homeworks.md

How to send your homework?

1. Create solutions for your tasks.
2. Put each solution in separate python file named `<homework_number>_<task_number>.py`. For example solution for the task **3.1.** should be placed in a file `3_1.py`. If you are working with Jupyter, don't worry – after creating solutions just copy them into new files with a proper name.
3. Put all of your solutions in the folder named with your index number. For example, if your University index number is *333444*, folder should be named `333444`.
4. Compress your files using zip extension (without password protection). Finally your file should be `333444.zip`.

For example if your index is *333444* and you are sending solutions for the first homework your files should have structure (after unzipping `333444.zip`):

```
333444
├── 1_1.py
├── 1_2.py
└── 1_3.py
```

Homework 1.

1.1. Write a script (or function) that takes as an input two numbers `width` and `height` and prints a rectangle with specified size.

Examples:

- `width=5` and `height=4` should result in terminal output:

```
#####
#   #
#   #
#####
```

- `width=7` and `height=2` should result in terminal output:

```
#######
#######
```

- `width=1` and `height=1` should result in terminal output:

```
#
```

1.2. Sum all perfect squares (numbers that are equal to the square of other number – 1, 4, ..., 9, 16, 25, 36, 49, ..., 9801, 10000) in range between 0 and 10000.

1.3. Write a script for setting an alarm, which ask users whether they are employed (yes / no) and whether they are currently on vacation (yes / no). User should answer typing either `Y` for yes or `N` for no. If user specify incorrect answer (anything that is not `Y` or `N`) program should warn user about incorrect answer and ask again.

The script output `True` if user is employed and not on vacation (because these are the circumstances under which you need to set an alarm). It should output `False` otherwise. Examples:

Examples:

- setting an alarm

```
> Are you employed? (Y/N):
> y
> Incorrect answer.
> Are you employed? (Y/N):
> Y
> Are you on vacation (Y/N):
> N
> True
```

- not setting an alarm

```
> Are you employed? (Y/N):
> N
> Are you on vacation (Y/N):
> N
> False
```

Homework 2.

2.1. Write a script approximating `cos(x)` as a first five terms of Taylor series expansion (see trigonometric functions section in Wikipedia). User should provide an input value of `x` and script should output calculated value for each term as well as current sum. Output should be formatted according to scheme:

```
> Calculating cos(1.0471975511965976) as Taylor expansion...
>
> Value for k=0 is +1.0000 (current sum is 1.0000)
> Value for k=1 is -0.5483 (current sum is 0.4517)
> Value for k=2 is +0.0501 (current sum is 0.5018)
> Value for k=3 is -0.0018 (current sum is 0.5000)
> Value for k=4 is +0.0000 (current sum is 0.5000)
```

Use either the `.format()` method or f-strings.

> You might need `factorial` function from `math` module

2.2. Write two scripts `coefficients.py` and `quadratic.py`. In `coefficients.py` define three variables `a`, `b` and `c` representing coefficients of the quadratic equation `ax^2 + bx + c`. In `quadratic.py` import coefficients, solve the equation and output properly formatted the solution. For example in `coefficients.py` define:

```
a = 1
b = -10
c = 25
```

Then running `quadratic.py` should produce:

```
> One solution found x=5.000
```

Output for two solutions should look like this:

```
> Two solutions found x=-2.414 and x=0.414
```

Output for no solutions (when delta is less than zero):

```
> No solutions found
```

> Imports will work only when these two files will be placed within the same directory

2.3. Write a script that takes an input string from the user and transforms it according to three rules:

1. delete all vowels,

2. leave all consonants and place `.` (dot) after them
3. transform all uppercase letters for lowercase letters

Examples:

- input `Programming` should be transformed into `p.r.g.r.m.m.n.g.`
- input `ABACC` should be transformed into `b.c.c.`
- input `aaa` should be transformed into an empty string

> Consider only inputs consisting of lowercase and uppercase ascii letters (a-z) and (A-Z)

Homework 3.

3.1. Write a function `coins(value)` which calculate coin combinations for a given value (amount of gr). For example if we want to pay 62gr, we have to prepare 1 x 50gr coin, 1 x 10gr coin and 1 x 2gr coin. Function should return a list with 6 elements: 1st element corresponding to the number of 50gr coins, 2nd element corresponding to the number of 20gr coins, and so on up to 6th element corresponding to the number of 1gr coins.

In this example function `coins(62)` should output `[1, 0, 1, 0, 1, 0]` because we have 1 x 50gr, 0 x 20gr, 1 x 10gr, 0 x 5gr, 1 x 2gr and 0 x 1gr.

More examples:

```
coins(70) # should return [1, 1, 0, 0, 0, 0]
coins(3) # should return [0, 0, 0, 0, 1. 1]
```

> `value` will be an integer between 0 and 99

3.2.

Imagine you are a secret agent and receive secret messages from your agency. Secret message is a string of random-looking characters like `kj4656%cwj4342dm`. Your job is to determine if you should change your location based on the secret message. If the first instance letter "c" in the string is immediately followed with the number "01" (operational code for changing location) you should change your location. Write a function `decipher(secret_string)` which returns either `True` or `False` based on the decision if you should change your location.

Examples:

```
decipher('qweqw34%c013fewca') # should return True, because first occurance of 'c' is followed by '01'
decipher('qceqw34%c013fewca') # should return False, because first occurance of 'c' is followed by 'eq'
decipher('sdwe6t544^d&fda65') # should return False, because there is no 'c' in secret string
```

Homework 4.

4.1. Write a function `moving_average(array, n)` that takes two inputs: `array` which is a list of numbers and `n` which is an integer. Return new list with values being a moving average with window size `n`. Moving average is an average calculated on `n` subsequent elements. For example:

```
array = [2, 2, 4, 5, 2]
n = 3 # window size
```

In this case moving average can be calculated as:

```
2, 2, 4, 5, 2
|     |
^^^^^^^
(2+2+4)/3 = 2.6666666666666665


2, 2, 4, 5, 2
```

```
   |     |
   ^^^^^^^
   (2+4+5)/3 = 3.6666666666666665


2, 2, 4, 5, 2
      |     |
      ^^^^^^^
      (4+5+2)/3 = 3.6666666666666665
```

so the function should oputut `[2.6666666666666665, 3.6666666666666665, 3.6666666666666665]`.

In the cases when moving average cannot be calculated, e.g. when `n` > `len(array)`, return `None`.

You may need these:

If `l` is our list, then:

- `sum(l)` returns the sum of all numbers in the list
- `len(l)` returns the length of the list
- `l.append(value)` adds new value to the end of the list
- `l[i:j]` returns sub-list from index i to index j (list slicing)

Additional reading about basic list operations.

4.2. The dragon's curve is a self-similar fractal which can be obtained by a recursive method.

Starting with the initial string 'Fa', at each step simultaneously perform the following operations:

- replace `'a'` with: `'aRbFR'`
- replace `'b'` with: `'LFaLb'`

This operations transforms the string (spaces were added to increase visibility):

- 1st iteration: `'Fa'` into `'F aRbFR '` (we don't replace `'b'` here, because `'b'` is not in `'Fa'`)
- 2nd iteration: `'FaRbFR'` into `'F aRbF R RLFaLb FR'` at second iteration
- 3rd iteration: `'FaRbFRRLFaLbFR'` into `'F aRbFR R LFaLb FRRLF aRbFR L LFaLb FR'`
- and so on...

After desired number of iterations, remove letters `'a'` and `'b'` from the output string. You will have a string with only `'R'`, `'L'`, and `'F'`. The goal of this task is to write a function `dragon(n)` wich takes one parameter `n`, the number of iterations needed and return the string of instruction as defined above.

Examples:

- `dragon(0)` should return `'F'`
- `dragon(1)` should return `'FRFR'`
- `dragon(2)` should return `'FRFRRLFLFR'`
- `dragon(3)` should return `'FRFRRLFLFRRLFRFRLLFLFR'`

> *Additional note*: Output string is a set of instruction. Starting at the origin of a grid looking in the (0, 1) direction, 'F' means a step forward, 'L' and 'R' mean respectively turn left and right. After executing all instructions, the trajectory will give a beautifull self-replicating pattern called Dragon Curve.

> *Tip*: String method `replace` might be useful for this exercise.

Homework 5.

5.1. Create a function `only_homogenous(array)` that takes as an input a list (`array`) of lists and return a new list containing only lists from the original list (`array`) that are homogenous (all items within a list have same type). We assume that empty lists are not homogenous. The result should contain lists with items in the same order as they were originally.

Example:

```
array = [[1, 2, 3], [1, '2'], [], ['a', '', 'ggg'], [1, 'f'], [2]]
only_homogenous(array) # should return [[1, 2, 3], ['a', '', 'ggg'], [2]]
                       # sub-lists [1, '2'], [] and [1, 'f'] should be discarded as they are not homogenous
```

For simplicty we will asume that lists will contain only integers and strings.

You may need:

- `isinstance(variable, int)` checks if a `variable` is an integer (works the same for `str`)
- `type(a) == type(b)` checks whether variables `a` and `b` are of the same type

> *Original array should not be modified*

5.2. Create a function `who_win(array)` that takes as an input 2D 3x3 array (represented as a list of lists) which corresponds to the board after the game of tic-tac-toe. Function should determine if the game was won by the player with `x` mark, `o` mark or ended as a draw. Example boards:

```
board_1 = [['o', ' ', ' '],
           ['x', 'x', 'x'],
           [' ', 'o', ' ']] # here 'x' wins
board_2 = [['o', ' ', ' '],
           ['x', 'o', 'x'],
           [' ', 'x', 'o']] # here 'o' wins
board_3 = [['x', 'x', 'o'],
           ['o', 'x', 'x'],
           ['x', 'o', 'o']] # here there is a draw
```

For these inputs function should work like this:

```
who_win(board_1) # should return 'x'
who_win(board_2) # should return 'o'
who_win(board_3) # should return 'd' because there was a draw
```

Homework 6.

6.1. Describe what are dictionaries in Python, how they work and why are they useful. Your homework should contain:

- description of dictionary data type
- cheatsheet containing basic and most useful operations on dictionaries (e.g. creating dictionary, retrieving data, changing, iterating, looking for items, etc.); cheatsheet may be in any form table, list of code snippets, etc.
- one **example programming exercise** and solution of this exercise that makes use of dictionary

> Please send solution as `.pdf` file named with your University index (e.g. `333333.pdf`)

Homework 7.

7.1. You will be given a list of non-negative integers and positive integer bin width.

Your task is to create the `histogram(data, bin_width)` function that will return histogram data corresponding to the input list. The histogram data is an array that stores under index i the count of numbers that belong to bin i. The first bin always starts with zero. Each bin has inclusive start and exclusive end.

For example if our list is `[0, 0, 0, 0, 1, 7, 9, 5, 1, 1, 2]` and our bin width is equal `3` then first bin will span from 0 (inclusive) to 3 (exclusive). In this range we have 4 x `0`, 3 x `1` and 1 x `2`, so the histogram value for the first bin will be 8. Second bin will span from 3 (inclusive) to 6 (exclusive) so the value for this bin will be 1 (there is only one `5` in the list). Third bin spanning from 6 to 9 (exclusive) will have a value of 1, and the last bin spanning from 9 (inclusive) to 12 (exclusive) will also have value of 1. Finally, the output should be `[8, 1, 1, 1]`

On empty input you should return empty output.

More examples:

- for input data [1, 1, 0, 1, 3, 2, 6] and `bin_width = 1` the result will be [1, 3, 1, 1, 0, 0, 1] as the data contains single element "0", 3 elements "1" etc.
- for the same data and `bin_width = 2` the result will be [4, 2, 0, 1]
- For input data [7] and `bin_width = 1` the result will be [0, 0, 0, 0, 0, 0, 0, 1]

7.2. Given two lists of names and phone numbers write a function `phonebook(names, numbers)` that returns a dictionary with names as dictionary keys and phone numbers as their vallues. Befory putting a name and the number in dictionary they both have to be validated. Invalid entries should be discarder.

Valid name should:

- contain only a-z letters
- start with capital letter
- have the rest of the letters lowercase

Examples of valid names are `Jack`, `Bob`, `Xayqyqy`, `C`. Examples of invalid names are `bob`, `jelly120`, `1Tom`, `AnA`, etc.

Valid number should be a string formatted like this `568-975-645` or `(+45)550-654-999`. Both types (with and without area code) should be assumed correct. Each area code has exactly two digits and starts with `+`. General format of valid number is either:

- `nnn-nnn-nnn` where `n` is a digit, or
- `(+nn)nnn-nnn-nnn`

Examples of invalid numbers: `(+5)999-999-645` (because area code has one digit), `5559-654-654` (incorrect number of digits), `654 654 654` (incorrect separator between number parts), `69X-546-132` (incorrect character in number), etc.

Some examples:

```
phonebook(['Bob', 'Kate', '1Tom'], ['555-555-555', '9654-654-654', '(+55)999-111-222']) # should return {'Bob': '555-5
phonebook(['Yi', 'Fo', 'Min'], ['111-222-333', '(+00)999-888-777', '222_555-999']) # should reuturn {'Yi': '111-222-33
phonebook(['jelly'], ['111-111-987']) # should return empty dict besause name is incorrect
```

Homework 8.

8.1. Hashing function is often used to store encrypted passwords in database. One popular (yet not secure!) hashing algorithm is called MD5. Your task is to crack a password being 4-digit number hashed with MD5 algorithm. Since they are not many combinations of 4-digit numbers, you can easily use brute force to acomplish this task. Write a function `crack(hash)` that takes an input `hash` which is 32 character string representation of a MD5 hash (written with hexadecimal characters).

To generate a hash from a password or message use `hashlib` built-in library in Python:

```
import hashlib

password = "1234"
hash = hashlib.md5(password.encode()).hexdigest()
```

If you print `hash` to the console, you will see `'81dc9bdb52d04dc20036dbd8313ed055'` hash string.

In addition to writing correctly working `crack` function you should protect it against incorrect input:

- raise `TypeError` if an input is not a string
- raise `ValueError` if an input is a string, but is not valid hash string (does not contain 32 characters or contain illegal characters – hexadecimal representation only contains digits 0-9 and characters a-f).

Examples:

```
crack('81dc9bdb52d04dc20036dbd8313ed055') # should return '1234'
crack('959ab9a0695c467e7caf75431a872e5c') # should return '6481'
crack(1) # should raise TypeError
crack('959a') # should raise ValueError
```

8.2. Write a helper function `wrap(text, tags)` which wraps a text in HTML tags. Some examples:

```
wrap('Hello', ['p', 'div', 'h1']) # should produce a string '<p><div><h1>Hello</h1></div></p>'
wrap('Hello', []) # should produce a string 'Hello'
wrap('Hi', ['h2', 'h3', 'p']) # should produce a string '<h2><h3><p>Hi</p></h3></h2>'
wrap('', ['p']) # should produce a string '<p></p>'
```

Note that first tag in a list should be outer tag enclosing all other tags, and last tag in a list should be inner tag directly enclosing a text. For simplicity assume that allowed tags are `p` (paragraph), `div` (block level logical division) `h1`, `h2`, `h3`, `h4`, `h5`, `h6` (section headings). You should validate your inputs:

- raise `TypeError` if `text` is not a string (although it can be empty string)
- raise `TypeError` if `tags` is not a list (althought it can be empty list)
- raise `ValueError` if an element of `tags` is not a valid tag (valid tags are `p`, `div`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`; each tag should be a string)

Homework 9.

9.1. Write a function `multiply_matrix(a, b)` which takes as an input two 2-D matrices of arbitrary size and returns new matrix being a product of matrix multiplication of `a` and `b`. For simplicity assume that input matrices are correct, i.e. they are valid list of lists containing only numbers. The only thing you need to check for are sizes of both matrices – for some sizes mutliplication is not possible.

Assuming matrix `A` is of size `n_a x m_a` and matrix `B` is of size `n_b x m_b`, multiplication is possible only if `m_a == n_b`. If the operation is not possible due to incorrect matrix sizes, function should return `ValueError` with an appropriate message.

Examples:

```
a = [[ 1, 2, 3, 1 ],
     [ 0, 2, 2, 1 ]]
b = [[ 1, 0, 0 ],
     [ 0, 2, 0 ],
     [ 0, 3, 0 ],
     [-2, 0, 4 ]]
c = multiply_matrix(a, b)
# c should be equal to
# [[ -1, 13, 4 ],
#  [ -2, 10, 4 ]]

a = [[ 1, 2, 3, 1 ],
     [ 0, 2, 2, 1 ]]
b = [[ 1, 0, 0 ],
     [ 0, 2, 0 ],
     [-2, 0, 4 ]]
c = multiply_matrix(a, b) # should raise ValueError because # of columns in a != # of rows in b
```

> You can use online calulator to test your function.

9.2. Write a function `linspace(start, stop, n)` which takes three integers start, stop and number as input and returns a list from start to stop with number total values in the list. All numbers in a list should be equally spaced between `start` and `stop`. Don't worry about input validation. Assume that `stop` is always greater than `start`.

Examples:

```
linspace(1, 5, 0) = []
linspace(1, 5, 1) = [1.0]
linspace(1, 5, 2) = [1.0, 5.0]
linspace(1, 5, 3) = [1.0, 3.0, 5.0]
linspace(1, 5, 4) = [1.0, 2.333333333333333, 3.6666666666666665, 5.0]
linspace(1, 5, 5) = [1.0, 2.0, 3.0, 4.0, 5.0]
```

Homework 10.

10.1. Write a function `transpose(a)` that takes 2-D array (matrix) as an input and returns it's transpose. Keep the function pure – do not modify input matrix. Matrix transposition swaps the columns and rows of the original matrix. Matrix will be represented as usual as a list of lists of numbers. You don't need to worry about input validation.

Example:

```
a = [[1, 2, 3],
     [4, 5, 6]]
a_t = transpose(a) # transposition of 2x3 matrix
# should return 3x2 matrix
# [[1, 4],
#  [2, 5],
#  [3, 5]]
```

10.2 Write a function `tricky(l)` that **will pass** these **exact** tests:

```
class TestTricky(unittest.TestCase):
     def test_tricky(self):
          self.assertEqual(tricky([2, 3, 3, 4, 5]), 10)
          self.assertEqual(tricky([-2, -1, 2, 3]), -6)
          self.assertEqual(tricky([0, 0, 0, 1]), 0)
          self.assertEqual(tricky([1, 2, 3]), 3)
          self.assertEqual(tricky([-1, 1]), -1)
          self.assertEqual(tricky([5]), 25)

          with self.assertRaises(ValueError):
               tricky([])
          with self.assertRaises(TypeError):
               tricky('[1, 2, 3]')
```

> Hint: There are multiple approaches to this problem. As long as your function will pass required tests it is fine.