Content delivered in class_16_25-August-2016

- Chapter 11: Debugging in Python
  - IDE-based debugging
    - pydev
  - Interactive-mode based debugging
    - pdb
    - Working with pdb, within the python scripts
- Chapter 15: Regular Expressions
  - Simple Character Matches
    - re.match()
    - re.search()
  - Special Characters
  - The Dot Character
  - Greedy Matches
  - Grouping
    - Perl based grouping
  - Matching at Beginning or End
    - ^ and $
  - re.findall()
  - re.finditer()
  - Compilation Flags
  - re.ASCII() vs re.UNICODE() vs re.LOCALE()
  - re.verbose()
  - Modifying Strings with re module
  - Summarizing all the Regular Expressions

Interview Questions Discussed:

Interview Question 1: Write a regular expression, to match all email IDs?

Interview Question 2: What is the difference between re.search() and re.match()?

Assignments Given:

Assignment 1: Try to find the dug, in this buggyFifth.py script.

Assignment 2: Analyze the script execution in this fifth.py

Assignment 3: In this sixth.py, Observe the call flow of function calls.

Assignment 4: Identify the email IDs in the resume in the following paragraph.

Debugger in Python

Debugging is essential to understand the call flow in run time. It is also used to debug an error.

There are various debuggers available in python, as listed here (https://wiki.python.org/moin/PythonDebuggingTools). These debuggers can be used in

1. Interactive mode, or
2. within the IDE

In all the cases, it is not feasible to work with an IDE. ex: Devops jobs of logging into a remote linux/Windows server, to solve an issue. Especially, in such cases, interactive debuggers such as pdb, ipdb, ... can be helpful.

pydev

- Majority of the IDEs will have pydev configured in it.
- Basic features of pydev are
  - step Over
  - step In
  - step Out
  - Placing debugger
  - Visualizing the objects in debug mode ....

pdb

- It is the basic interactive Debugger
- Features
  - pause a program
  - look at the values of variables
  - watch program execution step-by-step

Usage:

1. In the terminal :

   $ python -m pdb fileName.py
2. Within the script using pdb module

   import pdb

In the script, place the following statement from where we want to make debugging.

```
pdb.set_trace()
```

Now, run the program normally

At the output console, it returns pdb prompt, and wait for the commands.

```
(pdb)  l          # To list the complete script, irrespective of the line in which
 the set_trace() statement is placed.
(pdb)  l 1,5     # To list 1st line to 5th line (including) irrespective of place i
n which set_trace() statement is placed.
(pdb)  c        # Continue execution till a break point is encountered
```

It is recommended not to place set_trace() within a loop (for, while, ..)

```
(pdb) help        # To get help find all the options possible with pdb
(pdb) n         # Continue execution until the next line in current function is rea
ched or it returns
(pdb) r         # continue execution, until the current function returns
(pdb) u         # shows the flow. Shows previous step in execution flow. but, it wi
ll not execute
(pdb) d         # shows next step in execution flow.
```

Most commonly used among them are:

```
l(ist)
n(ext)
c(ontinue)
s(tep)
r(eturn)
b(reak)
```

```
pdb ? - to get help

pdb l - show the cursor position

pdb l 18 - to list line 18 in file
pdb passesLeft  - to get the number of passes left

pdb <any variable> - to get the value in variable

pdb b 18 - to place a breakpoint
pdb l

pdb n - to execute next step
```

pdb comes with four modes of operation:

```
- Script, postmortem, run and Trace modes
```

Working with pdb, within the python scripts

save the below scripts, and execute them in terminal, like general python scripts.

In [14]:
```
#!/usr/bin/python
# first.py
import pdb
version = '3.0'
def hello():
  ''' this is just for printing hello '''
  print "hello today is modules class"
def add(a=1,b=2):
  ''' this is an addition program '''
  sum = a + b
  return sum


pdb.set_trace()
if __name__ == '__main__':
  hello()
  sum = add()
  sum1 = add(5,6)
  print sum1
  print sum
  print version
else:
  print "Hello this has to be part of my modules"
```

```
--Return--
> <ipython-input-14-ca0a9d73001e>(14)<module>()->None
-> pdb.set_trace()
(Pdb) l
  9       ''' this is an addition program '''
 10       sum = a + b
 11       return sum
 12
 13
 14  -> pdb.set_trace()
 15     if __name__ == '__main__':
 16       hello()
 17       sum = add()
 18       sum1 = add(5,6)
 19       print sum1
(Pdb) l 9 25
*** Error in argument: '9 25'
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2872)run_cod
e()
-> sys.excepthook = old_excepthook
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2888)run_cod
e()
-> outflag = 0
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2889)run_cod
e()
-> return outflag
(Pdb) n
--Return--
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2889)run_cod
e()->0
-> return outflag
(Pdb) c
hello today is modules class
11
3
3.0
```

In [2]:
```
#!/usr/bin/python
# second.py
import pdb
pdb.set_trace()
import first as f
print f.hello()
print f.add(10,20)
print f.version
```

```
--Return--
> <ipython-input-2-6c3a48b9f2b1>(4)<module>()->None
-> pdb.set_trace()
(Pdb) ?

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pp       run      unt
a      c          continue  exit    l     q        s        until
alias  cl         d         h       list  quit     step     up
args   clear      debug     help    n     r        tbreak   w
b      commands   disable   ignore  next  restart  u        whatis
break  condition  down      j       p     return   unalias  where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv

(Pdb) h

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pp       run      unt
a      c          continue  exit    l     q        s        until
alias  cl         d         h       list  quit     step     up
args   clear      debug     help    n     r        tbreak   w
b      commands   disable   ignore  next  restart  u        whatis
break  condition  down      j       p     return   unalias  where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv

(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2872)run_cod
e()
-> sys.excepthook = old_excepthook
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2888)run_cod
e()
-> outflag = 0
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2889)run_cod
e()
-> return outflag
(Pdb) n
--Return--
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2889)run_cod
e()->0
```

```
-> return outflag
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2806)run_ast
_nodes()
-> for i, node in enumerate(to_run_exec):
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2807)run_ast
_nodes()
-> mod = ast.Module([node])
(Pdb) n 2
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2808)run_ast
_nodes()
-> code = compiler(mod, cell_name, "exec")
(Pdb) n 5
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2809)run_ast
_nodes()
-> if self.run_code(code, result):
(Pdb) n 5

---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-2-6c3a48b9f2b1> in <module>()
      3 import pdb
      4 pdb.set_trace()
----> 5 import first as f
      6 print f.hello()
      7 print f.add(10,20)

ImportError: No module named first

> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2810)run_ast
_nodes()
-> return True
(Pdb) q
```

```
---------------------------------------------------------------------------
BdbQuit                                   Traceback (most recent call last)
c:\python27\lib\site-packages\IPython\core\interactiveshell.pyc in run_ast_no
des(self, nodelist, cell_name, interactivity, compiler, result)
   2808                 code = compiler(mod, cell_name, "exec")
   2809                 if self.run_code(code, result):
-> 2810                     return True
   2811
   2812             for i, node in enumerate(to_run_interactive):

c:\python27\lib\bdb.pyc in trace_dispatch(self, frame, event, arg)
     47             return # None
     48         if event == 'line':
---> 49             return self.dispatch_line(frame)
     50         if event == 'call':
     51             return self.dispatch_call(frame, arg)

c:\python27\lib\bdb.pyc in dispatch_line(self, frame)
     66         if self.stop_here(frame) or self.break_here(frame):
     67             self.user_line(frame)
---> 68             if self.quitting: raise BdbQuit
     69         return self.trace_dispatch
     70

BdbQuit:
```

Note: You can observe some code which is getting executed here, though it is not part of our second.py. It is beacuse of ipython.

Also, with this debugging, we clearly identified that 'first.py' is not located in the current working directory.

Now, first.py is placed in cwd. Re-evaluating it.

In [11]:
```
#!/usr/bin/python
# second.py
import pdb
pdb.set_trace()
import first as f
print f.hello()
print f.add(10,20)
print f.version
```

```
--Return--
> <ipython-input-11-6c3a48b9f2b1>(4)<module>()->None
-> pdb.set_trace()
(Pdb) l 3
  1     #!/usr/bin/python
  2     # second.py
  3     import pdb
  4  -> pdb.set_trace()
  5     import first as f
  6     print f.hello()
  7     print f.add(10,20)
  8     print f.version
[EOF]
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2872)run_cod
e()
-> sys.excepthook = old_excepthook
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2888)run_cod
e()
-> outflag = 0
(Pdb) c
> c:\users\upethakamsetty\google drive\python\tut\kt group batch 2\class_16-2
5-august-2016\first.py(15)<module>()
-> if __name__ == '__main__':
(Pdb) n
> c:\users\upethakamsetty\google drive\python\tut\kt group batch 2\class_16-2
5-august-2016\first.py(23)<module>()
-> print "Hello this has to be part of my modules"
(Pdb) n
Hello this has to be part of my modules
--Return--
> c:\users\upethakamsetty\google drive\python\tut\kt group batch 2\class_16-2
5-august-2016\first.py(23)<module>()->None
-> print "Hello this has to be part of my modules"
(Pdb) c
hello today is modules class
None
30
3.0
```

In this third.py, you can observe the functionality of for loop, and how each for is getting executed.

In [12]:
```
#!/usr/bin/python
# third.py
import pdb

for i in (1,2,3,4,5,6,7):
  pdb.set_trace()
  print i
```

```
> <ipython-input-12-50f7b39ec9dd>(7)<module>()->None
-> print i
(Pdb) n
1
> <ipython-input-12-50f7b39ec9dd>(5)<module>()->None
-> for i in (1,2,3,4,5,6,7):
(Pdb) n
> <ipython-input-12-50f7b39ec9dd>(6)<module>()->None
-> pdb.set_trace()
(Pdb) n
> <ipython-input-12-50f7b39ec9dd>(7)<module>()->None
-> print i
(Pdb) n
2
> <ipython-input-12-50f7b39ec9dd>(5)<module>()->None
-> for i in (1,2,3,4,5,6,7):
(Pdb) n
> <ipython-input-12-50f7b39ec9dd>(6)<module>()->None
-> pdb.set_trace()
(Pdb) n
> <ipython-input-12-50f7b39ec9dd>(7)<module>()->None
-> print i
(Pdb) n
3
> <ipython-input-12-50f7b39ec9dd>(5)<module>()->None
-> for i in (1,2,3,4,5,6,7):
(Pdb) c
> <ipython-input-12-50f7b39ec9dd>(7)<module>()->None
-> print i
(Pdb) c
4
> <ipython-input-12-50f7b39ec9dd>(6)<module>()->None
-> pdb.set_trace()
(Pdb) c
5
> <ipython-input-12-50f7b39ec9dd>(7)<module>()->None
-> print i
(Pdb) c
6
> <ipython-input-12-50f7b39ec9dd>(6)<module>()->None
-> pdb.set_trace()
(Pdb) c
7
```

Similarly, in fourth.py, you can observe the functionality of for loop and if condition, in runtime.

In [13]:
```
#!/usr/bin/python
# fourth.py
import pdb

for i in range(1,11):
  if i == 5:
    pdb.set_trace()
    continue
  print i
```

```
1
2
3
4
> <ipython-input-13-f1378680eab5>(8)<module>()->None
-> continue
(Pdb) l
  3     import pdb
  4
  5     for i in range(1,11):
  6       if i == 5:
  7         pdb.set_trace()
  8  ->     continue
  9       print i
[EOF]
(Pdb) n
> <ipython-input-13-f1378680eab5>(5)<module>()->None
-> for i in range(1,11):
(Pdb) n
> <ipython-input-13-f1378680eab5>(6)<module>()->None
-> if i == 5:
(Pdb) n
> <ipython-input-13-f1378680eab5>(9)<module>()->None
-> print i
(Pdb) n
6
> <ipython-input-13-f1378680eab5>(5)<module>()->None
-> for i in range(1,11):
(Pdb) n
> <ipython-input-13-f1378680eab5>(6)<module>()->None
-> if i == 5:
(Pdb) n
> <ipython-input-13-f1378680eab5>(9)<module>()->None
-> print i
(Pdb) n
7
> <ipython-input-13-f1378680eab5>(5)<module>()->None
-> for i in range(1,11):
(Pdb) n
> <ipython-input-13-f1378680eab5>(6)<module>()->None
-> if i == 5:
(Pdb) n
> <ipython-input-13-f1378680eab5>(9)<module>()->None
-> print i
(Pdb) w
  c:\python27\lib\runpy.py(174)_run_module_as_main()
-> "__main__", fname, loader, pkg_name)
  c:\python27\lib\runpy.py(72)_run_code()
-> exec code in run_globals
  c:\python27\lib\site-packages\ipykernel\__main__.py(3)<module>()
-> app.launch_new_instance()
  c:\python27\lib\site-packages\traitlets\config\application.py(596)launch_in
stance()
-> app.start()
  c:\python27\lib\site-packages\ipykernel\kernelapp.py(442)start()
-> ioloop.IOLoop.instance().start()
  c:\python27\lib\site-packages\zmq\eventloop\ioloop.py(162)start()
```

```
-> super(ZMQIOLoop, self).start()
  c:\python27\lib\site-packages\tornado\ioloop.py(887)start()
-> handler_func(fd_obj, events)
  c:\python27\lib\site-packages\tornado\stack_context.py(275)null_wrapper()
-> return fn(*args, **kwargs)
  c:\python27\lib\site-packages\zmq\eventloop\zmqstream.py(440)_handle_events
()
-> self._handle_recv()
  c:\python27\lib\site-packages\zmq\eventloop\zmqstream.py(472)_handle_recv()
-> self._run_callback(callback, msg)
  c:\python27\lib\site-packages\zmq\eventloop\zmqstream.py(414)_run_callback
()
-> callback(*args, **kwargs)
  c:\python27\lib\site-packages\tornado\stack_context.py(275)null_wrapper()
-> return fn(*args, **kwargs)
  c:\python27\lib\site-packages\ipykernel\kernelbase.py(276)dispatcher()
-> return self.dispatch_shell(stream, msg)
  c:\python27\lib\site-packages\ipykernel\kernelbase.py(228)dispatch_shell()
-> handler(stream, idents, msg)
  c:\python27\lib\site-packages\ipykernel\kernelbase.py(391)execute_request()
-> user_expressions, allow_stdin)
  c:\python27\lib\site-packages\ipykernel\ipkernel.py(199)do_execute()
-> shell.run_cell(code, store_history=store_history, silent=silent)
  c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2705)run_cel
l()
-> interactivity=interactivity, compiler=compiler, result=result)
  c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2809)run_ast
_nodes()
-> if self.run_code(code, result):
  c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2869)run_cod
e()
-> exec(code_obj, self.user_global_ns, self.user_ns)
> <ipython-input-13-f1378680eab5>(9)<module>()->None
-> print i
(Pdb) where
  c:\python27\lib\runpy.py(174)_run_module_as_main()
-> "__main__", fname, loader, pkg_name)
  c:\python27\lib\runpy.py(72)_run_code()
-> exec code in run_globals
  c:\python27\lib\site-packages\ipykernel\__main__.py(3)<module>()
-> app.launch_new_instance()
  c:\python27\lib\site-packages\traitlets\config\application.py(596)launch_in
stance()
-> app.start()
  c:\python27\lib\site-packages\ipykernel\kernelapp.py(442)start()
-> ioloop.IOLoop.instance().start()
  c:\python27\lib\site-packages\zmq\eventloop\ioloop.py(162)start()
-> super(ZMQIOLoop, self).start()
  c:\python27\lib\site-packages\tornado\ioloop.py(887)start()
-> handler_func(fd_obj, events)
  c:\python27\lib\site-packages\tornado\stack_context.py(275)null_wrapper()
-> return fn(*args, **kwargs)
  c:\python27\lib\site-packages\zmq\eventloop\zmqstream.py(440)_handle_events
()
-> self._handle_recv()
  c:\python27\lib\site-packages\zmq\eventloop\zmqstream.py(472)_handle_recv()
-> self._run_callback(callback, msg)
```

```
  c:\python27\lib\site-packages\zmq\eventloop\zmqstream.py(414)_run_callback
()
-> callback(*args, **kwargs)
  c:\python27\lib\site-packages\tornado\stack_context.py(275)null_wrapper()
-> return fn(*args, **kwargs)
  c:\python27\lib\site-packages\ipykernel\kernelbase.py(276)dispatcher()
-> return self.dispatch_shell(stream, msg)
  c:\python27\lib\site-packages\ipykernel\kernelbase.py(228)dispatch_shell()
-> handler(stream, idents, msg)
  c:\python27\lib\site-packages\ipykernel\kernelbase.py(391)execute_request()
-> user_expressions, allow_stdin)
  c:\python27\lib\site-packages\ipykernel\ipkernel.py(199)do_execute()
-> shell.run_cell(code, store_history=store_history, silent=silent)
  c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2705)run_cel
l()
-> interactivity=interactivity, compiler=compiler, result=result)
  c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2809)run_ast
_nodes()
-> if self.run_code(code, result):
  c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2869)run_cod
e()
-> exec(code_obj, self.user_global_ns, self.user_ns)
> <ipython-input-13-f1378680eab5>(9)<module>()->None
-> print i
(Pdb) whois
*** NameError: name 'whois' is not defined
(Pdb) whatis
*** SyntaxError: SyntaxError('unexpected EOF while parsing', ('<string>', 0,
 0, ''))
(Pdb) whatis i
<type 'int'>
(Pdb) c
8
9
10
```

Now, let try to debug a script with some bugs.

Assignment 1: Try to find the dug, in this buggyFifth.py script.

```
#!/usr/bin/python
# buggyFifth.py
def eo(num):
  ''' fun for even odd numbers '''
  if num % 2 == 0:
    return 'even'
  else:
    return 'odd'

import pdb;pdb.set_trace()
print eo(2)
print eo(3)
print eo.func_doc
```

Assignment 2: Analyze the script execution in this fifth.py

```
#!/usr/bin/python
# fifth.py
def eo(num):
  ''' fun for even odd numbers '''
  if num % 2 == 0:
    print 'even'
  else:
    print 'odd'

import pdb
pdb.set_trace()
for i in range(100):
  print "The number1 is %s" %(i)
  print "The number2 is %s" %(i)
  print "The number3 is %s" %(i)
  print "The number4 is %s" %(i)
  print "The number5 is %s" %(i)
  eo(i)
```

Assignment 3: In this sixth.py, Observe the call flow of function calls.

```
#!/usr/bin/python
# sixth.py
import pdb

def fourth():
  second()
  print "this is my fourth function \n"
def third():
  print "this is my third function \n"
def second():
  third()
  print "this is my second function \n"
def first():
  fourth()
  print "this is my first function \n"

pdb.set_trace()
first()
```

In conclusion, pdb debugging starts from the line where the pdb.set_trace() is placed. It can be more helpful, in cases where using IDE or other GUI based debuggers is not possible. But, there are other debuggers such as pydev, ... which have much more functionality than pdb.

re module

Regular expression (or regex) are used for pattern matching in the date. It is available in almost all programming languages. Python has re module to deal with regular expressions.

In [1]:
```
import re
```

In [2]:
```
print dir(re)
```

```
['DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M', 'MULTILINE', 'S',
 'Scanner', 'T', 'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X', '_MAXCACHE', '__
all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__
version__', '_alphanum', '_cache', '_cache_repl', '_compile', '_compile_rep
l', '_expand', '_locale', '_pattern_type', '_pickle', '_subx', 'compile', 'co
py_reg', 'error', 'escape', 'findall', 'finditer', 'match', 'purge', 'searc
h', 'split', 'sre_compile', 'sre_parse', 'sub', 'subn', 'sys', 'template']
```

In [3]:
```
reg = re.compile("python")     # creating a regular expression Object
```

In [4]:
```
type(reg)
```

Out[4]:
```
_sre.SRE_Pattern
```

In [5]:
```
reg
```

Out[5]:
```
re.compile(r'python')
```

In [6]:
```
print reg
```

```
<_sre.SRE_Pattern object at 0x037C87B0>
```

re.match()

re.match(string[, pos[, endpos]])

- Matches zero or more characters at the beginning of the string

In [9]:
```
reg.match("python Programming is Good")    # The matched result is stored in r
e type object
```

Out[9]:
```
<_sre.SRE_Match at 0x46ef758>
```

In [10]:
```
print reg.match("python Programming is Good")
```

```
<_sre.SRE_Match object at 0x046EF790>
```

In [11]:
```
reg.match("python Programming is Good").group() # returns the matches stored i
n object
```

Out[11]:
```
'python'
```

In [13]:
```
reg.match('Python')  # observe the 'P' capital p
```

In [14]:
```
print reg.match('Python')
```

```
None
```

In [15]:
```
reg.match('Python').group()  # AttributeError came as that attrribute comes on
ly when there is atleast one matched result
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-15-656339889706> in <module>()
----> 1 reg.match('Python').group()  # AttributeError came as that attrribute
 comes only when there is atleast one matched result

AttributeError: 'NoneType' object has no attribute 'group'
```

Until and unless there are matches found, the group() attribute will not be present.

In [16]:
```
print reg.match('pythoN')  # results in None
```

```
None
```

In [17]:
```
print reg.match('')
```

```
None
```

In [22]:
```
#!/usr/bin/python
import os

checkString = 'python'
compiledCheckString = re.compile(checkString)   # re object will be created
matchs = compiledCheckString.match('python')    # match() will be initiated on
 the re object
print matchs
if matchs:
    print "Now lets print the matches word"
    print matchs.group()
```

```
<_sre.SRE_Match object at 0x047CAD78>
Now lets print the matches word
python
```

In [24]:
```
#!/usr/bin/python
import os

checkString = 'python'
compiledCheckString = re.compile(checkString)   # re object will be created
matchs = compiledCheckString.match('Python')    # match() will be initiated on
 the  re object
print matchs
if matchs:
    print "Now lets print the matches word"
    print matchs.group()
else:
    print "No matches found"
```

```
None
No matches found
```

Special Characters

^ (caret) - Matches the start of the string

In [33]:
```
string = "This is python class"
```

In [26]:
```
regObject = re.compile("^This")
```

In [27]:
```
result = regObject.match(string)
```

In [28]:
```
print type(result), type(regObject)
```

```
<type '_sre.SRE_Match'> <type '_sre.SRE_Pattern'>
```

In [29]:
```
print result
```

```
<_sre.SRE_Match object at 0x047CAF00>
```

In [30]:
```
print result.group()
```

```
This
```

Doing in other way

In [34]:
```
re.match('^This',string).group()
```

Out[34]:
```
'This'
```

In [35]:
```
string = "Yes!, This is python class"
```

In [37]:
```
print re.match('^This',string)
```

```
None
```

In [36]:
```
re.match('^This',string).group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-36-3d15a5a1e212> in <module>()
----> 1 re.match('^This',string).group()

AttributeError: 'NoneType' object has no attribute 'group'
```

$ - matches the end of the string, or just before the newline at the end of the string

In [38]:
```
#!/usr/bin/python
import re
string = 'foo foobar'
print string
regObject= re.compile('foobar$')
print "regObject = ",regObject
print regObject.match(string)    # re.match() vs re.search()
```

```
foo foobar
regObject =  <_sre.SRE_Pattern object at 0x04698AE8>
None
```

In [39]:
```
re.match('foobar$',string)  # pattern match only in the starting of line
```

In [40]:
```
print string
```

```
foo foobar
```

re.search()

In [41]:
```
re.search('foobar$',string)  # re.search tries to find the pattern anywhere in
 the string; whereas re.match fetches in the starting of line only
```

Out[41]:
```
<_sre.SRE_Match at 0x47d7330>
```

In [42]:
```
re.search('foobar$',string).group()
```

Out[42]:
```
'foobar'
```

In [43]:
```
#!/usr/bin/python
import re
string = 'foo foobar'
print string
regObject= re.compile('foobar$')
print "regObject = ",regObject
print regObject.search(string)
```

```
foo foobar
regObject =  <_sre.SRE_Pattern object at 0x04698AE8>
<_sre.SRE_Match object at 0x047CAD78>
```

Interview Question 2: What is the difference between re.search() and re.match()?

. (DOT)

- matches any character, except the newline. if DOTALL flag is enabled, it matches any character including newline

In [44]:
```
#!usr/bin/python

string = 'This'
print re.search('....', string)
```

```
<_sre.SRE_Match object at 0x046EF560>
```

In [45]:
```
print re.search('....', string).group()
```

```
This
```

In [46]:
```
print re.search('.....', string).group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-46-0f6879d7b95a> in <module>()
----> 1 print re.search('.....', string).group()

AttributeError: 'NoneType' object has no attribute 'group'
```

In [47]:
```
print re.search('...', string).group()
```

```
Thi
```

* causes the RE to match 0 or more repetitions of the preceeding RE.

```
ex: 'ab*'  - matches for 'a', 'ab', and 'a' followed by any number of 'b''s
                  ab, abb, abbbb ...
```

In [49]:
```
print string
```

```
This
```

In [50]:
```
re.search('.*',string).group()
```

Out[50]:
```
'This'
```

In [51]:
```
string = "This is the correct time to take a decision!"
```

In [52]:
```
re.search('.*',string).group()
```

Out[52]:
```
'This is the correct time to take a decision!'
```

In [53]:
```
re.search('.*', '').group()   # trying to find in null string
```

Out[53]:
```
''
```

+

- causes the RE to match 1 or more repetitions of the preceeding RE

ex: ab+ -> 'ab', 'abb', 'abbb', ...

```
In [55]: string = 'abbba'

In [56]: re.match('ab+a',string).group()
Out[56]: 'abbba'

In [57]: re.search('ab+a',string).group()
Out[57]: 'abbba'

In [58]: re.search('ab+a','aa')

In [59]: re.search('ab*a','aa')   # difference between * and +
Out[59]: <_sre.SRE_Match at 0x47d7800>
```

? causes the RE to match 0 or 1 time of the preceeding RE

ex: ab? --> 'a', 'ab'

```
In [60]: string = 'hello'

In [61]: re.match(r'hello?', string).group()
Out[61]: 'hello'

In [62]: re.match('hello?', 'hell')
Out[62]: <_sre.SRE_Match at 0x47d79f8>

In [63]: re.match('hell?o', 'hell')   # 'l' is repeating 2 times

In [64]: print re.match('hell?o', 'helll')   # 'l' is repeating 3 times

In [65]: print re.match('hell?o', 'he')   # 'o' is missing
         None

In [67]: print re.match('hell?o', 'helo')
         <_sre.SRE_Match object at 0x047D7BF0>
```

Gready Search Patterns

*?, +?, ?? - GREEDY SEARCH Patterns

In [69]:
```
string = '<H1>title</H1>'
```

In [70]:
```
re.match(r'<.*>', string)
```

Out[70]:
```
<_sre.SRE_Match at 0x47d7e90>
```

In [71]:
```
print re.match(r'<.*>', string).group()
```

```
<H1>title</H1>
```

In [73]:
```
print re.match(r'<.*?>', string).group()
```

```
<H1>
```

{m}

- specifies the exactly m copies of previous RE

ex: a{6} -- it maches six 'a' characters

In [74]:
```
string = 'aaashique'
```

In [75]:
```
re.match('a{3}shique', string).group()
```

Out[75]:
```
'aaashique'
```

In [76]:
```
print re.match('a{2}shique', string)
```

```
None
```

In [78]:
```
print re.match('aa{2}shique', string).group()
```

```
aaashique
```

{m,n}

- causes the resulting RE to match from m to n repititions of the preceding RE

ex: a{3,5} will match from 3 to 5 'a' characters

In [79]:
```
string = 'aaashique'
```

In [80]:
```
print re.match('a{2,3}shique', string).group()
```

```
aaashique
```

In [81]:
```
string = 'aashique'
```

In [82]:
```
print re.match('a{2,3}shique', string).group()
```

```
aashique
```

{m,n}? - combined regex pattern

In [83]:
```
string = 'aaaaaa'
```

In [84]:
```
re.match('a{2,3}',string).group()
```

Out[84]:
```
'aaa'
```

In [85]:
```
re.search('a{2,3}',string).group()
```

Out[85]:
```
'aaa'
```

In [86]:
```
re.match('a{2,3}?',string).group()
```

Out[86]:
```
'aa'
```

\ either escapes the special characters (permittin you to match characters like '*', '?') or used to signal a special sequence

In [87]:
```
string = '<H*>test<H*>'
```

In [88]:
```
re.match('<H\*>',string).group()
```

Out[88]:
```
'<H*>'
```

In [90]:
```
string = '<H?>test<H?>'
```

In [91]:
```
re.match('<H\?>',string).group()
```

Out[91]:
```
'<H?>'
```

[]

- used to indicate a set of characters.
- regular expression characters will lose their significance, within the [] (sqauare) braces

ex:

```
[mnk] - will match the characters 'm', 'n' and 'k'

[a-z] - will match all characters from 'a' to 'z'

[A-Z] - will match all characters from 'A' to 'Z'

[0-9] - will match all characters from 0 to 9

[a-m] - will match all characters from 'a' to 'm'
```

```
In [92]: string = 'hello'

In [93]: re.match('h[eE]llo',string).group()
Out[93]: 'hello'

In [94]: string = 'hEllo'

In [95]: re.match('h[eE]llo',string).group()
Out[95]: 'hEllo'

In [96]: re.match('[a-z].*',string).group()
Out[96]: 'hEllo'
```

Note - special characters lose their special meaning inside sets.

To match a literal ']' inside a set, precede it with a backslash, or place it at the beginning of the set. For example, both [()[]{}] and [] () {[]} will match a parenthesis

```
In [97]: string = '<h*>test<h*>'

In [98]: re.match('<h[*]>',string).group()
Out[98]: '<h*>'

In [99]: re.match('<h\*>',string).group()
Out[99]: '<h*>'
```

Interview Question 1: Write a regular expression, to match all email IDs?

Assignment 4: Identify the email IDs in the resume in the following paragraph.

```
Welcome to RegExr v2.1 by gskinner.com, proudly hosted by Media Temple!

Edit the Expression & Text to see matches. Roll over matches or the expression for
 details. Undo mistakes with ctrl-z. Save Favorites & Share expressions with friend
s or the Community. Explore your results with Tools. A full Reference & Help is ava
ilable in the Library, or watch the video Tutorial.
python@programm.com
Sample text for testing:
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 _+-.,!@#$%^&*();\/|<>"'python@gmail.com
12345 -98.7 3.141 .6180 9,000 +42
555.123.4567    +1-(800)-555-2468
foo@demo.net    bar.ba@test.co.uk
www.demo.com    http://foo.co.uk/
http://regexr.com/foo.html?q=bar
https://mediatemple.net
mediatepmple@outlook.com
mubeen.tom@hacker.com
1%453&harini_new@in.com
```

- **Popular Regular expression Generators :**
  - http://regexr.com/ (http://regexr.com/)
  - https://regex101.com/#python (https://regex101.com/#python)
  - http://www.regular-expressions.info/python.html (http://www.regular-expressions.info/python.html)

re.IGNORECASE

- to ignore the case (upper and lower)

In [100]:
```
reg = re.compile('python', re.IGNORECASE)  #compiling of regex object lets us
 to reuse it
```

In [103]:
```
result = reg.match('PYTHON')
```

In [104]:
```
print result.group()
```

```
PYTHON
```

In [105]:
```
reg = re.compile('python', re.I)   # Both re.I and re.IGNORECASE work in same
 way
```

In [110]:
```
result1 = reg.match('Python')
```

In [111]:
```
print result1.group()
```

```
Python
```

re.DOTALL

- special character match any character at all, includinh a newline

In [112]:
```
string = 'Today is Friday.\n Tomarrow is morning'
```

In [113]:
```
print string
```

```
Today is Friday.
 Tomarrow is morning
```

In [114]:
```
reg = re.compile('.*')
```

In [115]:
```
print reg.match(string).group()   # Observe that only first line is matched
```

```
Today is Friday.
```

In [116]:
```
reg = re.compile('.*', re.DOTALL)
```

In [117]:
```
print reg.match(string).group()  # Now, all the lines will be matched
```

```
Today is Friday.
 Tomarrow is morning
```

Perl based grouping pattern

(?P<name>)

In [118]:
```
m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Barack Obama" )
```

In [119]:
```
m.group('first_name')
```

Out[119]: 'Barack'

In [120]:
```
m.group('last_name')
```

Out[120]: 'Obama'

In [121]:
```
first_name    # Observe that those identifiers can't be used outside
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-121-1aebac703201> in <module>()
----> 1 first_name

NameError: name 'first_name' is not defined
```

In [125]:
```
m.group(0)
```

Out[125]: 'Barack Obama'

In [126]:
```
m.group(1)
```

Out[126]: 'Barack'

In [127]:
```
m.group(2)
```

Out[127]: 'Obama'

If a group matches multiple times, only the last match is accessible

In [130]:
```
re.match(r'(..)+', 'alb2cs').group()
```

Out[130]: 'alb2cs'

In [131]:
```
re.match(r'(..)+', 'alb2cs').group(0)
```

Out[131]: 'alb2cs'

In [132]:
```
re.match(r'(..)+', 'alb2cs').group(1)
```

Out[132]: 'cs'

In [133]:
```
re.match(r'(..)+', 'alb2cs').group(2)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-133-78100e216102> in <module>()
----> 1 re.match(r'(..)+', 'alb2cs').group(2)

IndexError: no such group
```

groups([default])

In [135]:
```
re.match(r'(..)+', 'alb2cs').groups()
```

Out[135]: ('cs',)

In [134]:
```
re.match(r'(\d+)\.(\d+)', '24.124123').groups()
```

Out[134]:
```
('24', '124123')
```

In [136]:
```
re.match(r'(\d+)\.(\d+)', '24').groups()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-136-0b2e55b3088e> in <module>()
----> 1 re.match(r'(\d+)\.(\d+)', '24').groups()

AttributeError: 'NoneType' object has no attribute 'groups'
```

In [138]:
```
re.match(r'(\d+)\.?(\d+)?', '24').groups()
```

Out[138]:
```
('24', None)
```

In [139]:
```
re.match(r'(\d+)\.?(\d+)?', '24').groups(0)
```

Out[139]:
```
('24', 0)
```

In [140]:
```
re.match(r'(\d+)\.?(\d+)?', '24').groups(1)
```

Out[140]:
```
('24', 1)
```

In [141]:
```
re.match(r'(\d+)\.?(\d+)?', '24').groups(34)
```

Out[141]:
```
('24', 34)
```

groupdict([default])

In [143]:
```
re.match(r'(?P<first_name>\w+) (?P<last_name>\w+)', 'sylvester stallone').grou
pdict()
```

Out[143]:
```
{'first_name': 'sylvester', 'last_name': 'stallone'}
```

start([group]) - Returns the starting position of the match

In [151]:
```
email = r'myEmail@domain.com'
```

In [152]:
```
m = re.search('dom', email)
if m:
    print m.group()
```

```
dom
```

In [153]:
```
email[:m.start()] + email[m.end():]   # Observe the 'dom' is absent
```

Out[153]:
```
'myEmail@ain.com'
```

In [154]:
```
m.start()   # results the position of start of the match
```

Out[154]: 8

In [155]:
```
email[0:m.start()]
```

Out[155]: 'myEmail@'

end([group]) - returns the ending position of the match

In [156]:
```
m.end()
```

Out[156]: 11

In [157]:
```
email[m.start():m.end()]
```

Out[157]: 'dom'

span([group])

- returns a tuple containing the (start, end) positions of the match

In [159]:
```
m.span()
```

Out[159]: (8, 11)

In [160]:
```
t = m.span()
email[t[0]:t[1]]
```

Out[160]: 'dom'

re.findall()

- Used to result in all the occurrances of given pattern

In [163]:
```
print re.findall(r'\d', '45345345354534545')
```

```
['4', '5', '3', '4', '5', '3', '4', '5', '3', '5', '4', '5', '3', '4', '5',
 '4', '5']
```

In [164]:
```
print re.findall(r'\w', 'this is friday')
```

```
['t', 'h', 'i', 's', 'i', 's', 'f', 'r', 'i', 'd', 'a', 'y']
```

In [167]:
```
print re.findall(r'(is)', 'this is friday')
```

```
['is', 'is']
```

In [168]:
```
print re.findall(r'\s', 'this is friday')   # \s is to used to match white spaces
```

```
[' ', ' ']
```

re.finditer()

- Used to result in all the occurrances of given pattern; But returns an iterator

In [220]:
```
re.finditer(r'\w', 'this is friday')
```

Out[220]:
```
<callable-iterator at 0x47f8710>
```

In [170]:
```
map(lambda x: x.group(), re.finditer(r'\w', 'this is friday'))
```

Out[170]:
```
['t', 'h', 'i', 's', 'i', 's', 'f', 'r', 'i', 'd', 'a', 'y']
```

In [171]:
```
re.findall('\d+', 'Today is 2nd week of July. 23 people came for 5th class')
```

Out[171]:
```
['2', '23', '5']
```

In [172]:
```
result = re.finditer('\d+', 'Today is 2nd week of July. 23 people came for 5th class')
```

In [173]:
```
type(result)
```

Out[173]:
```
callable-iterator
```

In [174]:
```
for mt in result:
    print mt.span(), mt.group()
```

```
(9, 10) 2
(27, 29) 23
(46, 47) 5
```

Compilation Flags

```
IGNORECASE, I  - Does case-insentitive matches. matches both upper and lower case alphabets
DOTALL,     S  - Make . match any charcater, including newlines
MULTILINE,  M  - Multi-line matching, affecting ^ and $
LOCALE,     L  - Does a locale-aware match
VERBOSE,    X  - Enable verbose REs, which can be organized more cleanly and understandably.
UNICODE,    U  - Makes several escapes like \w, \b, \s and \d dependent on the Unicode character database.
```

In [175]:
```
re.findall('python', 'Python python, PYTHON is PyThOnic- PYTHon', re.I) # both
 re.I and re.IGNORECASE work in the same way
```

Out[175]:
```
['Python', 'python', 'PYTHON', 'PyThOn', 'PYTHon']
```

In [176]:
```
re.findall('python', 'Python python, PYTHON is PyThOnic- PYTHon', re.IGNORECAS
E)
```

Out[176]:
```
['Python', 'python', 'PYTHON', 'PyThOn', 'PYTHon']
```

In [177]:
```
paragraph = '''
'python', \n'Python python,\n PYTHON is PyThOnic- PYTHon'
Working with python re module is interesting
'''
```

In [178]:
```
print paragraph
```

```
'python',
'Python python,
 PYTHON is PyThOnic- PYTHon'
Working with python re module is interesting
```

In [179]:
```
print re.findall('(.\w+)', paragraph, re.DOTALL)
```

```
["'python", "'Python", ' python', ' PYTHON', ' is', ' PyThOnic', ' PYTHon',
 '\nWorking', ' with', ' python', ' re', ' module', ' is', ' interesting']
```

In [180]:
```
print re.findall('(.\W+)', paragraph, re.DOTALL)   # \w and \W checks for occu
rrance and non-occurrance of word(s)
```

```
["  \n'", "n', \n'", 'n ', 'n,\n ', 'N ', 's ', 'c- ', "n'\n", 'g ', 'h ', 'n
 ', 'e ', 'e ', 's ', 'g\n']
```

In [181]:
```
paragraph = '''
This is good day.
this is 2nd week of July.
in this month there are 31 days.
'''
```

In [182]:
```
print paragraph
```

```
This is good day.
this is 2nd week of July.
in this month there are 31 days.
```

In [183]:
```
re.findall('this', paragraph, re.MULTILINE)
```

Out[183]:
```
['this', 'this']
```

In [184]:
```
re.findall('^this', paragraph, re.MULTILINE)
```

Out[184]:
```
['this']
```

In [185]:
```
re.findall('^this', paragraph, re.MULTILINE|re.IGNORECASE)  # combining the compilation flags
```

Out[185]:
```
['This', 'this']
```

In [186]:
```
re.findall('^this', paragraph, re.M|re.I)  # shortforms of these compilation flags
```

Out[186]:
```
['This', 'this']
```

re.ASCII() vs re.UNICODE() vs re.LOCALE()

re.ASCII is used to match all ASCII characters.

re.UNICODE is used to match all unicode characters.

re.LOCALE is used to match any local language (non-english) characters

re.UNICODE amd re.LOCALE are used together

In [187]:
```
re.findall("\w+", "this⇢is⇢an⇢example")
```

Out[187]:
```
['this', 'is', 'an', 'example']
```

Execute the above statement with a non-english language, if your system supports

In [188]:
```
re.findall(ur"\w+", u"这是一个例子", re.UNICODE)   # chinese
```

Out[188]:
```
[u'\u8fd9\u662f\u4e00\u4e2a\u4f8b\u5b50']
```

In [189]:
```
re.findall(ur"\w+", u"هذا مثال", re.UNICODE)         # Arabic
```

Out[189]:
```
[u'\u0647\u0630\u0627', u'\u0645\u062b\u0627\u0644']
```

re.VERBOSE is used to create more readable re objects. But, it works same as without placing this flag.

In [190]:
```
reObj = re.compile(r"""
 &[#]                # Start of a numeric entity reference
 (
     0[0-7]+         # Octal form
   | [0-9]+          # Decimal form
   | x[0-9a-fA-F]+   # Hexadecimal form
 )
 ;                   # Trailing semicolon
""", re.VERBOSE)
```

In [192]:
```
reObj
```

Out[192]:
```
re.compile(r'\n &[#]                # Start of a numeric entity reference\n
 (\n     0[0-7]+         # Octal form\n   | [0-9]+          # Decimal form\n
  | x[0-9a-fA-F]+   # Hexadecimal form\n )\n ;                   # Trailing s
emicolon\n',
re.VERBOSE)
```

In [193]:
```
print reObj
```

```
<_sre.SRE_Pattern object at 0x047D15A0>
```

In [194]:
```
#with out verbose flag, it will look like
reObj = re.compile("&#(0[0-7]+"
                   "|[0-9]+"
                   "|x[0-9a-fA-F]+);")
```

In [196]:
```
print reObj
```

```
<_sre.SRE_Pattern object at 0x047D1700>
```

NOTE: Run in python interpreter to observe the difference.

In [197]:
```
re.compile(r"[a-f|3-8]", re.DEBUG)  # to debug a pattern
```

```
in
  range (97, 102)
  literal 124
  range (51, 56)
```

Out[197]:
```
re.compile(r'[a-f|3-8]', re.DEBUG)
```

\b Word boundary. This is a zero-width assertion that matches only at the beginning or end of a word.

In [198]:
```
p = re.compile(r'\bclass\b')
```

In [199]:
```
print p.search("Today's class is on regular expressions")
```

```
<_sre.SRE_Match object at 0x047E8FA8>
```

In [200]:
```
print p.search("Today's class is on regular expressions").group()
```

```
class
```

In [201]:
```
print p.search('the declassified algorithm is classified again')
```

```
None
```

In [202]:
```
p = re.compile('\bclass\b')
```

In [203]:
```
print p.search('no class at all')  # no result as \b of regular expressions co
llide with backslash character (\b) of python
```

```
None
```

In [204]:
```
print p.search('\b' + 'class' + '\b')  # similarly, try print p.search('no \bc
lass\b at all')
```

```
<_sre.SRE_Match object at 0x047FB170>
```

In [205]:
```
print p.search('\b' + 'class' + '\b').group()  # \B works opposite to that of
 \b
```

```
class
```

Modifying Strings with re module

split() Split the string into a list, splitting it wherever the RE matches

sub() Find all substrings where the RE matches, and replace them with a different string

subn() Does the same thing as sub(), but returns the new string and the number of replacements

In [206]:
```
sentence = "It's the right right, right?"
```

In [207]:
```
p = re.compile(r'\W+')   #\W matches for absence of word
```

In [208]:
```
p.split(sentence)
```

Out[208]:
```
['It', 's', 'the', 'right', 'right', 'right', '']
```

In [209]:
```
re.findall(r'\w+', sentence)    # In this case, it is same as split()
```

Out[209]:
```
['It', 's', 'the', 'right', 'right', 'right']
```

In [210]:
```
p.split(sentence, 3)     # additionally, split() has option to specify maxspli
ts
```

Out[210]:
```
['It', 's', 'the', 'right right, right?']
```

In [211]:
```
p.split(sentence, 2)         # length will be maxsplits+1, when maxsplits!=0
```

Out[211]:
```
['It', 's', 'the right right, right?']
```

In [212]:
```
p.split(sentence, 0)     # default option
```

Out[212]:
```
['It', 's', 'the', 'right', 'right', 'right', '']
```

In [213]:
```
p = re.compile('(blue|white|red)')
```

In [214]:
```
p.sub('colour', 'blue Lorries and red Buses')  # returns a string
```

Out[214]:
```
'colour Lorries and colour Buses'
```

In [215]:
```
p.sub('colour', 'blue Lorries and red Buses', count=1)
```

Out[215]:
```
'colour Lorries and red Buses'
```

subn() does the same job as sub; but, returns a tuple containing the new string value, and the no. of replacements performed.

In [216]:
```
p.subn('colour', 'blue Lorries and red Buses')   # returns a tuple
```

Out[216]:
```
('colour Lorries and colour Buses', 2)
```

In [217]:
```
p.subn('colour', 'no colours at all')
```

Out[217]:
```
('no colours at all', 0)
```

kodos (http://kodos.sourceforge.net/home.html) - Tool for computing and practicing regular expressions.

pythex (http://pythex.org/online) regex generator created in python

Summarizing all the Regular Expressions

Non-special chars match themselves. Exceptions are special characters::

```
\       Escape special char or start a sequence.
.       Match any char except newline, see re.DOTALL
^       Match start of the string, see re.MULTILINE
$       Match end of the string, see re.MULTILINE
[]      Enclose a set of matchable chars
R|S     Match either regex R or regex S.
()      Create capture group, & indicate precedence
```

After '[', enclose a set, the only special chars are::

```
]   End the set, if not the 1st char
-   A range, eg. a-c matches a, b or c
^   Negate the set only if it is the 1st char
```

Quantifiers (append '?' for non-greedy)::

```
{m}     Exactly m repetitions
{m,n}   From m (default 0) to n (default infinity)
*       0 or more. Same as {,}
+       1 or more. Same as {1,}
?       0 or 1. Same as {,1}
```

Special sequences::

```
\A  Start of string
\b  Match empty string at word (\w+) boundary
\B  Match empty string not at word boundary
\d  Digit
\D  Non-digit
\s  Whitespace [ \t\n\r\f\v], see LOCALE,UNICODE
\S  Non-whitespace
\w  Alphanumeric: [0-9a-zA-Z_], see LOCALE
\W  Non-alphanumeric
\Z  End of string
\g<id>  Match prev named or numbered group,
        '<' & '>' are literal, e.g. \g<0>
        or \g<name> (not \g0 or \gname)
```

Special character escapes are much like those already escaped in Python string literals. Hence regex '\n' is same as regex '\\n'::

```
\a  ASCII Bell (BEL)
\f  ASCII Formfeed
\n  ASCII Linefeed
\r  ASCII Carriage return
\t  ASCII Tab
\v  ASCII Vertical tab
\\  A single backslash
\xHH   Two digit hexadecimal character goes here
\OOO   Three digit octal char (or just use an
       initial zero, e.g. \0, \09)
\DD    Decimal number 1 to 99, match
       previous numbered group
```

Extensions. Do not cause grouping, except '<name>'::

```
(?iLmsux)     Match empty string, sets re.X flags
(?:...)       Non-capturing version of regular parens
(?P<name>...) Create a named capturing group.
(?P=name)     Match whatever matched prev named group
(?#...)       A comment; ignored.
(?=...)       Lookahead assertion, match without consuming
(?!...)       Negative lookahead assertion
(?<=...)      Lookbehind assertion, match if preceded
(?<!...)      Negative lookbehind assertion
(?(id)y|n)    Match 'y' if group 'id' matched, else 'n'
```

Flags for re.compile(), etc. Combine with '|'::

```
re.I == re.IGNORECASE   Ignore case
re.L == re.LOCALE       Make \w, \b, and \s locale dependent
re.M == re.MULTILINE    Multiline
re.S == re.DOTALL       Dot matches all (including newline)
re.U == re.UNICODE      Make \w, \b, \d, and \s unicode dependent
re.X == re.VERBOSE      Verbose (unescaped whitespace in pattern
                        is ignored, and '#' marks comment lines)
```

Module level functions::

```
compile(pattern[, flags]) -> RegexObject
match(pattern, string[, flags]) -> MatchObject
search(pattner, string[, flags]) -> MatchObject
findall(pattern, string[, flags]) -> list of strings
finditer(pattern, string[, flags]) -> iter of MatchObjects
split(pattern, string[, maxsplit, flags]) -> list of strings
sub(pattern, repl, string[, count, flags]) -> string
subn(pattern, repl, string[, count, flags]) -> (string, int)
escape(string) -> string
purge() # the re cache
```

RegexObjects (returned from `compile()`)::

```
.match(string[, pos, endpos]) -> MatchObject
.search(string[, pos, endpos]) -> MatchObject
.findall(string[, pos, endpos]) -> list of strings
.finditer(string[, pos, endpos]) -> iter of MatchObjects
.split(string[, maxsplit]) -> list of strings
.sub(repl, string[, count]) -> string
.subn(repl, string[, count]) -> (string, int)
.flags      # int, Passed to compile()
.groups     # int, Number of capturing groups
.groupindex # {}, Maps group names to ints
.pattern    # string, Passed to compile()
```

MatchObjects (returned from `match()` and `search()`)::

```
.expand(template) -> string, Backslash & group expansion
.group([group1...]) -> string or tuple of strings, 1 per arg
.groups([default]) -> tuple of all groups, non-matching=default
.groupdict([default]) -> {}, Named groups, non-matching=default
.start([group]) -> int, Start/end of substring match by group
.end([group]) -> int, Group defaults to 0, the whole match
.span([group]) -> tuple (match.start(group), match.end(group))
.pos       int, Passed to search() or match()
.endpos    int, "
.lastindex int, Index of last matched capturing group
.lastgroup string, Name of last matched capturing group
.re        regex, As passed to search() or match()
.string    string, "
```

ref (https://github.com/tartley/python-regex-cheatsheet Version: v0.3.3) python-regex-cheatsheet