```
#  Populate test files
@'
Files full of
various words.
'@ | Out-File -FilePath C:\Test\File1.txt

@'
Create an index
of words.
'@ | Out-File -FilePath C:\Test\File2.txt

@'
Use the index
to find the files.
'@ | Out-File -FilePath C:\Test\File3.txt
```

```
#  Index files
Index-File C:\Test\File1.txt, C:\Test\File2.txt, C:\Test\File3.txt
```

Because PowerShell is a shell language, it is "a user interface to do a search". After running the script defining the functions and running a command to index the files, the user can simply run the search function at the PowerShell command prompt. Alternatively, one could create a more complex custom UI or GUI if desired.

```
#  Query index
Find-Word files
```

Output:

```
C:\Test\File1.txt
C:\Test\File3.txt
```

Python (/wiki/Category:Python)

Simple inverted index

First the simple inverted index from here (https://en.wikipedia.org/wiki/Inverted_index) together with an implementation of a search for (multiple) terms from that index.

```
'''
This implements: http://en.wikipedia.org/wiki/Inverted_index of 28/07/10
'''

from pprint import pprint as pp
from glob import glob
try: reduce
except: from functools import reduce
try:    raw_input
except: raw_input = input


def parsetexts(fileglob='InvertedIndex/T*.txt'):
    texts, words = {}, set()
    for txtfile in glob(fileglob):
        with open(txtfile, 'r') as f:
            txt = f.read().split()
            words |= set(txt)
            texts[txtfile.split('\\')[-1]] = txt
    return texts, words

def termsearch(terms): # Searches simple inverted index
    return reduce(set.intersection,
                  (invindex[term] for term in terms),
                  set(texts.keys()))

texts, words = parsetexts()
print('\nTexts')
pp(texts)
print('\nWords')
pp(sorted(words))

invindex = {word:set(txt
                     for txt, wrds in texts.items() if word in wrds)
            for word in words}
print('\nInverted Index')
pp({k:sorted(v) for k,v in invindex.items()})

terms = ["what", "is", "it"]
print('\nTerm Search for: ' + repr(terms))
pp(sorted(termsearch(terms)))
```

Sample Output

```
Texts
{'T0.txt': ['it', 'is', 'what', 'it', 'is'],
 'T1.txt': ['what', 'is', 'it'],
 'T2.txt': ['it', 'is', 'a', 'banana']}

Words
['a', 'banana', 'is', 'it', 'what']

Inverted Index
{'a': ['T2.txt'],
 'banana': ['T2.txt'],
 'is': ['T0.txt', 'T1.txt', 'T2.txt'],
 'it': ['T0.txt', 'T1.txt', 'T2.txt'],
 'what': ['T0.txt', 'T1.txt']}

Term Search for: ['what', 'is', 'it']
['T0.txt', 'T1.txt']
```

Full inverted index

There is a re-write of the `termsearch` function to work off this type of index, as well as a new `phrasesearch` function

The `phrasesearch` function will return multiple matches in a text, and goes on to show how this can be used to pick the text with most matches.

It is assumed that the following code is added to the end of the code for the simple case above and so shares its file opening and parsing results

```
from collections import Counter


def termsearch(terms): # Searches full inverted index
    if not set(terms).issubset(words):
        return set()
    return reduce(set.intersection,
                  (set(x[0] for x in txtindx)
                   for term, txtindx in finvindex.items()
                   if term in terms),
                  set(texts.keys()) )

def phrasesearch(phrase):
    wordsinphrase = phrase.strip().strip('"').split()
    if not set(wordsinphrase).issubset(words):
        return set()
    #firstword, *otherwords = wordsinphrase # Only Python 3
    firstword, otherwords = wordsinphrase[0], wordsinphrase[1:]
    found = []
    for txt in termsearch(wordsinphrase):
        # Possible text files
        for firstindx in (indx for t,indx in finvindex[firstword]
                          if t == txt):
            # Over all positions of the first word of the phrase in this txt
            if all( (txt, firstindx+1 + otherindx) in finvindex[otherword]
                    for otherindx, otherword in enumerate(otherwords) ):
                found.append(txt)
    return found


finvindex = {word:set((txt, wrdindx)
                      for txt, wrds in texts.items()
                      for wrdindx in (i for i,w in enumerate(wrds) if word==w)
                      if word in wrds)
             for word in words}
print('\nFull Inverted Index')
pp({k:sorted(v) for k,v in finvindex.items()})

print('\nTerm Search on full inverted index for: ' + repr(terms))
pp(sorted(termsearch(terms)))

phrase = '"what is it"'
print('\nPhrase Search for: ' + phrase)
print(phrasesearch(phrase))

# Show multiple match capability
phrase = '"it is"'
print('\nPhrase Search for: ' + phrase)
ans = phrasesearch(phrase)
print(ans)
ans = Counter(ans)
print('  The phrase is found most commonly in text: ' + repr(ans.most_common(1)[0][0]))
```

Sample Output

```
Full Inverted Index
{'a': [('T2.txt', 2)],
 'banana': [('T2.txt', 3)],
 'is': [('T0.txt', 1), ('T0.txt', 4), ('T1.txt', 1), ('T2.txt', 1)],
 'it': [('T0.txt', 0), ('T0.txt', 3), ('T1.txt', 2), ('T2.txt', 0)],
 'what': [('T0.txt', 2), ('T1.txt', 0)]}

Term Search on full inverted index for: ['what', 'is', 'it']
['T0.txt', 'T1.txt']

Phrase Search for: "what is it"
['T1.txt']

Phrase Search for: "it is"
['T0.txt', 'T0.txt', 'T2.txt']
  The phrase is found most commonly in text: 'T0.txt'
```

Racket (/wiki/Category:Racket)

```
#!/usr/bin/env racket
#lang racket
(command-line
 #:args (term . files)
 (define rindex (make-hasheq))
 (for ([file files])
   (call-with-input-file file
     (λ(in) (let loop ()
              (define w (regexp-match #px"\\w+" in))
              (when w
                (let* ([w (bytes->string/utf-8 (car w))]
                       [w (string->symbol (string-foldcase w))]
                       [r (hash-ref rindex w '())])
                  (unless (member file r) (hash-set! rindex w (cons file r)))
                  (loop)))))))
 (define res
   (for/list ([w (regexp-match* #px"\\w+" term)])
     (list->set (hash-ref rindex (string->symbol (string-foldcase w)) '()))))
 (define all (set->list (apply set-intersect res)))
 (if (null? all)
   (printf "No matching files.\n")
   (printf "Terms found at: ~a.\n" (string-join all ", "))))
```

Output: