List comprehensions:

```python
[<map exp> for <name> in <iter exp> if <filter exp>]
```

Short version:

```python
[<map exp> for <name> in <iter exp>]
```

A combined expression that evaluates to a list using this evaluation procedure:
1. Add a new frame with the current frame as its parent
2. Create an empty result list that is the value of the expression
3. For each element in the iterable value of <iter exp>:
   A. Bind <name> to that element in the new frame from step 1
   B. If <filter exp> evaluates to a true value, then add the value of <map exp> to the result list

The result of calling repr on a value is what Python prints in an interactive session.

The result of calling str on a value is what Python prints using the print function:

```python
print(str(123))
```

str and repr are both polymorphic; they apply to any object. repr invokes a zero-argument method repr on its argument.

```python
>>> today.__repr__ #=> today_repr()
>>> today.str() #=> today_str() 2014-10-13
```

Memoization: 
```
Memoization:
   def memoize(f):
       cache = {}  
       def memoized(n):
           if n not in cache:
               cache[n] = f(n)
               return cache[n]
           return cache[n]
```

Type dispatching: Look up a cross-type implementation of an operation based on the types of its arguments.

Type coercion: Look up a function for converting one type to another, then apply a type-specific implementation.

```python
θ(n)
```

Exponential growth. Recursive fib takes \( \Theta(\phi^n) \) steps, where \( \phi = \frac{1 + \sqrt{5}}{2} \approx 1.61802 \).

Incrementing the problem scales \( R(n) \) by a factor \( \Theta(\phi^n) \).

Quadratic growth. E.g., overlap doubling the problem only increments \( R(n) \) by the problem size \( n \).

Linear growth. E.g., factors or exp doubling the problem only increments \( R(n) \).

Logarithmic growth. E.g., exp doubling the problem only increments \( R(n) \).

Constant. The problem size doesn't matter.

Logarithms: The base of a logarithm does not affect the order of growth of a process \( \Theta(\log_b(n)) \).

Nesting: When an inner process is repeated for each step in an outer process, multiply the steps in the outer and inner processes to find the total number of steps \( \Theta(n^2) \).
Recursive description:
- A tree has a root label and a list of branches.
- Each branch is a tree.
- A tree with zero branches is called a leaf.

Relative description:
- Each location is a node.
- Each node has a label.
- One node can be the parent/child of another.

```
def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)

def is_tree(t):
    if not is_tree(branch):
        return False
    return True

def is_leaf(t):
    return not is_branches(t)

def leaves(t):
    """The leaf values in a tree.""
    if t.is_leaf():
        return [t.label()]
    else:
        return sum((leaves(b) for b in branches(t)), [])

class Tree:
    def __init__(self, label, branches=[]): self.label = label
        for branch in branches:
            assert is_tree(branch)
        self.branches = list(branches)
    def __len__(self):
        return sum(len(branches(b)) for b in branches(self))

A binary tree is a binary tree where each root is larger than all values in its left branch and smaller than all values in its right branch.
```