Programming and Debugging
Approaching Programming Problems

- Carefully read the specification
- Break it down into small, simple tasks
- Use pseudocode or a flowchart to help explain the algorithm
- Convert that into code. Code last!
Debugging

- Read the error messages!

- Look at the code where the error occurs. If you can't see the error there, look at lines before it.

- If you don't get an error but your code doesn't work, then examine your logic.

- If you're not sure what your program is doing, print out the values of variables at different points, so you can see how they are change and where the error happens.
The List Data Structure
Variables vs. Lists

- So far we have been working with variables, which can be thought of as “buckets” that hold a particular piece of data.

- Variables can only hold one piece of data at a time. Example:
  - x = 5
  - y = 5.0
  - z = ‘hello’
  - q = True

- However, there are times when we need to keep track of multiple pieces of data at the same time, and a single variable is limited to holding just one thing at a time.
Lists

- Lists are considered a “sequence” object. Sequence objects have the ability to hold multiple pieces of data at the same time.

- We can use a single sequence variable to hold any number of values.

- In most programming languages we call these “arrays.” In Python we call these “lists.”
Lists vs. Variables

List

Variable
Lists in Python

- You can create a list in Python by using bracket notation. Example:

  ```python
  my_list = [1, 2, 3]
  ```

- The above code will create a new list in Python that holds three integers – 1, 2 and 3 – in that order.

- Think of a list as a “book” that holds a series of sheets of paper (variables)
Lists in Python

- Lists can contain any data type that we have covered so far. Example:

  ```python
  my_list = ['Craig', 'John', 'Chris']
  ```

- Lists can also mix data types. Example:

  ```python
  my_list = ['Craig', 5.0, True, 67]
  ```

- You can print the value of a list using the `print()` function. Example:

  ```python
  print (my_list)
  ```
Indexing List Elements

- In a book you can reference a page by its page number.
- In a list you can reference an element by its index number.
- Indexes start at the number zero.

Example:

```python
my_list = ['Craig', 'John', 'Chris']
print (my_list[0])
```

```
>> Craig
```
Invalid indexes

You will get a runtime error if you attempt to access an element outside the range of a list. For example:

```python
my_list = ['Craig', 'John', 'Chris']

print (my_list[4])  # Index doesn’t exist!
```

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print (my_list[4])
  IndexError: list index out of range
List Operators
You can use the repetition operator ("*") to ask Python to repeat a list, much like how you would repeat a string. Example:

```python
my_list = [1, 2, 3] * 3
print (my_list)
```

```
>> [1, 2, 3, 1, 2, 3, 1, 2, 3]
```
List Concatenation

- You can use the concatenation operator ("+") to ask Python to combine lists, much like how you would combine strings. Example:

```python
my_list = [1, 2, 3] + [99, 100, 101]
print (my_list)

>> [1, 2, 3, 99, 100, 101]
```

- Note: You cannot concatenate a list with a non-list
Comparing Lists

- Lists with the identical contents are equal
  >>> a = [1,5,'hello']
  >>> b = [2,5,'hello']
  >>> a == b
  False
  >>> a = [2,5,'hello']
  >>> a == b
  True

- Other comparison operators also work, but have more complicated meanings.
Finding items in a list

- You can easily find a particular item in a list by using the “in” operator. Here’s an example:

  ```python
  my_list = ['pie', 'cake', 'pizza']
  if 'cake' in my_list:
      print("I found cake!")
  else:
      print("No cake found.")
  ```

- The “in” operator lets you search for any item in a list. It will return a Boolean value that indicates whether the item exists somewhere in the list.
Lists in Lists

- Unlike strings, you cannot use 'in' to test if one list is a subset of another.

```python
>>> a = [1, 2, 5, 'hello']
>>> b = [2, 5, 'hello']
>>> b in a
False
```

- However, a list can contain other lists. So you can use 'in' to test for that.

```python
>>> a = a + [b]
>>> a
[1, 2, 5, 'hello', [2, 5, 'hello']]
>>> b in a
True
```
Given the following lists, write a program that lets the user type in a product code. If the product name exists in our inventory you should print out that it is in our inventory. Otherwise you should print out that the product is not found.

```
products = ['X125', 'X127', 'Z121', 'Z991']
```
List Mechanics
Changing the value of an item in a list

- Lists are “mutable,” which means that they can be changed once they have been created (unlike strings)

Example:

```python
my_list = [1, 2, 3]
print (my_list)
>> [1,2,3]

my_list[0] = 99
print (my_list)
>> [99,2,3]
```
List Mechanics

- List variables are considered “references”

- This means that they “reference” or “point” to a specific region of your computer’s memory. This behavior can cause some interesting side effects. For example, the following two list variables refer to the same list in memory.

```python
mylist1 = [1,2,3]
mylist2 = mylist1

print (mylist1)
print (mylist2)

>> [1,2,3]
>> [1,2,3]
```
List Mechanics

- This means that you can change one of the lists and the change will be reflected in the other.

```python
mylist1 = [1,2,3]
mylist2 = mylist1

mylist1[0] = 999

print (mylist1)
print (mylist2)

>> [999,2,3]
>> [999,2,3]
```
Copying a List

- Python will only create new lists when you use `[]` syntax to define a list for the first time.

- You can take advantage of this behavior to create true copies of your list objects. For example:

  ```python
  mylist1 = [1,2,3]
  mylist2 = [] + mylist1

  mylist1[0] = 999

  print (mylist1)
  print (mylist2)
  
  >> [999,2,3]
  >> [1,2,3]
  ```
Creating Lists

- You can create an empty list with no elements using the following syntax:

```python
mylist = []
```

- Sometimes you want to create a list that contains a certain number of “pre-set” elements. For example, to create a list with 10 elements that are all set to zero you could do the following:

```python
mylist = [0] * 10
```
You can also create lists using the `range()` function. For example, to create a list of all even numbers between 0 and 100 you can do the following:

```python
even_numbers = list(range(0, 100, 2))
```
Iterating over a list
Using a “for” loop to iterate through a List

- You can use a for loop to iterate through a list. When you do this the target variable of your loop assumes each value of each element of the list in order. Example:

```python
my_list = [1,2,3]

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

```plaintext
>> 1
>> 2
>> 3
```
Programming Challenge: Count the A’s

- Given the following list:
  
  \[
  \text{grades} = [90, 100, 70, 45, 76, 84, 93, 21, 36, 99, 100]
  \]

- Write a program that counts the # of A’s (scores between 90 and 100)

- Extension: Count the # of B’s, C’s, D’s and F’s
Using Loops to Modify Lists

- A for loop is a convenient way to sequentially iterate through a list.

- The target variable in a for loop assumes the value of the current item in the list as you iterate.

- However, the target variable isn’t very helpful if you want to change the value of an item in a list since it is just a copy of the data that exists in the list. For example:

```python
mylist = [1,2,3]
for n in mylist:
    n = n * 5
print (mylist)
```

```python
>> [1,2,3]
```
Changing List Items

In order to change a list item you need to know the index of the item you wish to change. For example:

```python
mylist = [1,2,3]
mylist[0] = 999
print (mylist)
>> [999, 2, 3]
```
Iterating over a list using index values

- There are two main techniques for iterating over a list using index values. They include:
  - Setting up a counter variable outside the list and continually updating the variable as you move to the next position in the list
  - Using the `range()` function to create a custom range that represents the size of your list
Using a counter variable and a for loop to iterate over a list

- If you set up an accumulator variable outside of your loop you can use it to keep track of where you are in a list. For example:

```python
mylist = [1, 2, 3]
counter = 0

for num in mylist:
    mylist[counter] = mylist[counter] * 2
    counter += 1

print (mylist)

>> [2, 4, 6]
```
You can also use the `range()` function to construct a custom range that represents all of the indexes in a list. Example:

```python
mylist = [1,2,3]

for counter in range(0,len(mylist)):
    mylist[counter] = mylist[counter] * 2

print (mylist)

>> [2,4,6]
```
Programming Challenge

- Given the following list of prices, write a program that modifies the list to include 7% sales tax

Working with Lists
Creating empty lists for processing

Since you cannot access an element outside of the range of a list it is sometimes necessary to set up a correctly sized list before you begin working with it.

Example:

```python
# create a list of 7 zeros
daily_sales = [0] * 7
```
Programming Challenge

- Write a program that asks the user for daily sales figures for a full week (Sunday – Saturday)

- Store these values in a list and print them out at the end of your program
Slicing Lists

- Sometimes you need to extract multiple items from a list.

- Python contains some built in functions that make it easy for you to “slice” out a portion of a list. Example:

```python
list_1 = ['zero', 'one', 'two', 'three', 'four', 'five']
list_2 = list_1[1:3]

print (list_1)
print (list_2)

>> ['zero', 'one', 'two', 'three', 'four', 'five']
>> ['one', 'two']
```
Slicing Lists

- To slice a list you use a series of “slice indexes” to tell Python which elements you want to extract. Example:

```python
new_list = old_list[start:end]
```

- Python will copy out the elements from the list on the right side of the assignment operator based on the start and end indexes provided.

- Note that indexes work just like the `range()` function – you will grab items up until the end index, but you will not grab the end index itself.
Slicing Lists

- If you omit the `start_index` in a slice operation, Python will start at the first element of the list.

- If you omit the `end_index` in a slice operation, Python will go until the last element of the list.

- If you supply a third index, Python will assume you want to use a step value. This works the same as the step value you would pass to the `range()` function.
Programming Challenge

- Given the following list:

  ```python
  my_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
  ```

  Write a program that does the following:

  - Extract the first 3 elements of the list into a new list
  - Extract the characters b, c, and d into a new list
  - Extract the last 4 characters into a new list

- Write a program that creates a list of all #’s between 1 and 100. Then create a list of all even numbers using your first list as input.
Programming Challenge

- Given these two lists:

  a = [1, 2, 3, 4, 5]
  b = [2, 3, 10, 11, 12, 1]

- Write a program that finds all elements that exist in both lists (i.e. the integer 2 exists in both lists). Store your result in a list and print it out to the user.
List Methods
Adding items to a list

- You have already seen a few ways in which you can add items to lists:
  - Repeat the list using the “*” operator
  - Concatenate the list using the “+” operator

- Another way to add items to a list is to use the “append” method.

- The append method is a function that is built into the list datatype. It allows you to add items to the end of a list. Example:

```python
mylist = ['Christine', 'Jasmine', 'Renee']
mylist.append('Kate')
print(mylist)
```

```
>> ['Christine', 'Jasmine', 'Renee', 'Kate']
```
Programming Challenge

- Write a program that continually prompts a user to enter in a series of first names.
- Store these first names in a list.
- Print them out at the end of your program
You can have Python sort items in a list using the sort() method. Here’s an example:

```python
my_list = ['pie', 'cake', 'pizza']
my_list.append('apple')
print (my_list)

my_list.sort()
print (my_list)

>> ['pie', 'cake', 'pizza', 'apple']
>> ['apple', 'cake', 'pie', 'pizza']
```
Take the name program you wrote earlier and update it to print out the names entered in alphabetical order.
Finding the position of an item in a list

- You can use the “index” method to ask Python to tell you the index of an item in a list.

- The “index” method takes one argument – an element to search for – and returns an integer value of where that element can be found.

- Caution: The index method will throw an exception if it cannot find the item in the list.

- Here’s an example:

```python
def my_list = ['pizza', 'pie', 'cake']
if 'pie' in my_list:
    location = my_list.index('pie')
    print ('pie is at position #', location)
else:
    print ('not found!')
```
Given that the following lists match up with one another (i.e. the product at the first position of the products list matches the price at the first position in the prices list), write a product price lookup program.

```python
products = ['peanut butter', 'jelly', 'bread']
prices = [3.99, 2.99, 1.99]
```
Getting the largest and smallest values in a list

- Python has two built-in functions that let you get the highest and lowest values in a list. They are called “min” and “max” – here’s an example:

```python
prices = [3.99, 2.99, 1.99]
biggest = max(prices)
smallest = min(prices)
print (smallest, 'up to', biggest)
```
Removing items from a list

- You can remove an item from a list by using the “remove” method. Here’s an example:

```python
prices = [3.99, 2.99, 1.99]
prices.remove(2.99)
print(prices)
```

- Note that you will raise an exception if you try and remove something that is not in the list. Make sure to test to see if it is in the list first using the “in” operator, or use a try / except block to catch any errors you might raise.
Removing items from a list

- You can also remove an item from a list based on its index position. We can do this using the 'del' keyword, like this:

```python
prices = [3.99, 2.99, 1.99]
del prices[0] # remove whatever is at slot 0
print (prices)
```
Reversing a list

- You can use the reverse method on a list to reverse its elements.

- This will not sort the list in reverse order – it will simply shuffle the elements of a list such that the first element becomes the last element, the second element becomes the second to last element, etc.
Programming Problems
Programming Challenge

- Write a weight loss program that prompts the user to enter in 7 days of weight values

- At the end of the program, print out the following:
  - Weight on the first day
  - Weight on the last day
  - Change from the first day to the last day
  - Average weight for the period
  - Highest weight for the period
  - Lowest weight for the period
Programming Challenge

- Ask the user for 5 colors
- Don't allow duplicate values
- Sort the colors in both ascending and descending order
Programming Challenge

- Write a program that generates a random 5 digit lottery number
- Numbers must be between 1 and 60
- No duplicate numbers are permitted
- Sort the numbers in ascending order and print them out to the user
Programming Challenge

- Roulette is a game where a ball is rolled around a circular track – eventually the ball will come to rest in a slot that is labeled with the numbers 0 through 36.

- Gamblers can bet on an individual number – if they are successful they win a large prize (36 : 1).

- Write a program that asks the user for a number between 0 and 36.

- Then spin the roulette wheel 1000 times. Find out how many times the user’s # came up.

- Also tell the user the most frequent number and the least frequent number that came up.