Exercise set 6

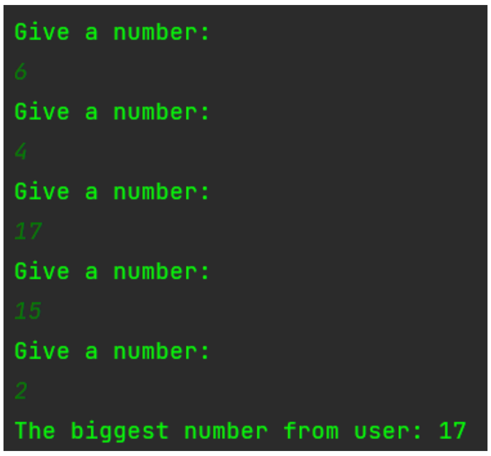
Combining repetition statements and conditional statements, break and continue -commands

1. Create an application that asks user to input 5 positive integers **in a loop**. Print **only** the biggest number after the loop.

   Tip: Create a helper variable **outside** the loop, which keeps track of the largest number at any moment. Change the number every time if the new number from the user is bigger than the previously biggest number!

   Extra task: Check also that the user inputs the number in correct format (positive integer)

   Example of the application running:



```
Give a number:
6
Give a number:
4
Give a number:
17
Give a number:
15
Give a number:
2
The biggest number from user: 17
```

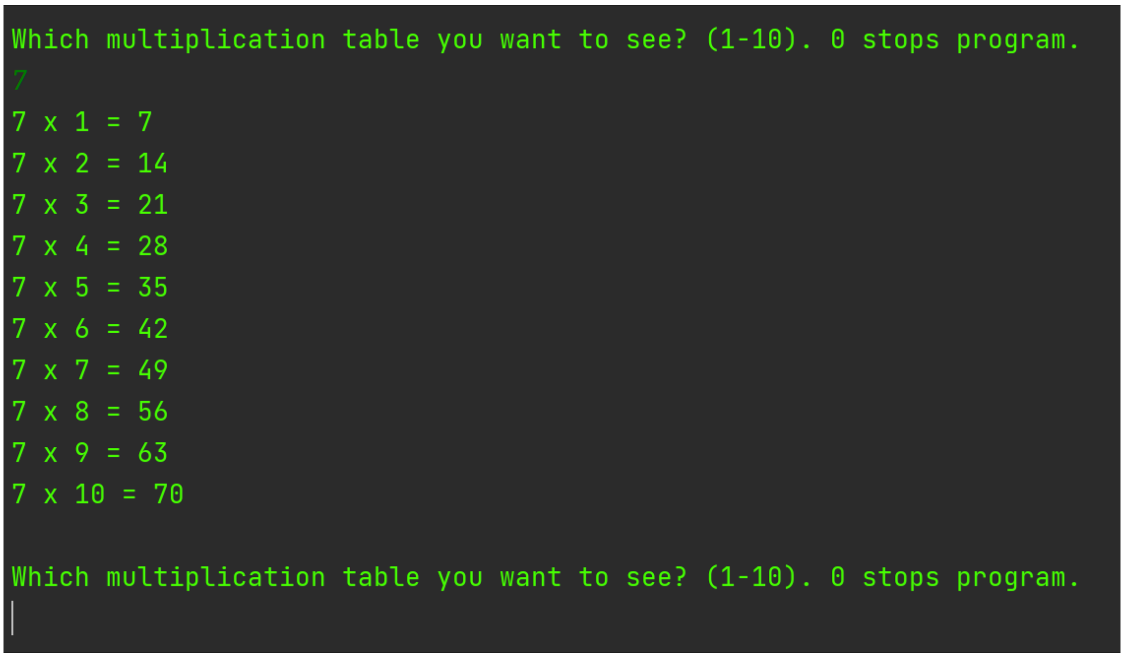
Filename of the exercise = ***exercise6_1.py***

Typical code amount : ***6-10 lines*** (empty lines/comments not included)

2. Create an application that allows the user to print the chosen multiplication table in the range of 1-10. Allow the user to use the application as many times as they want. Stop the application **only** if the user inputs **"0"** (zero).

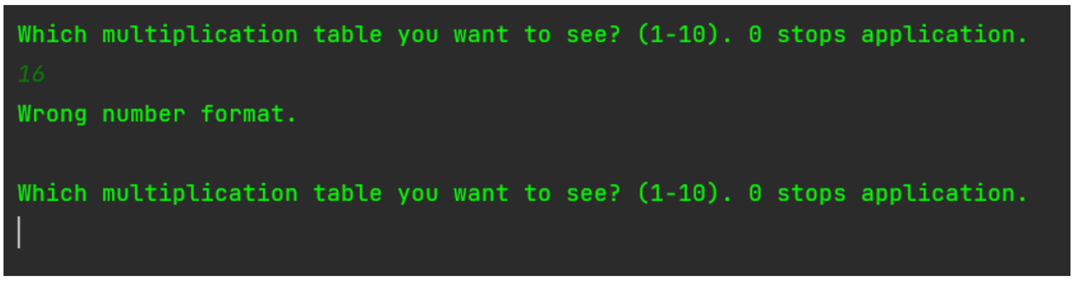
   In your application, check that the number given by the user is between 0 and 10. If this is not the case, print "Wrong number format.", and ask the number again until the user inputs a proper value.

   Examples of the application running:



```
Which multiplication table you want to see? (1-10). 0 stops program.
7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

Which multiplication table you want to see? (1-10). 0 stops program.
|
```



```
Which multiplication table you want to see? (1-10). 0 stops application.
16
Wrong number format.

Which multiplication table you want to see? (1-10). 0 stops application.
|
```

Filename of the exercise = ***exercise6_2.py***

Typical code amount: ***10-16 lines*** (empty lines/comments not included)

3. Create an application that allows the user to calculate statistics based on exam results by students. The application should work the following way:
    a) Ask the number of students (test with small values at first)

    b) Ask the exam grade of each student **in a loop.**

    c) Calculate the **average** of the grades and print it on the screen.

    d) In addition, print a text description of the average on the screen:

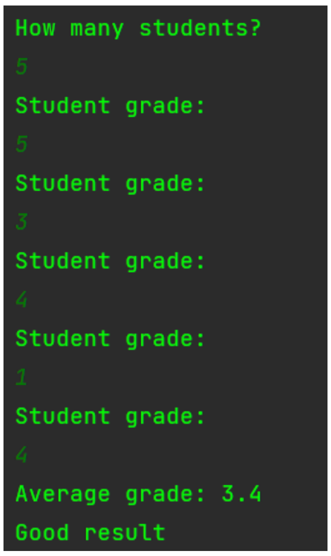
0 <= **average** < 1	**Bad result**
1 <= **average** < 2	**Weak result**
2 <= **average** < 3	**Average result**
3 <= **average** < 4	**Good result**
4 <= **average** <= 5	**Excellent result**

Extra task: Save all grades into a list, and use it to calculate the average.

Second extra task: Calculate also the median value of the grades.

Third extra task: Calculate also the most common grade
(**Tip:** use a list!)

Example of the application running:

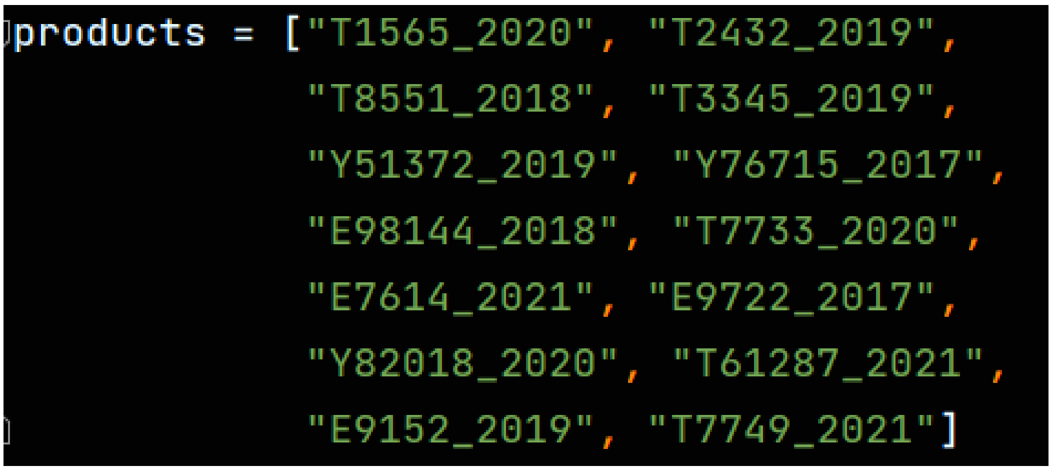


```
How many students?
5
Student grade:
5
Student grade:
3
Student grade:
4
Student grade:
1
Student grade:
4
Average grade: 3.4
Good result
```

Filename of the exercise = ***exercise6_3.py***

Typical code amount: ***18-28 lines*** (empty lines/comments not included)

4. Create an application that contains the following list of product identifiers:



```
products = ["T1565_2020", "T2432_2019",
            "T8551_2018", "T3345_2019",
            "Y51372_2019", "Y76715_2017",
            "E98144_2018", "T7733_2020",
            "E7614_2021", "E9722_2017",
            "Y82018_2020", "T61287_2021",
            "E9152_2019", "T7749_2021"]
```

NOTE! A copy-paste –version of this list exists in Moodle!

The list consists of simple product identifiers, where a single identifier follows the following formula:

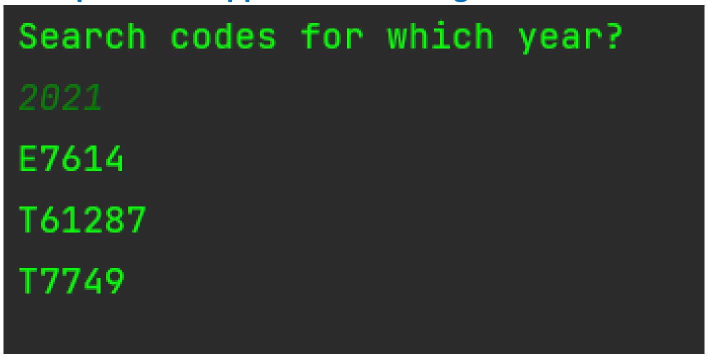
ORDERCODE_YEAR

Ask the user for a year, and **only print those order codes from the list, that have the matching year** (use a loop). For example, if the user inputs "2019", the codes ”T2432”, "T3345”, "Y51372" and "E9152" are printed. Notice that the order code might be either 4 or 5 characters long, depending on situation.

Tip: Use the split() –function! (examples in materials)

Note: Using a **if year in code** – type of solution **is not enough**, since some order codes might also contain a year-like number! **(see: code Y82018_2020)**

Example of the application running:



```
Search codes for which year?
2021
E7614
T61287
T7749
```

Filename of the exercise = ***exercise6_4.py***

Typical code amount: ***12-18 lines*** (empty lines/comments not included)

5. Use the same product identifier list **(products)** as in previous exercise.

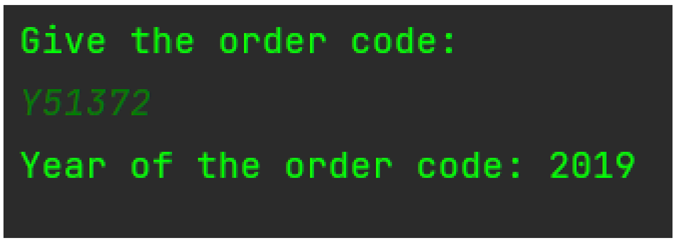
   Create an application that asks the user for an order code (for example, E9722). Then loop through the **products**-list, and when you find the user's order code, **print out only the year**, and use the **break-**command immediately.

   Note: The break-command is very useful in this situation, since the idea of the application is to find a certain order code's year. When the year has been found, **there's no reason to continue searching, because we already found what we were looking for**! The break-command stops the loop earlier than intended to improve application performance.

   Tip: Use the split() –function! (examples in materials)

   Note: Using a **if year in code** – type of solution **is not enough**, since some order codes might also contain a year-like number! **(see: code Y82018_2020)**

   Example of the application running:



```
Give the order code:
Y51372
Year of the order code: 2019
```

Filename of the exercise = ***exercise6_5.py***

Typical code amount: ***12-18 lines*** (empty lines/comments not included)

6. Create an application that contains a list called "**basket",** and add the following words in it:
    - "apple"
    - "banana"
    - "cherry"
    - "carrot"
    - "potato"
    - "tomato"
    - "cabbage"

    Then ask the user for a word. After this, print the contents of the **basket-**list on separate lines (by using a loop), **but do not print the word the user gave as input** (for example, by using the **continue –command).**

    If the given word is not found in the list, print **"Word not found.", and don't print the contents of the basket-variable at all!**

    Note: This can be done in many ways. The most straight-forward approach is to loop through all the food in the **basket-list,** and when you find the user's word in the loop, use the **continue-**command to skip it, for example:

    (e.g. **if word == userword**) => ***continue***

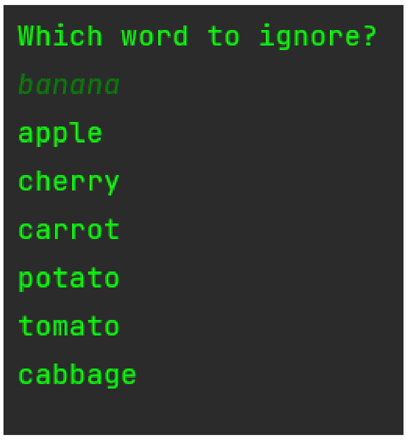
    *In this case, remember to use the above-mentioned if-statement **before** printing the food, so that the **continue** -command can react to the situation before printing.*

    You can also do this by using an **if/else** –structure or with collection functions.

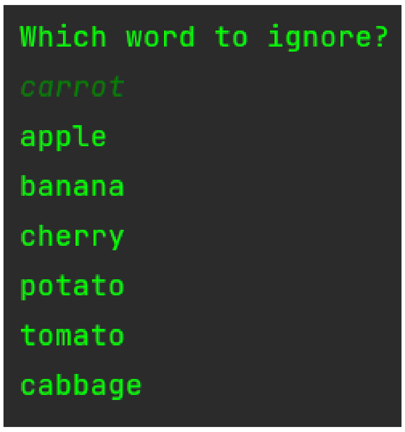
Extra task: If the user inputs a number (index) instead of a word, print all the other foods in the list, except the word that corresponds to the position of the word in the list based on the number. For example, if the user inputs "3", print everything else except "**cherry**" in that case.

Tip: Use the **text.isnumeric()** –function to figure out if the user inputed text or numbers!

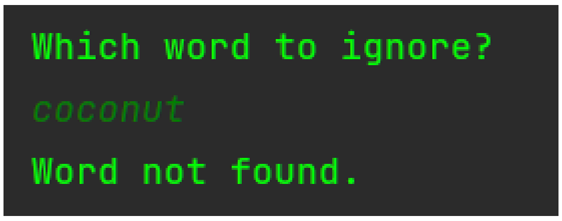
Examples of the application running:



```
Which word to ignore?
banana
apple
cherry
carrot
potato
tomato
cabbage
```



```
Which word to ignore?
carrot
apple
banana
cherry
potato
tomato
cabbage
```



```
Which word to ignore?
coconut
Word not found.
```

Filename of the exercise = ***exercise6_6.py***

Typical code amount: ***8-20 lines*** (empty lines/comments not included)

Advanced exercises:

Note: A good grade does not require all advanced exercises. Choose the ones that interest you the most!

7. Create an application that searches a suitable number between a range given by the user. In this case, a suitable number is divisible by both 5 and 7. The application works so, that the user is asked to input the start and the end of the range (two inputs). After this, loop through each number in that range, and print information whether the number is divisible by 5 and 7 or not.

   If the number is not divisible by 5, move to next number immediately. Stop the search immediately, if the application finds a suitable number. The end result could be something like this:

```
Give the range start: 15
Give the range end: 100
15 is not divisible by 7, skip.
16 is not divisible by 5, skip.
17 is not divisible by 5, skip.
18 is not divisible by 5, skip.
19 is not divisible by 5, skip.
20 is not divisible by 7, skip.
21 is not divisible by 5, skip.
22 is not divisible by 5, skip.
23 is not divisible by 5, skip.
24 is not divisible by 5, skip.
25 is not divisible by 7, skip.
26 is not divisible by 5, skip.
27 is not divisible by 5, skip.
28 is not divisible by 5, skip.
29 is not divisible by 5, skip.
30 is not divisible by 7, skip.
31 is not divisible by 5, skip.
32 is not divisible by 5, skip.
33 is not divisible by 5, skip.
34 is not divisible by 5, skip.
The number 35 is divisible by both 5 and 7!
Stopping the search.
>>> EXAMPLE 2:
Give the range start: 11
Give the range end: 13
11 is not divisible by 5, skip.
12 is not divisible by 5, skip.
13 is not divisible by 5, skip.
Suitable number not found within range.
```

Filename of the exercise = ***exercise6_7.py***

Typical code amount: ***20-36 lines*** (empty lines/comments not included)

8. Create an application that produces safe passwords of given length. Firstly, ask the user of the password length, and don't accept passwords less than 8 characters. After this, create a new password (by using a loop) of the desired length, which follows these rules:
   a) The password has to have at least one lower letter
   b) The password has to have at least one CAPITAL letter
   c) The password has to have at least one number
   d) The password has to have at least one of the following special characters:
      - _ ~ ? * $ # ! + % @

   Note: This exercise can be done in many different ways!

Filename of the exercise = ***exercise6_8.py***

Typical code amount: ***16-32 lines*** (empty lines/comments not included), this can take way more code lines than this, if the implementation is very sophisticated!

9. Create an application that calculates the new sale price for a second hand car based on these information (ask from the user):
    - Original price of the car
    - Year of manufacturing
    - Driven kilometers (mileage)
    - Price category of the manufacturer (1 or 2)
    - Is it an import car or not (y/n)

    The application should calculate the new sales price of the car by following this logic:

    - **If it's price category 2:**
        - If the driven kilometers are approximately 30000 km / year or more:
            - The original price decreases 10% by each year for the first five years, after which the price decreases 7% for each year
        - If the driven kilometers are less than 30000 km / year:
            - The original price of the car decreases 8% by each year for the first five years, after which the price decreases 5% for each year.
        - However, the price never decreases below 12% of the original price of the car.

- **If it's price category 1:**
    - If the driven kilometers are approximately 30000 km / year or more:
        - The original price decreases 7% by each year for the first five years, after which the price decreases 4% for each year
    - If the driven kilometers are less than 30000 km / year:
        - The original price of the car decreases 5% by each year for the first five years, after which the price decreases 3% for each year.

    - However, the price never decreases below 18% of the original price of the car.

- Any import car will get 24% of tax added to the new sale price of the car.

After the application has calculated the new sale price for the car, ask the user, whether they want to insert another car into the application. If they answer "n", stop the application, and thank the user for using the application. Otherwise start the application from the beginning.

Filename of the exercise = ***exercise6_9.py***

Typical code amount: ***40-60 lines*** (empty lines/comments not included)