RP – Rule Parser

0.91

L. Paris
December 2008

RP – Rule Parser....1
0.91....1
Introduction....3
sqs0.py - standard grep function....3
sqs1.py - standard grep function with rule....4
sqs2.py – standard grep function with separator characters for string to locate....6
sqs3.py – grep function with column specification....7
sqs4.py – grep function with two columns specification....8
sqs5.py – grep functions with multi ranges for string location....9
sqs6.py – reaching the select statement…....10
sqs7.py – appending columns to selection....12
sqs8.py – enhanced the where condition....14
Rule definition & keywords....18
Code definitions....20
Use of option verbose....20

History

Version	Date	Comments
0.9	Dec. 2008	Initial version posted
0.91	Jan. 2009	Minor changes + check rule name versus python reserved words

Introduction

RP is a small parser which parses string according to a rule defined in BNF format.
It is used as the re module for regular expressions.

To explain more in detail, let us show examples.

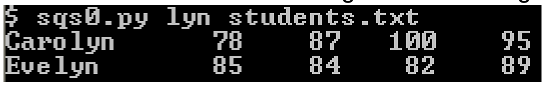
sqs0.py - standard grep function

```python
#!c:/python26/python.exe
#
# Simulate standard grep function.
#
# syntax:  sqs0.py  string_to_locate  fileid
#
import sys                          #sys module for argv

search,fileid=sys.argv[1:3]         #get the two words passed in parameter

try:
      id=open(fileid)                 #Open the file
      for l in id.readlines():         #loop on lines
            if l.find(search)>-1:     #if locate string,
                  print l[:-1]           #    then print the line
except Exception,e:
      print e
else:
      id.close()
```

Considering the following data of file: **students.txt**

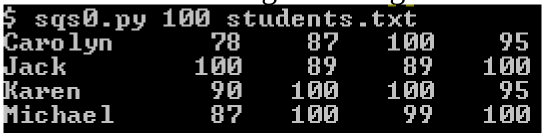
```
student    exam1 exam2 exam3 exam4
---------  ----- ----- ----- -----
Carolyn       78    87   100    95
Evelyn        85    84    82    89
Jack         100    89    89   100
Ken           88    79    79    93
Karen         90   100   100    95
Michael       87   100    99   100
```

Print lines containing the string “lyn”



```
$ sqs0.py lyn students.txt
Carolyn       78    87   100    95
Evelyn        85    84    82    89
```

Print lines containing the string “100”



```
$ sqs0.py 100 students.txt
Carolyn       78    87   100    95
Jack         100    89    89   100
Karen         90   100   100    95
Michael       87   100    99   100
```

sqs1.py - standard grep function with rule

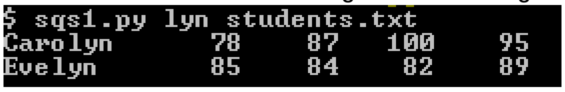
```
#!c:/python26/python.exe
#
# first sample using rule
#
import sys,rp                       #import the rp module
#
# we define the rule as a list of (sub)rules
#
# r"\S"*   means regular expression specifying
#          any character except blank
rule=['sqs  ::=  parms  fileid ',
      'parms::=  r"\S"* ',
      'fileid::= r"\S"* ']
#
# we concatenate arguments ... as words
parms=' '.join(sys.argv[1:])
#
# we make the parsing
cmp=rp.match(rule,parms)
#
#as re module, if the result object is None,
# the parsing is unsuccessful
if cmp==None:
      print "Error in parsing:"
else:
    #
    # now, to get values from parsing,
    # we use rule names as parser arguments.
    # cmp.sqs    will contain input parameters
    # cmp.parms  will contain string to locate
    # cmp.fileid will contain fileid to search in
      try:
            id=open(cmp.fileid)
            for l in id.readlines():
                  if l.find(cmp.parms)>-1:
                        print l[:-1]
      except Exception,e:
            print e
      else:
            id.close()
```

The rule contains three definitions as:

- ✓ **sqs** which is the main rule, containing two subrules, **parms** & **fileid**
- ✓ **parms** is a subrule specifying any characters, except blank
- ✓ **fileid** is a subrule specifying any characters, except blank

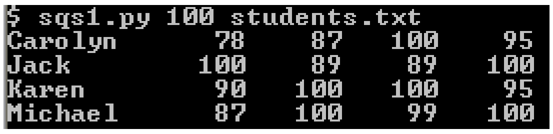
(details about rule syntax: Rule definitions)

Print lines containing the string “lyn”



```
$ sqs1.py lyn students.txt
Carolyn        78     87    100      95
Evelyn         85     84     82      89
```

Print lines containing the string “100”



```
$ sqs1.py 100 students.txt
Carolyn       78    87   100    95
Jack         100    89    89   100
Karen         90   100   100    95
Michael       87   100    99   100
```

sqs2.py – standard grep function with separator characters for string to locate

```
#!c:/python26/python.exe
#
# syntax:   sqs2.py  "/string to locate/ fileid"_
import sys,rp
rule=['sqs  ::=  parms  fileid ',
       'parms::=  sep car* sep ',
         'sep  ::=  r"\S" ',
         'car  ::=  r"." ^sep ',
         'fileid::= r"\S"* ']
#
# In its sample, we pass arguments as a single string enclosed in "
# (due to handle blanks if exist in string to locate)
parms=sys.argv[1]

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
      id=None
      try:
            id=open(cmp.fileid)
            for l in id.readlines():
                  if l.find(cmp.car)>-1:
                        print l[:-1]
      except Exception,e:
            print e
      else:
            if id!=None: id.close()
```

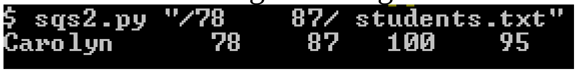
Now, for string delimited, we coded the rule **parms** as any character enclosed in separator character which can be any, except a blank ! (r”\S”)
The string to locate can contain any character except the separator itself, that is defined as:

```
      car  ::=  r"." ^sep
```

r”.” to specify any character
^sep to exclude the separator character   (unable to code in regular type: [^sep] !!)

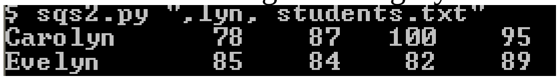
As we can search string containing blanks, we pass the entire parameters of the command as one enclosed in “

Print line containing the string: ’78   87’



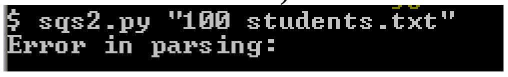
```
$ sqs2.py "/78     87/ students.txt"
Carolyn        78     87    100     95
```

Print lines containing the string ‘lyn’



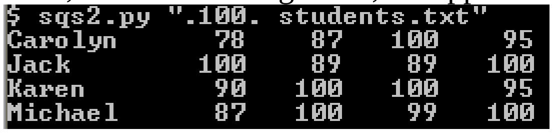
```
$ sqs2.py ",lyn, students.txt"
Carolyn        78     87    100     95
Evelyn         85     84     82     89
```

Try to locate lines containing the string 100, without any delimiter character, that encounters a parsing error. (NB: the parser discovered the first separator as the character ‘1’ and doesn’t find the end delimiter)



```
$ sqs2.py "100 students.txt"
Error in parsing:
```

Now, to find the string ‘100’, we appended delimiter: ‘.’



```
$ sqs2.py ".100. students.txt"
Carolyn        78     87    100      95
Jack          100     89     89     100
Karen          90    100    100      95
Michael        87    100     99     100
```

sqs3.py – grep function with column specification

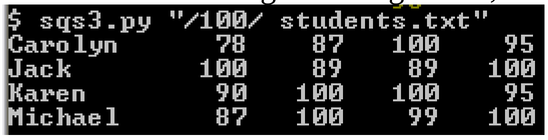
```
#!c:/python26/python.exe
import sys,rp
rule=['init n=0',
      'init m=999',
        'sqs  ::=  range? parms  fileid ',
      'range::=  n "-" m ',
        'n    ::=  r"[0-9]"* ',
        'm    ::=  r"[0-9]"* ',
      'parms::=  sep car* sep ',
        'sep  ::=  r"\S" ',
        'car  ::=  r"." ^sep ',
        'fileid::= r"\S"* ']
parms=sys.argv[1]

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
      id=None
      try:
            id=open(cmp.fileid)
            for l in id.readlines():
                  if l.find(cmp.car,int(cmp.n),int(cmp.m))>-1:
                        print l[:-1]
      except Exception,e:
            print e
      else:
            if id!=None: id.close()
```

In our rule, we can find new specifications as:

- ✓ `range?` meaning range is optional
- ✓ `range::=  n "-" m` range definition: column start – column end
- ✓ `n` & `m` are integer and defined using regular expression r”[0-9]”*
- ✓ `init n=0` initialisation of column start
- ✓ `init m=999` initialisation of column end

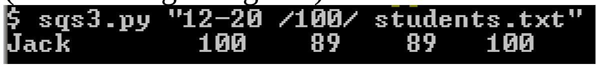
For a complete definition for rule, see chapter: rule definitions & keywords

Print lines containing the string ‘100’ , without columns specifications



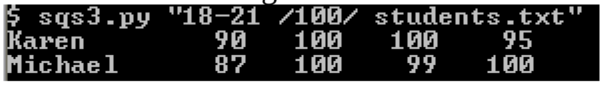
```
$ sqs3.py "/100/ students.txt"
Carolyn        78     87    100      95
Jack          100     89     89     100
Karen          90    100    100      95
Michael        87    100     99     100
```

Print line containg ‘100’ between columns 12 & 20
(columns beginning at 0)



```
$ sqs3.py "12-20 /100/ students.txt"
Jack        100    89    89   100
```

Print lines containing ‘100’ between columns 18 & 21



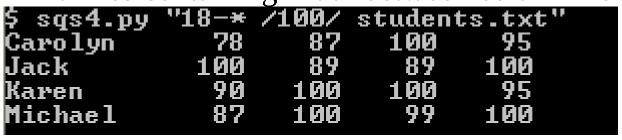
```
$ sqs3.py "18-21 /100/ students.txt"
Karen        90   100   100    95
Michael      87   100    99   100
```

sqs4.py – grep function with two columns specification

```
#!c:/python26/python.exe
import sys,rp
rule=['init n=0',
      'init m=999',
      'sqs  ::=  range? parms  fileid ',
      'range::=  n "-" "*" ',
      '      |   n "-" m ',
        'n    ::=  r"[0-9]"* ',
        'm    ::=  r"[0-9]"* ',
      'parms::=  sep car* sep ',
        'sep  ::=  r"\S" ',
        'car  ::=  r"." ^sep ',
        'fileid::= r"\S"* ']
parms=' '.join(sys.argv[1:])

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
      id=None
      n=int(cmp.n)
      m=int(cmp.m)
      try:
            id=open(cmp.fileid)
            for l in id.readlines():
                  if l.find(cmp.car,n,m)>-1:
                        print l[:-1]
      except Exception,e:
            print e
      else:
            if id!=None: id.close()
```

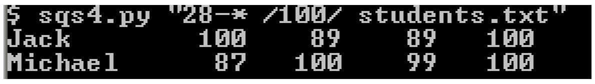
In this sample, we only appended another type of column specification, with a start column & ‘*’ as the end, indicating end of record.

Print lines containing ‘100’ between column 18 and end of record



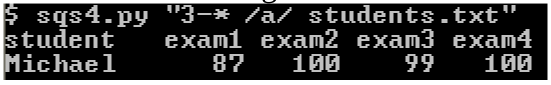
```
$ sqs4.py "18-* /100/ students.txt"
Carolyn      78    87   100    95
Jack        100    89    89   100
Karen        90   100   100    95
Michael      87   100    99   100
```

Now, from column 28 to end



```
$ sqs4.py "28-* /100/ students.txt"
Jack          100    89    89   100
Michael        87   100    99   100
```

Print lines containing ‘a’ between column 3 and end of record



```
$ sqs4.py "3-* /a/ students.txt"
student    exam1 exam2 exam3 exam4
Michael       87   100    99   100
```

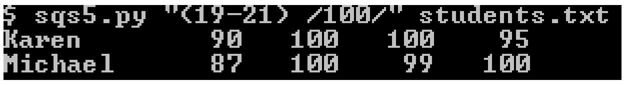
sqs5.py – grep functions with multi ranges for string location

With this sample we can make search on multi zones in the record.

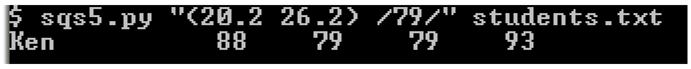
```
#!c:/python26/python.exe
import sys,rp
rule=['init sqs_ranges=[] ',
      'init sqs_locate="" ',
      'sqs  ::=  ranges? parms  fileid   @sqs_fileid="$fileid" ',
      'ranges::=  "(" range+ ")"  ',
        '       |   range ',
      '# for ranges, we appended two other types',
      '# and columns start at 1 (not 0), that explains the $n-1 ',
      'range::=  n "-" "*"              @sqs_ranges.append([$n-1,999]) ',
      '      |   n "-" m                @sqs_ranges.append([$n-1,$m]) ',
      '      |   n "." m                @sqs_ranges.append([$n-1,($m+$n-
1)]) ',
      '      |   n "-"                  @sqs_ranges.append([$n-1,999]) ',
        'n    ::=  r"[0-9]"*  ',
        'm    ::=  r"[0-9]"* ',
      'parms::=  sep car* sep          @sqs_locate="$car"',
        'sep  ::=  r"\S" ',
        'car  ::=  r"." ^sep ',
        'fileid::= r"\S"* ']
parms=' '.join(sys.argv[1:])

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
    id=None
    try:
        #
        #Opening...
        id=open(cmp.sqs_fileid)
        for l in id.readlines():
            #
            #if no ranges defined, then all the record
            if len(cmp.sqs_ranges)==0:
                data=l
            else:
                data=''
            #
            #Now, for each range, create the record
            for n,m in cmp.sqs_ranges:
                data+=l[n:m]
            if data.find(cmp.sqs_locate)>-1:
                 print l[:-1]
    except Exception,e:
         print e
    else:
          if id!=None: id.close()
```

As you can see in the rule, we appended, two new ranges definitions and parenthesis when we want to define more than one range on column selection.

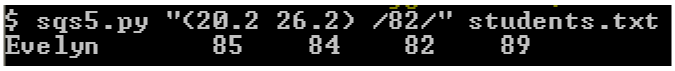
Below are commands and results:



```
$ sqs5.py "(19-21) /100/" students.txt
Karen            90    100    100     95
Michael          87    100     99    100
```



```
$ sqs5.py "(20.2 26.2) /79/" students.txt
Ken              88     79     79     93
```



```
$ sqs5.py "(20.2 26.2) /82/" students.txt
Evelyn           85     84     82     89
```

sqs6.py – reaching the select statement…

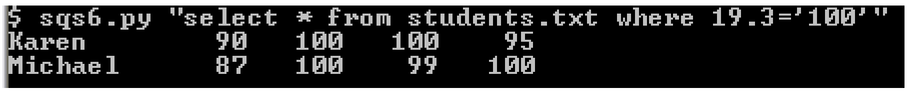
Now, we'll try to define our function as a select statement, in the form:

select * from *fileid* **where** *colrange operator string*

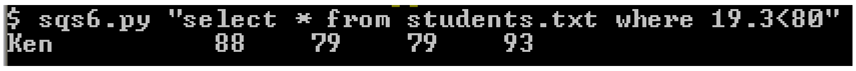
then code could appear as:

```
#!c:/python26/python.exe
import sys,rp

rule=['init sqs_range="" ',
      'sqs  ::=  SELECT "*" FROM fileid WHERE range operator value ',
      '                                     @sqs_fileid="$fileid"',
      '                                     @sqs_ope="$operator" ',

      'range::=  n "-" "*"                  @sqs_range=[$n-1,999] ',
      '      |   n "-" m                    @sqs_range=[$n-1,$m] ',
      '      |   n "." m                    @sqs_range=[$n-1,($m+$n-1)] ',
      '      |   n "-"                      @sqs_range=[$n-1,999] ',
        'n    ::=  r"[0-9]"*   ',
        'm    ::=  r"[0-9]"* ',
      'value::=  "\'" car* "\'"             @sqs_valtype=1  @sqs_value="$car"
',
      '      |   numeric                    @sqs_valtype=2  @sqs_value=numeric
',
      'operator::=  LIKE ',
      '         |    "="   ',
      '         |    ">"   ',
      '         |    "<"   ',
        'car  ::=  r"[^\']" ',
      'numeric::= r"[0-9]"* ',
        'fileid::= r"\S"* ']
parms=' '.join(sys.argv[1:])

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
    id=None
    try:
         id=open(cmp.sqs_fileid)
         for l in id.readlines():
            data=l[cmp.sqs_range[0]:cmp.sqs_range[1]]
            found=False
            if cmp.sqs_valtype==1:
                if cmp.sqs_ope.upper()=='LIKE':
                    if cmp.sqs_value.startswith('%'):
                        found=data.endswith(cmp.sqs_value[1:])
                    elif cmp.sqs_value.endswith('%'):
                        found=data.startswith(cmp.sqs_value[:-1])
                    else:
                        found=(data.find(cmp.sqs_value)>-1)
                elif cmp.sqs_ope=='=':
                    found=(data == cmp.sqs_value)
                else:
                    raise Exception('Operator not allowed with alpha')
            else:
                try:
                    v1=int(data.strip())
                    v2=int(cmp.sqs_value.strip())
                    if cmp.sqs_ope=='=': found=(v1==v2)
                    elif cmp.sqs_ope=='>': found=(v1>v2)
                    elif cmp.sqs_ope=='<': found=(v1<v2)
                except:
                    pass
            if found:
                print l[:-1]
    except Exception,e:
          print e
    else:
        if id!=None: id.close()
```

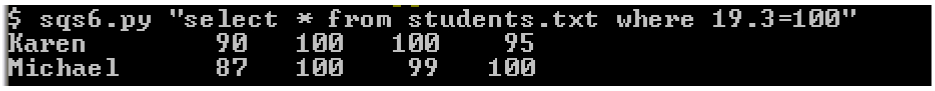
Samples run and results



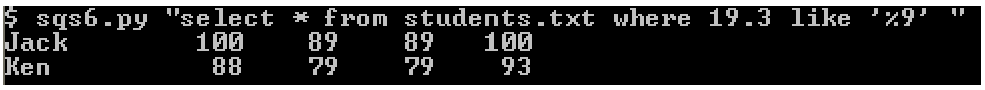
```
$ sqs6.py "select * from students.txt where 19.3='100'"
Karen           90    100    100     95
Michael         87    100     99    100
```



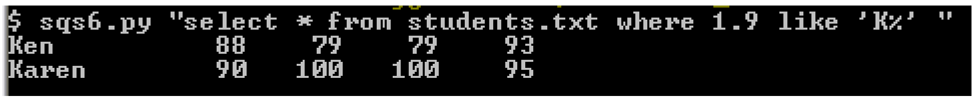
```
$ sqs6.py "select * from students.txt where 19.3<80"
Ken             88     79     79     93
```



```
$ sqs6.py "select * from students.txt where 19.3=100"
Karen           90    100    100     95
Michael         87    100     99    100
```



```
$ sqs6.py "select * from students.txt where 19.3 like '%9' "
Jack           100     89     89    100
Ken             88     79     79     93
```



```
$ sqs6.py "select * from students.txt where 1.9 like 'K%' "
Ken             88     79     79     93
Karen           90    100    100     95
```

sqs7.py – appending columns to selection

In this sample, we appended column selection and column name in where condition, in the form:

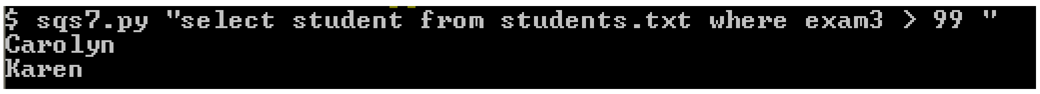
select *col* [, *col* [, *col* […]]] **from** *fileid* **where** *col operator value*

Sample code could appear as:

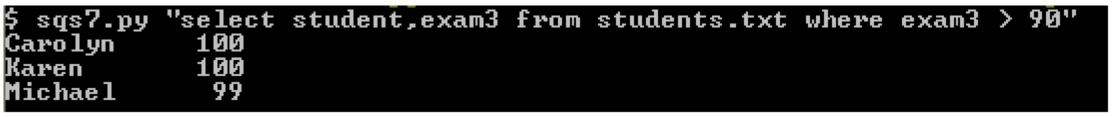
```
#!c:/python26/python.exe
import sys,rp

rule=['init sqs_selection=[] ',
      'sqs  ::=  SELECT selection FROM fileid WHERE column operator value
',
      '                                 @sqs_fileid="$fileid"',
      '                                 @sqs_column="$column"',
      '                                 @sqs_ope="$operator"',

      'selection::=  col0 cols*      ',
      'col0     ::=  col                @sqs_selection.append("$col")',
      'cols     ::=  "," col            @sqs_selection.append("$col")',
      'col      ::=  r"[A-Za-z0-9]"*  ',
      'column   ::=  r"[A-Za-z0-9]"*  ',
      'value::=  "\'" car* "\'"        @sqs_valtype=1  @sqs_value="$car"
',
      '      |   numeric               @sqs_valtype=2  @sqs_value=numeric
',
      'operator::=  LIKE ',
      '         |   "="  ',
      '         |   ">"  ',
      '         |   "<"  ',
        'car  ::=  r"[^\']" ',
      'numeric::= r"[0-9]"* ',
        'fileid::= r"\S"* ']
parms=' '.join(sys.argv[1:])

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
    id=None
    try:
        id=open(cmp.sqs_fileid)
        #
        #Read the first two lines that contain columns name & length
        #and prepare dictionary 'columns'
        colnames=[x for x in id.readline()[:-1].split(' ') if x!=""]
        collengths=id.readline()[:-1].split(' ')
        deb=0
        columns={}
        for i,name in enumerate(colnames):
           colsize=len(collengths[i])
           columns[name]=(deb,colsize+deb)
           deb+=colsize+1
        #
        #now loop on lines until end of file
        for l in id.readlines():
            indexes=columns[cmp.sqs_column]     #get indexes associated to
column
            data=l[indexes[0]:indexes[1]]       #get the data of where
condition
```

```python
                found=False
                if cmp.sqs_valtype==1:
                    if cmp.sqs_ope.upper()=='LIKE':
                        if cmp.sqs_value.startswith('%'):
                            found=data.endswith(cmp.sqs_value[1:])
                        elif cmp.sqs_value.endswith('%'):
                            found=data.startswith(cmp.sqs_value[:-1])
                        else:
                            found=(data.find(cmp.sqs_value)>-1)
                    elif cmp.sqs_ope=='=':
                        found=(data == cmp.sqs_value)
                    else:
                        raise Exception('Operator not allowed with alpha')
                else:
                    try:
                        v1=int(data.strip())
                        v2=int(cmp.sqs_value.strip())
                        if cmp.sqs_ope=='=': found=(v1==v2)
                        elif cmp.sqs_ope=='>': found=(v1>v2)
                        elif cmp.sqs_ope=='<': found=(v1<v2)
                    except:
                        pass
                if found:
                    txt=''
                    for col in cmp.sqs_selection:
                        c=columns[col]
                        txt+=l[c[0]:c[1]]+' '
                    print txt
    except Exception,e:
          print e
    else:
        if id!=None: id.close()
```

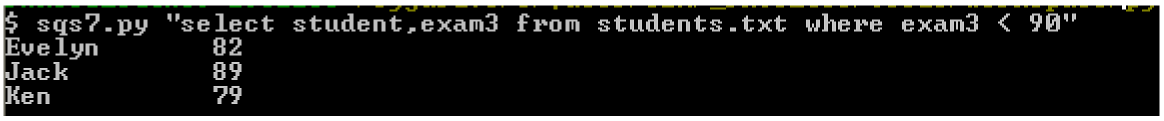
Samples runs & results:



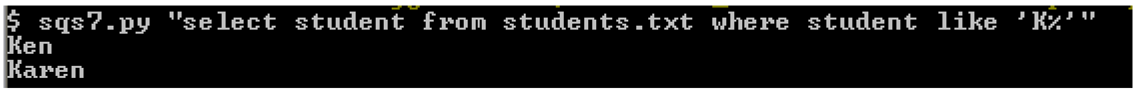
```
$ sqs7.py "select student from students.txt where exam3 > 99 "
Carolyn
Karen
```



```
$ sqs7.py "select student,exam3 from students.txt where exam3 > 90"
Carolyn      100
Karen        100
Michael       99
```



```
$ sqs7.py "select student,exam3 from students.txt where exam3 < 90"
Evelyn        82
Jack          89
Ken           79
```



```
$ sqs7.py "select student from students.txt where student like 'K%'"
Ken
Karen
```

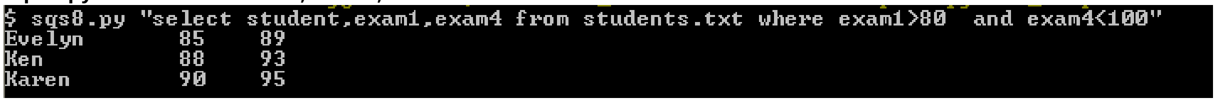
sqs8.py – enhanced the where condition

In this sample, we define the where condition as a list boolean conditions with the operators OR, AND and also ( ) .
To process the ‘where condition’, we will create a string in which we will append conditions and that ‘where condition’ will be interpreted for each line retrieved.

In order to achive that, we can code our sample as:

```
#!c:/python26/python.exe
import sys,rp

def process(data,cols,col,ope,val):
    ret=False
    indexes=cols[col]
    data=data[indexes[0]:indexes[1]]
    #print data,ope,val
    if isinstance(val,int):
        exec("ret=("+data+ope+str(val)+")")
    else:
        if ope.upper()=='LIKE':
            if val.startswith('%'):
                ret=data.endswith(val[1:])
            elif val.endswith('%'):
                ret=data.startswith(val[:-1])
            else:
                ret=(data.find(val)>-1)
        elif ope=='=':
            ret=(data == val)
    return ret

rule=['option norun',             #norun, means that lines of code
                                  #will not be interpreted while parsing
string
      'init sqs_selection=[] ',
      'init sqs_cond="" ',
      'sqs  ::=  SELECT selection FROM fileid WHERE condition ',
      '                                   @sqs_fileid="$fileid"',
      'selection::=  col0 cols*      ',
      'col0     ::=  col                  @sqs_selection.append("$col")',
      'cols     ::=  "," col              @sqs_selection.append("$col")',
      'col      ::=  r"[A-Za-z0-9]"*   ',

        'fileid::= r"\S"* ',

      'condition ::=  cond1  cond2* ',
      'cond2     ::=  op_or  cond1  ',
      'op_or     ::=  OR                  @sqs_cond+=" or "',
      'cond1     ::=  cond3 cond4*  ',
      'cond4     ::=  op_and  cond3 ',
      'op_and    ::=  AND                 @sqs_cond+=" and "',
      'cond3     ::=  column operator value  ',
      '@sqs_cond+= "process(data,cols,\'$column\',\'$operator\'," ',
      '@if sqs_valtype==1:',
      '@   sqs_cond+="\'"+sqs_value+"\'"',
      '@else:',
      '@   sqs_cond+=str(sqs_value)',
      '@sqs_cond+=")"',
      '           |   leftpar condition rightpar  ',
      '           |   condition ',
      'column   ::=  r"[A-Za-z0-9]"*  ',
      'leftpar   ::=  "("               @sqs_cond+=" ( " ',
      'rightpar  ::=  ")"               @sqs_cond+=" ) " ',
      'operator  ::=  LIKE ',
      '           | "=" ',
      '           | "<=" ',
      '           | ">=" ',
      '           | "<" ',
      '           | ">" ',
      'value     ::=  "\'" car* "\'"        @sqs_valtype=1
@sqs_value="$car" ',
      '           |   numeric             @sqs_valtype=2
@sqs_value=$numeric ',
      'car       ::=  r"[^\']" ',
      'numeric   ::=  r"[0-9]"* ' ]
parms=' '.join(sys.argv[1:])

cmp=rp.match(rule,parms)
if cmp==None:
      print "Error in parsing:"
else:
    #
    #cmp.code contains the complete lines of code of the parsing
    #so, we need to exec it in order to have values set.
    #After that, to access values, no need to prefix with cmp.
    """
    print '----- the code -----'
    print cmp.code
    print '--------------------'
    """
    exec(cmp.code)
    id=None
    try:
        id=open(sqs_fileid)
        #
        #Read the first two lines that contain columns name & length
        #and prepare dictionary 'columns'
        colnames=[x for x in id.readline()[:-1].split(' ') if x!=""]
        collengths=id.readline()[:-1].split(' ')
        deb=0
        cols={}
        for i,name in enumerate(colnames):
           colsize=len(collengths[i])
           cols[name]=(deb,colsize+deb)
           deb+=colsize+1
        #
        #now loop on lines until end of file
        for data in id.readlines():
            """
            print '---- sqs_cond -----'
            print sqs_cond
            print '-------------------'
            """
            exec("found="+sqs_cond)
            if found:
                txt=''
                for col in sqs_selection:
                    c=cols[col]
                    txt+=data[c[0]:c[1]]+' '
                print txt
    except Exception,e:
            print e
    else:
        if id!=None: id.close()
```

sqs8.py "select student,exam1,exam4 from students.txt where exam1>80 and exam4<100"

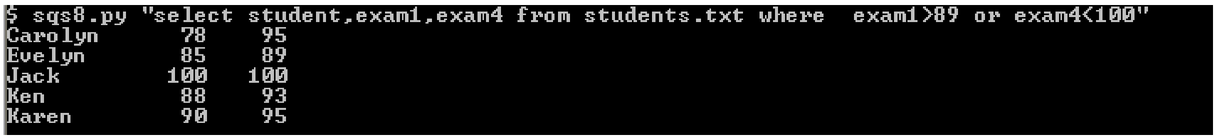


```
$ sqs8.py "select student,exam1,exam4 from students.txt where exam1>80  and exam4<100"
Evelyn        85     89
Ken           88     93
Karen         90     95
```

for that sample, we can display detail about run:

```
$ sqs8.py "select student,exam1,exam4 from students.txt where exam1>80  and exam4<100"
----- the code -----
sqs_selection=[]
sqs_cond=""
sqs_selection.append("student")
sqs_selection.append("exam1")
sqs_selection.append("exam4")
sqs_valtype=2
sqs_value=80
sqs_cond+= "process(data,cols,'exam1','>',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_cond+=" and "
sqs_valtype=2
sqs_value=100
sqs_cond+= "process(data,cols,'exam4','<',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_fileid="students.txt"

--------------------
..... sqs_cond .....
process(data,cols,'exam1','>',80) and process(data,cols,'exam4','<',100)
....................
..... sqs_cond .....
process(data,cols,'exam1','>',80) and process(data,cols,'exam4','<',100)
....................
Evelyn       85    89
..... sqs_cond .....
process(data,cols,'exam1','>',80) and process(data,cols,'exam4','<',100)
....................
..... sqs_cond .....
process(data,cols,'exam1','>',80) and process(data,cols,'exam4','<',100)
....................
Ken          88    93
..... sqs_cond .....
process(data,cols,'exam1','>',80) and process(data,cols,'exam4','<',100)
....................
Karen        90    95
..... sqs_cond .....
process(data,cols,'exam1','>',80) and process(data,cols,'exam4','<',100)
....................
```

sqs8.py "select student,exam1,exam4 from students.txt where exam1>89 or exam4<100"

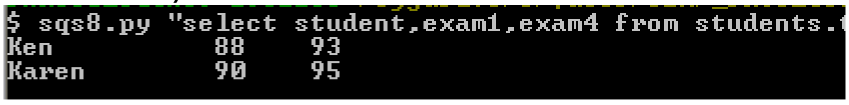


```
$ sqs8.py "select student,exam1,exam4 from students.txt where  exam1>89 or exam4<100"
Carolyn       78    95
Evelyn        85    89
Jack         100   100
Ken           88    93
Karen         90    95
```

the detail about process is:

```
$  sqs8.py "select student,exam1,exam4 from students.txt where  exam1>89 or exam4<100"
----- the code -----
sqs_selection=[]
sqs_cond=""
sqs_selection.append("student")
sqs_selection.append("exam1")
sqs_selection.append("exam4")
sqs_valtype=2
sqs_value=89
sqs_cond+= "process(data,cols,'exam1','>',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_cond+=" or "
sqs_valtype=2
sqs_value=100
sqs_cond+= "process(data,cols,'exam4','<',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_fileid="students.txt"

--------------------
..... sqs_cond .....
process(data,cols,'exam1','>',89) or process(data,cols,'exam4','<',100)
....................
Carolyn      78    95
..... sqs_cond .....
process(data,cols,'exam1','>',89) or process(data,cols,'exam4','<',100)
....................
Evelyn       85    89
..... sqs_cond .....
process(data,cols,'exam1','>',89) or process(data,cols,'exam4','<',100)
....................
Jack        100   100
..... sqs_cond .....
process(data,cols,'exam1','>',89) or process(data,cols,'exam4','<',100)
....................
Ken          88    93
..... sqs_cond .....
process(data,cols,'exam1','>',89) or process(data,cols,'exam4','<',100)
....................
Karen        90    95
..... sqs_cond .....
process(data,cols,'exam1','>',89) or process(data,cols,'exam4','<',100)
....................
```

sqs8.py "select student,exam1,exam4 from students.txt where student like 'K%' and (exam1>89 or exam4<100)"



```
$ sqs8.py "select student,exam1,exam4 from students.
Ken           88     93
Karen         90     95
```

detail is:

```
$ sqs8.py "select student,exam1,exam4 from students.txt where student like 'K%' and ( exam1>89
or exam4<100 )"
----- the code -----
sqs_selection=[]
sqs_cond=""
sqs_selection.append("student")
sqs_selection.append("exam1")
sqs_selection.append("exam4")
sqs_valtype=1
sqs_value="K%"
```

```
sqs_cond+= "process(data,cols,'student','like',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_cond+=" and "
sqs_cond+=" ( "
sqs_valtype=2
sqs_value=89
sqs_cond+= "process(data,cols,'exam1','>',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_cond+=" or "
sqs_valtype=2
sqs_value=100
sqs_cond+= "process(data,cols,'exam4','<',"
if sqs_valtype==1:
   sqs_cond+="'"+sqs_value+"'"
else:
   sqs_cond+=str(sqs_value)
sqs_cond+=")"
sqs_cond+=" ) "
sqs_fileid="students.txt"

--------------------
..... sqs_cond .....
process(data,cols,'student','like','K%') and  ( process(data,cols,'exam1','>',89) or
process(data,cols,'exam4','<',100) )
....................
..... sqs_cond .....
process(data,cols,'student','like','K%') and  ( process(data,cols,'exam1','>',89) or
process(data,cols,'exam4','<',100) )
....................
..... sqs_cond .....
process(data,cols,'student','like','K%') and  ( process(data,cols,'exam1','>',89) or
process(data,cols,'exam4','<',100) )
....................
..... sqs_cond .....
process(data,cols,'student','like','K%') and  ( process(data,cols,'exam1','>',89) or
process(data,cols,'exam4','<',100) )
....................
Ken           88    93
..... sqs_cond .....
process(data,cols,'student','like','K%') and  ( process(data,cols,'exam1','>',89) or
process(data,cols,'exam4','<',100) )
....................
Karen         90    95
..... sqs_cond .....
process(data,cols,'student','like','K%') and  ( process(data,cols,'exam1','>',89) or
process(data,cols,'exam4','<',100) )
....................
```

Rule definition & keywords

A rule is coded in a list, each item of the list consists on:

- ✓ a rule definition, in the forms:
  - o *name* **::=** *definitions* *@line_of_code*
  - o | *definitions* *@line_of_code*
  - o *@line_of_code*

- ✓ a special directive as:
  - o **init** *line_of_code*
  - o **post** *line_of_code*
  - o **import** *fileid*

  - **option verbose**
  - **option blanks**
  - **option norun**
  - **option sepcode** *char*

- ✓ a comment as:
  - *#line of comment*

Where:

- ✓ *name* is a name of a rule and composed with alphanumeric + _

- ✓ *definitions* is a list of rules, terminals, regular expression, such as:
  - *rule* rule that will be defined in next item in the list
  - **TERMInal** a terminal (for which abbreviation **TERMI** is allowed)
  - **r"***exp***"** a regular expression *exp*
  - **"*"** or **"|"** or any special character enclosed in **"**

- ✓ each word in definitions can be enforced with special character:
  - ? to indicate 0 or 1 time
  - + to indicate at least 1 time and more
  - * to indicate 0 to n times

- ✓ *line_of_code* is a python statement that will be interpreted during parsing, only if the associated rule is successfully passed.

- ✓ **init** *line_of_code* corresponds to code that will be interpreted at the beginning of the parsing

- ✓ **post** *line_of_code* corresponds to code that will be interpreted at the end of the parsing

- ✓ **import** *fileid* allows to import content of fileid as rule items to append.

- ✓ **option verbose** allows to display phases parsing, that is necessary for debugging rule definitions, and tune them

- ✓ **option blanks** allows to take blanks into account during parsing
  In that case, you MUST define a rule such as:
  blank ::= r"\s" and use this rule in your definitions.

- ✓ **option norun** tells rp to not interpret code allowing developer to make the interpretation in his process, when he wants.
  (see samples to use this option)

- ✓ **option sepcode** *char* allows you to define a character in place of '@' to specify line_of_code.

You can use this option, is you need to use ‘@’ in your definitions.

Code definitions

Corresponding to each rule, you can define code to be interpreted (or not), in order to make process during the parsing.
Code can be specified on more than one line, ex:

```
rule   ::=  n “-“ m   @rule_type=1
                      @rule_n=$n    @rule_m=$m
         |   n “.” m   @rule_type=2
                      @rule_n=$n    @rule_m=$m
n      ::=  r”[0-9]”*
m      ::=  r”[0-9]”*
```

If you need to define a process with indentations, you can code as:

```
@if toto==None:
@   print ‘it is null..’
@else:
@   print ‘it is ok, now!’
```

(in that case, you have to specify ‘option norun’ in the rule, in order to tell rp to not interpret individual lines of code, else you encounter an error)

Use of option verbose

The verbose option allows you to display detail about parsing…
see following:
(‘Parsing ‘ means start of a rule parsing)
(‘<Parsed’ means parsing is ok for the rule)

```
$ sqs8.py "select student,exam1,exam4 from students.txt where exam1>80  and exam4<100"
Parsing rule: "sqs" for string:"select student,exam1,exam4 from students.txt where exam1>80
and exam4<100"
.Parsing terminal:"SELECT" for string:"select student,exam1,exam4 from students.txt where
exam1>80  and exam4<100"
<Parsed terminal :"SELECT" - value="select"
.Parsing rule: "selection" for string:" student,exam1,exam4 from students.txt where exam1>80
and exam4<100"
..Parsing rule: "col0" for string:"student,exam1,exam4 from students.txt where exam1>80  and
exam4<100"
...Parsing rule: "col" for string:"student,exam1,exam4 from students.txt where exam1>80  and
exam4<100"
....Parsing regular:"[A-Za-z0-9]" for string:"student,exam1,exam4 from students.txt where
exam1>80  and exam4<100"
<<<<Parsed regular :"[A-Za-z0-9]" - value="student"
<<<Parsed rule :"col" - value="student"
<<Parsed rule :"col0" - value="student"
..Parsing rule: "cols" for string:",exam1,exam4 from students.txt where exam1>80  and
exam4<100"
...Parsing terminal:"," for string:",exam1,exam4 from students.txt where exam1>80  and
exam4<100"
<<<Parsed terminal :"," - value=","
...Parsing rule: "col" for string:"exam1,exam4 from students.txt where exam1>80  and
exam4<100"
....Parsing regular:"[A-Za-z0-9]" for string:"exam1,exam4 from students.txt where exam1>80
and exam4<100"
```

```
<<<<Parsed regular :"[A-Za-z0-9]" - value="exam1"
<<<Parsed rule :"col" - value="exam1"
...Parsing terminal:"," for string:",exam4 from students.txt where exam1>80  and exam4<100"
<<<Parsed terminal :"," - value=","
...Parsing rule: "col" for string:"exam4 from students.txt where exam1>80  and exam4<100"
....Parsing regular:"[A-Za-z0-9]" for string:"exam4 from students.txt where exam1>80  and
exam4<100"
<<<<Parsed regular :"[A-Za-z0-9]" - value="exam4"
<<<Parsed rule :"col" - value="exam4"
...Parsing terminal:"," for string:" from students.txt where exam1>80  and exam4<100"
...Parsed terminal :"," - value=""
<<Parsed rule :"cols" - value=",exam1,exam4"
<Parsed rule :"selection" - value="student,exam1,exam4"
.Parsing terminal:"FROM" for string:" from students.txt where exam1>80  and exam4<100"
<Parsed terminal :"FROM" - value="from"
.Parsing rule: "fileid" for string:" students.txt where exam1>80  and exam4<100"
..Parsing regular:"\S" for string:"students.txt where exam1>80  and exam4<100"
<<Parsed regular :"\S" - value="students.txt"
<Parsed rule :"fileid" - value="students.txt"
.Parsing terminal:"WHERE" for string:" where exam1>80  and exam4<100"
<Parsed terminal :"WHERE" - value="where"
.Parsing rule: "condition" for string:" exam1>80  and exam4<100"
..Parsing rule: "cond1" for string:"exam1>80  and exam4<100"
...Parsing rule: "cond3" for string:"exam1>80  and exam4<100"
....Parsing rule: "column" for string:"exam1>80  and exam4<100"
.....Parsing regular:"[A-Za-z0-9]" for string:"exam1>80  and exam4<100"
<<<<<Parsed regular :"[A-Za-z0-9]" - value="exam1"
<<<<Parsed rule :"column" - value="exam1"
....Parsing rule: "operator" for string:">80  and exam4<100"
.....Parsing terminal:"LIKE" for string:">80  and exam4<100"
.....Parsed terminal :"LIKE" - value=""
.....Parsing terminal:"=" for string:">80  and exam4<100"
.....Parsed terminal :"=" - value=""
.....Parsing terminal:"<=" for string:">80  and exam4<100"
.....Parsed terminal :"<=" - value=""
.....Parsing terminal:">=" for string:">80  and exam4<100"
.....Parsed terminal :">=" - value=""
.....Parsing terminal:"<" for string:">80  and exam4<100"
.....Parsed terminal :"<" - value=""
.....Parsing terminal:">" for string:">80  and exam4<100"
<<<<<Parsed terminal :">" - value=">"
<<<<Parsed rule :"operator" - value=">"
....Parsing rule: "value" for string:"80  and exam4<100"
.....Parsing terminal:"'" for string:"80  and exam4<100"
.....Parsed terminal :"'" - value=""
.....Parsing rule: "numeric" for string:"80  and exam4<100"
......Parsing regular:"[0-9]" for string:"80  and exam4<100"
<<<<<<Parsed regular :"[0-9]" - value="80"
<<<<<Parsed rule :"numeric" - value="80"
<<<<Parsed rule :"value" - value="80"
<<<Parsed rule :"cond3" - value="exam1>80"
...Parsing rule: "cond4" for string:"  and exam4<100"
....Parsing rule: "op_and" for string:"and exam4<100"
.....Parsing terminal:"AND" for string:"and exam4<100"
<<<<<Parsed terminal :"AND" - value="and"
<<<<Parsed rule :"op_and" - value="and"
....Parsing rule: "cond3" for string:" exam4<100"
.....Parsing rule: "column" for string:"exam4<100"
......Parsing regular:"[A-Za-z0-9]" for string:"exam4<100"
<<<<<<Parsed regular :"[A-Za-z0-9]" - value="exam4"
<<<<<Parsed rule :"column" - value="exam4"
.....Parsing rule: "operator" for string:"<100"
......Parsing terminal:"LIKE" for string:"<100"
......Parsed terminal :"LIKE" - value=""
......Parsing terminal:"=" for string:"<100"
......Parsed terminal :"=" - value=""
......Parsing terminal:"<=" for string:"<100"
......Parsed terminal :"<=" - value=""
......Parsing terminal:">=" for string:"<100"
......Parsed terminal :">=" - value=""
......Parsing terminal:"<" for string:"<100"
<<<<<<Parsed terminal :"<" - value="<"
<<<<<Parsed rule :"operator" - value="<"
.....Parsing rule: "value" for string:"100"
......Parsing terminal:"'" for string:"100"
......Parsed terminal :"'" - value=""
......Parsing rule: "numeric" for string:"100"
```

```
.......Parsing regular:"[0-9]" for string:"100"
<<<<<<<Parsed regular :"[0-9]" - value="100"
<<<<<<Parsed rule :"numeric" - value="100"
<<<<<Parsed rule :"value" - value="100"
<<<<Parsed rule :"cond3" - value="exam4<100"
....Parsing rule: "op_and" for string:""
.....Parsing terminal:"AND" for string:""
.....Parsed terminal :"AND" - value=""
....Parsed rule :"op_and" - value=""
<<<Parsed rule :"cond4" - value="and exam4<100"
<<Parsed rule :"cond1" - value="exam1>80  and exam4<100"
..Parsing rule: "cond2" for string:""
...Parsing rule: "op_or" for string:""
....Parsing terminal:"OR" for string:""
....Parsed terminal :"OR" - value=""
...Parsed rule :"op_or" - value=""
..Parsed rule :"cond2" - value=""
<Parsed rule :"condition" - value="exam1>80  and exam4<100"
Parsed rule :"sqs" - value="select student,exam1,exam4 from students.txt where exam1>80  and
exam4<100"
```

In case of bad parsing, the parser show where is located the error, as:

```
<<Parsed rule :"cond1" - value="exam1>80  "
..Parsing rule: "cond2" for string:"exam4<100"
...Parsing rule: "op_or" for string:"exam4<100"
....Parsing terminal:"OR" for string:"exam4<100"
....Parsed terminal :"OR" - value=""
...Parsed rule :"op_or" - value=""
..Parsed rule :"cond2" - value=""
<Parsed rule :"condition" - value="exam1>80  "
Parsed rule :"sqs" - value="select student,exam1,exam4 from students.txt where exam1>80  "
select student,exam1,exam4 from students.txt where exam1>80  exam4<100
-------------------------------------------------------------^
```

in which, ‘and’ is missing.