recommend taking the [Learn the Command Line course](https://www.codecademy.com/learn/learn-the-command-line).
2. Install Python by following the directions in this article on [Installing Python](https://www.codecademy.com/articles/install-python3).
3. Learn about [Jupyter Notebooks](https://www.codecademy.com/articles/how-to-use-jupyter-notebooks), a cool way of combining Python code with explanations or instruction in a web terminal.
4. Download the [Linear Regression project](https://s3.amazonaws.com/codecademy-content/programs/programming-with-python/Reggie's+Linear+Regression.zip). Unzip it by double-clicking on it. In the terminal, navigate to the directory containing the project, and type:

jupyter notebook

This should open a browser tab.

1. Click on **Reggie\_Linear\_Regression\_Skeleton.ipynb** in the browser tab. This will open up your Jupyter Notebook.
2. Follow the steps in the Jupyter Notebook. If you get stuck, you can look at **Reggie\_Linear\_Regression\_Solution.ipynb** for the answer.

See Jupyter notebook for solution and examples.

This concludes Week 5.