5.23 Write function `pay()` that takes as input an hourly wage and the number of hours an employee worked in the last week. The function should compute and return the employee’s pay. Overtime work should be paid in this way: Any hours beyond 40 but less than or equal to 60 should be paid at 1.5 times the regular hourly wage. Any hours beyond 60 should be paid at 2 times the regular hourly wage.

```python
>>> pay(10, 35)
350
>>> pay(10, 45)
475
>>> pay(10, 61)
720
```

5.24 Write function `case()` that takes a string as input and returns 'capitalized', 'not capitalized', or 'unknown', depending on whether the string starts with an uppercase letter, lowercase letter, or something other than a letter in the English alphabet, respectively.

```python
>>> case('Android')
'capitalized'
>>> case('3M')
'unknown'
```

5.25 Implement function `leap()` that takes one input argument—a year—and returns True if the year is a leap year and False otherwise. (A year is a leap year if it is divisible by 4 but not by 100, unless it is divisible by 400 in which case it is a leap year. For example, 1700, 1800 and 1900 are not leap years but 1600 and 2000 are.)

```python
>>> leap(2008)
True
>>> leap(1900)
False
>>> leap(2000)
True
```

5.26 Rock, Paper, Scissors is a two-player game in which each player chooses one of three items. If both player choose the same item, the game is tied. Otherwise, the rules that determine the winner are:
(a) Rock always beats Scissors (Rock crushes Scissors)
(b) Scissors always beats Paper (Scissors cut Paper)
(c) Paper always beats Rock (Paper covers Rock)
Implement function `rps()` that takes the choice ('R', 'P', or 'S') of player 1 and the choice of player 2, and returns -1 if player 1 wins, 1 if player 2 wins, or 0 if there is a tie.

```python
>>> rps('R', 'P')
1
>>> rps('R', 'S')
-1
>>> rps('S', 'S')
0
```
5.27 Write function `letter2number()` that takes as input a letter grade (A, B, C, D, F, possibly with a - or +) and returns the corresponding number grade. The numeric values for A, B, C, D, and F are 4, 3, 2, 1, 0. A + increases the number grade value by 0.3 and a - decreases it by 0.3.

```python
>>> letter2number('A-')
3.7
>>> letter2number('B+')
3.3
>>> letter2number('D')
1.0
```

5.28 Write function `geometric()` that takes a list of integers as input and returns True if the integers in the list form a geometric sequence. A sequence $a_0, a_1, a_2, a_3, a_4, \ldots, a_n$ is a geometric sequence if the ratios $a_1/a_0, a_2/a_1, a_3/a_2, a_4/a_3, \ldots, a_n/a_{n-1}$ are all equal.

```python
>>> geometric([2, 4, 8, 16, 32, 64, 128, 256])
True
>>> geometric([2, 4, 6, 8])
False
```

5.29 Write function `lastfirst()` that takes one argument—a list of strings of the format `<LastName, FirstName>`—and returns a list consisting two lists:
(a) A list of all the first names
(b) A list of all the last names

```python
>>> lastfirst(['Gerber, Len', 'Fox, Kate', 'Dunn, Bob'])
[['Len', 'Kate', 'Bob'], ['Gerber', 'Fox', 'Dunn']]
```

5.30 Develop the function `many()` that takes as input the name of a file in the current directory (as a string) and outputs the number of words of length 1, 2, 3, and 4. Test your function on file sample.txt.

File: sample.txt

```python
>>> many('sample.txt')
Words of length 1 : 2
Words of length 2 : 5
Words of length 3 : 1
Words of length 4 : 10
```

5.31 Write a function `subsetSum()` that takes as input a list of positive numbers and a positive number `target`. Your function should return True if there are three numbers in the list that add up to `target`. For example, if the input list is `[5, 4, 10, 20, 15, 19]` and `target` is 38, then True should be returned since $4 + 15 + 19 = 38$. However, if the input list is the same but the target value is 10, then the returned value should be False because 10 is not the sum of any three numbers in the given list.

```python
>>> subsetSum([5, 4, 10, 20, 15, 19], 38)
True
>>> subsetSum([5, 4, 10, 20, 15, 19], 10)
False
```
5.37 Write function mssl() (minimum sum sublist) that takes as input a list of integers. It then computes and returns the sum of the maximum sum sublist of the input list. The maximum sum sublist is a sublist (slice) of the input list whose sum of entries is largest. The empty sublist is defined to have sum 0. For example, the maximum sum sublist of the list

\[ [4, -2, -8, 5, -2, 7, 7, 2, -6, 5] \]
is \[ [5, -2, 7, 7, 2] \] and the sum of its entries is 19.

```python
>>> 1 = [4, -2, -8, 5, -2, 7, 7, 2, -6, 5]
>>> mssl(1)
19
>>> mssl([3,4,5])
12
>>> mssl([-2,-3,-5])
0
```

In the last example, the maximum sum sublist is the empty sublist because all list items are negative.

5.38 Write function collatz() that takes a positive integer \( x \) as input and prints the Collatz sequence starting at \( x \). A Collatz sequence is obtained by repeatedly applying this rule to the previous number \( x \) in the sequence:

\[
\begin{align*}
  x &= \begin{cases} 
    x/2 & \text{if } x \text{ is even} \\
    3x + 1 & \text{if } x \text{ is odd.}
  \end{cases}
\end{align*}
\]

Your function should stop when the sequence gets to number 1. *Note:* It is an open question whether the Collatz sequence of every positive integer always ends at 1.

```python
>>> collatz(10)
10
5
16
8
4
2
1
```

5.39 Write function exclamation() that takes as input a string and returns it with this modification: Every vowel is replaced by four consecutive copies of itself and an exclamation mark (!) is added at the end.

```python
>>> exclamation('argh')
'aaaargh!'
>>> exclamation('hello')
'heeeellooo!''
```

5.40 The constant \( \pi \) is an irrational number with value approximately 3.1415928\ldots The precise value of \( \pi \) is equal to this infinite sum:

\[
\pi = 4/1 + 4/3 + 4/5 + 4/7 + 4/9 + 4/11 + \ldots
\]

We can get a good approximation of \( \pi \) by computing the sum of the first few terms. Write a function approxPi() that takes as input a float-value error and approximates