1.First Programs

♥Hello World ★

Print the string `"Hello world!"`

2.Simple Data

♥Hypotenuse ★

Define a function called `hypotenuse(n)` that given the length `n` which corresponds to two sides of a right-angled triangle, it returns the length of the hypotenuse. Please use round with two decimal places.

- for the number `n=2`, the result is the float `2.83`
- for the number `n=6`, the result is the float `8.49`

♥Alarm clock ★★

Define a function `alarm(hour, minutes)` that returns at what time the alarm clock will ring, knowing that the current hour is `hour` and the current minutes are `minutes`. The clock goes off after 6 hour and 51 minutes.

- for the time `hour=5` and `minutes=10`, the alarm should ring at `12:01`

3.Program Flow

♥Sum numbers ★

Define a function `f(n)` that returns the sum of numbers 1+2+...+n.

4.Conditionals and iteration

♥Dog's age in dog's years ★

Write a Python function `dogs(h_age)` that receives the dog’s age in human years `h_age` and returns the corresponding dog's age in dog's years. For the first two years, a dog year is equal to 10.5 human years. After that, each dog year equals 4 human years.

♥Quadratic equation solver ★

Define a function `solver(a, b, c)` that returns the solution to the quadratic formula of the type: $ax^2+bx+c=0$. Return the result in the form of a list: empty list if no solution exists in $\mathbb{R}$, a list with one or two elements if one or two solutions exist, respectively; if there are two solutions, use ascending order.

5.Functions

♥No pairs of 1s ★★

Write a Python function `no_consecutives1(k)` that, given an integer `k`, determines the number of binary numbers of length `k` that do not have consecutive 1s in them.

For example, if `k=3`, then the following binaries exist: 000, 001, 010, 011, 100, 101, 110, 111. Of these, only 5 do not have a contiguous 1.

Adapted from: www.geeksforgeeks.org

For example:

- `no_consecutives1(3)` should return `5`
- `no_consecutives1(4)` should return `8`

♥Pascal's triangle ★★

Write a Python function `pascal(n)` that, for a given integer `n`, returns a string with the first `n` rows of the Pascal's triangle. Each row finishes by `"\n"` (newline) and each value is separated by a single space. The value at the *i*-th row and *j*-th column of the triangle is equal to *i!/(j!*(i-j)!)*. Note: *i* and *j* start in zero.

- for the number `n=3`, the result is the string `1\n1 1\n1 2 1`
- for the number `n=5`, the result is the string `1\n1 1\n1 2 1\n1 3 3 1\n1 4 6 4 1`

♥The Collatz Sequence ★★

Write a Python function `collatz(n)` that returns a comma-separated string with the Collatz sequence. Starting by the positive integer `n`, the next term of the Collatz sequence is obtained from the previous by: (i) diving the previous term by 2, if the previous term is even; or (ii) multiplying the previous term by 3 and then summing 1, if the previous term is odd. The sequence ends when it reaches 1.

- for the number `n=3`, the result is the string `3,10,5,16,8,4,2,1`

- for the number `n=12`, the result is the string `12,6,3,10,5,16,8,4,2,1`

♥Ugly number ★★

Write a Python function `ugly(n)` to check whether a given positive number `n` is an ugly number. Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. To check if a number is ugly, divide the number by the prime factors 2, 3 or 5, if the number becomes 1 then it is an ugly number, otherwise it is not. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7. By convention, 1 is also an ugly number.

- for the number `n=6`, the result is `True`
- for the number `n=14`, the result is `False`

6.Strings

♥Discarding the middle ★

Write a Python function `discard_middle(s)` that, given a string `s`, returns a new string consisting of its first and last 2 characters. If the string is less or equal to 3 characters, the result should be the empty string.

- `discard_middle("string")` returns the string: `stng`
- `discard_middle("n")` returns the empty string

♥Longest word ★

Write a Python function `longest(s)` that, given a string `s`, returns the length of its longest word.

- `longest("A list with some words")` returns the integer: `5` (the longest word is *words*)
- `longest("Unnecessarily long sentence since the longest word is the first one")` returns the integer: `13` (the longest word is *Unnecessarily*)

♥Palindrome index ★★

Write a Python function `palindrome_index(s)` that, given a string `s`, returns the index of the first character that, if removed, turns the string into a palindrome. If there is no such character or the string already is a palindrome, it should return `-1`.

- `palindrome_index("aaab")`returns the integer: `3`
- `palindrome_index("tattarrattat")` returns the integer: `-1`

♥Remove leading zeros ★★

Write a Python function `remove_leading(ip)` that, given a string with an IPv4 address `ip`, returns a new string with all leading zeros removed from `ip`. An IPv4 address is an address in the format N.N.N.N, where N is an integer in the interval [0, 255].

- `remove_leading("255.024.01.01")` returns the string: `"255.24.1.1"`
- `remove_leading("192.168.0.24")` returns the string: `"192.168.0.24"`

♥Summary ranges ★★

Given a comma-separated string `astring` with sorted integers without duplicates, write a Python function `summary_ranges(astring)` that returns a string showing the contiguous intervals in the set.

- `summary_ranges("0,1,2,3,4,5,7,10,11,20,21")` returns the string: `0->5,7,10->11,20->21`
- `summary_ranges("1,3,4,6,7,9,10")` returns the string: `1,3->4,6->7,9->10`

Alphabet Rangoli ★★★

Given an user-defined integer `N`, write a Python function `rangoli(N)` to return an alphabet rangoli of size `N`. (Rangoli is a form of Indian folk art based on creation of patterns.)

- for `N=1`, the function should return the alphabet rangoli as follows:

```
a
```

- for `N=3`, the function should return the alphabet rangoli as follows:

```
----c----

--c-b-c–

c-b-a-b-c

--c-b-c–

----c----
```

- for `N=6`, the function should return the alphabet rangoli as follows:

```
----------f----------

--------f-e-f--------

------f-e-d-e-f------

----f-e-d-c-d-e-f----

--f-e-d-c-b-c-d-e-f—

f-e-d-c-b-a-b-c-d-e-f

--f-e-d-c-b-c-d-e-f—

----f-e-d-c-d-e-f----

------f-e-d-e-f------

--------f-e-f--------

----------f----------
```

Adapted from [hackerrank]

Minion game ★★★

Kevin and *Stuart* want to play the 'The Minion Game'. The goal of game is to make substrings using the letters of a given string `astring`. While *Stuart* has to make words starting with *consonants*, *Kevin* has to make words starting with *vowels*. The game ends when both players have made all possible substrings. A player gets +1 point for each occurrence of the substring in the string `astring`. Write a Python function `minion(astring)` that receives a string and computes the score as well as the substrings made by each player. The output should return the total score of the winner as well all substring/score pairs.

Notes: You can assume that `astring` will contain only uppercase letters. If the game is a draw, return the string: `It was a draw!`

For example:

- `minion("ANANAS")`, the function should return the following strings:

```
The winner was Kevin with a total of 12 points:

- A: 3

- AN: 2

- AS: 1
```

```
- ANA: 2

- ANAN: 1

- ANAS: 1

- ANANA: 1

- ANANAS: 1
```

- `minion("BANANA")` the output should return:

```
The winner was Stuart with a total of 12 points:

- B: 1

- N: 2

- BA: 1

- NA: 2

- BAN: 1

- NAN: 1

- BANA: 1

- NANA: 1

- BANAN: 1

- BANANA: 1
```

Adapted from hackerrank

7.Tuples

Add item ★

Write a Python function `add_item(tup, idx, item)` that, given a tuple `tup`, inserts the value `item` at position `idx`. If `idx` points to the beginning or the end of the tuple, it should replace the old value. Assume that index always holds a valid value.

- for `tup=(1,2,3)`, `idx=1` and `item=[4,5]`, the result is the tuple `(1,[4,5],2,3)`
- for `tup=('a',2,'c')`, `idx=2` and `item='new item'`, the result is the tuple `('a',2,'new item')`

Counting elements ★

Write a function `count_until(tup)` that, given a tuple `tup`, returns the number of elements in it before an element of the type tuple appears. If there isn't a nested tuple, it should return -1.

- for `tup=(1,'2', 3, 4.0, 5j)`, the result is the integer -1
- for `tup=(1,2,(3,), 4.0, 5j)`, the result is the integer 2

Indexing tuples ★

Write a Python program to print the first item in the `fruits` tuple.

```
fruits = ("apple", "banana", "cherry")
```

Output esperado: `"apple"`

More counting ★

Define a function `count_elems(tup)` that, given a tuple `tup`, returns a new tuple in which each element corresponds to: the number of elements in the element at the same position in `tup`, if a tuple or a list; the number of characters, if a string; 1 otherwise.

- for `tup=(1,'234', 3, 4.0, (5j,))`, the result is the tuple `(1, 3, 1, 1, 1)`
- for `tup=('12',2,(3, 4), [4.0, 'str', 'str2'], 5j)` the result is the tuple `(2, 1, 2, 3, 1)`

Unpacking tuples ★

Write a Python function `unpack_rev(atuple)` to unpack the tuple `atuple` and return the items as a string, separated by spaces and in **reverse** order. The tuple does not contain compound items.

- If the given `atuple` is `("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")`, the result should be the string `Saturday Friday Thursday Wednesday Tuesday Monday Sunday`

Text summary ★★

Given a text -- perhaps a poem, a speech, instructions to bake a cake, some inspirational verses, etc. -- write a function called `summary(text)` that given a string `text`, returns a tuple with the number of total words, and the number of words that contain at least an "e" (upper or lower case).

- `summary("A fool thinks himself to be wise, but a wise man knows himself to be a fool.")` returns the tuple: `(17, 6)`

8.Lists

Fibonacci numbers ★

Write a Python function `fib(n)` that returns a list of the first `n` Fibonacci numbers. The next number in a Fibonacci sequence is defined as the sum of the previous two numbers, and the first two numbers are defined as 0 and 1, respectively.

- `fib(1)` should return the list: `[0]`
- `fib(5)` should return the list: `[0, 1, 1, 2, 3]`

N-th lowest ★

Write a Python function `nth_lowest(lnum, n)` that, given a list of numbers `l` and a positive integer `n`, returns the `n`-th lowest value in the list. Assume `n` always holds a valid value (i.e. does not go out of bounds) and that the list never has repeated values.

- for the list `lnum=[42, 234, 135, 21, 232, 12312, -2343]` and `n=2`, the result is the number `21`
- for the list `lnum=[73,100,23,18,84,61,56,75,92,38,54,73,3,13]` and `n=12`, the result is the number `84`

Rotating ★

Write a Python function `rotate_list(l)` that, given a list `l` of length >= 3, shifts its elements to the left 3 times, returning the new list. If an element is at the beginning, it should circle back to the end.

- for the list `l=[1,2,3,4,5,6]`, the result is the list `[4,5,6,1,2,3]`
- for the list `l=[1,2,3,4,5,6,7,8]`, the result is the list `[4,5,6,7,8,1,2,3]`

Adapted from CodingBat

Fetching the middle ★★

Write a Python function `fetch_middle(lists)` that, given a list of lists, returns a new list containing the middle element from each list in the lists. If the list's length is even, consider the middle element to be the average between its two central elements (i.e. for the list `[1,2,3,4]`, the middle element would be (2+3)/2 = 2.5).

- for the list of lists `lists=[[1,2,3],[4,5,6],[7,8,9,10]]`, the result is the list `[2,5,8.5]`
- for the list of lists `lists=[[10,25,35,45],[100,-1,3,4],[-3,-5,-10,-20,-100]]`, the result is the list `[30.0,1.0,-10]`

Adapted from CodingBat

♥Pattern hunting ★★

Write a function `pattern_hunting(l1, l2, p)` that given two equal length lists of strings `l1` and `l2` and a pattern (string) `p`, builds a new list with elements from the original lists where the pattern occurs not in that element, but the corresponding (i.e. same index) element in the other list. The resulting list should be ordered alphabetically in descending order.

- for the list `l1=['a string', 'two strings', 'not here']`, another list `l2=['choose me', 'me too', 'but not me']` and the pattern `p='string'`, the result is the list `['me too', 'choose me']`
- for the list `l1=['not found', 'pattern here', 'skip', 'next... or not?']`, another list `l2=['pattern here indeed', 'i want to be chosen', 'not my day', 'sneakypatternbutitisthere']` and the pattern `p='pattern'`, the result is the list `['not found', 'next... or not?', 'i want to be chosen']`

Orthogonal matrices ★★★

Write a function `is_orthogonal(mx)` that given a square matrix (list of lists where each list represents a row), determines if the matrix is orthogonal, returning the appropriate boolean value. A matrix M is orthogonal if $MM^T = I$ (i.e. the product of M by its transpose is equal to the identity matrix).

- for the matrix `mx=[[-1, 0], [0,1]]`, the output should be `True`
- for the matrix `[[-2,3], [4,-1]]`, the output should be `False`

9.Dictionaries

Academy Awards ★

Write a Python function `academy_awards(alist)` which receives a list of tuples, `alist`, where each tuple holds the category and the corresponding winning movie and returns a dictionary that maps each movie to the number of awards that it won.

For example:

- `academy_awards([("Best Picture", "Moonlight"), ("Best Director", "La La Land"), ("Best Actor", "Manchester by the Sea"), ("Best`

`Actress", "La La Land"), ("Best Supporting Actor", "Moonlight"), ("Best Supporting Actress", "Fences"), ("Best Original Screenplay", "Manchester by the Sea"), ("Best Original Score", "La La Land")])` should return the dictionary: `{'Moonlight': 2, 'La La Land': 3, 'Manchester by the Sea': 2, 'Fences': 1}`

Heroes and Villains ★

Write a Python function `fight(heroes, villain)` that, given a list of `heroes` and a `villain`, checks if the good side can prevail. Each hero in the list has a set of properties stored in a dictionary: name, health, category and score (number of victories). The villain has a similar structure, but does not record his score. The fight occurs within the following set of restrictions:

1. The villain only fights heroes of the same category as him.
2. The villain fights the heroes in the order they appear in the list.
3. In a hero versus villain fight, the winner is the one with bigger health (in case of a tie, hero wins). If the villain wins, diminish his health by half of the hero's health.
4. The fight ends when the villain dies or there are no more heroes.

The function should return a properly formatted string according to the fight result (hero or villain win), respectively:

```
<h> defeated the villain and now has a score of <s>
```

or

```
<v> prevailed with <hp>HP left
```

where `<h>` is the name of the hero that (finally) defeated the villain, `<s>` his updated score, `<v>` the name of the villain and `<hp>` his final remaining health (a decimal **rounded to 1 decimal point**).

For example:

- `fight([{'name': 'Genos', 'health': 3000, 'category': 'S', 'score': 0}, {'name': 'Blizzard of Hell', 'health': 1000, 'category': 'B', 'score': 0}, {'name': 'Saitama', 'health': 9001, 'category': 'C', 'score': 0}, {'name': 'King', 'health': 3750, 'category': 'S', 'score': 1}], {'name': 'Hero Killer', 'health': 4000, 'category': 'S'})` returns the string: `King defeated the villain and now has a score of 2`
- `fight([{'name': 'Genos', 'health': 3000, 'category': 'S', 'score': 0}, {'name': 'Blizzard of Hell', 'health': 1000, 'category': 'B', 'score': 0}, {'name': 'Saitama', 'health': 9001, 'category': 'C', 'score': 0}, {'name': 'King', 'health': 2000, 'category': 'S', 'score': 1}], {'name': 'Hero Killer', 'health': 4000, 'category': 'S'})` returns the string: `Hero Killer prevailed with 1500.0HP left`

Most Frequent ★

Write a Python function `most_frequent(alist)` that, given a list `alist` of integers, using a dictionary, returns the most frequent element in `alist`. In case of ties, return the element with the greatest value.

- `most_frequent([-1, 2, 5, -1, 3, 3, 3, 4, 4, 2, 4, 5, -1, -1])` returns the integer: `-1`
- `most_frequent([-1, 111, 1, 11, -1, 11, 1, 111])` returns the integer: `111`

Sort by key ★

Write a Python function `sort_by_key(dict)` that, given a dictionary `dict` of colors (key is the color name and value is its hexadecimal value), returns a list of pairs ordered by color. Note that It is not possible to sort a dictionary, only to get a representation of a dictionary that is sorted.

For example:

- `sort_by_key({"Blue":"#0000FF", "Green":"#008000", "Black":"#000000", "White":"#FFFFFF"})` returns the list: `[("Black", "#000000"), ("Blue", "#0000FF"), ("Green", "#008000"), ("White", "#FFFFFF")]`

Sort by value ★

Write a Python function `sort_by_value(dict)` that, given a dictionary `dict` of colors (key is the color name and value is its hexadecimal value), returns a list of pairs ordered by color. Note that It is not possible to sort a dictionary, only to get a representation of a dictionary that is sorted.

For example:

- `sort_by_value({"Blue":"#0000FF", "Green":"#008000", "Black":"#000000", "White":"#FFFFFF"})` returns the list: `[("Black", "#000000"), ("Blue", "#0000FF"), ("Green", "#008000"), ("White", "#FFFFFF")]`

TF-IDF ★★★

Google or any other text-based search engine must have efficent ways of finding the relevant documents in a large corpus. To weight the importance of words in documents for search a numerical statistic, called tf-idf may be used. Typically, for each document a vector is used. You will need to implement the following steps:

1. Build a vocabulary: create a list with all words in all documents, as lower-case
2. Build the text frequency vector (TF): create a dictionary where, for each word, specify a list of how many times it occurs for each document
3. Multiply each value in the dictionary by a factor known as the inverse document frequency (IDF): *log(N/n)*, where *N* is the total number of documents and *n* is the number of documents where the word appears, and round the result to 3 digits.

The reason for the IDF step is to scale the vector so that words that appear often (like "the" or "and") are given a lower value because they are frequent not only in that document, but in all document. The resulting element of each vector is its *tf*idf*.

Write a function called `tfidf(documents)` that receives a list of `documents` (as its string content) and returns a list of a dictionary for each document associating each word to its tf-idf vector.

For example:

- `tfidf(["To be or not to be", "Impossible is a word to be found only in the dictionary of fools", "There is nothing impossible to him who will try"])` returns the list of tf-idf dictionaries: `{'to': [0.0, 0.0, 0.0], 'be': [0.811, 0.405, 0.0], 'or': [1.099, 0.0, 0.0], 'not': [1.099, 0.0, 0.0], 'impossible': [0.0, 0.405, 0.405], 'is': [0.0, 0.405, 0.405], 'a': [0.0, 1.099, 0.0], 'word': [0.0, 1.099, 0.0], 'found': [0.0, 1.099, 0.0], 'only': [0.0, 1.099, 0.0], 'in': [0.0, 1.099, 0.0], 'the': [0.0, 1.099, 0.0], 'dictionary': [0.0, 1.099, 0.0], 'of': [0.0, 1.099, 0.0], 'fools': [0.0, 1.099, 0.0], 'there': [0.0, 0.0, 1.099], 'nothing': [0.0, 0.0, 1.099], 'him': [0.0, 0.0, 1.099], 'who': [0.0, 0.0, 1.099], 'will': [0.0, 0.0, 1.099], 'try': [0.0, 0.0, 1.099]}`

10.Recursion

Factorial ★

The factorial of a number can be written recursively. For example, *5!=4!*5*. In other words:

- factorial(n) = n*factorial(n-1)
- factorial(0) = 1

Write a recursive Python function `factorial(n)` which computes the factorial of the number `n` using recursion.

For example:

- `factorial(5)` should return `120`

Fibonacci ★

Write a recursive Python function `fib(n)` that returns the Fibonacci series for `n` terms. Remember, the Fibonacci series is of a given number is defined as the sum of the previous two Fibonacci numbers, with 0 and 1 being the base conditions.

For example:

- `fib(10)` should return `55`

Greatest common divisor ★

Write a recursive Python function `rec_gcd(n1, n2)` to find the greatest common divisor (*gcd*) of two integers `n1` and `n2`.

Adapted from: www.w3resource.com

For example:

- `rec_gcd(12,14)` should return `2`

Juggler Sequence ★

Write a recursive Python function `juggler(n, p)` that, given two integers `n` and `p`, calculates the `p`-th term in the juggler sequence for `n`. The juggler sequence for a non-negative integer `n` is defined as follows:

- juggler(n, 0) = n
- juggler(n, p) = ⌊juggler(n, p-1)$^{1}/_{2}$⌋, if juggler(n, p-1) is even
- juggler(n, p) = ⌊juggler(n, p-1)$^{3}/_{2}$⌋, if juggler(n, p-1) is odd

For example:

- `juggler(3,1)` should return `5`
- `juggler(3,2)` should return `11`

Sum of digits ★

Write a recursive Python function `rec_sum_digits(n)` that returns the sum of the digits of an integer number.

Adapted from: www.w3resource.com

For example:

- `rec_sum_digits(12)` should return `3`
- `rec_sum_digits(45)` should return `9`

Change to base ★★

Write a recursive Python function `rec_to_base(n, base)` to convert an integer `n` to a string in any base, given as an integer ∈ [0, 16].

Adapted from: www.w3resource.com

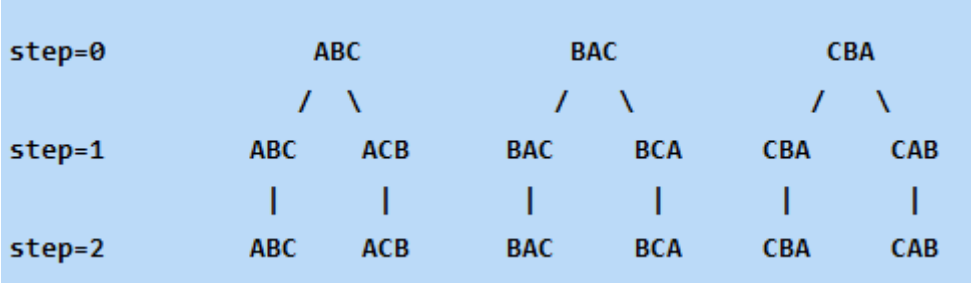
For example:

- `rec_to_base(20,16)` should return `14`
- `rec_to_base(20,2)` should return `10100`

Permutations ★★

The full permutation of a list can be easily programmed using recursive algorithms. The number of the full permutation results is *n!* where *n* is the number of elements to permutate. Please return the list of permutations in the same order as in the example.

For example, `permuta(['A', 'B', 'C'])`,



In each call, the function permutates the element at position `step`. In the start, it permutatted the first element with the following elements (`step=0`), then the second element with the following elements (`step=1`), and finally the third element with the following elements (`step=2`). In `step=2`, since there were no *following elements*, the list remained the same. This is the base case and is the one expected for the answer.

Write a recursive Python function `permuta(alist, step=0)` that returns the lists of permutations.

For example:

- `permuta(['Joe', 'Mary', 'John'])` should return `[['Joe', 'Mary', 'John'], ['Joe', 'John', 'Mary'], ['Mary', 'Joe', 'John'], ['Mary', 'John', 'Joe'], ['John', 'Mary', 'Joe'], ['John', 'Joe', 'Mary']]`

Painter's problem ★★★

Write a Python recursive function `paint(n, boards)` that returns the minimum time necessary to paint the list of `boards` by `n` painters. `boards` is a list of the time necessary to paint each board.

For example, if there are `boards=[6, 2, 5, 1, 9]`, and painter 1 has to paint the first three (`[6, 2, 5]`) then he will need time=6, and if painter 2 paints the rest (`[1, 9]`) then he will need time=9 to paint all boards. The time required of each painter to paint the board is the maximum element in that list.

The objective is to find the minimum time required to paint the boards by choosing which boards are painted by which painters. Return the minimum time at the end of the function. Notice that since each painter must paint at least one board, the minimum time can increase when we increase painters.

Adapted from: www.geeksforgeeks.org

For example:

- `paint(3,[60, 70, 10, 20, 40, 50, 10, 40])` should return `120`
- `paint(2,[60, 70, 10, 20, 40, 50, 10, 40])` should return `110`

11. Functional Programming with Collections

Celsius ★

Write a Python function `to_celsius(t)` that, given a list `t` of temperatures in Fahrenheit degrees, uses `map()` with a lambda function to compute the corresponding Celsius degrees, rounded to 1 decimals.

For example:

- `to_celsius([84.56, 79.7, 81.14, 82.4, 82.04])` should return `[29.2, 26.5, 27.3, 28.0, 27.8]`
- `to_celsius([23.0, 28.4, 32.0, 35.6])` should return `[-5.0, -2.0, 0.0, 2.0]`

Fahrenheit ★

Write a Python function `to_fahrenheit(t)` that, given a list `t` of temperatures in Celsius degrees, uses `map()` with a lambda function to compute the corresponding Fahrenheit degrees, rounded to 2 decimals.

For example:

- `to_fahrenheit([29.2, 26.5, 27.3, 28, 27.8])` should return `[84.56, 79.7, 81.14, 82.4, 82.04]`
- `to_fahrenheit([-5, -2, 0, 2])` should return `[23.0, 28.4, 32.0, 35.6]`

Rearranging ★

Write a function `rearrange(l)` that, given a list of numbers `l`, rearranges it so that all non-positive numbers appear before the positive ones, but does **not** alter the numbers' relative order (i.e., all positive numbers must appear in the same order relative to each other as in the original list; same goes for non-positive numbers).

For example:

- `rearrange([12, 11, -13, -5, 6, -7, 5, -3, -6, 0])` should return `[-13, -5, -7, -3, -6, 0, 12, 11, 6, 5]`
- `rearrange([-19, -15, -14, -9, -18, -1, -4])` should return `[-19, -15, -14, -9, -18, -1, -4]`

Map, Filter & Reduce ★★

Write a function `map_filter_reduce(lst, f1, f2, f3)` that receives a list `lst` and three functions: `f1`, `f2` and `f3`. The function filters the elements of `lst` using `f1`, applies `f2` to each element of the result, and finally applies reduce to convert the list to a scalar by using `f3`.

For example:

- `map_filter_reduce([1, 2, 3, 4, 5, 6, 7, 8],lambda i: i % 2 == 0,lambda i: i**2,lambda a, b: a+b)` should return `120`

Median Batch Normalization ★★

Write a *generator* function `batch_norm(alist, batch_size)` that, given a list of integers `alist`, *yields* a normalized batch of elements of `alist` of a given size `batch_size`. Each batch must be normalized by the median value of the batch. That is, each batch is subtracted by the median within that batch. A batch is a sublist with size=`batch_size`.

For example:

- `batch_norm([-1, 0, 1, 1, 2, 4], 3)` returns a generator that produces `[[-1, 0, 1], [-1, 0, 2]]`
- `batch_norm([10, 1, -12, 5, 1, 3, 7, 3, 3], 5)` returns a generator that produces `[[9, 0, -13, 4, 0], [0.0, 4.0, 0.0, 0.0]]`