* Course Overview
* Iterators and Iterables
  + Iterators decouple retrieval from structure
  + Iterable: object which you can iterate
  + Iterators: encapsulate the current position in the iterable
    - Interface that allows us to get next value
    - And when iterable is exhausted
  + Python uses forward iterators
    - Traverse iterable in a forward direction
    - No option to jump around and go back
  + Can create reverse iterator
    - Forward iterator with their sense of forward reverse
  + Obtaining and advancing an iterator
    - iterator = iter(iterable)
    - try:
    - item = next(iterator)
    - print(item)
    - except StopIteration:
    - print(“No more items”)
  + Creating iterables
    - list = []
    - tuple = ()
    - dictionary = dict(a=””, b=””)
    - def iterable\_function()
    - yield value1
    - yield value2
    - iterable\_square = (x \* x for x in range(10))
  + Iterable
    - Object that can be passed successfully to built in iter function
    - iter call \_\_iter\_\_() on object, which returns an iterator
  + Iterator
    - Object that fulfills the iterator protocol
    - \_\_iter\_\_()
    - All iterator must also be iterables
    - \_\_next\_\_(): returns the next value or stop iteration
    - When next is called, it class \_\_next\_\_()
* Motivating Iterators: Tree Trav..
  + Binary tree can be represented in as a sequence
* Breath-first, Level-order Iterator
  + Perfect tree is (2^h) -1, h being the height of the tree
* Depth-first, Pre-order Iterator
  + Python doesn’t have a dedicated stack collection
  + But list has all the methods to use list as a stack
* Depth-first, In-order Iterator
* Filtering Iterators
* Transforming Iterators
* Iterables
  + Iterators are usually used indirectly, through iterable-consuming functions and for-loops