Regular Expressions
(in Python)

Python or Egrep

- We will use Python.
- In some scripting languages you can call the command “grep” or “egrep”
- egrep pattern file.txt
- E.g. egrep “^A” file.txt
- Will print all the line of file.txt which start with (^) the letter A (capital A)

Regular expression

- (abbreviated **regex** or **regexp**) a search pattern, mainly for use in pattern matching with strings, i.e. "find and replace"-like operations.
- Each character in a regular expression is either understood to be a metacharacter with its special meaning, or a regular character with its literal meaning.
- We ask the question – does a given string match a certain pattern?

List of Meta characters

1. .
2. +
3. ?
4. *
5. ^
6. $
7. [...]
8. -
9. [^...]
10. |
11. ()
12. {m,n}

. (dot)

- Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor-, character-encoding-, and platform-specific, but it is safe to assume that the line feed character is included).
- Within POSIX bracket expressions, the dot character matches a literal dot. For example, a.c matches "abc", etc., but [a.c] matches only "a", ".", or "c".

Example .

- `string1 = "Hello, world."`
- `if re.search(r".....", string1):`
- `    print string1 + " has length >= 5"`

Example [.] literally a dot

```
string1 = "Hello, world."
if re.search(r"....[.]", string1):
    print string1 + " has length >= 5 and ends with a ."
```

+

- Matches the preceding element one or more times. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac".
- `string1 = "Hello, world."`
- `if re.search(r"l+", string1):`
- `    print 'There are one or more consecutive letter "l"' +\`
- `            "'s in " + string1`

?

- Matches the preceding pattern element zero or one times.
- #?
- #Matches the preceding pattern element zero or one times.
- string1 = *"Hello, world."*
- if re.search(r*"H.?e", string1):*
- print *"There is an 'H' and a 'e' separated by 0-1 characters (Ex: He Hoe)"*

*

- Matches the preceding element zero or more times. For example, ab*c matches "ac", "abc", "abbbc", etc. [xyz]* matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. (ab)* matches "", "ab", "abab", "ababab", and so on.
- `string1 = "Hello, world."`
- `if re.search(r"e(ll)*o", string1):`
- `    print "'e' followed by zero to many'll' followed by 'o' (eo, ello, ellllo)"`

^

- Matches the beginning of a line or string.
- #^ Matches the beginning of a line or string.
- string1 = "Hello World"
- if re.search(r"^He", string1):
- print string1, "starts with the characters 'He'"

$

- Matches the end of a line or string.
- `string1 = "Hello World"`
- `if re.search(r"rld$", string1):`
- `    print string1, "is a line or string that ends with 'rld'"`

[ ]

- A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z", as does [a-cx-z].
- The - character is treated as a literal character if it is the last or the first (after the ^, if present) character within the brackets: [abc-], [-abc]. Note that backslash escapes are not allowed. The] character can be included in a bracket expression if it is the first (after the ^) character: []abc].

Example []

- #[] Denotes a set of possible character matches.
- string1 = *"Hello, world."*
- if re.search(r*"[aeiou]+", string1):*
- print string1 + *" contains one or more vowels."*

[^]

- Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than "a", "b", or "c". [^a-z] matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.

Example [^]

- #[^...] Matches every character except the ones inside brackets.
- string1 = *"Hello World\n"*
- if re.search(r*"[^abc]", string1):*
- print string1 + *" contains a character other than a, b, and c"*

Example |

- #| Separates alternate possibilities.
- string1 = *"Hello, world."*
- if re.search(r*"(Hello|Hi|Pogo)", string1):*
- print *"At least one of Hello, Hi, or Pogo is contained in " + string1*

()

- Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, *n*). A marked subexpression is also called a block or capturing group.

Example ()

- `string1 = "Hello, world."`
- `m_obj = re.search(r"(H..).(o..)(...)", string1)`
- `if re.search(r"(H..).(o..)(...)", string1):`
- `    print "We matched '" + m_obj.group(1) + "' and '" + m_obj.group(2) + "' and '" + m_obj.group(3)+ "'"`

\n **{*m*,*n*}**

- \n Matches what the *n*th marked subexpression matched, where *n* is a digit from 1 to 9. This construct is vaguely defined in the POSIX.2 standard. Some tools allow referencing more than nine capturing groups.
- **{*m*,*n*}** Matches the preceding element at least *m* and not more than *n* times. For example, a{3,5} matches only "aaa", "aaaa", and "aaaaa". This is not found in a few older instances of regular expressions. **BRE mode requires\{*m*,*n*\}**.

-v option

- A regex in Python, either the search or match methods, returns a Match object or None. For grep -v equivalent, you might use:
- import re for line in sys.stdin: if re.search(r'[a-z]', line) is None: sys.stdout.write(line)

e.g. Username

- /^[a-z0-9_-]{3,16}$/
- Starts and ends with 3-16 numbers, letters, underscores or hyphens
- Any lowercase letter (a-z), number (0-9), an underscore, or a hyphen.
- At least 3 to 16 characters.
- Matches E.g. my-us3r_n4m3 but not th1s1s-wayt00_l0ngt0beausername

e.g. Password

- /^[a-z0-9_-]{6,18}$/
- Starts and ends with 6-18 letters, numbers, underscores, hyphens.
- Matches e.g. myp4ssw0rd but not mypa$$w0rd

e.g. Hex Value

- /^#?([a-f0-9]{6}|[a-f0-9]{3})$/
- ***Starts with a +/- (optional) followed by one or more***
- Matches e.g. #a3c113 but not #4d82h4

e.g. Email

- /^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
- ***String that matches:***
- john@doe.com
- ***String that doesn't match:***
- john@doe.something (TLD is too long)

Match n characters

- egrep.exe "^...$" data1.txt
- Will match any line with exactly 3 characters
- ^ starts with .
- And contains “…” (i.e. 3 characters)
- $ ends with
- Or just egrep.exe "^.{3}$" data1.txt
- What about egrep.exe "(..){2}" data1.txt?