

Practice 9

COMP9021, Term 3, 2019

1 Word ladders

Write a program `word_ladder.py` that computes all transformations of a word `word_1` into a word `word_2`, consisting of sequences of words of minimal length, starting with `word_1`, ending in `word_2`, and such that two consecutive words in the sequence differ by at most one letter. All words have to occur in a dictionary with name `dictionary.txt`, stored in the working directory.

It is convenient and effective to first create a dictionary whose keys are all words in the dictionary with one letter replaced by a “slot”, the value for a given key being the list of words that match the key with the “slot” being replaced by an appropriate letter. From this dictionary, one can then build a dictionary with words as keys, and as value for a given key the list of words that differ in only one letter from the key.

The program implements a function `word_ladder(word_1, word_2)` that returns the list of all solutions, a solution being as previously described.

Next is a possible interaction.

```
$ python3  
...  
>>> from word_ladder import *  
>>> for ladder in word_ladder('cold', 'warm'): print(ladder)  
...  
['COLD', 'CORD', 'WORD', 'WORM', 'WARM']  
['COLD', 'CORD', 'WORD', 'WARD', 'WARM']  
['COLD', 'CORD', 'CARD', 'WARD', 'WARM']  
>>> for ladder in word_ladder('three', 'seven'): print(ladder)  
...  
['THREE', 'THREW', 'SHREW', 'SHRED', 'SIRED', 'SITED', 'SATED', 'SAVED', 'SAVER', 'SEVER', 'SEVEN']  
  
>>> for ladder in word_ladder('train', 'bikes'): print(ladder)  
...  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'BROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'BROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'BROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'BROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'BORES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'BROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'PORES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'BORES', 'BAKES', 'BIKES']  
['TRAIN', 'BRAIN', 'BRAUN', 'BROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'PORES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'GROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'GROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'GROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'GROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'BAKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'GROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'BORES', 'BAKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'GROWS', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'PORES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'POKES', 'PIKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'CAKES', 'BAKES', 'BIKES']  
['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CARES', 'BAKES', 'BIKES']
```

['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'BORES', 'BARES', 'BAKES', 'BIKES']
 ['TRAIN', 'GRAIN', 'GROIN', 'GROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'PORES', 'POKES', 'PIKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'CAKES', 'BAKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'COKES', 'POKES', 'PIKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CAKES', 'BAKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'CAKES', 'BAKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'BORES', 'BARES', 'BAKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DROWN', 'CROWN', 'CROWS', 'CROPS', 'COOPS', 'CORPS', 'CORES', 'PORES', 'POKES', 'PIKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DRAWS', 'DRAGS', 'BRAGS', 'BRATS', 'BEATS', 'BESTS', 'BUSTS', 'BUSES', 'BASES', 'BAKES', 'BIKES']
 ['TRAIN', 'DRAIN', 'DRAWN', 'DRAWS', 'DRAGS', 'BRAGS', 'BRATS', 'BEATS', 'BELTS', 'BELLS', 'BALLS', 'BALES', 'BAKES', 'BIKES']

2 The Word Search puzzle

The Word Search puzzle consists of a grid of letters and a number of words, that have to be read horizontally, vertically or diagonally, in either direction. Write a program `word_search.py` that defines a class `WordSearch` with the following properties.

- To create a `WordSearch` object, the name of a file has to be provided. This file is meant to store a number of lines all with the same number of uppercase letters, those lines possibly containing spaces anywhere, and the file possibly containing extra blank lines.
- `__str__()` is implemented.
- It has a method `number_of_solutions()` to display the number of solutions for each word length for which a solution exists.
- It has a method `locate_word_in_grid()` that takes a word as argument; it returns `None` if the word cannot be read in the grid, and otherwise returns the x and y coordinates of an occurrence of the first letter of the word in the grid and the direction to follow (N, NE, E, SE, S, SW, W or NW) to read the whole word from that point onwards. Coordinates start from 0, with the x-axis pointing East, and the y-axis pointing South.
- It has a method `locate_words_in_grid()` that takes any number of words as arguments, and returns a dictionary whose keys are those words and whose values are `None` or the triple returned by `locate_word_in_grid()` when called with that word as argument.
- It has a method `display_word_in_grid()` that takes a word as argument and in case the word can be read from the grid, prints out the grid with all characters being displayed in lowercase, except for those that make up word, displayed in uppercase.

Here is a possible interaction.

```
$ cat word_search_1.txt
N D A O E L D L O G B M N E
I T D C M E A I N R U T S L
C L U U E I C G G G O L I I
K M U I M U I D I R I A L T
E U R T U N G S T E N B V H
L I L S L T T U L R U O E I
C M A T E T I U R D R C R U
I D S C A M A G N E S I U M
M A M P D M U I N A T I T I
P C N P L A T I N U M D L L
H Z E M A N G A N E S E I G
M G I T I N R U N O R I T C
R I A N N A M E R C U R Y N
U O T C C R E P P O C E E R
```

```

$ python3
...
>>> from word_search import *
>>> import pprint
>>> ws = WordSearch('word_search_1.txt')
>>> print(ws)
N D A O E L D L O G B M N E
I T D C M E A I N R U T S L
C L U U E I C G G G O L I I
K M U I M U I D I R I A L T
E U R T U N G S T E N B V H
L I L S L T T U L R U O E I
C M A T E T I U R D R C R U
I D S C A M A G N E S I U M
M A M P D M U I N A T I T I
P C N P L A T I N U M D L L
H Z E M A N G A N E S E I G
M G I T I N R U N O R I T C
R I A N N A M E R C U R Y N
U O T C C R E P P O C E E R
>>> metal = 'PLATINUM'
>>> print(f'{metal}: ws.locate_word_in_grid(metal)')
PLATINUM: (3, 9, 'E')
>>> metal = 'SODIUM'
>>> print(f'{metal}: ws.locate_word_in_grid(metal)')
SODIUM: None
>>> metals = ('PLATINUM', 'COPPER', 'MERCURY', 'TUNGSTEN', 'MAGNESIUM', 'ZINC', 'MANGANESE',
... 'TITANIUM', 'TIN', 'IRON', 'LITHIUM', 'CADMIUM', 'GOLD', 'COBALT', 'SILVER',
... 'NICKEL', 'LEAD', 'IRIDIUM', 'URANIUM', 'SODIUM')
>>> located_metals = ws.locate_words_in_grid(*metals)
>>> pprint.pprint(located_metals)
{'CADMIUM': (1, 9, 'N'),
 'COBALT': (11, 6, 'N'),
 'COPPER': (10, 13, 'W'),
 'GOLD': (9, 0, 'W'),
 'IRIDIUM': (10, 3, 'W'),
 'IRON': (11, 11, 'W'),
 'LEAD': (4, 5, 'S'),
 'LITHIUM': (13, 1, 'S'),
 'MAGNESIUM': (5, 7, 'E'),
 'MANGANESE': (3, 10, 'E'),
 'MERCURY': (6, 12, 'E'),
 'NICKEL': (0, 0, 'S'),
 'PLATINUM': (3, 9, 'E'),
 'SILVER': (12, 1, 'S'),
 'SODIUM': None,
 'TIN': (6, 9, 'NE'),
 'TITANIUM': (12, 8, 'W'),
 'TUNGSTEN': (3, 4, 'E'),
 'URANIUM': None,
 'ZINC': (1, 10, 'SE')}
>>> for metal in metals:
...     print(metal, end = ':\n')
...     ws.display_word_in_grid(metal)
...     print()
...
PLATINUM:
n d a o e l d l o g b m n e
i t d c m e a i n r u t s l
c l u u e i c g g g o l i i
k m u i m u i d i r i a l t
e u r t u n g s t e n b v h
l i l s l t t u l r u o e i
c m a t e t i u r d r c r u
i d s c a m a g n e s i u m
m a m p d m u i n a t i t i
p c n P L A T I N U M d l l
h z e m a n g a n e s e i g
m g i t i n r u n o r i t c
r i a n n a m e r c u r y n
u o t c c r e p p o c e e r

COPPER:
n d a o e l d l o g b m n e
i t d c m e a i n r u t s l
c l u u e i c g g g o l i i
k m u i m u i d i r i a l t
e u r t u n g s t e n b v h
l i l s l t t u l r u o e i
c m a t e t i u r d r c r u
i d s c a m a g n e s i u m
m a m p d m u i n a t i t i
p c n p l a t i n u m d l l
h z e m a n g a n e s e i g
m g i t i n r u n o r i t c
r i a n n a m e r c u r y n
u o t c c R E P P O C e e r

MERCURY:
n d a o e l d l o g b m n e
i t d c m e a i n r u t s l
c l u u e i c g g g o l i i

```

kmuiuidirialt
eurtungstenbvh
lilslttulruoei
cmattet iurdr cr u
idscamagnesium
mampdmuinatiti
pcnplatinumdll
hzemanganeseig
mgitinrunoritc
riannaMERCURYn
uotcreppoceer

TUNGSTEN:

ndaoelddlogbmne
itdcmeainruts l
cluueicgggoli i
kmuiuidirialt
eurTUNGSTENbvh
lilslttulruoei
cmattet iurdr cr u
idscamagnesium
mampdmuinatiti
pcnplatinumdll
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotcreppoceer

MAGNESIUM:

ndaoelddlogbmne
itdcmeainruts l
cluueicgggoli i
kmuiuidirialt
eurtungstenbvh
lilslttulruoei
cmattet iurdr cr u
idscamagnesium
mampdmuinatiti
pcnplatinumdll
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotcreppoceer

ZINC:

ndaoelddlogbmne
itdcmeainruts l
cluueicgggoli i
kmuiuidirialt
eurtungstenbvh
lilslttulruoei
cmattet iurdr cr u
idscamagnesium
mampdmuinatiti
pcnplatinumdll
hZemanganeseig
mgItinrunoritc
riaNnamercuryn
uotcCreppoceer

MANGANESE:

ndaoelddlogbmne
itdcmeainruts l
cluueicgggoli i
kmuiuidirialt
eurtungstenbvh
lilslttulruoei
cmattet iurdr cr u
idscamagnesium
mampdmuinatiti
pcnplatinumdll
hzeMANGANESEig
mgitinrunoritc
riannamercuryn
uotcreppoceer

TITANIUM:

ndaoelddlogbmne
itdcmeainruts l
cluueicgggoli i
kmuiuidirialt
eurtungstenbvh
lilslttulruoei
cmattet iurdr cr u
idscamagnesium
mampdMUINATITi
pcnplatinumdll
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotcreppoceer

TIN:

ndaoelddlogbmne
itdcmeainruts1
cluueicgggoli
kmuimuidirialt
eurtungstenbvh
lilsltttulruoei
cmattet iurdr cr
idscamagNesium
mampdmuInatiti
pcnplaTinumd11
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

IRON:

ndaoelddlogbmne
itdcmeainruts1
cluueicgggoli
kmuimuidirialt
eurtungstenbvh
lilsltttulruoei
cmattet iurdr cr
idscamagnesium
mampdmuinatiti
pcnplatinumd11
hzemanganeseig
mgitinrunORItc
riannamercuryn
uotccreppoceer

LITHIUM:

ndaoelddlogbmne
itdcmeainrutsL
cluueicgggoliI
kmuimuidirialT
eurtungstenbvh
lilsltttulruoeI
cmattet iurdr crU
idscamagnesium
mampdmuinatiti
pcnplatinumd11
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

CADMIUM:

ndaoelddlogbmne
itdcmeainruts1
cluueicgggoli
kMuimuidirialt
eUrtungstenbvh
lIlsltttulruoei
cMatettiurdr cr
iDscamagnesium
mAmpdmuinatiti
pCnplatinumd11
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

GOLD:

ndaoelDLOGbmne
itdcmeainruts1
cluueicgggoli
kmuimuidirialt
eurtungstenbvh
lilsltttulruoei
cmattet iurdr cr
idscamagnesium
mampdmuinatiti
pcnplatinumd11
hzemanganeseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

COBALT:

ndaoelddlogbmne
itdcmeainruTsl
cluueicgggoLi
kmuimuidiriAlt
eurtungstenBvh
lilsltttulruOei
cmattet iurdrCru
idscamagnesium
mampdmuinatiti
pcnplatinumd11
hzemanganeseig

mgitinrunoritc
riannamercuryn
uotccreppoceer

SILVER:

ndaoelddlogbmne
itdcmeainrutsSl
cluueicgggollii
kmuiuidirialt
eurtungstenbVh
lilslttulruoEi
cmattetiurdrCRu
idscamagnesium
mampdmuinatititi
pcnplatinumdll
hzemanganesseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

NICKEL:

Ndaoelddlogbmne
ItdcmeainrutsSl
Cluueicgggollii
Kmuimuidirialt
Eurtungstenbvh
Lilslttulruoei
cmattetiurdrCRu
idscamagnesium
mampdmuinatititi
pcnplatinumdll
hzemanganesseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

LEAD:

ndaoelddlogbmne
itdcmeainrutsSl
cluueicgggollii
kmuiuidirialt
eurtungstenbvh
lilslttulruoei
cmatEtiurdrCRu
idscAmagnesium
mampDmuinatititi
pcnplatinumdll
hzemanganesseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

IRIDIUM:

ndaoelddlogbmne
itdcmeainrutsSl
cluueicgggollii
kmuiMUIDIRIalt
eurtungstenbvh
lilslttulruoei
cmattetiurdrCRu
idscamagnesium
mampdmuinatititi
pcnplatinumdll
hzemanganesseig
mgitinrunoritc
riannamercuryn
uotccreppoceer

URANIUM:

SODIUM:

3 Possible subtractions yielding a given sum

Write a program `subtractions.py` that takes as input an iterable `L` of nonnegative integers and an integer `N`, and displays all ways of inserting minus signs and parentheses in `L`, resulting in an expression that evaluates to `N`. For this question we make use of `eval()`.

Next is a possible interaction.

```
$ python3
...
>>> from subtractions import *
>>> subtractions((1, 2, 3, 4, 5), 1)
1 - ((2 - 3) - (4 - 5))
(1 - ((2 - 3) - 4)) - 5
>>> subtractions((1, 2, 3, 4, 5), 2)
>>> subtractions((1, 2, 3, 4, 5), 3)
1 - (2 - (3 - (4 - 5)))
1 - ((2 - (3 - 4)) - 5)
(1 - (2 - 3)) - (4 - 5)
>>> subtractions((1, 2, 3, 4, 5), 4)
>>> subtractions((1, 2, 3, 4, 5), 5)
(1 - 2) - ((3 - 4) - 5)
>>> subtractions((1, 3, 2, 5, 11, 9, 10, 8, 4, 7, 6), 40)
1 - (((((3 - 2) - 5) - 11) - (9 - (((10 - 8) - 4) - 7) - 6)))
1 - ((((((3 - 2) - 5) - 11) - 9) - 10) - (8 - (4 - (7 - 6))))
1 - (((((((3 - 2) - 5) - 11) - 9) - 10) - ((8 - (4 - 7)) - 6))
1 - (((((((((3 - 2) - 5) - 11) - 9) - 10) - (8 - 4)) - (7 - 6))
1 - ((((((3 - 2) - 5) - 11) - (9 - (((10 - 8) - 4) - 7))) - 6)
1 - (((((((3 - 2) - 5) - 11) - (9 - ((10 - 8) - 4))) - 7) - 6)
1 - (((((((((3 - 2) - 5) - 11) - (9 - (10 - 8))) - 4) - 7) - 6)
1 - ((((((((((3 - 2) - 5) - 11) - (9 - 10)) - 8) - 4) - 7) - 6)
(1 - 3) - (((((2 - 5) - 11) - 9) - (10 - (((8 - 4) - 7) - 6)))
(1 - 3) - ((((((2 - 5) - 11) - 9) - (10 - ((8 - 4) - 7))) - 6)
(1 - 3) - (((((((2 - 5) - 11) - 9) - (10 - (8 - 4))) - 7) - 6)
(1 - 3) - (((((((((2 - 5) - 11) - 9) - (10 - 8)) - 4) - 7) - 6)
(1 - (((((3 - 2) - 5) - 11) - 9)) - (((((10 - 8) - 4) - 7) - 6)
((1 - 3) - (((((2 - 5) - 11) - 9) - 10)) - (((8 - 4) - 7) - 6)
(1 - (((((((3 - 2) - 5) - 11) - 9) - 10) - 8)) - (4 - (7 - 6))
(1 - (((((((3 - 2) - 5) - 11) - 9) - 10) - (8 - (4 - 7)))) - 6
(1 - (((((((((3 - 2) - 5) - 11) - 9) - 10) - (8 - 4)) - 7)) - 6
((1 - (((((((3 - 2) - 5) - 11) - 9) - 10) - 8)) - (4 - 7)) - 6
```


4 Voting systems (optional)

Find out (e.g., in Wikipedia) about these voting systems: (a) one round method, (b) two round method, (c) elimination method, (d) De Borda count, and (e) De Condorcet count.

The elimination method works as follows. One adds up the tallies of all candidates who rank 1st and eliminate the candidate(s) who get the minimal number of votes (as ranked 1st candidates). For a given ordering, the candidates who remain and were ranked after the eliminated candidate(s) see their ranking go up so that the ordering is preserved, and rankings range from 1 up to the number of candidates that remain (for instance, if to start with, there are 5 candidates, A, B, C, D and E who are ranked 1, 2, 3, 4 and 5, respectively, and if B and D are eliminated because they get the least number of votes as 1st candidates across all rankings, then for that particular ranking, A remains ranked 1st, C becomes ranked 2nd, and E becomes ranked third). The process is repeated until there is only one candidate left, or all candidates that remain get exactly the same number of votes as preferred candidates.

Then design a program `election.py` that defines a class `Election`, with objects of this class created from Excel files of the kind provided as examples, to which the methods

- `one_round_winners()`,
- `two_round_winners()`,
- `elimination_winner()`,
- `de_borda_winners()`, and
- `de_condorcet_winners()`

can be applied. Also, the `__str__()` method is implemented so as to display in textual form the election results recorded in the Excel file.

Next is a possible interaction.

```

$ python3
...
>>> from election import *
>>> election = Election('election_1.xlsx')
>>> print(election)
Number of votes  Albert  Emily  Oscar  Maria  Max
3273            1      5      4      2      3
2182            5      1      4      3      2
1818            5      2      1      4      3
1636            5      4      2      1      3
727             5      2      4      3      1
364             5      4      2      3      1
>>> election.one_round_winners()
The winner is Albert.
>>> election.two_round_winners()
The winner is Emily.
>>> election.elimination_winners()
The winner is Oscar.
>>> election.de_borda_winners()
The winner is Maria.
>>> election.de_condorcet_winners()
The winner is Max.
>>> election = Election('election_2.xlsx')
Number of votes  Albert  Emily  Oscar  Maria  Max
1000            1      2      3      4      5
>>> election.one_round_winners()
The winner is Albert.
>>> election.two_round_winners()
The winner is Albert.
>>> election.elimination_winners()
The winner is Max.
>>> election.de_borda_winners()
The winner is Albert.
>>> election.de_condorcet_winners()
The winner is Albert.

```

```

>>> election = Election('election_3.xlsx')
>>> print(election)
Number of votes  Albert
      1000         1
      1000         1
      1000         1
      1000         1
      1000         1
      1000         1
>>> election.one_round_winners()
All candidates are winners.
>>> election.two_round_winners()
All candidates are winners.
>>> election.elimination_winners()
All candidates are winners.
>>> election.de_borda_winners()
All candidates are winners.
>>> election.de_condorcet_winners()
All candidates are winners.
>>> election = Election('election_4.xlsx')
>>> print(election)
Number of votes  Albert  Emily  Oscar
      1000         1      2      3
      1000         2      1      3
>>> election.one_round_winners()
The winners are Albert and Emily.
>>> election.two_round_winners()
The winners are Albert and Emily.
>>> election.elimination_winners()
The winner is Oscar.
>>> election.de_borda_winners()
The winners are Albert and Emily.
>>> election.de_condorcet_winners()
The winners are Albert and Emily.

```

```

>>> election = Election('election_5.xlsx')
>>> print(election)
Number of votes  Albert  Emily  Oscar  Maria
1000             1       2       3       4
1000             2       3       1       4
1000             3       1       2       4
>>> election.one_round_winners()
The winners are Albert, Emily and Oscar.
>>> election.two_round_winners()
The winners are Albert, Emily and Oscar.
>>> election.elimination_winners()
The winner is Maria.
>>> election.de_borda_winners()
The winners are Albert, Emily and Oscar.
>>> election.de_condorcet_winners()
There is no winner.
>>> election = Election('election_6.xlsx')
>>> print(election)
Number of votes  Albert  Emily  Oscar
1000             1       2       3
1000             2       1       3
250              2       3       1
250              3       2       1
>>> election.one_round_winners()
The winners are Albert and Emily.
>>> election.two_round_winners()
The winners are Albert and Emily.
>>> election.elimination_winners()
The winners are Albert and Emily.
>>> election.de_borda_winners()
The winners are Albert and Emily.
>>> election.de_condorcet_winners()
The winners are Albert and Emily.

```

5 Context free grammars (advanced, optional)

A *context free* grammar is a set of *production rules* of the form

$$\text{symbol}_0 \rightarrow \text{symbol}_1 \dots \text{symbol}_n$$

where $\text{symbol}_0, \dots, \text{symbol}_n$ are either *terminal* or *nonterminal symbols*, with symbol_0 being necessarily nonterminal. A symbol is a nonterminal symbol iff it is denoted by a word built from underscores or uppercase letters. A special nonterminal symbol is called the *start symbol*. The language *generated* by the grammar is the set of sequences of terminal symbols obtained by replacing a nonterminal symbol by the sequence on the right hand side of a rule having that nonterminal symbol on the left hand side, starting with the start symbol. For instance, the following, where **EXPRESSION** is the start symbol, is a context free grammar for a set of arithmetic expressions.

```
EXPRESSION --> EXPRESSION TERM_OPERATOR TERM
EXPRESSION --> TERM
TERM --> TERM FACTOR_OPERATOR FACTOR
TERM --> FACTOR
FACTOR --> NUMBER
FACTOR --> (EXPRESSION)
NUMBER --> DIGIT NUMBER
NUMBER --> DIGIT
DIGIT --> 0
...
DIGIT --> 9
TERM_OPERATOR --> +
TERM_OPERATOR --> -
FACTOR_OPERATOR --> *
FACTOR_OPERATOR --> /
```

Moreover, blank characters (spaces or tabs) can be inserted anywhere except inside a number. For instance, $(2 + 3) * (10 - 2) - 12 * (1000 + 15)$ is an arithmetic expression generated by the grammar.

Note that operators associate to the left. The grammar is *unambiguous*, in the sense that every expression generated by the grammar has a unique evaluation.

Write down a program `context_free_grammar.py` that implements a function `evaluate()` which takes a string representing an expression as an argument, checks whether the expression can be generated by the grammar, and in case the answer is yes, returns the value of the expression, provided that no division by 0 is attempted; otherwise, the function returns `None`.

Next is a possible interaction.

```
$ python3
...
>>> from context_free_grammar import *
>>> evaluate('100')
100
>>> evaluate('(100)')
100
>>> evaluate('1 - 20 + 300')
281
>>> evaluate('((((1))-((20))+((300))))')
281
>>> evaluate('20 * 4 / 5')
16.0
>>> evaluate('((((20))*((4))/((5))))')
16.0
>>> evaluate('1 + 20 * 30 - 400 / 500')
600.2
>>> evaluate('1 + (20*30-400) / 500')
1.4
>>> evaluate('1+(20 / 30 * 400)- 500')
-232.33333333333337
>>> evaluate('1 + 2 * (3+4*5) / (6*7-8/9)')
2.1189189189189186
>>> evaluate('100')
100
>>> evaluate('100 + ')
100
>>> evaluate('100 + -3')
97
>>> evaluate('100 # 50')
100
>>> evaluate('100 / 0')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "context_free_grammar.py", line 10, in evaluate
    raise ValueError("Division by zero")
ValueError: Division by zero
```

Before you tackle the exercise, find out about *recursive descent parsers*. To easily tokenise the string, check out the `findall()` function from the `re` module.