Memory Model

- abstraction: representation of important aspects of the execution of Python code
- goal: help us focus on important aspects of Python code
- principles:
  1. EVERY name has a VALUE and that value is a "reference" (memory address)
  2. references can refer to:
     - objects
     - functions
     - classes
     - modules

Example:
1. a = [1, 2, 3]
2. b = a
3. a[0]= 4
4. print(b)
Tree ADT

- made up of nodes
- each node has potentially many "children"
  * each node has exactly one "parent",
  except for one node: the "root"
- single node with no parent = "root"
- node with no children = "leaf"
- all other nodes = "internal nodes"
  * a tree contains no cycles (impossible to follow links and get back where you started)
  * a tree is connected: every node can be reached by starting at the root and following links

Terminology:
- "siblings": nodes with the same parent
- "ancestor", etc.: defined as you would expect...
- "branching factor": maximum number of children of any node
  (example: tree above has branching factor 3)
- "path": sequence of connected links (example: A-B-E; F-G)
- length of path = number of edges (links) on the path
  (BEWARE: some authors use number of nodes instead)

- "depth": length of the path from a node back to the root (example: depth(A) = 0, depth(B) = depth(F) = 1, etc.)
- "height": length of a longest path from a node down to a leaf (example:
  height(C) = 0, height(B) = 1, height(A) = 3)
- height of a tree = height of the root

Tree traversal: going through every node systematically to carry out a particular task at each node.
- preorder: first, carry out the task for the root, then do it recursively for each of the root's children
- postorder: first, carry out the task recursively for each of the root's children, then finish with the root itself

pre = left
post = right

- preorder: A B C D E F G H
- postorder: C D E B H G F A