The Stack

The stack is LIFO (last in, first out) – we add items to the top and we take them from the top.

1. `push(item)` – place `item` on the stack
2. `pop()` – remove and return the top element from the stack
3. `top()` – report the top element on the stack
4. `length()` – report the number of elements on the stack
5. `is_empty()` – report whether or not the stack is empty

The Queue

The queue is FIFO (first in, first out) – when we add an item, we add to the back, and when we take one, we take from the front.

1. `enqueue(item)` – place `item` in the queue
2. `dequeue()` – remove and return the next element from the queue
3. `front()` – report the next element from the queue
4. `length()` – report the number of elements in the queue
5. `is_empty()` – report whether or not the stack is empty

The Priority Queue

The priority queue is similar to a queue, but items are stored with their priority. The element with the highest priority is always removed next.

1. `add(item, key)` – place `item` on the queue with key `key`
2. `remove_min()` – remove and return the element in the queue with minimum key (highest priority)

3. `min()` – report the element with minimum key (highest priority)
4. `length()` – report the number of elements in the queue
5. `is_empty()` – report whether or not the queue is empty

The List

Lists are mutable, sequential data structures that can be navigated using index positions, and possibly iterated over.

1. `add(pos, item)` – add a new element containing `item` after position `pos`
2. `add_current(item)` – add a new element containing `item` after the cursor
3. `add_first(item)` – add a new element containing `item` to the start of the list
4. `add_last(item)` – add a new element containing `item` to the end of the list
5. `remove(pos)` – remove the element after position `pos`
6. `remove_current()` – remove the element at the cursor
7. `remove_first()` – remove the first element from the list
8. `remove_last()` – remove the last element from the list
9. `replace(pos, item)` – replace the element at `pos` with `item`
10. `replace_current(item)` – replace the element at the cursor with `item`
11. `replace_first(item)` – replace the first element of the list with `item`
12. `replace_last(item)` – replace the last element of the list with `item`
13. `clear()` – remove all elements

    ----
14. `next()` – move the cursor forward 1 place
15. `prev()` – move the cursor backward 1 place
16. `move_to_front()` – move the cursor to the first element
17. `move_to_last()` – move the cursor to the last element

    ----

18. `get(pos)` – return the element at position `pos`
19. `get_current()` – return the element at the cursor
20. `get_first()` – return the first element in the list
21. `get_last()` – return the last element in the list

    ----

22. `find(item)` – report the position of the first occurrence of `item`, if it is in the list
23. `has_next()` – report whether or not there are more elements after the cursor
24. `length()` – report how many elements are in the list
25. `is_empty()` – report whether or not the list is empty

The Set

A set is a collection of elements, with no duplicates, and with no particular order among the elements.

1. `add(item)` – add `item` to this set, if it is not already there
2. `delete(item)` – delete `item` from this set, if it is there
3. `pop()` – remove and return an aribitrary element
4. `clear()` – remove all elements from the set
5. `contains(item)` – report whether or not the set contains `item`
6. `length()` – report the number of elements in the set
7. `is_empty()` – report whether or not the set is empty

    ----

8. `union(other)` – return the set containing all elements in this set and all elements in `other`
9. `intersection(other)` – return the set containing all elements in this set that are also in `other`
10. `subset(other)` – report whether or not this set is a subset of `other`

11. `proper_subset(other)` – report whether or not this set is a proper subset of `other`
12. `is_disjoint(other)` – report whether or not this set contains no elements that are also in `other`
    ----
13. `next()` – move the cursor to the next element, in some order
14. `move_to_first()` – move the cursor to the first element, in some order
15. `current()` – return the element at the cursor
16. `has_next()` – report whether or not there is another element in the order after the cursor

The Position

A position represents a node or index in some data structure, and maintains a reference to an element.

1. `element()` – return the element at this position

The Binary Tree

This is a tree in which each node has at most two children. The left child of a node contains an element which is less than the element in the parent node, and the right child of a node contains an element which is greater than the element in the parent node.

1. `add(item)` – add `item` as a new node in the tree
2. `remove(item)` – remove `item` from the tree
3. `search(item)` – return the Position of `item` in the tree
4. `root()` – return the Position of the root node of the tree
5. `positions()` – return a list of all Positions in the tree
6. `elements()` – return a list of all items in the tree

7. `inorder()` – return a string in-order traversal of the tree
8. `depth()` – report the depth of the tree
9. `size()` – report the number of items in the tree
10. `is_empty()` – report whether or not the tree is empty

    ----

11. `parent(p)` – return the parent of `p`
12. `left(p)` – return the left child of `p`
13. `right(p)` – return the right child of `p`
14. `children(p)` – return a list of `p`'s children
15. `is_root(p)` – report whether or not `p` is the root Position
16. `is_leaf(p)` – report whether or not `p` is a leaf
17. `has_left(p)` – report whether or not `p` has a left child
18. `has_right(p)` – report whether or not `p` has a right child
19. `height(p)` – report the height of Position `p`
20. `num_children(p)` – report the number of children of `p`

The Map

A dictionary or map is a storage and look-up structure maintaining (key, value) pairs. Each value has a unique key. Specifying the key allows us to retrieve the value from the structure.

1. `setitem(key, value)` – assign `value` to the element paired with `key`, or create a new element with value `value` and key `key`
2. `delitem(key)` – remove the element with key `key` and return its value
3. `getitem(key)` – return the value paired with `key`
4. `contains(key)` – report whether or not the map has an element with the key `key`
5. `length()` – report the number of elements in the map
6. `is_empty()` – report whether or not the map is empty