What do we know so far?

- Class – lecture, lab, Repfactor, Quick Checks, R&R, Easter eggs
- Solve problems, computers useful, user vs. programmer
- Sequence of instructions, algorithm is step-by-step
- Python is 3rd most popular language, core principles
- Syntax, runtime, & logic errors, testing & debugging, hardware vs. software
- Control flow – step-by-step, function call, conditional, loop
- IDE shell, editor, install Python, Hello World
- Interpreter, compiler, Python Standard Library
- Variables, assignment, numeric expr., precedence
- Print function, Strings, concatenation, indexes, in, *
- Interactive programs, if, if-else, if-elseif-else, int, float
- Boolean expressions: ==, !=, <, <=, >, >=, not, and, or
- Input function, comparing strings, programming conventions
- Variable & function names lowercase, CONSTANTS, indent
- while, for range, augmented assignments, palindromes
- Turtle Graphics, forward, left, right, pensize, pencolor, dot, circle
- goto, pensp, pensdon, fill, begin_fill, end_fill, speed
- Calling & defining functions, import, parameters vs. arguments, return
- Positional args, default args, variable args, keyword args, local variables
- String methods, replace, method vs. function, built-in & external functions
- Using loops and functions to create graphics, random numbers, design process

Lists

A list is a sequence. The items or elements of a list can be just about anything, including a character, string, integer, or even another list.

```python
letters = ['a', 'b', 'c', 'd', 'e']
numbers = [867, 5, 3, 0, 9]
beatles = ['John', 'Paul', 'George', 'Ringo']
```

The elements in a list can be different types.

```python
jackson5 = [['a', 'b', 'c'], [1, 2, 3]]
```

List Indexing

The elements in a list can be accessed individually using their indexes similar to string indexing.

```python
beatles = ['John', 'Paul', 'George', 'Ringo']
names = beatles[3]
print(names)
```

```python
Ringo
```

Element values also can be changed using their index, because lists are mutable meaning their values can be changed.

```python
beatles[3] = 'Pete'
print(beatles)
```

```python
['John', 'Paul', 'George', 'Pete']
```

Iterating a List

Iterating the elements in a list can be done with a for loop.

```python
beatles = ['John', 'Paul', 'George', 'Ringo']
for name in beatles:
    print(name)
```

```python
John
Paul
George
Ringo
```

List Concatenation

Concatenation with lists works the same as with strings, using the concatenation operator (+).

```python
list1 = ['a', 'm', 'g']
list2 = ['n', 'f', 'w']
exclam = list1 + list2
print(exclam)
```

```python
['a', 'm', 'g', 'n', 'f', 'w']
```

Augmented assignment works on lists, as well.

```python
exclam += ['!']
print(exclam)
```

```python
['a', 'm', 'g', 'n', 'f', 'w', '!']
```
List Containment

Containment works with lists using the `in` operator.

```python
list1 = ['o', 'm', 'g']
if 'm' in list1:
    print('Item found in list')
else:
    print('Item not found in list')
```

The `not in` operator works on lists, as well.

```python
if 'a' not in list1:
    print('Item doesn’t belong in list anyway!')
```

Containment is an example of a membership operation.

Repeating Lists

The repetition operator (*) will create a new list by repeating another list.

```python
food = ['Spam']
meal = food * 5
print(meal)
```

Membership

The `in` and `not in` operators determine membership in any sequence, including lists and strings.

```python
menu = 'spam, baked beans, ice cream, strawberries'
if 'beans' in menu:
    print('Beans are good for your heart')
if 'cupcake' not in menu:
    print('Good! I am avoiding carbs!')
```

Beans are good for your heart
Good! I am avoiding carbs!

Identity

The is and is not operators determine whether two variables refer to the exact same object.

```python
x = 7
y = x
if x is y:
    print('x and y refer to the same object')
else:
    print('x and y do NOT refer to the same object')
```

x and y refer to the same object

When you assign one variable to another variable, the two refer to the same underlying object.

Type Checking

The is and is not operators also can determine the type of an object.

```python
x = 7
if type(x) is int:
    print('x refers to an integer')
else:
    print('x does NOT refer to an integer')
x refers to an integer
```

Use the `isinstance` function for more complicated types.

```python
if isinstance(some_variable, str):
    print('The variable is a string')
```

Common List Algorithms: Minimum Value

A common algorithm used with lists is finding a minimum.

```python
values = [3, 11, 99, 5, 17, 2, 73, 3, 9, 12, 8]
min_index = 0
for i in range(1, len(values) - 1):
    if values[i] < values[min_index]:
        min_index = i
print('min_index is', min_index)
print('min value is', values[min_index])
```

```
min_index is 5
min value is 2
```

Lists enable the use of many other common algorithms:

- Filling and printing a list
- Finding an average, minimum or maximum
- Searching for and swapping elements
Slicing Strings and Lists

Slicing a string or list means getting a sub-sequence.

```python
mystring = 'Hello, world!
part = mystring[7:12]
print(part)
world
```

A slice starts with the item at the first index and includes everything up to but not including the second index.

```python
letters = [ 'a', 'b', 'c', 'd', 'e', 'f', 'g']
slice = letters[2:5]
print('Original list:', letters)
print('Slice of list:', slice)
Original list: ['a', 'b', 'c', 'd', 'e', 'f', 'g']
Slice of list: ['c', 'd', 'e']
```

Slicing can be done from one end or the other.

```python
prefix = letters[:3]  # list up to not incl index 3
suffix = letters[4:]  # list from index 4 to the end
```

List Methods

Adding, deleting, finding, counting, reordering, copying.

```python
# Add using append, extend or insert
names.append('Jimmy')
# Delete using remove, pop or clear
names.remove('Elisabeth')
# Find and count using index and count
print('Mary is at index:', names.find('Mary'))
print('Joes in the list:', names.count('Joe'))
# Reorder using sort, reverse, and variations
names.sort()
# Copy using copy, list() or slicing[:]
newnames = names.copy()
```