Session11-Pandas (RegEx in Python, Data Input and Output)

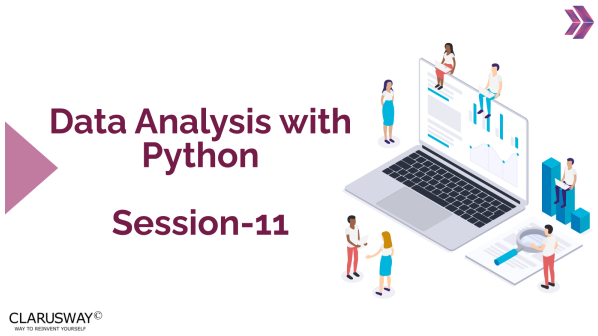
DAwithPython S11
Training Clarusway
Pear Deck - May 12, 2022 at 7:33PM

Part 1 - Summary

Use this space to summarize your thoughts on the lesson

Part 2 - Responses

Slide 1

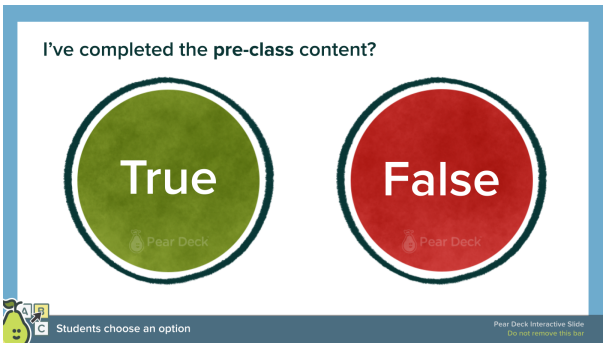


Use this space to take notes:

Slide 2

Use this space to take notes:

Slide 3



Your Response

You Chose
- **False**

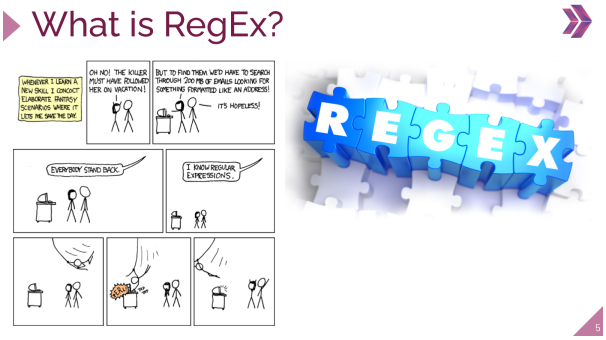
Other Choices
- True

Use this space to take notes:

Slide 4

Use this space to take notes:

Slide 5



Use this space to take notes:

Slide 6

What is RegEx?

- A regular expression is a **sequence of characters** that specifies a **search pattern** in text.
- Fields of application range from validation to **parsing/replacing strings**, passing through **translating data to other formats** and **web scraping**.
- It can be used in (almost) all programming languages (JavaScript, Java, VB, C #, C / C++, **Python**, Perl, Ruby, Delphi, R, Tcl, and many others) with the **slightest distinctions** about the support of the most advanced features and syntax versions supported by the engines.

6

Use this space to take notes:

Slide 7

Common Expressions

`\d`
Matches any decimal digit; this is equivalent to the class `[0-9]`.

`\D`
Matches any non-digit character; this is equivalent to the class `[^0-9]`.

`\s`
Matches any whitespace character; this is equivalent to the class `[ \t\n\r\f\v]`.

`\S`
Matches any non-whitespace character; this is equivalent to the class `[^ \t\n\r\f\v]`.

`\w`
Matches any alphanumeric character; this is equivalent to the class `[a-zA-Z0-9_]`.

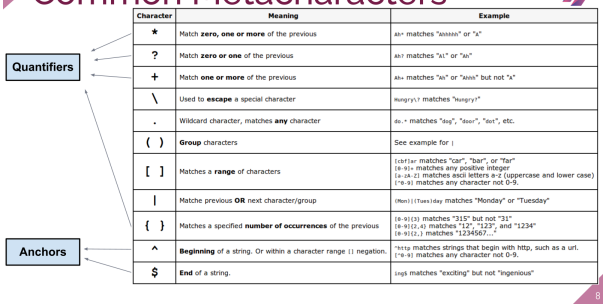
`\W`
Matches any non-alphanumeric character; this is equivalent to the class `[^a-zA-Z0-9_]`.

7

Use this space to take notes:

Slide 8

Common Metacharacters



Character	Meaning	Example
*	Match **zero, one or more** of the previous	`ab*` matches "a", "ab", or "abbbb"
?	Match **zero or one** of the previous	`ab?` matches "a" or "ab"
+	Match **one or more** of the previous	`ab+` matches "ab" or "abbb" but not "a"
\	Used to **escape** a special character	`Hungry\?` matches "Hungry?"
.	Wildcard character, matches **any** character	`do.*` matches "dog", "door", "dot", etc.
( )	**Group** characters	See example for |
[ ]	Matches a **range** of characters	`[cbf]ar` matches "car", "bar", or "far" `[0-9]+` matches any positive integer `[a-zA-Z]` matches ascii letters a-z (uppercase and lower case) `[^0-9]` matches any character not 0-9.
|	Matche previous **OR** next character/group	`(Mon)|(Tues)day` matches "Monday" or "Tuesday"
{ }	Matches a specified **number of occurrences** of the previous	`[0-9]{3}` matches "315" but not "31" `[0-9]{2,4}` matches "12", "123", and "1234" `[0-9]{2,}` matches "1234567..."
^	**Beginning** of a string. Or within a character range [] negation.	`^http` matches strings that begin with http, such as a url. `[^0-9]` matches any character not 0-9.
$	**End** of a string.	`ing$` matches "exciting" but not "ingenious"

Use this space to take notes:

Slide 9

Raw String (r/R"string")

- Python raw string is created by prefixing a string literal with **'r'** or **'R'**.
- Python raw string treats backslash (\) as a literal character. This is useful when we want to have a string that contains backslash and don't want it to be treated as an escape character.


```
my_string = "Hello\nWorld"
print(my_string)
```

```
Hello
World
```

```
my_string = r"Hello\nWorld"
print(my_string)
```

```
Hello\nWorld
```


Use this space to take notes:

Slide 10

Expressions and Special Characters

expression	matches...
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the *beginning* of the string
abc$	abc at the *end* of the string
a|b	either of a and b
^abc|abc$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabcacaa)
[^abc]+	any (nonempty) string which does *not* contain any of a, b and c (such as defg)
\d\d	any two decimal digits, such as 42; same as \d{2}
\w+	a "word": a nonempty sequence of alphanumeric characters and low lines (underscores), such as foo and 12bar8 and foo_1
100\s*mk	the strings 100 and mk optionally separated by any amount of white space (spaces, tabs, newlines)
abc\b	abc when followed by a word boundary (e.g. in abc! but not in abcd)

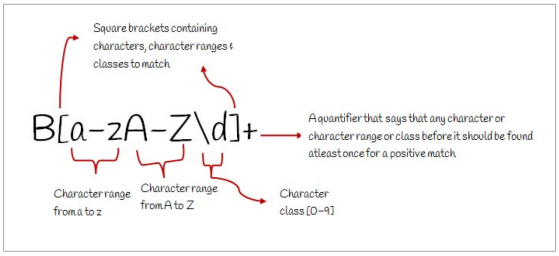
Some Basic Examples

10

Use this space to take notes:

Slide 11

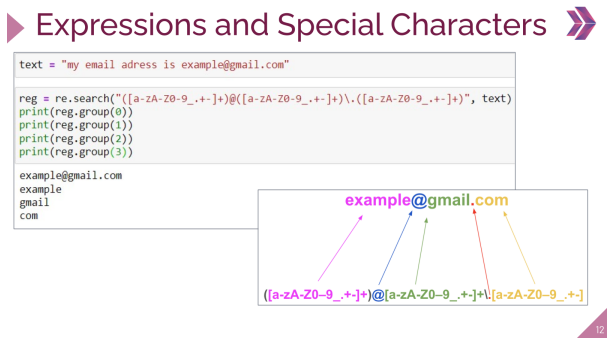
Expressions and Special Characters



11

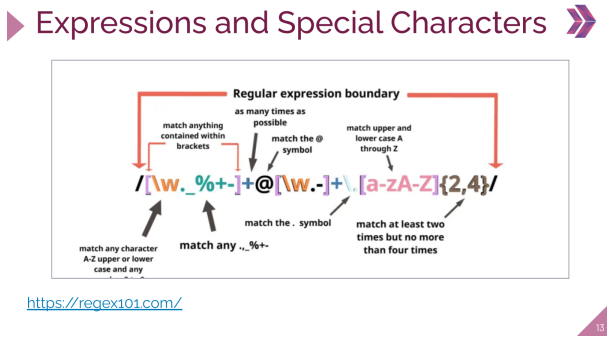
Use this space to take notes:

Slide 12



Use this space to take notes:

Slide 13



Link(s) on this slide:

- https://regex101.com/

Use this space to take notes:

Slide 14

Common Python RegEx Functions

- **re.search()** : Scan through string looking for a match to the pattern.
- **re.match()** : Try to apply the pattern at the start of the string.
- **re.fullmatch()** : Try to apply the pattern to all of the string.
- **re.findall()** : Return a list of all non-overlapping matches in the string.
- **re.sub()** : Return the string obtained by replacing the leftmost non-overlapping occurrences of the pattern in string by the replacement repl.
- **re.split()** : Split the source string by the occurrences of the pattern, returning a list containing the resulting substrings.

Use this space to take notes:

Slide 15

Pandas Functions Accepting RegEx

- **count()** : Count occurrences of pattern in each string of the Series/Index
- **replace()** : Replace each occurrence of pattern/regex in the Series/Index.
- **contains()**: Test if pattern or regex is contained within a string of a Series or Index.
- **findall()** : Find all occurrences of pattern or regex in the Series/Index.
- **match()** : Determine if each string matches a regular expression.
- **split()** : Split strings around given separator/delimiter.
- **extract()** : Extract capture groups in the regex pat as columns in a DataFrame and returns the captured groups

Use this space to take notes:

Slide 16

RegEx Func. vs Pandas Func.

Common Python RegEx Func	Pandas Func. Accepting RegEx
re.search()	count()
re.match()	replace()
re.fullmatch()	contains()
re.findall()	findall()
re.sub()	match()
re.split()	split()
	extract()

16

Use this space to take notes:

Slide 17

Data Input & Output

CLARUSWAY©

Use this space to take notes:

Slide 18

Data Input & Output

Input	Output
pd.read_csv("file_name")	df.to_csv("file_name")
pd.read_excel()	pd.to_excel()
pd.read_jason()	pd.to_jason()
pd.read_html()	pd.to_html()
pd.read_sql()	pd.to_sql()

18

Use this space to take notes:

Slide 19

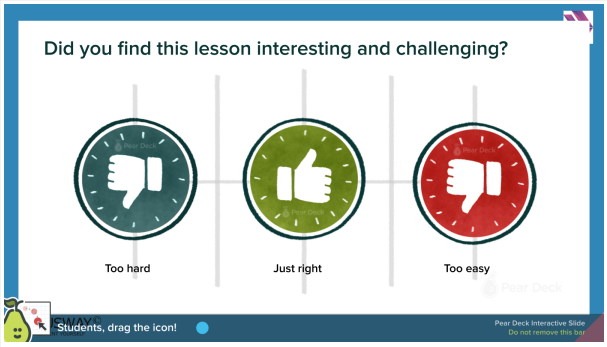
Data Analysis with Python

let's start the hands-on phase

19

Use this space to take notes:

Slide 20	Your Response



Use this space to take notes:

Slide 21

Use this space to take notes: