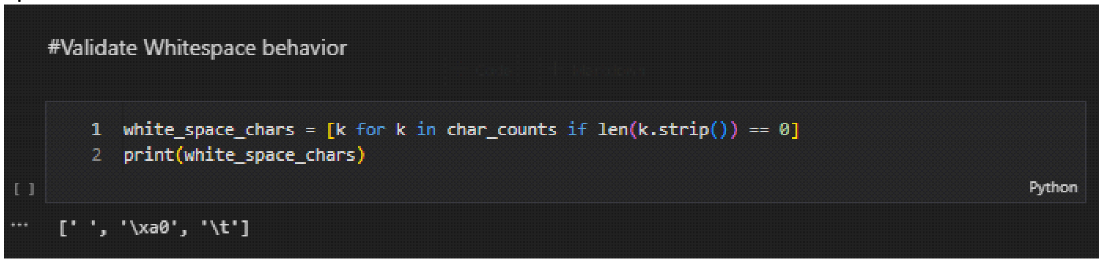
Language Processing

Tuesday, July 26, 2022 4:12 PM

Assessing and verifying text and python functions with the characters found in this text

Whitespace

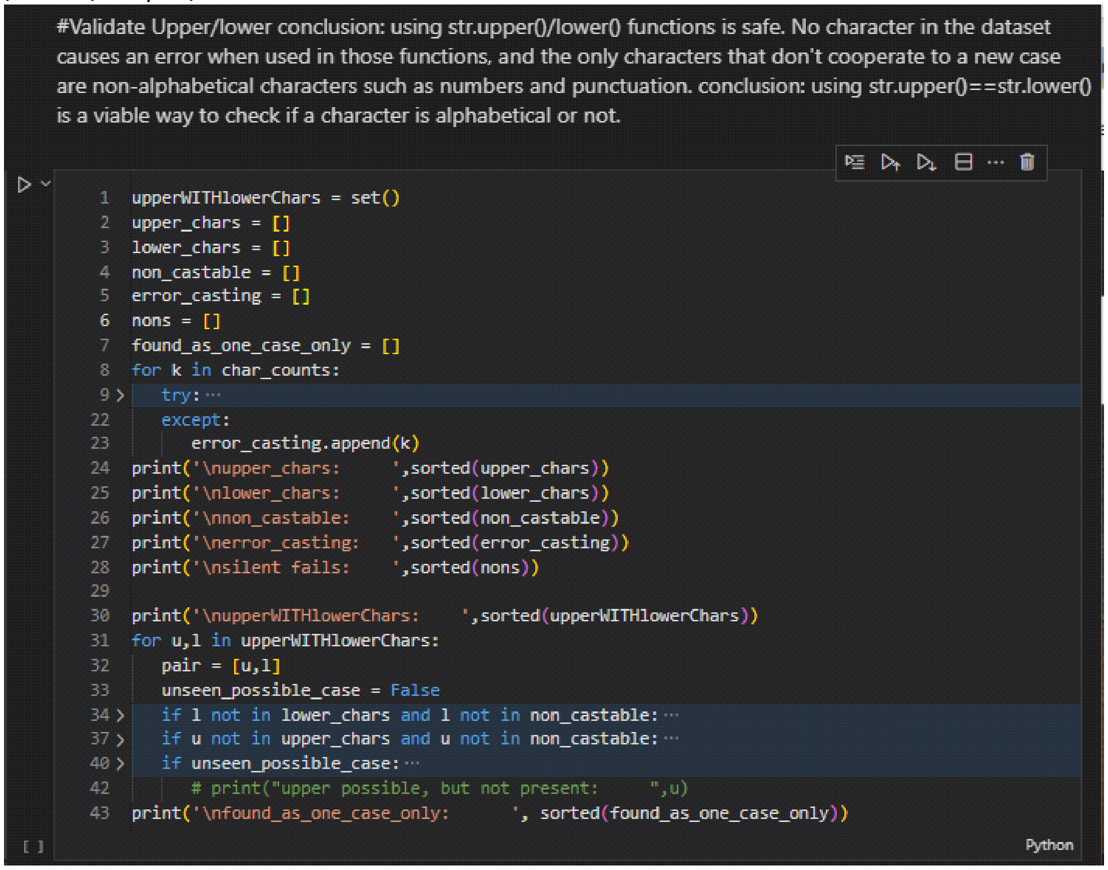


#Validate Whitespace behavior

```python
white_space_chars = [k for k in char_counts if len(k.strip()) == 0]
print(white_space_chars)
```

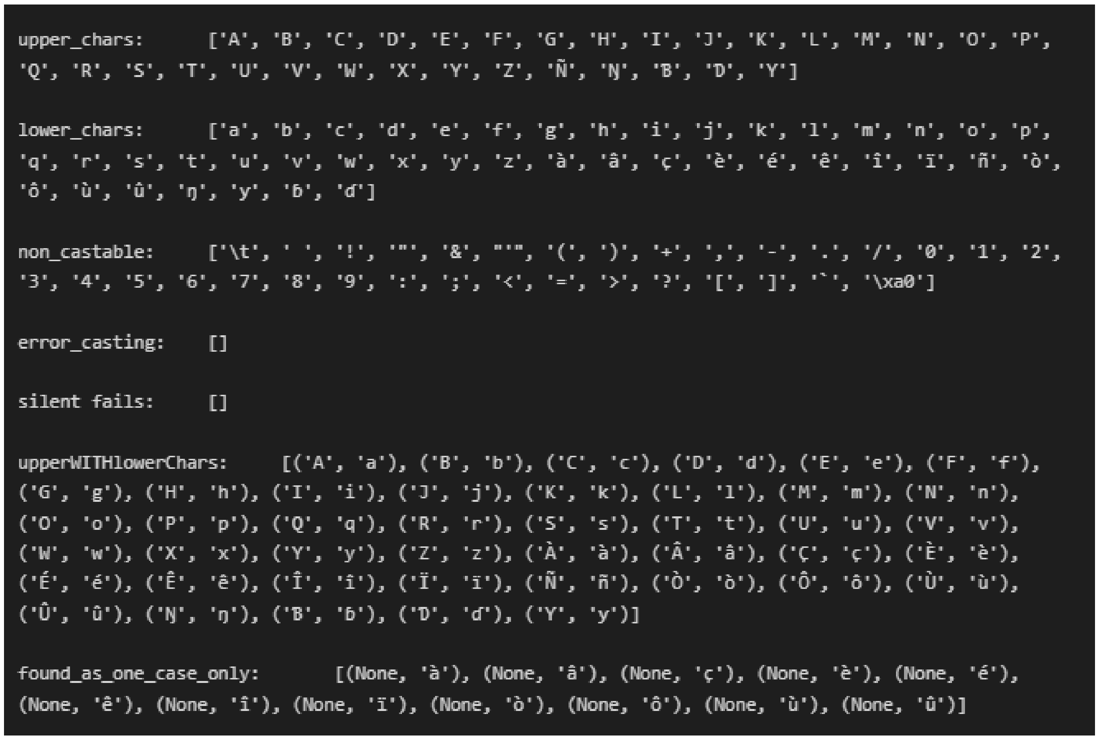
```
['  ', '\xa0', '\t']
```

Upper/Lower/IsAlpha/Punctuation

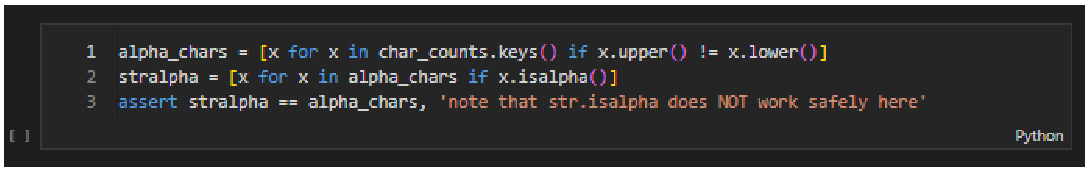


#Validate Upper/lower conclusion: using str.upper()/lower() functions is safe. No character in the dataset causes an error when used in those functions, and the only characters that don't cooperate to a new case are non-alphabetical characters such as numbers and punctuation. conclusion: using str.upper()==str.lower() is a viable way to check if a character is alphabetical or not.

```python
upperWITHlowerChars = set()
upper_chars = []
lower_chars = []
non_castable = []
error_casting = []
nons = []
found_as_one_case_only = []
for k in char_counts:
    try:···
    except:
        error_casting.append(k)
print('\nupper_chars:      ',sorted(upper_chars))
print('\nlower_chars:      ',sorted(lower_chars))
print('\nnon_castable:     ',sorted(non_castable))
print('\nerror_casting:    ',sorted(error_casting))
print('\nsilent fails:     ',sorted(nons))

print('\nupperWITHlowerChars:     ',sorted(upperWITHlowerChars))
for u,l in upperWITHlowerChars:
    pair = [u,l]
    unseen_possible_case = False
    if l not in lower_chars and l not in non_castable:···
    if u not in upper_chars and u not in non_castable:···
    if unseen_possible_case:···
        # print("upper possible, but not present:     ",u)
print('\nfound_as_one_case_only:      ', sorted(found_as_one_case_only))
```



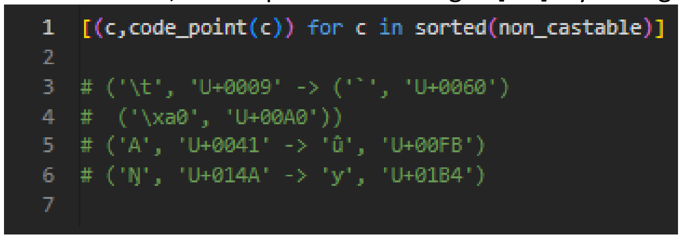
```
upper_chars:      ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'Ñ', 'Ŋ', 'Ɓ', 'Ɗ', 'Ƴ']

lower_chars:      ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'à', 'â', 'ç', 'è', 'é', 'ê', 'î', 'ï', 'ñ', 'ò',
'ô', 'ù', 'û', 'ŋ', 'ƴ', 'ɓ', 'ɗ']

non_castable:     ['\t', ' ', '!', '"', '&', "'", '(', ')', '+', ',', '-', '.', '/', '0', '1', '2',
'3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '[', ']', '`', '\xa0']

error_casting:    []

silent fails:     []

upperWITHlowerChars:     [('A', 'a'), ('B', 'b'), ('C', 'c'), ('D', 'd'), ('E', 'e'), ('F', 'f'),
('G', 'g'), ('H', 'h'), ('I', 'i'), ('J', 'j'), ('K', 'k'), ('L', 'l'), ('M', 'm'), ('N', 'n'),
('O', 'o'), ('P', 'p'), ('Q', 'q'), ('R', 'r'), ('S', 's'), ('T', 't'), ('U', 'u'), ('V', 'v'),
('W', 'w'), ('X', 'x'), ('Y', 'y'), ('Z', 'z'), ('À', 'à'), ('Â', 'â'), ('Ç', 'ç'), ('È', 'è'),
('É', 'é'), ('Ê', 'ê'), ('Î', 'î'), ('Ï', 'ï'), ('Ñ', 'ñ'), ('Ò', 'ò'), ('Ô', 'ô'), ('Ù', 'ù'),
('Û', 'û'), ('Ŋ', 'ŋ'), ('Ɓ', 'ɓ'), ('Ɗ', 'ɗ'), ('Ƴ', 'ƴ')]

found_as_one_case_only:      [(None, 'à'), (None, 'â'), (None, 'ç'), (None, 'è'), (None, 'é'),
(None, 'ê'), (None, 'î'), (None, 'ï'), (None, 'ò'), (None, 'ô'), (None, 'ù'), (None, 'û')]
```



```python
alpha_chars = [x for x in char_counts.keys() if x.upper() != x.lower()]
stralpha = [x for x in alpha_chars if x.isalpha()]
assert stralpha == alpha_chars, 'note that str.isalpha does NOT work safely here'
```

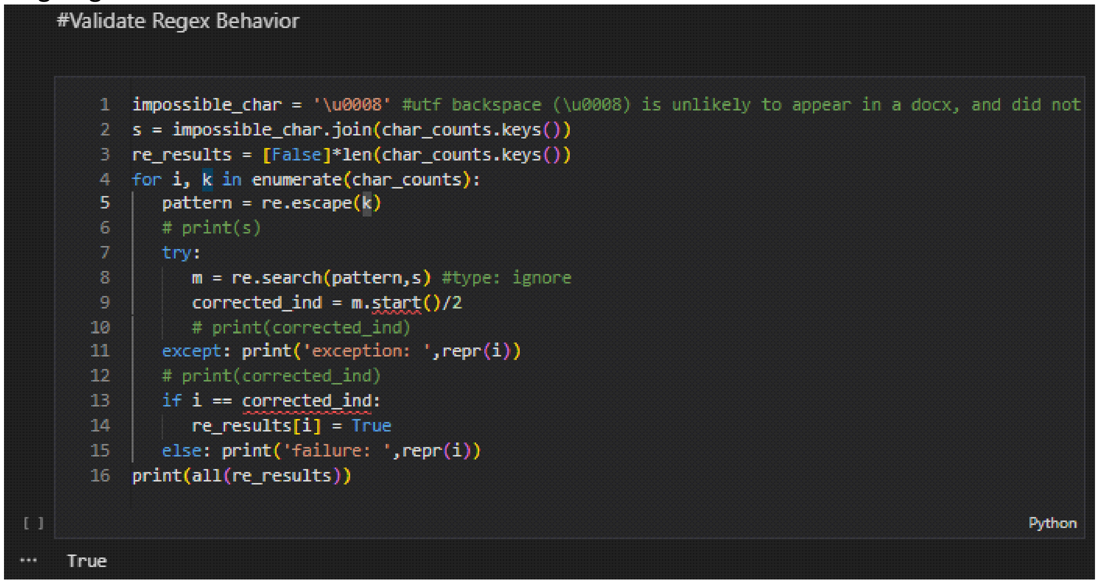
[ ] Python

Hex/Unicode values, and implications on regex [a-z] style ranges



```python
[(c,code_point(c)) for c in sorted(non_castable)]

# ('\t', 'U+0009' -> ('`', 'U+0060')
#  ('\xa0', 'U+00A0'))
# ('A', 'U+0041' -> 'û', 'U+00FB')
# ('Ŋ', 'U+014A' -> 'ƴ', 'U+01B4')
```

Validating Regex basics



#Validate Regex Behavior

```python
impossible_char = '\u0008' #utf backspace (\u0008) is unlikely to appear in a docx, and did not
s = impossible_char.join(char_counts.keys())
re_results = [False]*len(char_counts.keys())
for i, k in enumerate(char_counts):
    pattern = re.escape(k)
    # print(s)
    try:
        m = re.search(pattern,s) #type: ignore
        corrected_ind = m.start()/2
        # print(corrected_ind)
    except: print('exception: ',repr(i))
    # print(corrected_ind)
    if i == corrected_ind:
        re_results[i] = True
    else: print('failure: ',repr(i))
print(all(re_results))
```

[ ] Python

```
... True
```