**Python Part 3 - Pattern Matching**

|  |  |
| --- | --- |
| **Pattern Matching with regex**  Python's pattern matching with regular expressions is based on Perl's regular expressions.   * To use these capabilities, we must import the regular expression module:   **import re**   * Define a regular expression object using **re.compile**(*regEx*)   *regExObj* = re.compile(r'\d\d\d\d')   * We can use the regex object to search for a regular expression in a string   *regExObj*.search(*string*)   * We can use that regex object to match an entire string   *regExObj*.match(*string*)   * We can find all occurrences that match that regular expression using   *regExObj*.findall(*string*)   * We usually save the results of one of those functions in a match object   *matchObj* = *regExObj*.search(*string*)   * Assuming *matchObj* isn't None, we can reference the matching string   *matchObj*.group() | >>> # we will assume that re has been imported in all subsequent  # regex examples  import re  # define a regular expression to match a phone number 999-999-9999  phoneRE = re.compile(r'\d{3}-\d{3}-\d{4}')  >>> text1 = "Please call me at 210-555-1234"  matchObj = phoneRE.search(text1)  if matchObj != None:  print( matchObj.group())  210-555-1234  >>> text2 = "Please call me as soon as possible."  matchObj = phoneRE.search(text2)  if matchObj != None:  print( matchObj.group())  else:  print("not found")  not found  >>> text3 = "210-555-6666"  matchObj = phoneRE.match(text3)  if matchObj != None:  print("matched")  matched  >>> text3 = "xx210-555-6666"  matchObj = phoneRE.match(text3)  if matchObj != None:  print("matched")  else:  print("not matched")  not matched |
| **Saving Values Using Grouping**  It is often necessary to save particular portions of what is matched. This can be done by placing parentheses around the portions to be saved. You can reference each group*k* by specifying  *matchObj*.group(*k*)  The first matched group will be  *matchObj*.group(1).  *matchObj* will be **None** if the regular expression did not match. | >>> # define a regular expression to match an phone number 999-999-9999  # saving the area code and number  phoneRE = re.compile(r'(\d{3})-(\d{3}-\d{4})')  matchObj = phoneRE.search(text1)  if matchObj != None:  print( "Area Code=", matchObj.group(1)  , "Phone=", matchObj.group(2))  Area Code= 210 Phone= 555-1234 |
| **Special Characters in Regular Expressions**  **\d** matches any numeric digit 0 thru 9  **\D** matches any character that is **not** a numeric digit  **\w** matches any letter, numeric digit or underscore  **\W** matches any character that is **not** a letter, numeric digit or underscore  **\s** matches a space, tab or newline character  **\S** matches any character that is **not** a space, tab or newline character  **. a dot** any character other than a newline character  **(*ex*)** defines a group for *ex*  **[*values*]** matches any character listed between the brackets  **[^*values*]** matches any character that is **not** listed between the brackets | >>> # define a regular expression to match an ABC123 ID  abc123RE = re.compile(r'[a-z]{3}\d{3}')  text4 = "My ID is xyz123."  matchObj = abc123RE.search(text4)  if matchObj != None:  abc123Id = matchObj.group()  else:  abc123Id = None  >>> abc123Id  xyz123  >>> # define a regular expression which matches Ho Ho Ho with  # any of these characters in between the words:  # comma or exclamation point  # It also has a space between those characters  # and the next Ho.  # The last Ho must be immediately followed by a !  # It should be case insensitive (specify re.I)  hohohoRE = re.compile(r'ho[,!]\sho[,!]\sho!', re.I)  matchObjHo = hohohoRE.search('Santa said, "Ho! Ho! Ho!"') |
| **Special Characters Applied to a Preceding Expression**  The following special characters are applied to a preceding expression:  **?** matches **zero** or **one** of the preceding expression  **\*** matches **zero** or **more** of the preceding expression  **+** matches **one** or **more** of the preceding expression  **{*n*}** matches **exactly *n*** of the preceding expression  **{*n,*}** matches ***n* or more** of the preceding expression  **{*,m*}** matches ***0* to *m*** of the preceding expression  **{*n,m*}** matches **at least *n* and at most *m*** of the preceding expression  **$** preceding expression must match at the end of the string  **| *ex*** matches the preceding expression or the following one | >>> # define a regex that matches Mickey Mouse or Minnie Mouse  # Mouse is optional  mouseRE = re.compile(r'(Mickey|Minnie)\s?(Mouse)?')  mouseMO = mouseRE.search("Goofy yelled at Mickey Mouse")  >>> mouseMO.group(1)  Mickey  >>> mouseMO.group(2)  Mouse  >>> mouseMO = mouseRE.search("Pluto licked Minnie's hand")  >>> mouseMO.group(1)  Minnie  >>> mouseMO.group(2)  None |
| **Values that Must be Escaped to be Matched**  There are several symbols that have special meanings to regular expressions. To match the literal values, these must be escaped:  **\.** matches a period  **\\** matches a backslash  **\(** matches a left parenthesis  **\)** matches a right parenthesis  **\[** matches a left bracket  **\]** matches a right bracket  **\{** matches a left curly brace  **\}** matches a right curly brace  **\^** matches a carat  **\$** matches a dollar sign | >>> # define a regular expression for a phone number (999)-999-9999  # saving the area code and number  import re  phoneRE = re.compile(r'\((\d{3})\)-(\d{3}-\d{4})')  text1 = "Please call me at (210)-555-1234"  phoneMO = phoneRE.search(text1)  if phoneMO!= None:  print( "Area Code=", phoneMO.group(1)  , "Phone=", phoneMO.group(2))  Area Code= 210 Phone= 555-1234 |
| **Splitting a string using re.split()**  **re.split(***regExpr, string***)**  *regExpr* is a regular expression. *string* is the string to be split. | >>> # Split on semicolon, comma, period or space. Also ignore 0 to many  # spaces after the delimiter.  import re  text3 = "He loved playing basketball; however, he hated watching it on TV."  wordM = re.split("[;,\s\.]\s\*", text3)  >>> wordM  ['He', 'loved', 'playing', 'basketball', 'however', 'he', 'hated', 'watching', 'it', 'on', 'TV', ''] |
| **Greedy vs Nongreedy Matches**  Like regular expression matching in other languages and tools, Python uses a greedy matching mode for multi-occuring patterns unless told to be nongreedy. With **greedy matching mode**, it tries to match as much text as possible.  Placing a ? after a multi-occuring pattern, causes **re** to use a nongreedy matching mode. This is not the same meaning of ? as above. With **nongreedy matching mode**, it tries to match as little text as possible. | >>> # Greedy matching lab.cs.utsa.edu  # The matching pattern matches any character until a dot is found.  # With greedy, the .\* matches as much text as possible.  location = "lab.cs.utsa.edu"  gDotRE = re.compile(r'(.\*)\.')  gDotMO = gDotRE.search(location)  print (gDotMO.group())  **lab.cs.utsa.**  >>> # Nongreedy matching lab.cs.utsa.edu  location = "lab.cs.utsa.edu"  ngDotRE = re.compile(r'(.\*?)\.')  ngDotMO = ngDotRE.search(location)  print (ngDotMO.group())  **lab.** |
| **More Examples of Greedy vs Nongreedy** | >>> # Greedy matching of Ho!  text = "Ho! Ho! Ho! Ho! Ho! Ho! "  gSantaRE = re.compile(r'(Ho!\s){2,5}') #minimum 2 max 5  gSantaMO = gSantaRE.search(text)  print (gSantaMO.group())  **Ho! Ho! Ho! Ho! Ho!**  >>> # Nongreedy matching of Ho!  text = "Ho! Ho! Ho! Ho! Ho! "  ngSantaRE = re.compile(r'(Ho!\s){2,5}?')  ngSantaMO = ngSantaRE.search(text)  print (ngSantaMO.group())  **Ho! Ho!**  >>> # Matching of ho, ho, ho, kid  text = "ho, ho, ho, kid"  ngSantaRE = re.compile(r'(ho,\s){1,5}?kid')  ngSantaMO = ngSantaRE.search(text)  print (ngSantaMO.group())  **ho, ho ,ho, kid**  Why?  Matching starts from he left side. #has to have kid so it reaches it |
| **Looping for Multiple of Similar Items on One Text String**  Suppose we have the recipients of medals in any order on a text line. Example:  GOLD Fred SILVER Barney  SILVER Shaggy GOLD Scooby BRONZE Fred  One approach is to loop until we run out of values matching. | import re  # what does this pattern do?  # skip over leading white spaces  # matches type of medal and saves it in group(1)  # skip over a singles white space(can be a tab)  # matches a word and saves it in group(2)  # matches anything until end of sting and saves it in group(3)  # group(3) is our remaining stuff  medalRE = re.compile(r'\s\*(GOLD|SILVER|BRONZE)\s(\w\*)(.\*)$')  # loop until eof  while True:  try:  inputLine = input() # reads from stdin  except(EOFError):  break  # loop while there are medals  medalMO = medalRE.search(inputLine)  while medalMO != None:  medal = medalMO.group(1)  recipient = medalMO.group(2)  # group(3) has the rest of the line after the name  restOfLine = medalMO.group(3)  print(medal, recipient)  medalMO = medalRE.search(restOfLine) |
| How could split be used to separate those? | # loop until eof  while True:  try:  inputLine = input() # reads from stdin  except(EOFError):  break  rest = inputLine  # loop while there are at least two parts using split  tokenM = rest.split(‘ ‘, 2) #only care about 2 parts  while len(tokenM) >= 2:  medal = tokenM[0]  recipient = tokenM[1]  print (medal, recipient)  if len(tokenM) <= 2:  break  rest = tokenM[2]  tokenM = rest.split(' ',2) |
| **Non-exact matches - difflib**  Python has several libraries which support non-exact matches. The **difflib** module has several methods.  To get the difference of two strings separated as lists of text lines:  *dobj* = difflib.Differ()  *result*=list(*dobj.*compare(*list1, list2*))  Results contain lines with a two character code   |  |  | | --- | --- | | **Code** | **Meaning** | | '- ' | line unique to *list1* | | '+ ' | line unique to *list2* | | ' ' | line is in both lists | | '? ' | line shows where the two lines are different | | >>> import difflib  text1 = """This is an example where the first line is the same, but  the second line is different.# 3 quotes = multiple lines  How about the third line? It is in this only.  Oh boy! It is the same here!  The last line in text1.""".splitlines()  text2 = """This is an example where the first line is the same, but  the second line is a little different.  Oh boy! It is the same here!  This line isn't in text1.  This line also isn't in text1.  The last line in text1.""".splitlines()  dobj = difflib.Differ()  result = list(dobj.compare(text1, text2))  from pprint import pprint  pprint(result)  **[' This is an example where the first line is the same, but',**  **'- the second line is different.',**  **'+ the second line is a little different.',**  **'? +++++++++\n',**  **'- How about the third line? It is in this only.',**  **' Oh boy! It is the same here!',**  **"+ This line isn't in text1.",**  **"+ This line also isn't in text1.",**  **' The last line in text1.']** |
| **Non-exact matches - difflib continued**  difflib also provides a function for checking sequences of matched characters.  from difflib import SequenceMatcher as SM  *result =* SM(None, *str1, str2*).ratio()  Returns a float, between 0 and 1, measuring the *similarity* of the sequences. In general, matches less than 0.6 are not good.  In some situations, you may prefer to specify the first parameter to eliminate characters of no interest:  *result =* SM(lambda x: x == " "  , *str1, str2*).ratio()  That would not consider blank spaces.  #if char is a space -> throw it away. | >>> # Example using SequenceMatcher  from difflib import SequenceMatcher as SM  str1 = "SAN ANTONIO"  str2 = "SAN ANTONOI"  str3 = "SNA ANTONIO"  str4 = "SAN DIEGO"  str5 = "SAN FRANCISCO"  print (str1, str2, SM(None, str1, str2).ratio())  print (str1, str3, SM(None, str1, str3).ratio())  print (str2, str3, SM(None, str2, str3).ratio())  print (str1, str4, SM(None, str1, str4).ratio())  print (str1, str5, SM(None, str1, str5).ratio())  **SAN ANTONIO SAN ANTONOI 0.90909090909090906**  **SAN ANTONIO SNA ANTONIO 0.90909090909090906**  **SAN ANTONOI SNA ANTONIO 0.81818181818181823**  **SAN ANTONIO SAN DIEGO 0.5**  **SAN ANTONIO SAN FRANCISCO 0.58333333333333337** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |