HOF (Higher Order Functions), (Tree) Recursion

- Tree recursion is useful for exploring different options: Making decisions, aggregating results, etc.
- def f1(*args):
  for i in args: ...

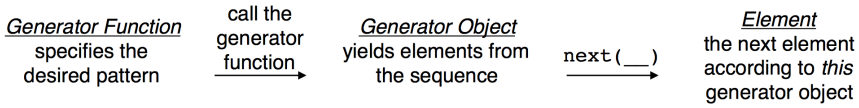
Python Lists (Env. Diag.), Trees

- Cloning a list: if there are lists as elements of the original list, then they point to the same thing.
- When you concatenate two lists, you are creating a new list.
  - lst1 = [0, [1]]; lst2 = lst1[:]. Then, lst1 is not lst2, lst1[1] is lst2[1].
- lst = [1, 2, 3]
  - lst * $\mathbb{Z}^-$ = [ ]
  - [1] is not [1]; [1] == [1]; [1].append != [1].append
  - lst[−1] = 3; lst[: −1] = [1, 2]; lst[: : −1] = [3, 2, 1]
  - lst[3:] = [ ], lst[3] = IndexError
  - new = lst[:] makes a copy of lst; new = lst makes a pointer to the same lst
  - lst.insert(100,7); lst == [1, 2, 3, 7]
  - lst.index(10) gives Error; lst.remove(10) gives Error.
  - lst = [0,1,2]; lst2 = [9,8]
    lst[2:], lst2[1:] = lst2[1:], lst[2:]
    lst == [0, 1, 8]; lst2 == [9, 2]
- lst = [1]; lst = lst.append(0); lst is None; lst = lst.append(0) leads to ERROR
- Draw env. diag of lst = list(other_lst)
- prune_leaves(t, vals):
  if is_leaf(t): return None if label(t) in vals else tree(label(t))
  else: return tree(label(t), [prune_leaves(b, vals) for b in branches(t) if prune_leaves(b, vals)])
- def reverse(L): # Reverses and mutate list:
  for i in range(len(L)//2):
  L[i], L[-i-1] = L[-i-1], L[i]

Nonlocal (Mutable Values/Functions), Iterators & Generators

- Global names cannot be modified using the nonlocal keyword.
- Names in the current frame cannot be overridden using the nonlocal keyword. This means we cannot have both a local and nonlocal binding with the same name in a single frame.
- Note that most iterators are also iterables - that is, calling iter on them will return an iterator. This means that we can use them inside for loops. However, calling iter on an iterator will return that same iterator, not a copy.
- If a dictionary changes in structure because a key is added or removed, then all iterators become invalid gives Error. On the other hand, changing the value of an existing key does not invalidate iterators or change the order of their contents.

- lst = [0, 1, 2]; i = iter(lst); next(i); lst.append(3); next(i); next(i); next(i) – okay
- * If you want to yield infinitely many elements, then use while True; if you want to yield finitely many elements, then use for.
- Calling the generator function returns a **generator** object. First next call to generator object:
  1. Start at the top of the generator function body (Start at previous start position for Subsequent next calls to generator object)
  2. Run until immediately after the first yield encountered
  3. Return the value in the yield line
  4. Save start position at line following yield
- lst = [0, 1, 2]; i = iter(lst); next(i); next(i); next(i) (almost reaches StopIteration). Now, if we use: lst.append(3) or lst.extend[3] or lst += [3], we could still call next(i) (returns 3). However, if we use: lst = lst + [3], calling next(i) would raise ERROR.
- lst(iter) will take the iterator to the endpoint, i.e. next(iter) will raise StopIteration



OOP (Object-Oriented Programming)

- repr, str, print
  - Displaying Instance, uses the __repr__ (output has no quotation marks)
  - repr(instance) calls __repr__ (always has quotation marks)
  - print(Instance) calls __str__ and displays the value of the returned string, and if __str__ isn't found, use the __repr__ (always no quotation marks)
  - str(Instance) calls __str__ and returns the value of the returned string (always with quotation marks)
- As an attribute of a class, a method is just a function; but as an attribute of an instance, it is a bound method (instance.func is a method).
- If an instance calls a parent class' function, then that instance will pass itself in as the first argument to the function. Otherwise, everything works like normal and you have to pass in all the arguments manually.
- The parent of a function has to be a function frame, and the parent of an object has to be an object frame.

Orders of Growth, Linked Lists

- $\Theta(1) < \Theta(\log n) < \Theta(n) < \Theta(n^b) < \Theta(b^n) < \Theta(n!)$
- For recursion (recursive tree) structure: $\sum_{\text{layers}}$ (word per node) $\times$ (nodes per layer)
- Loops: (runtime to execute 1 loop) * (number of times looped)
  Recursion: (runtime of 1 call) * (number of calls)

- Link(1) != Link(1) (while [1] == [1])
- Order matters: lnk is Link.empty (or lnk.rest is Link.empty)

Extra Sanity Checks

- Dictionary keys can't be lists. Dictionary keys and values will be evaluated when being created. In essence, only functions will not evaluate their bodies when being created.
- print(a, b) first evaluates all of its parameters and then prints everything: print(1, Error) only prints Error
- A function only prints stuff due to (1) print statements or (2) the final return to Global frame
- Whenever you see an equals sign:
  1. Do not look at the left side of the equals sign;
  2. Evaluate the right side of the equals sign. Draw the result somewhere with enough space;
  3. Now look at the left side of the equals sign, and bind that value to what you just drew.
- Mutation:
  - lst.append(item) takes element
  - lst.extend([item]) takes iterable
  - lst += [item] takes list (equivalent to .extend)

  Cloning:
  - list(lst) takes iterable
  - lst = lst + [item] takes list (concatenates)
  - lst[:]
- For Tree problems, typically the base case is either a leaf or a condition on label (or a condition on the other passed-in variable if there exists), and the recursive calls get applied on branches.
- txt = 'a\tb'
  - txt[1] gives: '\t'
  - txt gives: 'a\tb'
  - print(txt) gives: a (tab) b
- def make_anonymous_factorial():
  return (lambda f: lambda k: f(f, k))(lambda f, k: k if k == 1 else mul(k, f(f, sub(k, 1))))
- (1,) gives one-element (1,) tuple
- a = range(0, 3); type(x for x in a) is a generator; type([x for x in a]) is a list. (Both could be sum()-ed)
- Can add two tuples: (1,) + (2,) == (1,2)
- x = (1,2); a,b = x gives a = 1, b = 2
- Weird list comps etc.
  - [i for i in range(3) if $i > 1$] gives [2]

  - [i if $i > 1$ else 100 for i in range(3)] gives [100, 100, 2]
  - 7 if 1 == 1 else 83 gives 7 (else cond. necessary)
- Concatenate string with int: my_str + str(my_int)
- my_dict.get(key, default_value)
- Calling repr on anything (mostly) basically adds quotation marks around it
- Set literals follow the mathematical notation of elements enclosed in {braces}. Duplicate elements are removed upon construction.
- 'range' object is not an iterator

```
Lists:
                                Complexity
Operation      | Example       | Class          | Notes
---------------+---------------+----------------+-------------------------------
Index          | l[i]          | O(1)           |
Store          | l[i] = 0      | O(1)           |
Length         | len(l)        | O(1)           |
Append         | l.append(5)   | O(1)           | mostly: ICS-46 covers details
Pop            | l.pop()       | O(1)           | same as l.pop(-1), popping at end
Clear          | l.clear()     | O(1)           | similar to l = []

Slice          | l[a:b]        | O(b-a)         | l[1:5]:O(l)/l[:]:O(len(l)-0)=O(N)
Extend         | l.extend(...) | O(len(...))    | depends only on len of extension
Construction   | list(...)     | O(len(...))    | depends on length of ... iterable

check ==, !=   | l1 == l2      | O(N)           |
Insert         | l[a:b] = ...  | O(N)           |
Delete         | del l[i]      | O(N)           | depends on i; O(N) in worst case
Containment    | x in/not in l | O(N)           | linearly searches list
Copy           | l.copy()      | O(N)           | Same as l[:] which is O(N)
Remove         | l.remove(...) | O(N)           |
Pop            | l.pop(i)      | O(N)           | O(N-i): l.pop(0):O(N) (see above)
Extreme value  | min(l)/max(l) | O(N)           | linearly searches list for value
Reverse        | l.reverse()   | O(N)           |
Iteration      | for v in l:   | O(N)           | Worst: no return/break in loop

Sort           | l.sort()      | O(N Log N)     | key/reverse mostly doesn't change
Multiply       | k*l           | O(k N)         | 5*l is O(N): len(l)*l is O(N**2)

Tuples support all operations that do not mutate the data structure (and they
have the same complexity classes).
```