Parsing

Preparation

Test-driven development (TDD) is a technique that is used by software developers. The technique is to first write down a few examples that show how the code should work, before the actual program code is written to resolve the problem in question. Kent Beck that developed the technique in 2003, says the following about it: " "*TDD encourages simple designs and inspires confidence*".

The Python Standard Library contains a module doctest with which the TDD principle can be easily applied when writing Python program code. Indeed, this module includes a function testmod which looks for pieces of text in the docstrings of the program code that look like interactive Python sessions. These snippets are then actually carried out, to determine whether they indeed work as indicated. To submit your program code to a doctest it is therefore sufficient to include representative examples in the docstrings of your functions and adding the following snippet at the end of your Python modules:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Assignment

We want to write a Python module in which we bring together a number of functions that can be used to analyze words and sentences. The following code fragment already contains the first step in this direction. We have already worked out the header of the functions that determines the name and parameters. For the parameter `s` a string should always be passed, and for the parameter `l` a list of strings must always be passed. Through a few examples in the form of interactive Python sessions we have already indicated in the docstring of the functions how the functions should work .

Based on the examples, it is now your job to derive what result the function should return for arbitrary values of the appropriate data type that are passed as an argument to the function. It is envisaged that the function does not only work correctly for the given examples, but also for other values of the arguments. Implement the necessary Python instructions in the body of each of the functions, so the functions pass the doctest. Also add a brief description of the procedure to the docstring of each function.

```
def onlyletters(s):

    """
    >>> onlyletters('what?')
    'what'
    >>> onlyletters('"now!"')
    'now'
    >>> onlyletters('?+="word!,@$()"')
    'word'
    """

def doublehyphen(s):

    """
    >>> doublehyphen('well--being')
    True
    >>> doublehyphen('more')
    False
    >>> doublehyphen('creature')
    False
    >>> doublehyphen('merry-go--round')
    True
    >>> doublehyphen('yo---yo')
    False
    """

def wordlist(s):

    """
    >>> wordlist('I wish to wish the wish you wish to wish, but if you wish the wish the witch wishes, I won't wish the wish you wish to wish.')
    ['what', 'was', 'was', 'before', 'was', 'was', 'was']
    >>> wordlist('Peter Piper picked a peck of pickled peppers as a pick-me-up.')
    ['Peter', 'Piper', 'picked', 'a', 'peck', 'of', 'pickled', 'peppers', 'as', 'pickmeup']
    >>> wordlist('I sell unsifted--thistles to thistle-sifters.')
    ['I', 'sell', 'unsifted', 'thistles', 'to', 'thistlesifters']
    """

def numberofwords(s, l):

    """
    >>> numberofwords('to', ['i', 'wish', 'to', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish', 'but', 'if', 'you', 'wish', 'the', 'wish', 'the', 'witch', 'wishes', 'i', 'will', 'not', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish'])
    ['to', 3]
    >>> numberofwords('wish', ['i', 'wish', 'to', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish', 'but', 'if', 'you', 'wish', 'the', 'wish', 'the', 'witch', 'wishes', 'i', 'will', 'not', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish']
    ['wish', 11]
    >>> numberofwords('i', ['i', 'wish', 'to', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish', 'but', 'if', 'you', 'wish', 'the', 'wish', 'the', 'witch', 'wishes', 'i', 'will', 'not', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish'])
    ['i', 2]
    >>> numberofwords('is', ['i', 'wish', 'to', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish', 'but', 'if', 'you', 'wish', 'the', 'wish', 'the', 'witch', 'wishes', 'i', 'will', 'not', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish'])
    ['is', 0]
    """

def uniquewords(l):

    """
    >>> uniquewords(['I', 'wish', 'to', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish', 'but', 'if', 'you', 'wish', 'the', 'wish', 'the', 'witch', 'wishes', 'I', 'will', 'not', 'wish', 'the', 'wish', 'you', 'wish', 'to', 'wish'])
    ['but', 'i', 'if', 'not', 'the', 'to', 'will', 'wish', 'wishes', 'witch', 'you']
    >>> uniquewords(['If', 'Stu', 'chews', 'shoes', 'should', 'Stu', 'choose', 'the', 'shoes', 'he', 'chews'])
    ['chews', 'choose', 'he', 'if', 'shoes', 'should', 'stu', 'the']
    """

def longestword(l):
```

```
    """
    >>> longestword(['apple', 'pear', 'mango'])
    5
    >>> longestword(['I', 'am', 'the', 'coolest'])
    7
    >>> longestword(['foo', 'supercalifragilisticexpialidocious', 'bar'])
    34
    """

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Voorbereiding

Test-driven development (TDD) is een techniek die gebruikt wordt door software-ontwikkelaars. De techniek bestaat erin om eerst enkele voorbeelden neer te schrijven die aangeven hoe de programmacode moeten werken, vooraleer de eigenlijke programmacode geschreven wordt waarmee het probleem in kwestie kan opgelost worden. Kent Beck die de techniek in 2003 heeft ontwikkeld, zegt er zelf het volgende over: "*TDD encourages simple designs and inspires confidence*".

De Python Standard Library bevat een module `doctest` waarmee het TDD-principe makkelijk kan toegepast worden bij het schrijven van Python programmacode. Deze module bevat immers een functie `testmod` die op zoek gaat naar stukken tekst in de docstrings van de programmacode die eruit zien als interactieve Python sessies. Deze tekstfragmenten worden daarna ook daadwerkelijk uitgevoerd, om na te gaan of ze wel degelijk werken zoals wordt aangegeven. Om je programmacode aan een doctest te onderwerpen, volstaat het dus om representatieve voorbeelden op te nemen in de docstrings van je functies en achteraan je Python modules het volgende fragment toe te voegen:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Opgave

We willen een Python module schrijven waarin we een aantal functies bijeenbrengen die kunnen gebruikt worden om woorden en zinnen te analyseren. Onderstaand codefragment bevat reeds de eerste aanzet hiertoe. We hebben reeds de hoofding van de functies uitgewerkt, die de naam en de parameters vastlegt. Hierbij moet voor de parameter `s` steeds een string doorgegeven worden, en moet voor de parameter `l` steeds een lijst van strings doorgegeven worden. Via enkele representatieve voorbeelden onder de vorm van interactieve Python sessies hebben we in de docstring van de functies ook reeds aangegeven hoe de functies moeten werken.

Het is jouw taak om op basis van de voorbeelden af te leiden welk resultaat de functie moet teruggeven voor willekeurige waarden van het juiste gegevenstype die als argument aan de functie doorgegeven worden. Het is de bedoeling dat de functie niet enkel correct werkt voor de opgegeven voorbeelden, maar ook voor andere waarden van de argumenten. Implementeer hiervoor de nodige Python instructies in de body van elk van de functies, zodat de functies de doctest doorstaan. Voeg ook een beknopte omschrijving van de werking toe aan de docstring van elke functie.

```
def enkelletters(s):

    """
    >>> enkelletters('wat?')
    'wat'
    >>> enkelletters('"nu!"')
    'nu'
    >>> enkelletters('?+="woord!,@$()"')
    'woord'
    """

def dubbelkoppel(s):

    """
    >>> dubbelkoppel('gala--avond')
    True
    >>> dubbelkoppel('meerdere')
    False
    >>> dubbelkoppel('schepsel')
    False
    >>> dubbelkoppel('doe-het--zelver')
    True
    >>> dubbelkoppel('yo---yo')
    False
    """

def woordenlijst(s):

    """
    >>> woordenlijst('Wat was was, eer was was was?')
    ['wat', 'was', 'was', 'eer', 'was', 'was', 'was']
    >>> woordenlijst('Als apen apen na-apen, apen apen apen na.')
    ['als', 'apen', 'apen', 'naapen', 'apen', 'apen', 'apen', 'na']
    >>> woordenlijst('Een niet--machine niet, maar een naai-machine niet.')
    ['een', 'niet', 'machine', 'niet', 'maar', 'een', 'naaimachine', 'niet']
    """

def aantalwoorden(s, l):

    """
    >>> aantalwoorden('wat', ['wat', 'was', 'was', 'eer', 'was', 'was', 'was'])
    ['wat', 1]
    >>> aantalwoorden('was', ['wat', 'was', 'was', 'eer', 'was', 'was', 'was'])
    ['was', 5]
    >>> aantalwoorden('eer', ['wat', 'was', 'was', 'eer', 'was', 'was', 'was'])
    ['eer', 1]
    >>> aantalwoorden('is', ['wat', 'was', 'was', 'eer', 'was', 'was', 'was'])
    ['is', 0]
```

```
    """

def uniekewoorden(l):

    """
    >>> uniekewoorden(['Wat', 'was', 'was', 'eer', 'was', 'was', 'was'])
    ['eer', 'was', 'wat']
    >>> uniekewoorden(['Als', 'apen', 'naapen', 'apen', 'apen', 'apen', 'na'])
    ['als', 'apen', 'na', 'naapen']
    """

def langstewoord(l):

    """
    >>> langstewoord(['appel', 'peer', 'mango'])
    5
    >>> langstewoord(['ik', 'ben', 'de', 'peulschil'])
    9
    >>> langstewoord(['foo', 'supercalifragilisticexpialidocious', 'bar'])
    34
    """

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```