Python
Feature Summary: Regexes

RULES

- patterns can match anywhere in a string (unless anchored with **^** or **$**)
- patterns must match on consecutive characters
- quantifiers will match on as many characters as possible ("greedy" matching)
- matches are case sensitive (unless flagged insensitive)
- "raw" string (marked with **r** before the first quote) must be used for patterns

FUNCTIONS

- **re.search()** function: ***match a pattern to a string***
  *returned "match" object reads as True in an **if** expression if pattern matches. It can also be queried to retrieve grouped text*

  arguments: "raw" string pattern, text to search
  return value: **re** "Match" object

```
matchobj = re.search(r'\d', text)
if matchobj:
  print('matched')
```

- **re.findall()** function: ***return a list of all matches in a string***
  pattern matches as many times as possible until reaching end of string . if pattern has no group, will return matched text; if pattern has a group, will return grouped text; if multiple groups, will return a list of tuples of grouped text

  arguments: raw string pattern, text to search
  return value: a list of strings of matched text, or list of tuples if there are multiple groupings in pattern

```
strlist = re.findall(r'pat', text)
```

- **re.sub()** function: ***string replacement using a pattern***
  searches for pattern and replaces with supplied string

  arguments: raw string pattern, replacement text, text to search
  return value: a string with replacements made

```
rstr = re.sub(r'pat', rplce, text)
```

- **re.compile()** function: ***precompile a pattern***
  *lets **re** do preprocessing before evaluating the match on a string -- used for matching on numerous strings*

  arguments: raw string pattern, replacement text, text to search
  return value: a string with replacements made

```
cpat = re.compile(r'pat')
  for line in many_text_lines:
    matchobj = cpat.search(line)
```

FLAGS

These are passed as additional arguments; they modify the behavior of the match. If multiple, flags are needed, they should be separated by a vertical bar.

- re.I / re.IGNORECASE: ***case-insensitive match*** `re.search(r'pat', text, re.I)`
- re.M / re.MULTILINE: ***^ and $ will match on start and end of line in a "multi-line" string*** `re.search(r'pat', text, re.M)`
- re.S / re.DOTALL: ***. (wildcard) matches on newlines*** `re.search(r'pat', text, re.S)`

ANCHORS

Anchors require that the match start at the first character or end at the last character.

- **^** ***match from start of string*** `m = re.search(r'^pat', text)`
- **$** ***match to end of string*** `m = re.search(r'pat$', text)`
- **\b** ***match at end of word*** `m = re.search(r'\bword\b', text)`
  this "zero width" matcher does not match on a character but rather the boundary between a letter and a non-letter (space, punctuation or the start or end of string)

"BUILT IN" CHARACTER CLASSES

A character class matches on one character in the string. If quantified, it may match on more than one.

- **\d** ***matches any numeric character 0-9*** `m = re.search(r'\d+', text)`
- **\s** ***matches tab, space or newline*** `m = re.search(r'\s+$', text)`
- **\w** ***matches any letter, number or underscore*** `m = re.findall(r'\w+', text)`
- **\D** ***matches any character other than \d*** `if re.search(r'\D', text):`
- **\S** ***matches any character other than \s*** `if re.search(r'\S', text):`
- **\W** ***matches any character other than \w*** `if re.search(r'\W', text):`
- **.** **(period):** ***"wildcard" -- matches on any character other than newline*** `m = re.search(r'\w+.+\w+', text)`

CUSTOM CHARACTER CLASSES

A custom character class defines specific members of a class, and will match only on those characters. A range (x-z) may be used. "Built in" character classes may be used inside custom character classes. Characters may be listed individually, as a range (a-z) or as a built-in character class.

- **[a-fxyz\s]:** ***matches any character listed*** `m = re.search(r'[a-z]+', text)`
- **[^a-fxyz\s]:** ***matches any character other than those listed*** `m = re.search(r'[^a-z]+', text)`

QUANTIFIERS

A quantifier placed after any character, character class or grouped (parenthetical) pattern will match them on as many characters as possible

- **+** ***one or more*** `m = re.search(r'\d+', text)`
- ***** ***zero or more*** `m = re.search(r'\d*\.\d{2}', text)`

? ***zero or one***

```
m = re.search(r'\d?\.\d{2}', text)
```

- **{0,3}** **(*custom): between x and y***
  to specify "or more" for max, omit the 2nd number

```
m = re.search(r'\w{0,3}\d+', text)
```

- **?** ***"non-greedy" modifier***
  when placed after any quantifier, will match on "as few as possible" instead of "as many as possible"

```
m = re.search(r'\d+.+?\d+', text)
```

PARENTHETICAL GROUPINGS

Parentheses are used to group characters within a pattern. There are 3 possible purposes for groupings.

- ***grouping for alternates***
  grouping will match on one of the alternate patterns separated by the vertical bar

```
m = re.search(r'this (and|or) that')
```

- ***grouping for quantifying***
  quantifier placed right after a grouping quantifies the entire group

```
m = re.search(r'Rich (M. )?Nixon',text)
```

- ***grouping for text extraction***
  matched characters within a grouping are retrievable through the ***Match*** *object*

```
tt = 'cost:  23.95'
m = re.search(r'cost: (\d+\.\d\d)', tt)
val = m.group(1)   # 23.95
```

re.Match METHODS

A ***Match*** *object is returned from a successful match. The object can be queried to retrieve matched text or learn about the match.*

- ***.group()*** *method: retrieve matched text from a grouping*
  groups are numbered 1-n, counting each left parenthesis
  arguments: integer index starting at 1 (0 is entire match)
  return value: string of text that matched the pattern in the grouping

```
val = matchobj.group(1)
```

- ***.groups()*** *method: retrieve matched text from all groupings*
  groups are ordered counting each left parenthesis
  arguments: none
  return value: list of strings, each the text matched in each grouping

```
values = matchobj.groups()
```