− . − −./... − −/. − .. .. − ./

Reminder : it is strictly FORBIDDEN to use the @ operator.

"Beginning in 1836, the American artist Samuel F. B. Morse, the American physicist Joseph Henry, and Alfred Vail developed an electrical telegraph system. This system sent pulses of electric current along wires which controlled an electromagnet that was located at the receiving end of the telegraph system. A code was needed to transmit natural language using only these pulses, and the silence between them. Morse therefore developed the forerunner to modern International Morse code."

– wikipedia

The morse code is a kind of binary format for letters and punctuations. A letter is only build using `.` (short signal - dits) and `-` (long signal - dahs), the more frequent the letters, the shorter the conversion in morse (for example a `e` is only a `.`). The purpose of this practical is to provides a library to convert sentences to morse and morse to sentences. We will consider that a letter is a list of characters (`.` or `-`), and a word is a list of letters (a list of list of characters).

Examples :

```
# latin_letter_to_morse 'X';;
- : char list = ['-'; '.'; '.'; '-']

# latin_word_to_morse "42";;
- : char list list = [['.'; '.'; '.'; '.'; '-']; ['.'; '.'; '-'; '-'; '-']]
```

Stage 0 - must-do

0.1 Equality

Write the function `are_equal` which tests if two lists are equal.

```
# are_equal [1, 2, 3] [1, 2, 3];;
- : bool = true

# are_equal ['a', 'b', 'c'] ['a', 'b', 'c', 'd'];;
- : bool = false
```

0.2 Concatenation

Write the function `append` which concatenates the second list at the end of the first one.

```
# append [1; 2; 3] [42; 0; 21];;
- : int list = [1; 2; 3; 42; 0; 21]
```

0.3 Reverse

Write the function `reverse` which reverses a list.

```
# reverse ['a'; 'l'; 'l'; 'o'];;
- : char list = ['o'; 'l'; 'l'; 'a']
```

0.4 Display

Write a function which displays a list of characters.

Stage 1 - characters

1.1 Validity

Write the `is_morse` which checks if a list of characters is in proper morse code

```
# is_morse ['-', '-', '-'];;
- : bool = true

# is_morse ['.', '.', 'k'];;
- : bool = false
```

1.2 Conversion

Write the function `letter_to_morse` which returns the morse code of the letter given as parameter.

```
# letter_to_morse 'S';;
- : char list = ['.'; '.'; '.']
```

Stage 2 - words

2.1 Conversion

Write the function `word_to_morse` which converts a list of letters to a list of morse code.

```
# word_to_morse ['S', 'O', 'S'];;
- : char list list = [['.'; '.'; '.']; ['-'; '-'; '-']; ['.'; '.'; '.']]
```

2.2 Another way to see things

A standard way to display morse is to separate letters with spaces (`' '`). Here we will use this character to convert a list of list of characters (a word in morse) to a list of characters.

```
# to_single_list  [['-'; '-'; '-']; ['-'; '.'; '-']];;
- : char list = ['-'; '-'; '-'; ' '; '-'; '.'; '-']
```

2.3 Printing

Write a function which display a word (in morse - `char list list` format)

Stage 3 - sentences

3.1 Conversion (again)

Write the function `sentence_to_morse` which returns the conversion of a sentence (list of list of characters) into morse (list of list of list of characters).

```
# sentence_to_morse [['t'; 'o']; ['m'; 'e']];;
- : char list list list = [[['-']; ['-'; '-'; '-']]; [['-'; '-']; ['.']]]
```

3.2 char list list ...

Rest assured, we will not add another `list` level. Like one of the preceding function, we will use the character `'/'` to separate word in order to remove one list level.

```
# sentence_to_single_list [[['-']; ['-'; '-'; '-']]; [['-'; '-']; ['.']]];;
- : char list = ['-'; ' '; '-'; '-'; '-'; '/'; '-'; '-'; ' '; '.'; '/']
```

3.3 Nonstop

Now that we know how to reduce the list depth, we can not stop at this point. We want to convert a word (list of characters) or a sentence (list of list of characters) to a single list of morse code, using the proper separation characters (`' '`, `'/'`)

```
# to_single_morse ['S'; 'O'; 'S'] ;;
- : char list = ['.'; '.'; '.'; ' '; '-'; '-'; '-'; ' '; '.'; '.'; '.']

# latin_sentence_to_single [['T'; 'O']; ['M'; 'E']] ;;
- : char list = ['-'; ' '; '-'; '-'; '-'; '/'; '-'; '-'; ' '; '.'; '/']
```

Stage 4 - encode me

Write the functions to translate a sentence (string) to its equivalent morse encoding (string also).

```
# latin_to_morse "Vive Les Listes" ;;
- : string = "...- .. ...- ./.-.. . .../.-.. .. ... - . .../"
```

Stage 5 - decode me

Write the function to translate a morse sentence (string) to its equivalent sentence (string also).

```
# morse_to_latin "...- .. ...- ./.-.. . .../.-.. .. ... - . .../";;
- : string = "VIVE LES LISTES"
```

Stage 6 - Caml spree

6.1 Rhythm !

Write a function which displays a morse sentence in rhythm
- A '-' is as long as 3 '.'
- The signals for a same letter are separated by a blank as long as a '.'
- Two letters are separated by a blank as long as a '-'
- Two words are separated by a blank of 7 '.'

6.2 Bip bip bip !

Write a function which uses the sounds engine to play the morse sequence in rhythm.

Stage 7 - Genericity ?

Morse code is `<fun>`, but with all the stuff you have already done for handeling the translation, it could be interesting not to have to redo everything for each other alphabet (other than morse) we may have to do. Fortunately the files are here to help you. Find a proper way to store the "dictionnary" (alphabet to morse) and how to use it in your functions. Most of the stuff will not change, and you will be able to handle as many languages as we want :)