Readwordlist

```
def readwordlist(filename):
    file = open(filename)
    text = file.read().upper()
    wordset = set(word for word in text.splitlines())
    prefixset = set(p for word in wordset for p in prefixes(word))
    return wordset, prefixset
```

Extend Prefix

```
def find_words(letters):
    results = set()

def extend_prefix(w, letters):
    if w in WORDS: results.add(pre)
    if w not in PREFIXES: return
        for L in letters:
            extend_prefix(pre+L,letters.replace(l,'',1),results)
    return results
```

Adding Suffixes

```
9
0 def add_suffixes(hand, pre, results):
1     """Return the set of words that can be formed by extending p
2     if pre in WORDS: results.add(pre)
3     if pre in PREFIXES:
4         for L in hand:
5             add_suffixes(hand.replace(L,'',1), pre+L, results)
6     return results
```

Longest Words

```
71
72 def longest_words(hand, board_letters):
73     "Return all word plays, longest first."
74     words = word_plays(hand, board_letters)
75     return sorted(words, reverse=True, key=len)
76
77
```

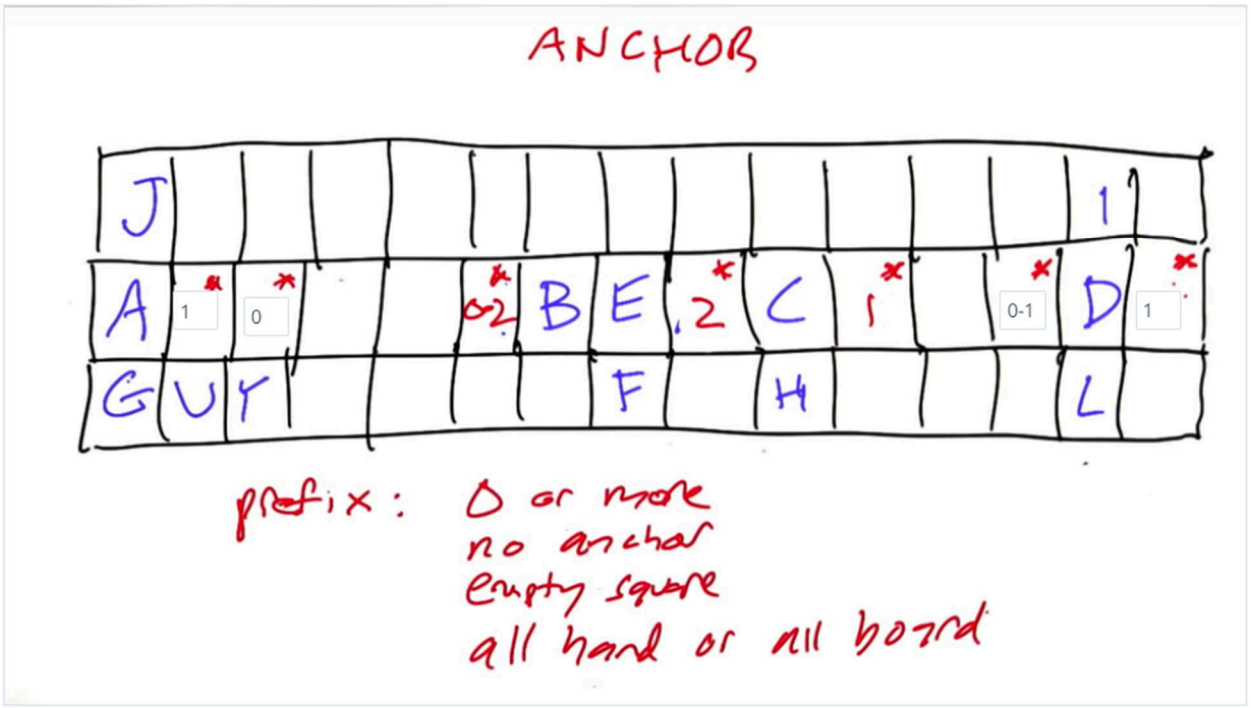
Word Score

```
11
12 def word_score(word):
13     word_score = 0
14     for letter in word:
15         word_score += POINTS[letter]
16     return word_score
17
```

Top N Hands

```
def topn(hand, board_letters, n=10):
    "Return a list of the top n words that hand can play, sorte
    words = word_plays(hand, board_letters)
    return sorted(words,reverse=True, key=word_score)[:n]
```

Anchors



Legal Prefixes

```
    return ('', i-s)

def is_empty(sq):
    "Is this an empty square (no letters, but a valid position on board)."
    return sq  == '.' or sq == '*' or isinstance(sq, anchor)

def is_letter(sq):
    return isinstance(sq, str) and sq in LETTERS

def test_row():
    assert legal_prefix(2, a_row) == ('A', 1)
    assert legal_prefix(3, a_row) == ('',0)
    assert legal_prefix(6, a_row) == ('',2)
    assert legal_prefix(9, a_row) == ('BE',2)
    assert legal_prefix(11, a_row) == ('C',1)
    assert legal_prefix(13, a_row) == ('',1)
    return 'test_row passes'
```

Increasing Efficiency

```
66
67  def find_prefixes(hand, pre='', results=None):
68      """Find all prefixes (of words) that can be made from letters in hand."""
69      global prev_hand, prev_results
70      if hand == prev_hand: return prev_results
71      if results is None: results = set()
72      if pre == '': prev_hand, prev_results = hand, results
73      if pre in WORDS or pre in PREFIXES: results.add(pre)
74      if pre in PREFIXES:
75          for L in hand:
76              find_prefixes(hand.replace(L, '', 1), pre+L, results)
77      return results
```

Show And Spell

```
17
18  def show(board):
19      "Print the board."
20      for i in board:
21          print i
22
```

Horizontal Plays

```
134
135 def horizontal_plays(hand, board):
136     "Find all horizontal plays -- (score, pos, word) pairs -- across all rows."
137     results = set()
138     for (j, row) in enumerate(board[1:-1], 1):
139         set_anchors(row, j, board)
140         for (i, word) in row_plays(hand, row):
141             results.add(((i,j), word))
142     return results
143
```

All Plays

```
151 def all_plays(hand, board):
152     """All plays in both directions. A play is a (score, pos, dir, word) tuple,
153     where pos is an (i, j) pair, and dir is a (delta-_i, delta_j) pair."""
154     hplays = horizontal_plays(hand, board)
155     vplays = horizontal_plays(hand, transpose(board))
156     return (set(((i, j), ACROSS, w) for ( (i, j), w) in hplays) |
157             set(( (i, j), DOWN, w) for ( (j, i), w) in vplays))
158
159
160
```

Making The Board

```
219
220 def show(board):
221     "Print the board."
222     for j, fila in enumerate(board):
223         for i, ss in enumerate(fila):
224             print( ss if (is_letter(ss) or ss == '|') else BONUS[j][i])
225         print
226
```

Making Plays

```
20
21 def make_play(play, board):
22     "Put the word down on the board."
23     (score, (i, j), (di, dj), word) = play
24     for (n, L) in enumerate(word):
25         board[j+ n*dj][i + n*di] = L
26     return board
```

Best Play

```
def best_play(hand, board):
    "Return the highest-scoring play.  Or None."
    plays = all_plays(hand, board)
    return sorted(plays)[-1] if plays else NOPLAY

NOPLAY = None
```