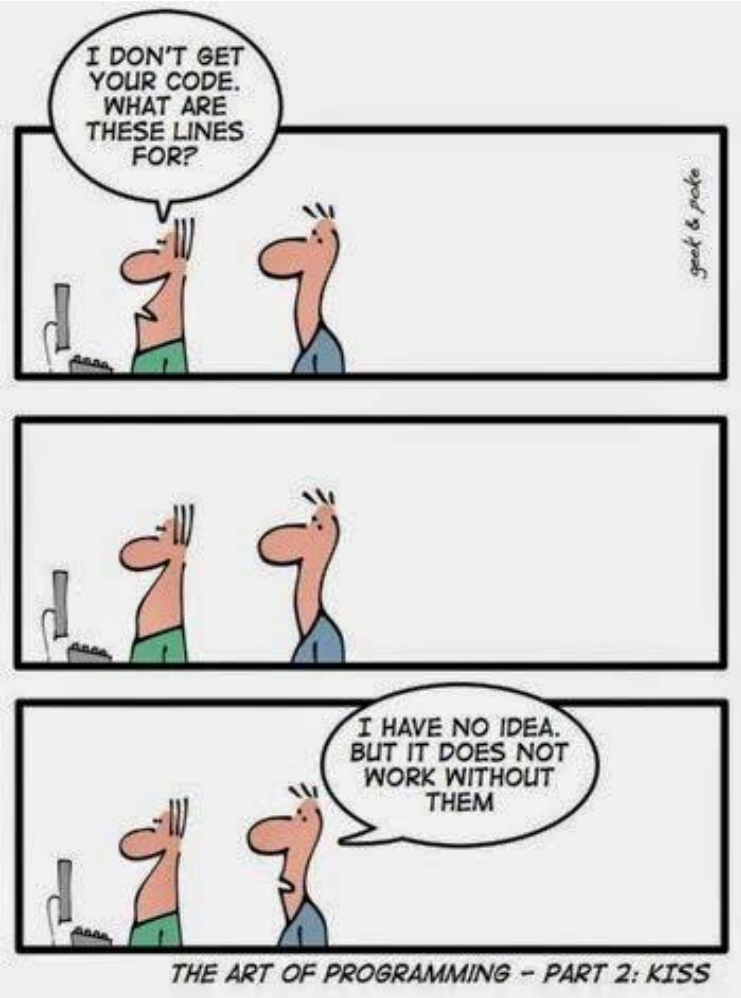
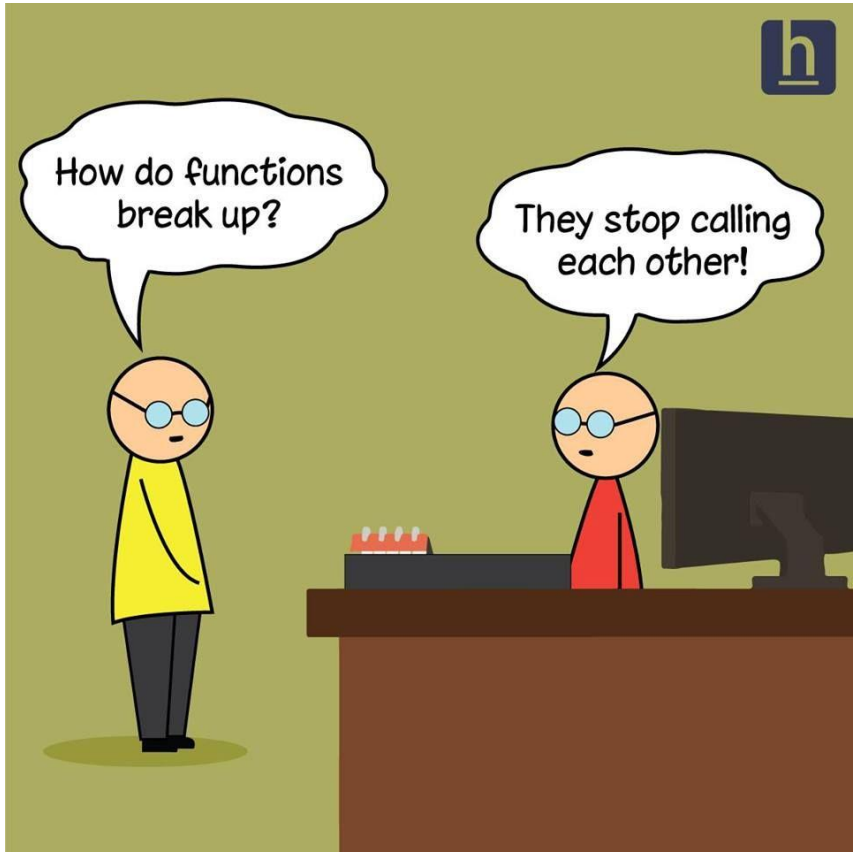
PYTHON BOOT CAMP

Module 7:
Lists

CS Jokes


I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM
THE ART OF PROGRAMMING - PART 2: KISS


How do functions break up?
They stop calling each other!

Program 1

- Write a program that asks the user how many numbers they would like to enter (between 5 and 20). Your program should then read those numbers (one per line) and should then find the sum of those numbers.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 1

- Step 1: Problem-solving Phase
  - First, let's check out a sample run:

```
>>> %Run sum_numbers.py
 How many numbers do you want to enter: 7

 Great, please enter those 7 numbers (one per line) below.

 Enter a number: 1
 Enter a number: 3
 Enter a number: 5
 Enter a number: 7
 Enter a number: 11
 Enter a number: 13
 Enter a number: 17

 The sum of the 7 entered numbers is 57

 The average of the 7 entered numbers is 8.14.
```

Program 1

- Step 1: Problem-solving Phase
  - There's not really much to discuss here
  - Ask how many numbers they plan to enter
    - Let's say then enter 8, for example
  - Make a for loop to loop that many times
  - Each time, print ("Enter the next number: ") and scan the next value
  - Keep a running sum
  - Print out the final sum
  - Yes, it's intended to be very easy
  - Go ahead and code this

Program 1

- Step 2: Implementation Phase

```
sum = 0
n = int(input("How many numbers do you want to enter: "))

print("\nGreat, please enter those", n, "numbers (one per line) below.\n")

for i in range(n):
    sum += int(input("Enter a number: "))

print("\nThe sum of the", n, "entered numbers is", sum)
print("\nThe average of the {} entered numbers is {:.2f}.".format(n, sum/n))
```

Program 1

- Truth: that was intended to be easy
  - It was just a warmup
  - And it allowed you to make a starter code, which you can use to solve what should be an easy problem…
  - The following is, logically, only a slight modification…
- Find out how many of the user-inputted numbers are above the average

Program 2

- Write a program that asks the user how many numbers they would like to enter (between 5 and 20). Your program should then read those numbers (one per line), should find both the sum and the average of those numbers, and should print out how many of the numbers are above the average.
- Remember:
  - Step 1: Problem-solving Phase
  - Step 2: Implementation Phase

Program 2

- Step 1: Problem-solving Phase
  - So the only thing different here is the need to count how many numbers are above the average
  - Give it a shot
    - Talk with a neighbor and see what you come up with
  - Ideas:
    - ?

Program 2

- Step 1: Problem-solving Phase
  - Now that we have the average, how can we find and print the user-values that were above the average?
    - Hmmm. We did not save each of those 10 values.
      - We just read them and added the values to sum.
    - That's a problem!
      - Cause after we find the average, we need to again loop over the numbers to see which ones are above the average
    - Okay. Well, one possible solution is to make 10 variables!
    - Is that an acceptable solution?
    - What if the number of user-inputted values was 10,000?
    - Would you make 10,000 variables?
  - This is a problem. But it is EASY to solve with **Lists**!

Program 2

- Step 2: Implementation Phase

```
numbers = []  # create an empty list
sum = 0
n = int(input("How many numbers do you want to enter: "))

print("\nGreat, please enter those", n, "numbers (one per line) below.\n")

for i in range(n):
    value = int(input("Enter a number: "))
    numbers.append(value)
    sum += value

average = sum / n

count = 0

for i in range(n):
    if numbers[i] > average:
        count += 1

print("\nThe average of the {} entered numbers is {:.2f}.".format(n, sum/n))
print("Number of elements above the average: {}".format(count))
```

Program 2

- Step 2: Implementation Phase
  - Note:
    - We don't need to understand all the details right now
    - This problem is just to introduce the idea of how a list in Python works
    - Summary: instead of using 10 different variables, we can use one single variable, a list of size 10
    - This one list allows us to save the 10 user values
  - Let us now study lists in detail…

List Basics

- The `list` class
  - Python provides a class called `list` that stores a sequential collection of elements
  - This class contains the methods for creating, manipulating, and processing lists
  - Elements in a `list` can be accessed through an index

List Basics

- Creating Lists
  - Here's several ways to create lists:
    - `list1 = list()`
      - Create an empty list
    - `list2 = list([2, 3, 4])`
      - Create a list with elements 2, 3, 4
    - `list3 = list(["red", "green", "blue"])`
      - Create a list with strings
    - `list4 = list(range(3, 6))`
      - Create a list with elements 3, 4, 5
    - `list5 = list("abcd")`
      - Create a list with characters a, b, c, d

List Basics

- Creating Lists
  - You can also use a simpler syntax:
    - `list1 = []`
      - Same as list()
    - `list2 = [2, 3, 4]`
      - Same as list([2, 3, 4])
    - `list3 = ["red", "green"]`
      - Same as list(["red", "green"])
  - You can see that the elements in a list are separated by commas and enclosed by a pair of brackets (`[]`)
  - A list can also contain elements of mixed type:
    - `list4 = [2, "three", 4]`

List Basics

- Lists is a Sequence Type
  - Like strings, lists are sequence types in Python
    - A string is a sequence of characters
    - A list is a sequence of any element
  - Python has built-in operators that work on sequences
  - Examples:
    - Given a sequence, s,
      - `s[i]` will give the **i**th element in the list
      - "`x in s`" will return True if the element **x** is in the sequence **s**
      - `min(s)` will give the smallest element in the sequence **s**
      - `max(s)` will give the largest element in the sequence **s**
  - The following table summarizes sequence operators…

List Basics

- Common Operators for a Sequence `s`

Operation	*Description*
`x in s`	True if element x is in sequence s.
`x not in s`	True if element x is not in sequence s.
`s1 + s2`	Concatenates two sequences s1 and s2.
`s * n, n * s`	n copies of sequence s concatenated.
`s[i]`	ith element in sequence s.
`s[i : j]`	Slice of sequence s from index i to j − 1.
`len(s)`	Length of sequence s, i.e., the number of elements in s.
`min(s)`	Smallest element in sequence s.
`max(s)`	Largest element in sequence s.
`sum(s)`	Sum of all numbers in sequence s.
`for loop`	Traverses elements from left to right in a `for` loop.
`<, <=, >, >=, =, !=`	Compares two sequences.

List Basics

- Examples:

```
from random import *

list1 = [2, 3, 4, 1, 32]

# We can print the whole list quite easily:
print("List values:", list1)

# Or we can print using a for loop
for i in list1:
    print("\t{}".format(i))
print()
print("Length of list:", len(list1))
print("Min value in list:", min(list1))
print("Max value in list:", max(list1))
print("Sum of list values:", sum(list1))

list1.sort()
print("List after sorting:", list1)

shuffle(list1) # from random class
print("List after shuffling:", list1)
```

```
>>> %Run list_operators.py
 List values: [2, 3, 4, 1, 32]

 Print using for loop:
        2
        3
        4
        1
        32

 Length of list: 5
 Min value in list: 1
 Max value in list: 32
 Sum of list values: 42
 List after sorting: [1, 2, 3, 4, 32]
 List after shuffling: [2, 1, 4, 3, 32]
```

List Basics

- Index Operator `[]`
  - So how do you access individual elements in a list?
  - Easy: you use the index operator, `[]`
  - Syntax:

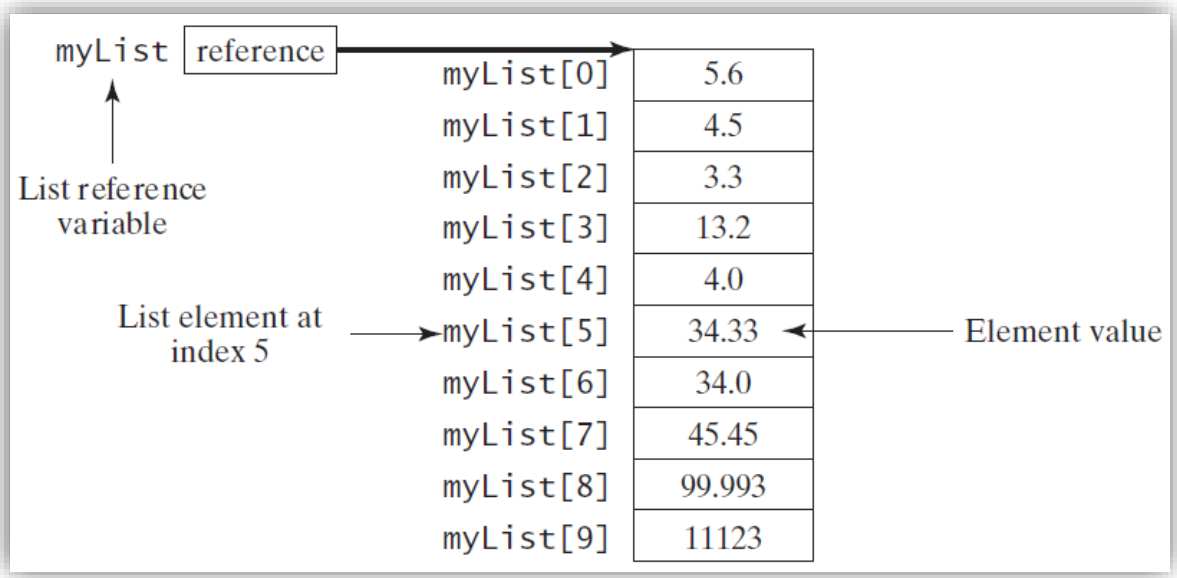
    ```
    my_list[index]
    ```

    - Think of index as being the numerical spot where the given elements is located within the list
  - Note:
    - List indexes are `0` based
    - Translation:
      the range of list indexes is from `0` to `len(my_list)-1`

List Basics

- Index Operator `[]`
  - Example:

```
myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33,
          34.0, 45.45, 99.993, 11123]
```



- So this list has ten elements
- And the elements range from index 0 to index 9

List Basics

- Index Operator `[]`
  - `myList[index]` can be used just like a variable
    - As such, it is also known as an *indexed variable*
  - Consider the following code:

```
myList[2] = myList[0] + myList[1]
```

    - The values in myList[0] and myList[1] are added to myList[2]
  - Another example:

```
for i in range(len(myList)):
    myList[i] = i
```

    - The above loop assigns 0 to `myList[0]`, 1 to `myList[1]`, …, and 9 to `myList[9]`

List Basics

- Index Operator [ ]
  - Caution:
    - A common mistake with new programmers is to access lists outside of their given range (out of bounds)
    - This results in a runtime error
      - an error that occurs when you run your program
    - How do you avoid this mistake?
      - It's not easy
      - But in short, never use an index beyond `len(myList) - 1`
      - So if your list is length 80, never use an index beyond 79

List Basics

- Index Operator [ ]
  - Caution:
    - Another mistake is to called the *off-by-one* error
    - This happens when programmers reference the first element in the list with a **1**
      - Remember: the first element should be referenced with a **0**
    - Example:

```
i = 0
while i <= len(myList):
    print(myList[i])
    i += 1
```

    - Here, the **<=** should be replaced by **<**.

List Basics

- Index Operator [ ]
  - Negative indexes:
    - Python also allows you to use a negative index
      - This is something many (or most) other languages do not allow!
    - So how are negative indexes used?
      - The negative number is a reference position relative to the end of the list
      - actual position is obtained by adding the length of the list with the negative index
  - Example:

```
list1 = [2, 3, 5, 2, 33, 21]
list1[-1]    # refers to 21
list1[-3]    # refers to 2
```

List Basics

- List Slicing
  - Now this is something cool Python does
    - And something that many other languages do not do
  - Python has a slicing operator
  - This operator returns a slice of the list
  - Syntax:

```
list[start : end]
```

    - The slice is a **sublist** from index **`start`** to index **`end - 1`**
  - Example:

```
list1 = [2, 3, 5, 7, 9, 1]
list1[2 : 4]    # resulting slice is [5, 7]
```

List Basics

- List Slicing
  - You can omit the starting or ending index with slicing
  - Examples:

```
list1 = [2, 3, 5, 2, 33, 21]
list1[ : 2]  # resulting slice is [2, 3]
list1[3 : ]  # resulting slice is [2, 33, 21]
```

    - Note:
      - `list1[ : 2]` is the same as `list1[0 : 2]`
      - `list1[3 : ]` is the same as `list1[3 : len(list1)]`
  - g

List Basics

- List Slicing
  - You can also use negative indices with slicing
  - Examples:

```
list1 = [2, 3, 5, 2, 33, 21]
list1[1 : -3]  # resulting slice is [3, 5]
list1[-4 : -2]  # resulting slice is [3, 5]
```

    - Note:
      - list1[1 : -3] is the same as list1[1 : -3 + len(list1)]
      - list1[-4 : -2] is the same as list1[-4 + len(list1) : -2 + len(list1)]
  - Important:
    - If **`start >= end`**, list[start : end] returns an empty list
    - If **`end`** specifies a position beyond the end of the list, Python will use the length of the list for **`end`** instead

List Basics

- The `+`, `*`, `in`, and `not in` Operators
  - The concatenation operator (`+`) can be used to join two lists
  - The repetition operator (`*`) can be used to replicate elements in a list
  - Examples:

```
 1  >>> list1 = [2, 3]
 2  >>> list2 = [1, 9]
 3  >>> list3 = list1 + list2
 4  >>> list3
 5  [2, 3, 1, 9]
 6  >>>
 7  >>> list4 = 3 * list1
 8  >>> list4
 9  [2, 3, 2, 3, 2, 3]
10  >>>
```

List Basics

- The `+`, `*`, `in`, and `not in` Operators
  - The `in` and `not in` operators are straightforward
    - We use them to test if a given value is `in` a list or if a given value is `not in` a list
  - Examples:

```
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2 in list1
True
>>> 2 not in list1
False
>>>
```

List Basics

- Traversing Elements in a List
  - Python has an easy for loop to access each element in a list
  - Syntax:

```
for u in my_list:
    print(u)
```

    - Note that we do not need range
  - What if you wanted to traverse the list in a different order? Or if you wanted to change the list?
    - You use a standard for loop with an index variable:

```
for i in range(0, len(myList), 2):
    print(myList[i])
```

List Basics

- Comparing Lists
  - Lists can be compared using the comparison operators:
    - `>`, `>=`, `<`, `<=`, `==`, and `!=`
  - But for this to work, the two lists must contain the same type of elements (all int values, all strings, etc.)
  - The comparison uses lexicographical ordering:
    - The first two elements are compared
    - If they differ, this determines the outcome of the comparison
    - If they are the same, the next two elements are compared
    - And so on

List Basics

- Comparing Lists

```
 1  >>> list1 = ["green", "red", "blue"]
 2  >>> list2 = ["red", "blue", "green"]
 3  >>> list2 == list1
 4  False
 5  >>> list2 != list1
 6  True
 7  >>> list2 >= list1
 8  False
 9  >>> list2 > list1
10  False
11  >>> list2 < list1
12  True
13  >>> list2 <= list1
14  True
15  >>>
```

List Basics

- List Comprehensions
  - Sounds weird, right.?.
  - This is a cool way of making lists in Python
  - The list is made with an expression followed by a for clause, and even perhaps followed by if statements
  - The result of this “List Comprehension” is a new list produced after evaluating the expression

List Basics

- List Comprehensions

```
>>> list1 = [x for x in range(5)] # Returns a list of 0, 1, 2, 3, 4
>>> list1
[0, 1, 2, 3, 4]
>>>
>>> list2 = [0.5 * x for x in list1]
>>> list2
[0.0, 0.5, 1.0, 1.5, 2.0]
>>>
>>> list3 = [x for x in list2 if x < 1.5]
>>> list3
[0.0, 0.5, 1.0]
>>>
```

List Basics

- List Methods
  - Once you have created a list, you can use the built-in methods to manipulate the list

list	
`append(x: object): None`	Adds an element x to the end of the list.
`count(x: object): int`	Returns the number of times element x appears in the list.
`extend(l: list): None`	Appends all the elements in l to the list.
`index(x: object): int`	Returns the index of the first occurrence of element x in the list.
`insert(index: int, x: object): None`	Inserts an element x at a given index. Note that the first element in the list has index 0.
`pop(i): object`	Removes the element at the given position and returns it. The parameter i is optional. If it is not specified, list.pop() removes and returns the last element in the list.
`remove(x: object): None`	Removes the first occurrence of element x from the list.
`reverse(): None`	Reverses the elements in the list.
`sort(): None`	Sorts the elements in the list in ascending order.

List Basics

- List Methods
  - Examples:

```
 1  >>> list1 = [2, 3, 4, 1, 32, 4]
 2  >>> list1.append(19)
 3  >>> list1
 4  [2, 3, 4, 1, 32, 4, 19]
 5  >>> list1.count(4) # Return the count for number 4
 6  2
 7  >>> list2 = [99, 54]
 8  >>> list1.extend(list2)
 9  >>> list1
10  [2, 3, 4, 1, 32, 4, 19, 99, 54]
11  >>> list1.index(4) # Return the index of number 4
12  2
13  >>> list1.insert(1, 25) # Insert 25 at position index 1
14  >>> list1
15  [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
16  >>>
```

List Basics

- List Methods
  - And more examples:

```
 1  >>> list1 = [2, 25, 3, 4, 1, 32, 4, 19, 99, 54]
 2  >>> list1.pop(2)
 3  3
 4  >>> list1
 5  [2, 25, 4, 1, 32, 4, 19, 99, 54]
 6  >>> list1.pop()
 7  54
 8  >>> list1
 9  [2, 25, 4, 1, 32, 4, 19, 99]
10  >>> list1.remove(32) # Remove number 32
11  >>> list1
12  [2, 25, 4, 1, 4, 19, 99]
13  >>> list1.reverse() # Reverse the list
14  >>> list1
15  [99, 19, 4, 1, 4, 25, 2]
16  >>> list1.sort() # Sort the list
17  >>> list1
18  [1, 2, 4, 4, 19, 25, 99]
19  >>>
```

List Basics

- Splitting a String into a List
  - The **`str`** class contains a method called **`split`**
  - Example:
    - Consider the following code:

```
items = "Jane John Peter Susan"
items = items.split()
```

    - This splits the string, **`items`**, based on the spaces
    - The result is a list with four elements:
      - `['Jane', 'John', 'Peter', 'Susan']`
    - Again, the **delimiter** used to split was a space
    - We could also split using a different delimiter…

List Basics

- Splitting a String into a List
  - Example:
    - Consider the following code:

```
items = "11/28/2018"
Items = items.split("/")
```

    - This splits the string, **`items`**, based on the forward slash (/)
    - The result is a list with three elements:
      - `['11', '28', '2018']`

List Basics

- Inputting Lists
  - It's often helpful to read data directly into a list
  - Consider the following code:

```
lst = [] # Create a list
print("Enter 10 numbers: ")
for i in range(10):
    lst.append(eval(input()))
```

  - Note:
    - This assumes entering one number per line

List Basics

- Inputting Lists
  - What if the user wants to enter all numbers on the same line? How can we do that?
  - Read the whole line
  - Split the line based on spaces
    - That results in a list of strings
      - Even though we know it's really a list of numbers (inside strings)
  - Then use a "List Comprehension" to make a new list of numbers from this list of strings…

List Basics

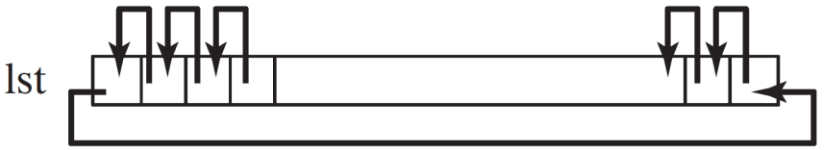
- Inputting Lists
  - What if the user wants to enter all numbers on the same line? How can we do that?
  - Here's the code:

```
line = input("Enter 10 vals separated by spaces: ")
items = line.split()
lst = [eval(x) for x in items]
```

    - So we read the full input string as a single line of characters
    - And we saved it into a variable called "line"
    - We then split this line based on spaces
    - And we make a new list of numbers based on that

List Basics

- Shifting Lists
  - What about shifting the elements in your list all to the left? How would you do this?
  - So assume you have a list of 10 elements
  - You want to perform the following moves:
    - index 1 should move to index 0
    - index 2 should move to index 1
    - index 3 should move to index 2
    - And the original value at index 0 should be placed at index 9
  - Try to do this…



List Basics

- Shifting Lists
  - Solution

```
def shift(lst):
    temp = lst[0] # Retain the first element
    # Shift elements left

    for i in range(1, len(lst)):
        lst[i - 1] = lst[i]

    # Move the first element to fill in the last position
    lst[len(lst) - 1] = temp
```

List Basics

- Lists can Simplify your code!!!
  - Example:
    - Suppose you want to get the English name for a particular month number
    - How would you traditionally do this?
    - Answer:
      - Have a long if/elif/else statement to get the correct month name
    - Code:

```
if monthNumber == 0:
    print("The month is January")
elif monthNumber == 1:
    print("The month is February")
...
else:
    print("The month is December")
```

List Basics

- Lists can Simplify your code!!!
  - Example:
    - But now, using lists:
      - Make a list of strings representing the months
      - And for ease you can make 13 strings in this list, where the first string is empty
      - This let's January go at the natural position of index 1
    - Code:

```
months = ["", "January", "February", "March", ..., "December"]
month_number = int(input("Enter a month number (1 to 12): "))
print("The month is", months[month_number])
```

List Basics

Check Yourself

- Given `lst = [30, 1, 2, 1, 0]`, what is the list after applying each of the following statements? Assume that each line of code is independent.

```
lst.append(40)
lst.insert(1, 43)
lst.remove(1)
lst.pop(1)
lst.pop()
lst.sort()
lst.reverse()
random.shuffle(lst)
```

List Basics

Check Yourself

- Given `lst = [30, 1, 2, 1, 0]`, what is the list after applying each of the following statements? Assume that each line of code is independent.

```
lst.append(40) => [30, 1, 2, 1, 0, 40]
lst.insert(1, 43) => [30, 43, 1, 2, 1, 0]
lst.remove(1) => [30, 2, 1, 0]
lst.pop(1) => [30, 2, 1, 0]
lst.pop() => [30, 1, 2, 1]
lst.sort() => [0, 1, 1, 2, 30]
lst.reverse() => [0, 1, 2, 1, 30]
random.shuffle(list) => list is randomly shuffled
```

List Basics

Check Yourself

- Given `lst = [30, 1, 2, 1, 0]`, what is the return value of each of the following statements?

```
lst.index(1)
lst.count(1)
len(lst)
max(lst)
min(lst)
sum(lst)
```

List Basics

Check Yourself

- Given `lst = [30, 1, 2, 1, 0]`, what is the return value of each of the following statements?

```
lst.index(1) => 1
lst.count(1) => 2
len(list) => 5
max(list) => 30
min(list) => 0
sum(list) => 34
```

List Basics

Check Yourself

- Given **list1 = [30, 1, 2, 1, 0]** and **list2 = [1, 21, 13]**, what is the return value of each of the following statements?

```
list1 + list2
2 * list2
list2 * 2
list1[1 : 3]
list1[3]
```

List Basics

Check Yourself

- Given **list1 = [30, 1, 2, 1, 0]** and **list2 = [1, 21, 13]**, what is the return value of each of the following statements?

```
list1 + list2  => [30, 1, 2, 1, 0, 1, 21, 13]
2 * list2 => [1, 21, 13, 1, 21, 13]
list2 * 2 => [1, 21, 13, 1, 21, 13]
list1[1 : 3] => [1, 2]
list1[3] => 1
```

List Basics

Check Yourself

- Given `list1 = [30, 1, 2, 1, 0]`, what is the return value of each of the following statements?

```
[x for x in list1 if x > 1]
[x for x in range(0, 10, 2)]
[x for x in range(10, 0, -2)]
```

List Basics

Check Yourself

- Given `list1 = [30, 1, 2, 1, 0]`, what is the return value of each of the following statements?

```
[x for x in list1 if x > 1]
[x for x in range(0, 10, 2)]
[x for x in range(10, 0, -2)]
```

- Answers:

```
[x for x in list1 if x > 1]  => [30, 2]
[x for x in range(0, 10, 2)]  =>  [0, 2, 4, 6, 8]
[x for x in range(10, 0, -2)] => [10, 8, 6, 4, 2]
```

List Basics

Check Yourself

- Given **list1 = [30, 1, 2, 1, 0]** and **list2 = [1, 21, 13]**, what is the return value of each of the following statements?

```
list1 < list2
list1 <= list2
list1 == list2
list1 != list2
list1 > list2
list1 >= list2
```

List Basics

Check Yourself

- Given **list1 = [30, 1, 2, 1, 0]** and **list2 = [1, 21, 13]**, what is the return value of each of the following statements?

```
list1 < list2       False
list1 <= list2      False
list1 == list2      False
list1 != list2      True
list1 > list2       True
list1 >= list2      True
```

List Basics

Check Yourself

- What are **list1** and **list2** after the following lines of code?

```
list1 = [1, 43]
list2 = [x for x in list1]
list1[0] = 22
```

- Answer:
  - list1 is [22, 43]
  - list2 is [1, 43]

List Basics

Check Yourself

- Write statements to do the following:
  - Create a list with 100 Boolean False values.
  - Assign the value 5.5 to the last element in the list.
  - Display the sum of the first two elements.
  - Compute the sum of the first five elements in the list.
  - Find the minimum element in the list.
  - Randomly generate an index and display the element of this index in the list.

List Basics

Check Yourself

- Write statements to do the following:
  - Create a list with 100 Boolean False values.
  - Code:

```
lst = [False] * 100
```

List Basics

Check Yourself

- Write statements to do the following:
  - Assign the value 5.5 to the last element in the list.
  - Code:

```
lst[len(lst) - 1] = 5.5
```

List Basics

Check Yourself

- Write statements to do the following:
  - Display the sum of the first two elements.
  - Code:

```
print(lst[0] + lst[1])
```

List Basics

Check Yourself

- Write statements to do the following:
  - Compute the sum of the first five elements in the list.
  - Code:

```
total = sum(lst[0:5])
```

List Basics

Check Yourself

- Write statements to do the following:
  - Find the minimum element in the list.
  - Code:

```
minimum = min(lst)
```

List Basics

Check Yourself

- Write statements to do the following:
  - Randomly generate an index and display the element of this index in the list.
  - Code:

```
print(list[random.randint(0, len(lst) - 1)])
```

Problem: Lotto Numbers

- So, you wanna buy lotto tickets, and each ticket apparently has ten lotto numbers, with each of the ten numbers chosen from a range of numbers from 1 to 99. You've purchased a certain amount of tickets and would like all 99 possible numbers to be represented on those tickets.
- In summary:
  - Write a program to determine if all lotto numbers fully cover the numbers from 1 to 99.

Problem: Lotto Numbers

- Write a program to determine if all lotto numbers fully cover the numbers from 1 to 99.
  - You will read ticket numbers, one per line
  - The last line of input will contain a single 0
    - This is how you know to stop reading (break out of loop)
  - Check Portal for a text file containing test cases
  - Also check portal for three different solutions:
    - One using a boolean array to store used numbers
    - One adding newly seen elements to a new list
    - One making a list of all numbers between 1 and 99 and then removing from that list each number seen in the tickets

PYTHON BOOT CAMP

Module 7:
Lists