exercises.md

1. Basic types

1.1. Write a simple script that asks user for the name and year of birth and print greeting and calculated age in one sentence.

1.2. Write a simple script that asks user for speed in km/h and converts into mph.

2. Flow control

2.1. Print decision if user age is greater than 20 and user name has at most 5 characters.

2.2. Write a program that calculates the distance of thrown ball.

Program should ask user for:

- initial velocity (in m/s)
- initial angle (in degrees)
- if there is an initial height

Program should validate user input checking if:

- initial velocity is a number > 0
- initial anlge is a number > 0 and < 90
- initial height is a number > 0 (if it is used)

Wikipedia article about projectile motion.

You may need:

- program exit: `from sys import exit`
- trigonometric functions: `from math import sqrt, sin, cos`
- limiting number to two decimal points: `{:.2f}'.format(my_number)`

2.3. Write a program using while loop that ask user for password exactly three times. If the password is correct display hello message, otherwise display account blocked message. On each incorrect password give appropriate message with remaining number of attempts. Password is `test123`.

2.4. Write a program which brute force guess user selected password. Password should be three characters long and may contain lowercase ascii characters, numbers and uppercase ascii characters. Program should terminate after 1 million incorrect password guesses. If the password is found program should output the password and number of attempts required to correctly guess the password.

You may need:

- strings with ascii characters and digits: `from string import ascii_lowercase, ascii_uppercase, digits`
- function choosing random character from the string: `from random import choice`

> *Play around with different character sets and see how it impacts number of guesses required to find correct password*

> Does the number of guesses depend on the password complexity or character set lenght?

2.5. Write a program (using `while` loop) that determines whether integer is a prime number. Wikipedia article about primality tesst

3. Iteration

3.1. Sum all numbers from 1 to 1000.

3.2. Sum all numbers that doesn't contain digit 1 from 1 to 1000.

You may need:

- checking wheter specific letter is inside string: `letter in string`, e.g. `'a' in 'abc'` should return `True`

3.3. Ask user to specify two integers: `a` and `b`. Calculate the sum of all numbers divisible by 3 between `a` and `b`.

3.4. Use `for` loop, `range()` function and one of formatting techniques to produce output:

```
sub-0080
sub-0085
sub-0090
sub-0095
sub-0100
sub-0105
sub-0110
sub-0115
sub-0120
```

> *Checking formatting documantiation may be useful*

3.5. Write a program that selects random number between 1 and 100 (inclusive) and ask user to take subsequent guesses until the number is guessed correctly. After each guess program should give a feedback if guessed number is too high or too low.

You may need:

- function that returns random intiger from specified range: `from random import randint`

3.6. Solve Kata (7 kyu): Coloured Triangles

You may need:

- getting first letter from a string `s`: `s[0]`
- getting i-th letter from a string `s`: `s[i-1]`
- getting last letter from a string `s`: `s[-1]`

> You don't have to write a function at this point. Just write a script which solves the puzzle.

4. Functions

4.1. Write a function that simulate process of rolling a dice. It should output sum of all throws. If 6 is thrown, then the program should roll again (as many times as needed to throw number other than 6).

Example:

- sequence: `4` should output `4`
- sequence: `6, 1` should output `7`
- sequence: `6, 6, 6, 5` should output `23`

4.2. Prove that functions without return statement implicitely return `None`.

4.3. Assume we have function with three arguments:

```
    print(f'a={a}, b={b}, c={c}')
```

Write down all possible and unique ways to call function `fun` to produce output `a=1, b=2, c=3`.

4.4. Solve Kata (8 kyu): Are You Playing Banjo?

4.5. Write a function calulating factorial.

4.6. Solve Kata (7 kyu): Credit Card Mask?

4.7. Given two numbers (m and n) :

- convert both of them to binary

- sum them as if they were in base 10
- convert the result to binary
- return as string

Example:

```
binary_pyramid(1,4) # should return "1111010"
```

You may need:

- `bin()` built-in function converting numbers to their binary representation

5. Lists

5.1. Ask user to name three vegetables. If `'carrot'` is among them print the list replicated two times. Othewise print the list with `'carrot'` added three more times.

5.2. Create function `ski_jump_score(judges_points, distance, k_point)` which calulates and returns total number of points that ski jumper is awarded for his jump. Briefly, total score is score for distance (`1.8 * (distance - k_point)`) plus score from judges points (highest and lowest score is discarded and the sum is calculated) plus initial number of 60 points. You can read more here. Assume default value for `k_point = 120`.

Example:

```
> ski_jump_score([17, 15, 18, 17, 19], 126)
> 122.8
```

5.3. Write a script that asks the user to input shopping information: product names and product prices. User can specify as many products as he wants. Function should print (nicely formatted) aggregated informations about the products.

Running script should first prompt user for products:

```
> Product 1 name:
> Bread
> Product 1 price:
> 3
> Product 2 name (ENTER to skip):
> Ham
> Product 2 price:
> 5
> Product 3 name (ENTER to skip):
> Pepsi
> Product 3 price:
> 5.50
> Product 4 name (ENTE to skip):
>
```

And then output statistics:

```
Cheapest.........................Bread
Most expensive.....................Pepsi
Avg. price.........................4.17
```

5.4. Solve Kata (7 kyu): Maximum Product

Given an array of integers, find the maximum product obtained from multiplying 2 adjacent numbers in the array.

Example:

```
adjacent_element_product([9, 5, 10, 2, 24, -1, -48]) # should return 50
```

5.5. Solve Kata (7 kyu): Row Weights

Several people are standing in a row divided into two teams. The first person goes into team 1, the second goes into team 2, the third goes into team 1, and so on.

Given an array of positive integers (the weights of the people), return a new array / tuple of two integers, where the first one is the total weight of team 1, and the second one is the total weight of team 2.

Example:

```
row_weights([50, 60, 70, 80]) # should return (120, 140)
```

5.6. Solve Kata (7 kyu): Peak array index

Given an array of ints, return the index such that the sum of the elements to the right of that index equals the sum of the elements to the left of that index. If there is no such index, return -1. If there is more than one such index, return the left-most index.

Example:

```
peak([1, 2, 3, 5, 3, 2, 1]) # should return 3, because 1+2+3 = 3+2+1
peak([1, 12, 3, 3, 6, 3, 1]) # should return 2, because 1+12 = 3+6+3+1
peak([10, 20, 30, 40]) # should return -1
```

6. Variables revisited

6.1. Write a function `rotate(vector, angle)` that takes two parameters: `vector` which is tuple with 2 floats, and angle (in degrees) and returns coordinates of rotated vector. Here you can find formula for 2D rotation. Set default value of 0 degrees for angle parameter.

7. Dictionaries, function testing

7.1. Given following input data:

```
from string import ascii_lowercase
from itertools import product
from random import randint

# map, lambda functions, string join method, product
names = list(map(lambda x: ''.join(x), product(ascii_lowercase, repeat=3)))

# list comprehension
numbers = [f'{randint(0, b=999_999):06}' for _ in names]

# dictionary comprehension, zip function
phonebook = {k: v for k, v in zip(names, numbers)}
```

Write two functions `find_list` and `find_dict` which finds phone number for specific name required number of times. Each function should take two inputs:

- three character name (e.g. `abc` and get, but not return or display, corresponding phone number)
- operation of finding a number should be repeated `reps` number of times (default 10,000 times)

7.2. Solve **5.4.** and **5.5.** from `lesson_05` using list comprehensions.

7.3. Write a function `add_matrix` that accepts two matrices (n x m two dimensional arrays), and return the sum of the two. Both matrices being passed into the function will be represented as a list of the list.

How to sum two matrices:

Take each value `matrix_1[n][m]` from the first matrix, and add it with the same (corresponding) `matrix_2[n][m]` value from the second matrix. This will be a value `[n][m]` of the solution matrix.

7.4. Extend function `add_matrix` and validate user provided matrices:

- check if both inputs are lists (if not raise `TypeError`)
- check if inner lists have equal lenght (if not raise `ValueError`)
- check if both matrices have same size (if not raise `ValueError`)

7.5. Use `assert` statement to test function `adjacent_element_products`.

> When writing test try to predict *edge cases*, i.e. input combinations thate are likely to brake the function.

7.6. Use `input` function to ask the user to input the number. Repeat asking until correct number is passed. If a user pass a string that cannot be converted to number, warn him with a message. If a correct number is passed print this number multiplied by 10.

> You may need `.isnumeric()` string method

7.7. Use `input` function again to ask the user to input the number. All rules stay the same, except now you want to use `try`, `except` statements to achieve the same result.