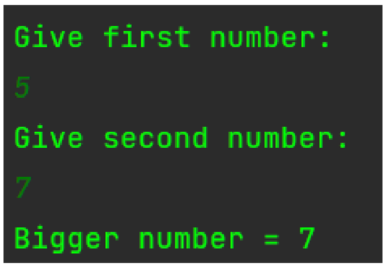
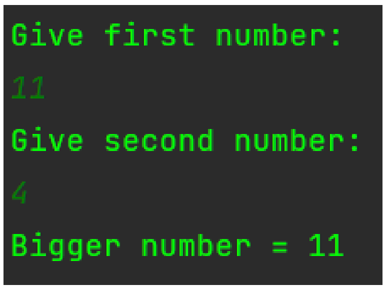
Exercise set 3
Conditional statements (**if / elif / else)**

1. Create an application that asks the user for two integers, and **prints only the bigger integer** on screen. If the numbers are equal, print *"Numbers are equal."*.

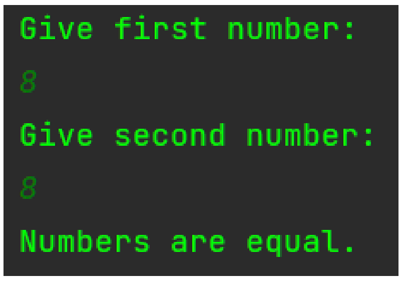
   Examples of the application running:



```
Give first number:
5
Give second number:
7
Bigger number = 7
```



```
Give first number:
11
Give second number:
4
Bigger number = 11
```



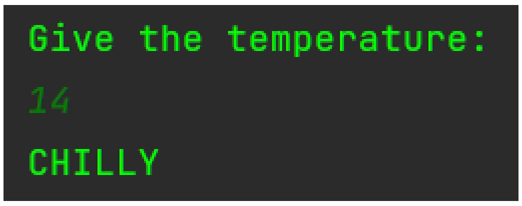
```
Give first number:
8
Give second number:
8
Numbers are equal.
```

   Filename of the exercise = ***exercise3_1.py***

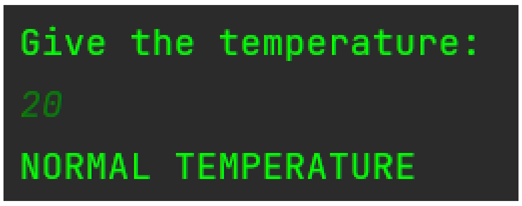
   Typical code amount: **8-12 lines** (empty lines/comments not included)

2. Create an application that asks the user for a summer day's temperature as an integer. Depending on what the user inputs as the temperature, print one of the following statements on the screen:
    - "COLD", if temperature is 0-10.
    - "CHILLY", if temperature is 11-15.
    - "NORMAL TEMPERATURE", if temperature is 16-20.
    - "WARM", if temperature is 21-25.
    - "HOT", if temperature is 26-30.

Examples of the application running:



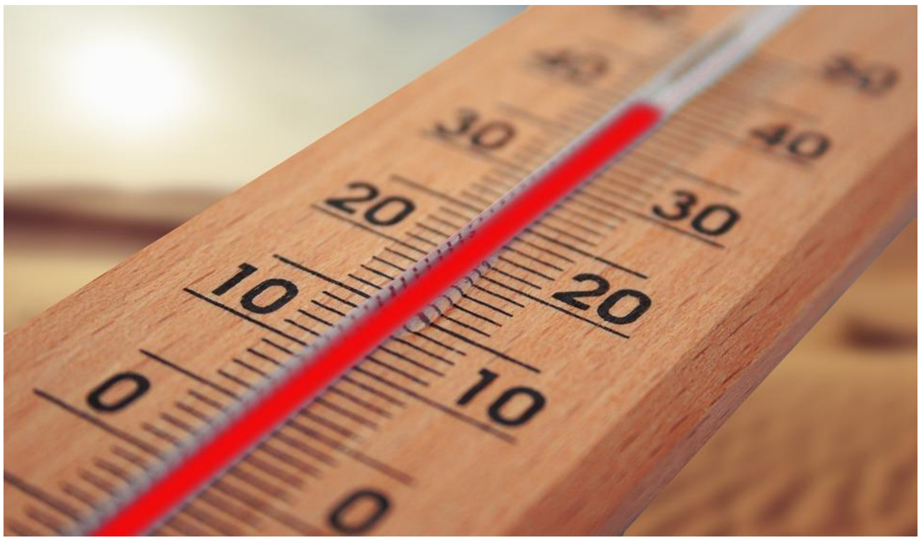
```
Give the temperature:
14
CHILLY
```



```
Give the temperature:
20
NORMAL TEMPERATURE
```

Filename of the exercise = ***exercise3_2.py***

Typical code amount: **11-16 lines** (empty lines/comments not included)



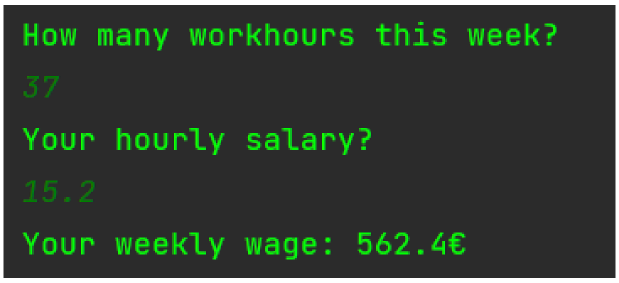
3. Create an application that asks the user the number of workhours for the week and their hourly wage. Based on the given workhours and the hourly wage, calculate the user's weekly wage.

   If the user has worked overtime during the week, the overtime hours have a 50% increase to their hourly wage. All hours above 40 are overtime hours.

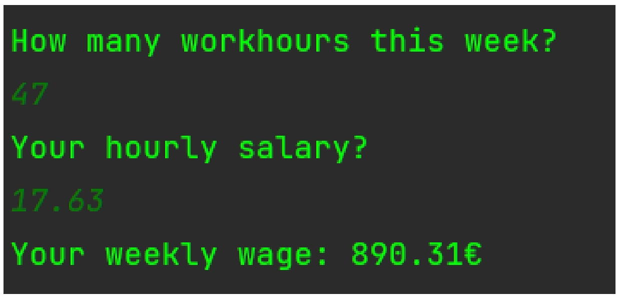
   Round the result into two decimals.

   Note: The overtime wage increase **should only consider overtime hours**, **not all hours**! (it's a good idea to calculate the wage of the first 40 hours first and then the wage of the hours that go over 40 hourss, and then sum them together)

   Examples of the application running:



```
How many workhours this week?
37
Your hourly salary?
15.2
Your weekly wage: 562.4€
```



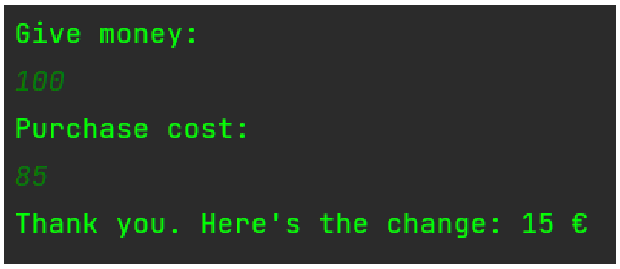
```
How many workhours this week?
47
Your hourly salary?
17.63
Your weekly wage: 890.31€
```

Filename of the exercise = ***exercise3_3.py***

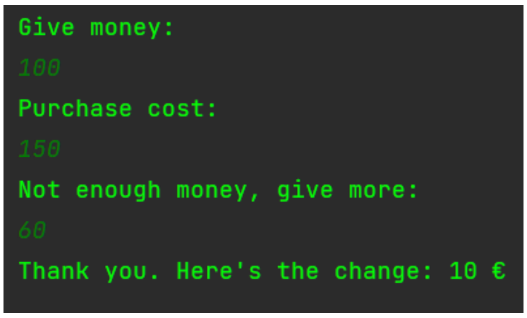
Typical code amount: **10-16 lines** (empty lines/comments not included)

4. Create an application that asks the user for thew amount of money and the cost of the purchase. The idea of the application is to inspect the cost and the money and decide, whether the user provided enough money or not.
   - If the money is enough on the first try, print "Thank you" and also the amount of change.
   - If there's not enough money, ask the user again to input more money for one more time
     - After this, check again whether the money is now enough for the purchase. If the money is still not sufficient, print "You don't have enough money." If the money is enough, print "Thank you" and the amount of change.

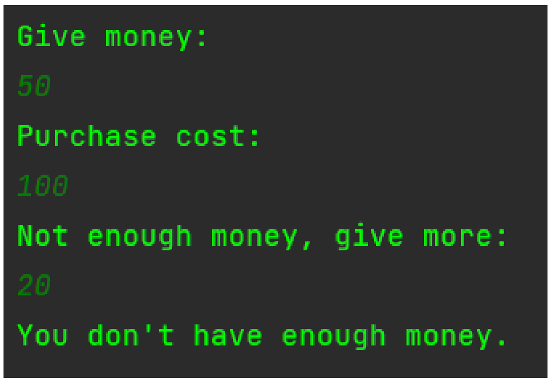
Examples of the application running:



```
Give money:
100
Purchase cost:
85
Thank you. Here's the change: 15 €
```



```
Give money:
100
Purchase cost:
150
Not enough money, give more:
60
Thank you. Here's the change: 10 €
```



```
Give money:
50
Purchase cost:
100
Not enough money, give more:
20
You don't have enough money.
```

Tip: Use the same words in your printing as in examples above, this will help ***codePost*** to understand your code better!

Filename of the exercise = ***exercise3_4.py***

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

5. Create an application that allows a teacher to get the grade of an exam based on the points table included.

   Ask the user the amount of points, and print the matching grade based on it.

   If the user inputs an impossible score, inform the user: "Invalid points value." (smaller than 0 or bigger than 100)

Points	Grade
0 - 50	0
51 - 60	1
61 - 70	2
71 - 80	3
81 - 90	4
91 - 100	5

Examples of the application running:

```
Exam points:
36
Grade: 0
```

```
Exam points:
72
Grade: 3
```

```
Exam points:
87
Grade: 4
```

```
Exam points:
60
Grade: 1
```

```
Exam points:
-15
Invalid points value.
```

```
Exam points:
150
Invalid points value.
```

Filename of the exercise = ***exercise3_5.py***

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

6. A leap year is divisible by 4, but not by 100. However, if a year is divisible by 400, it's considered a leap year always. Create an application that asks the user for a year, and print on the screen whether the given year was a leap year or not. If the year was a leap year, print "Leap year: YES", and if not, print "Leap year: NO".

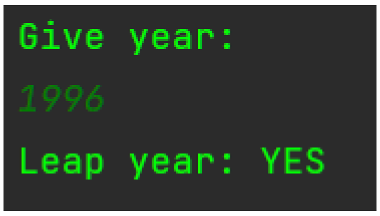
   Tips:

   This can be done either by a single if/else –statement, or with nested if-statements. You can use the approach that suits you best!

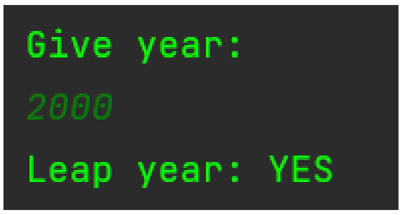
Examples of leap years:	**Examples of years, that are not leap years**
1996 (divisible by 4)	2015 (not divisible by 4 or 400)
2000 (divisible by 400)	1994 (not divisible by 4 or 400)
2016 (divisible by 4)	1900 (divisible by 100, but not by 400)

   Small extra task: Use a Boolean –variable to indicate, whether a year is a leap year or not (True / False), and use it to print "Leap year: YES" or "Leap year: NO"

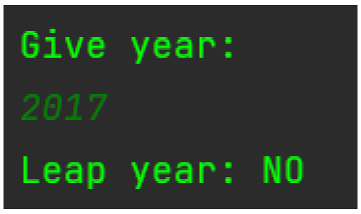
Examples of the application running:



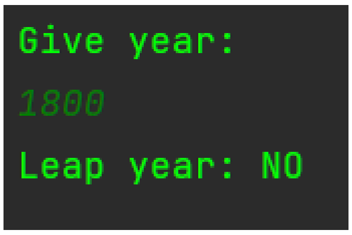
```
Give year:
1996
Leap year: YES
```



```
Give year:
2000
Leap year: YES
```



```
Give year:
2017
Leap year: NO
```



```
Give year:
1800
Leap year: NO
```

Filename of the exercise = ***exercise3_6.py***

Typical code amount: **10-20 lines** (empty lines/comments not included)

Tip! One way to solve this is to follow a code structure like this:

```
If the year is divisible by 400, it's always a leap year.
If the year is not divisible by 400, then this is the logic:
    If the year is divisible by 100, it's not a leap year
    If the year is divisible by 4, it's a leap year
    In other cases, it's not a leap year
```

Advanced exercises!

7. Create an application that functions as a shipment cost / postage calculator. The application should support both letters and parcels. Firstly, ask the user if it's a letter or a parcel, and then the weight of the shipment. Use the following pricing table to create the calculations.

Type	**Base cost**	**Weight class**		
		< 200g	**200g - 500g**	**> 500g**
Letter	50 cents	no extra cost	4 cent per 100g	7 cent per 100g
Parcel	2 €	no extra cost	8 cent per 100g	14 cent per 100g

- The extra costs based on the weight class of the delivery are measured by full 100 grams -> for example, a letter weighing 499g, the extra cost will be 400g * 4 cent per 100g => 16 cents

- **If a letter is more than 500g and it does not fit the mailbox,** add an extra 2 € to the price (ask the user if the letter fits the mailbox or not)

Filename of the exercise = ***exercise3_7.py***

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

Note: *It's possible to do this with less than 20 code lines, but it'll require helper variables. (tip: create a helper variable for the price modifier, which keeps track of the current price based on if it's a letter or a parcel and the weight. For example, if the user wants to send a parcel of 400g, the modifier would be 0.08, which is 8 cent etc.)*

8. Create an application that asks the user for total cost of the order (€), whether the user is a student or not (Y / N) and whether the users is VIP customer or not (Y / N).

   If the user is a VIP customer, ask the user how many VIP points the user has from previous purchases. Finally, ask the user for a possible special sales code. Acceptable special sales codes are:

   - ***FALL22***
   - ***BK2SCHOOL***

   Use the following logic to alter the final cost:

   - Firstly, handle the sales codes:
     - The special sales code ***FALL22*** reduces the amount of total cost by 10%. The reduction does not apply to shipment costs.
     - The special sales code ***BK2SCHOOL*** reduces the amount of total costs by 20%, but only if the user is a student. The reduction does not apply to shipment costs.
     - The user can only use one of the special sales codes in the order

   - Secondly, if the user is a VIP customer: For each whole 10€ in the remaining total cost (after sales codes), give the user 100 new VIP points (do not include shipment costs)
     - If the user is a VIP customer, reduce the total cost by 5€ for each full 1000 VIP points
       - The new VIP points from the new purchase are included in this reduction

   - Finally, add the shipment costs (7.95€) on top of the total cost
     - If the total cost exceeds 99€, shipment costs are free

   Print out the final total cost including possible shipment costs.

Filename of the exercise = ***exercise3_8.py***

Typical code amount: **20-40 lines** (empty lines/comments not included)