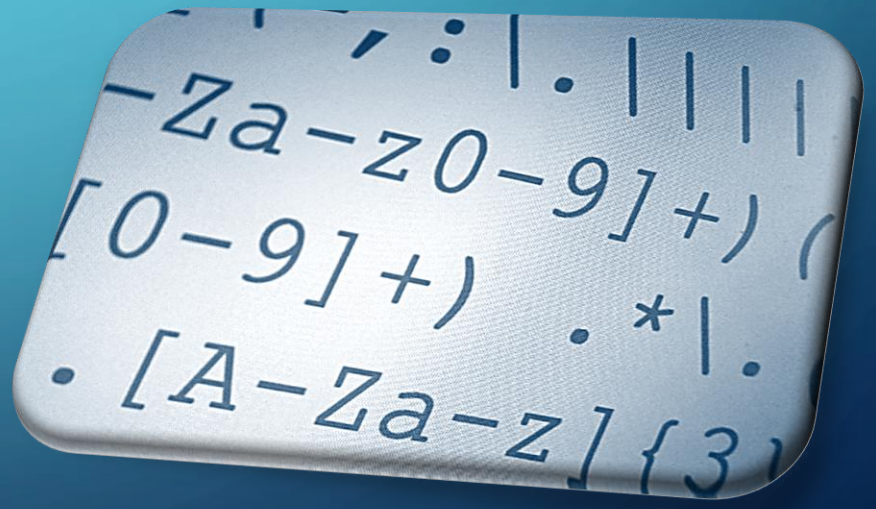
REGULAR EXPRESSIONS

LANGUAGE FOR STRING PARSING



- Regular expressions is a computer language for specifying a pattern of text.
- Regular expressions are encoded text strings used as patterns for matching sets of strings.

Regex Syntax Example

\d	Any numeric digit from 0 to 9
\D	Any character that is not a numeric digit from 0 to 9
\w	Any letter, numeric digit, or underscore
\W	Any character that is not a letter, numeric digit, underscore
\s	Any space, tab, or newline character
\S	Any character that is not a space, tab, or newline

Regex Syntax Example

- The ? matches zero or one
- The * matches zero or more
- The + matches one or more
- The {n} matches exactly n
- The {n,} matches n or more
- The {,m} matches 0 to m
- The . matches any character

- [ABC] matches character between brackets
- [^ABC] matches character that isn’t between brackets
- The {n,m} matches at least n and at most m
- {n,m}? or *? or +? performs a match

Matching

Phone number: 408-864-5300

- Option #1

  [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]

- Option #2

  \d\d\d-\d\d\d-\d\d\d\d

- Option #3

  \d{3}-?\d{3}-?\d{4}

- Option #4

  (\d{3,4}[.-]?)+

Matching Words & Spaces

Address: 21250 Stevens Creek blvd., Cupertino

\w

21250 Stevens Creek Blvd., Cuptertino

\w+

21250 Stevens Creek Blvd., Cuptertino

\W

21250 Stevens Creek Blvd., Cuptertino

\W+

21250 Stevens Creek blvd., Cupertino

Boundaries

Address: 21250 Stevens Creek blvd., Cupertino

[0-9]*

[^0-9]*

[a-zA-Z0-9]*

[^a-zA-Z0-9]*

Boundaries

[0-9]*

```
21250 Stevens Creek blvd., Cupertino
```

[^0-9]*

```
21250 Stevens Creek blvd., Cupertino
```

[a-zA-Z0-9]*

```
21250 Stevens Creek blvd., Cupertino
```

[^a-zA-Z0-9]*

```
21250 Stevens Creek blvd., Cupertino
```

Example 1

```
number = ‘800-555-1212’
phone = '(\d{3})-(\d{3})-(\d{4})’
phonePattern = re.compile(r‘’+phone)
plist = phonePattern.search(number)
print(plist)
Output:('800', '555', '1212')
```

Example 2

```
# \d is equivalent to [0-9].
p = re.compile('\d’)
print(p.findall("I left at 11 A.M. on 4th July 1886"))
Output:['1', '1', '4', '1', '8', '8', '6’]

# \d+ will match a group on [0-9], group of one or greater size
p = re.compile('\d+’)
print(p.findall("I left at 11 A.M. on 4th July 1886"))
Output:['11', '4', '1886']
```

Summary

- Regular Expressions (Regex) is a language for parsing strings.
- The language is cryptic and complex and because of this takes some time to master, unlike Python.
- Regex is based on a standard, so learning it in one language will transfer to other languages with minor exceptions.

Cheat Sheet

`.`	Any character except newline.
`\.`	A period (and so on for `\*`, `\(`, `\\`, etc.)
`^`	The start of the string.
`$`	The end of the string.
`\d`,`\w`,`\s`	A digit, word character `[A-Za-z0-9_]`, or whitespace.
`\D`,`\W`,`\S`	Anything except a digit, word character, or whitespace.
`[abc]`	Character a, b, or c.
`[a-z]`	a through z.
`[^abc]`	Any character except a, b, or c.
`aa|bb`	Either aa or bb.
`?`	Zero or one of the preceding element.
`*`	Zero or more of the preceding element.
`+`	One or more of the preceding element.
`{`*n*`}`	Exactly *n* of the preceding element.
`{`*n*`,}`	*n* or more of the preceding element.
`{`*m*`,`*n*`}`	Between *m* and *n* of the preceding element.
`??`,`*?`,`+?`, `{`*n*`}?`, etc.	Same as above, but as few times as possible.
(*expr*)	Capture *expr* for use with `\1`, etc.
(`?:`*expr*)	Non-capturing group.
(`?=`*expr*)	Followed by *expr*.
(`?!`*expr*)	Not followed by *expr*.