INTRODUCTION TO SETS

LEARNING OBJECTIVES:

* What are sets?
* Why sets?
* Intro to working with sets (syntax)
* Basic operations with sets

WHAT ARE SETS?

In Python, a set is an object that is made up of an unordered collection of unique elements.

That means, similar to lists and dictionaries, sets are objects made up of other elements. But, unlike lists, elements in sets have no order. And, unlike dictionaries, elements in sets are not associated with any corresponding value. And, unlike lists and dictionaries, sets cannot have more than one of the same element.

Individual elements of sets must be “hashable” (meaning, each element of the set must a hash value which never changes during its lifetime). Thus, only immutable objects (numbers, strings, and some tuples, and not lists, dictionaries, or other sets) can be elements of sets.

Sets may be “frozen sets”, which are immutable once created. These can actually be elements in sets as well.

Sets are a unique datatype with inherent advantages and disadvantages. Which brings us to our next section…

WHY USE SETS?

For one, speed! Unlike lists, sets are unordered. Thus, checking for membership of a value in a set is faster than checking to see if a value is included in a list.

For two, memory! Unlike dictionaries, sets do not have (indexes and keys). Thus, sets take up less memory than dictionaries.

So, sets are a useful datatype to use in situations that require checking for membership, don’t need to keep track of item order or duplicate items, and have hashable elements.

WORKING WITH SETS: SYNTAX

THE “SET()” FUNCTION

Sets can be declared by passing an iterable (such as a list, string, or tuple) through the “set()” function.

The order in which the elements are entered is unimportant, and duplicate elements are made irrelevant.

print set(["h", "e", "l", "l", "o"])

print set(["o", "l", "e", "h"])

print set(["o", "l", "e", "h"]) == set(["h", "e", "l", "l", "o"])

set(['h', 'e', 'l', 'o'])

set(['h', 'e', 'l', 'o'])

True

Passing a string through the “set()” function will create a set of its characters.

print set("hello")

set(['h', 'e', 'l', 'o'])

Frozen sets are declared using the “frozenset()” function. Although they cannot be modified like sets, they are evaluated as though they are sets.

print frozenset(["h", "e", "l", "l", "o"]) == set(["h", "e", "l", "l", "o"])

True

CURLY BRACKETS

In newer versions of Python (2.7 and after), sets can be declared using curly brackets.

print {"h", "e", "l", "l", "o"}

set(['h', 'e', 'l', 'o'])

This can be handy because it requires less typing. However, curly brackets cannot express empty sets, and will not transform strings into sets of their characters.

print type(set())

print type({})

<type 'set'>

<type 'dict'>

a = "hello"

print set(a)

print {a}

set(['h', 'e', 'l', 'o'])

set(['hello'])

WORKING WITH SETS: MODIFYING SETS

Sets (but not frozen sets) can be modified using a few basic operations…