Part 1

# Conditional statements and conditional operation

## Learning Objectives

- Become familiar with the idea of a conditional statement and know how to create a program containing optional operations through the use of conditional statements.
- Become familiar with comparison and logical operators commonly used in conditional statements.
- Know how to compare both numbers and strings, remembering the equals-command for strings.
- Become familiar with the order of execution for a conditional statement, and know that the parsing of a conditional statement stops at the first condition whose statement evaluates to true.

Our programs have so far been linear. In other words, the programs have executed from top to bottom without major surprises or conditional behavior. However, we usually want to add conditional logic to our programs. By this we mean functionality that's in one way or another dependent on the state of the program's variables.

To branch the execution of a program based on user input, for example, we need to use something known as a **conditional statement**. The simplest conditional statement looks something like this.

```
System.out.println("Hello, world!");
if (true) {
```

```
    System.out.println("This code is unavoidable!");
}
```

Sample output

```
Hello, world!
This code is unavoidable!
```

A conditional statement begins with the keyword `if` followed by parentheses. An expression is placed inside the parentheses, which is evaluated when the conditional statement is reached. The result of the evaluation is a boolean value. No evaluation occurred above. Instead, a boolean value was explicitly used in the conditional statement.

The parentheses are followed by a block, which is defined inside opening-`{` and closing `}` curly brackets. The source code inside the block is executed if the expression inside the parentheses evaluates to *true*.

Let's look at an example where we compare numbers in the conditional statement.

```
int number = 11;
if (number > 10) {
    System.out.println("The number was greater than 10");
}
```

If the expression in the conditional statement evaluates to true, the execution of the program progresses to the block defined by the conditional statement. In the example above, the conditional is "if the number in the variable is greater than 10". On the other hand, if the expression evaluates to false, the execution moves on to the statement after the closing curly bracket of the current conditional statement.

NB! An `if` -statement is not followed by a semicolon since the statement doesn't end after the conditional.

Programming exercise:

## Speeding Ticket

Points
1/1

Write a program that asks the user for an integer and prints the string "Speeding ticket!" if the input is greater than 120.

Sample output

```
Give speed:
15
```

Sample output

```
Give speed:
135
Speeding ticket!
```

Exercise submission instructions

How to see the solution

# Code Indentation and Block Statements

A code block refers to a section enclosed by a pair of curly brackets. The source file containing the program includes the string `public class`, which is followed by the name of the program and the opening curly bracket of the block. The block ends in a closing curly bracket. In the picture below, the program block is highlighted.

```
public class Program {

    public static void main(String[] args) {

        // Source code can be written here. You can run
        // the program by choosing Run->Run File in
        // the menu or by pressing Shift+F6
    }
}
```

The recurring snippet `public static void main(String[] args)` in the programs begins a block, and the source code within it is executed when the program is run — this snippet is, in fact, the starting point of all programs. We actually have two blocks in the example above, as can be seen from the image below.

```
public class Program {

    public static void main(String[] args) {

        // Source code can be written here. You can run
        // the program by choosing Run->Run File in
        // the menu or by pressing Shift+F6
    }
}
```

Blocks define a program's structure and its bounds. A curly bracket must always have a matching pair: any code that's missing a closing (or opening) curly bracket is erroneous.

A conditional statement also marks the start of a new code block.

In addition to the defining program structure and functionality, block statements also have an effect on the readability of a program. Code living inside a block is indented. For example, any source code inside the block of a conditional statement is indented four spaces deeper than the `if` command that started the conditional statement. Four spaces can also be added by pressing the tab key (the key to the left of 'q'). When the block ends, i.e., when we encounter a `}` character, the indentation also ends. The `}` character is at the same level of indentation as the `if`-command that started the conditional statement.

The example below is incorrectly indented.

```
if (number > 10) {
number = 9;
}
```

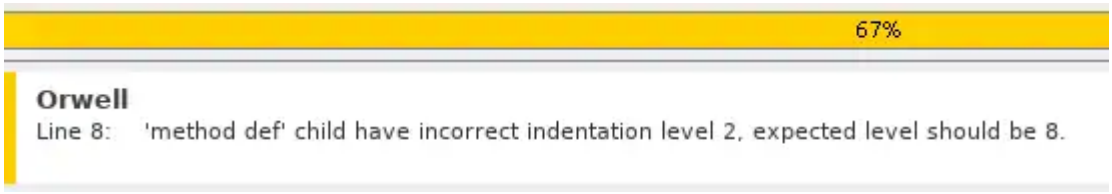
The example below is correctly indented.

```
if (number > 10) {
    number = 9;
}
```

## Automatic Code Indentation

Code in Java is indented either by four spaces or a single tab for each block. Use either spaces or tabs for indentation, not both. The indentation might break in some cases if you use both at the same time. NetBeans will help you with this if you hit the "alt + shift + f" (macOS "control + shift + f") key combination.

From now on the our program code needs to be indented correctly in the exercises as well. If the indentation is incorrect, the development environment will not accept the solution. You will see indentation errors highlighted in yellow in the test results.




The above error message says that there should have been 8 spaces at the beginning of line 8 - there were only 2. In this case, we can fix the indentation by adding 6 more spaces to the beginning of line 8.

Programming exercise:

# Check Your Indentation

Points

1/1

The exercise template contains a program demonstrating the use of conditional statements. It is, however, incorrectly indented.

Try to run the tests before doing anything. TMC shows the indentation errors differently compared to errors in program logic.

When you notice how indentation errors are shown, correct them. Now would be a good time to give the automatic code formatting a try.

Exercise submission instructions

How to see the solution

# Comparison Operators

- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- == equal to
- != not equal to

```
int number = 55;

if (number != 0) {
    System.out.println("The number is not equal to 0");
}

if (number >= 1000) {
    System.out.println("The number is at least 1000");
}
```

Sample output

```
The number was not equal to 0
```

Programming exercise:

## Orwell

Points
1/1

Write a program that prompts the user for an integer and prints the string "Orwell" if the number is exactly 1984.

Sample output

```
Give a number:
1983
```

Sample output

```
Give a number:
1984
Orwell
```

Exercise submission instructions

How to see the solution

Programming exercise:

## Ancient

Points
1/1

Write a program that prompts the user for a year. If the user inputs a number that is smaller than 2015, then the program prints the string "Ancient history!".

Sample output

```
Give a year:
2017
```

Sample output

Give a year:
2013
Ancient history!

Exercise submission instructions

How to see the solution

# Else

If the expression inside the parentheses of the conditional statement evaluates to false, then the execution of the code moves to the statement following the closing curly bracket of the current conditional statement. This is not always desired, and usually we want to create an alternative option for when the conditional expression evaluates to false.

This can be done with the help of the `else` command, which is used together with the `if` command.

```
int number = 4;

if (number > 5) {
    System.out.println("Your number is greater than five!");
} else {
    System.out.println("Your number is five or less!");
}
```

Sample output

```
Your number is five or less!
```

If an `else` branch has been specified for a conditional statement, the block defined by the else branch is run in the case that the condition of the conditional statement is false. The `else`-command is placed on the same line as the closing bracket of the block defined by the `if`-command.

Programming exercise:

## Positivity

Points
1/1

Write a program that prompts the user for an integer and informs the user whether or not it is positive (greater than zero).

Sample output

```
Give a number:
5
The number is positive.
```

Sample output

```
Give a number:
-2
The number is not positive.
```

Exercise submission instructions

How to see the solution

Programming exercise:

## Adulthood

Points
1/1

Write a program that prompts the user for their age and tells them whether or not they are an adult (18 years old or older).

Sample output

```
How old are you?
12
You are not an adult
```

Sample output

```
How old are you?
32
You are an adult
```

Exercise submission instructions

How to see the solution

# More Conditionals: else if

In the case of multiple conditionals, we use the `else if`-command. The command `else if` is like `else`, but with an additional condition. `else if` follows the `if`-condition, and they may be multiple.

```
int number = 3;

if (number == 1) {
    System.out.println("The number is one");
} else if (number == 2) {
    System.out.println("The given number is two");
} else if (number == 3) {
    System.out.println("The number must be three!");
} else {
    System.out.println("Something else!");
}
```

Sample output

```
The number must be three!
```

Let's read out the example above: 'If the number is one, then print "The number is one", else if the number is two, then print "The given number is two", else if the number is three, then print "The number must be three!". Otherwise, print "Something else!"'

The step-by-step visualization of the code above:

```java
public class Example {
  public static void main(String[] args) {
    int number = 3;

    if (number == 1) {
      System.out.println("The number is one");
    } else if (number == 2) {
      System.out.println("The given number is two");
    } else if (number == 3) {
      System.out.println("The number must be three!");
    } else {
      System.out.println("Something else!");
    }
  }
}
```

main:3

| Name | Value |
|---|---|

## Output

Prev 1 / 7 Next

Programming exercise:

# Larger Than or Equal To

Points 1/1

Write a program that prompts the user for two integers and prints the larger of the two. If the numbers are the same, then the program informs us about this as well.

Sample outputs:

Sample output

```
Give the first number:
5
```

Give the second number:
3
Greater number is: 5

Sample output

Give the first number:
5
Give the second number:
8
Greater number is: 8

Sample output

Give the first number:
5
Give the second number:
5
The numbers are equal!

Exercise submission instructions

How to see the solution

# Order of Execution for Comparisons

The comparisons are executed top down. When execution reaches a conditional statement whose condition is true, its block is executed and the comparison stops.

```
int number = 5;

if (number == 0) {
    System.out.println("The number is zero.");
} else if (number > 0) {
    System.out.println("The number is greater than zero.");
```

```
} else if (number > 2) {
    System.out.println("The number is greater than two.");
} else {
    System.out.println("The number is less than zero.");
}
```

Sample output

```
The number is greater than zero.
```

The example above prints the string "The number is greater than zero." even if the condition `number > 2` is true. The comparison stops at the first condition that evaluates to true.

Programming exercise:

## Grades and Points

Points 1/1

The table below describes how the grade for a particular course is determined. Write a program that gives a course grade according to the provided table.

| points | grade |
|---|---|
| < 0 | impossible! |
| 0-49 | failed |
| 50-59 | 1 |
| 60-69 | 2 |
| 70-79 | 3 |
| 80-89 | 4 |
| 90-100 | 5 |

| points | grade |
| --- | --- |
| > 100 | incredible! |

Sample outputs:

Sample output

```
Give points [0-100]:
37
Grade: failed
```

Sample output

```
Give points [0-100]:
76
Grade: 3
```

Sample output

```
Give points [0-100]:
95
Grade: 5
```

Sample output

```
Give points [0-100]:
-3
Grade: impossible!
```

Exercise submission instructions

How to see the solution

# Conditional Statement Expression and the Boolean Variable

The value that goes between the parentheses of the conditional statement should be of type boolean after the evaluation. `boolean` type variables are either *true* or *false*.

```
boolean isItTrue = true;
System.out.println("The value of the boolean variable is " + isItTrue);
```

Sample output

```
The value of the boolean variable is true
```

The conditional statement can also be done as follows:

```
boolean isItTrue = true;
if (isItTrue) {
    System.out.println("Pretty wild!");
}
```

Sample output

```
Pretty wild!
```

Comparison operators can also be used outside of conditionals. In those cases, the boolean value resulting from the comparison is stored in a boolean variable for later use.

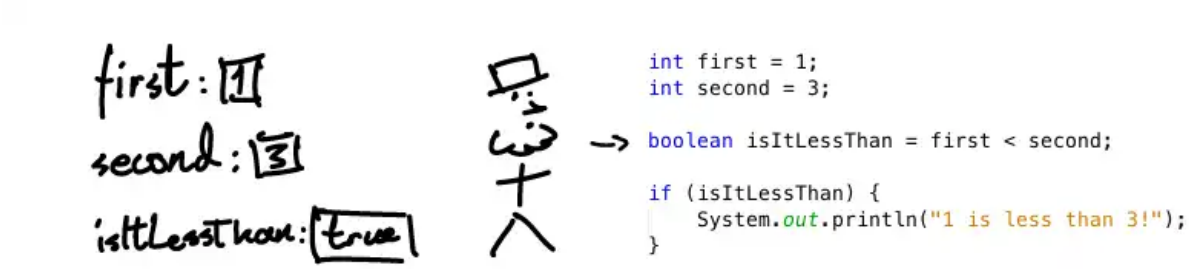
```
int first = 1;
int second = 3;
boolean isGreater = first > second;
```

In the example above, the boolean variable `isGreater` now contains the boolean value *false*. We can extend the previous example by adding a conditional statement to it.

```
int first = 1;
int second = 3;
boolean isLessThan = first < second;

if (isLessThan) {
```

```
    System.out.println("1 is less than 3!");
}
```




The code in the image above has been executed to the point where the program's variables have been created and assigned values. The variable `isLessThan` has `true` as its value. Next in the execution is the comparison `if (isLessThan)` — the value for the variable `isLessThan` is found in its container, and the program finally prints:

Sample output

```
1 is less than 3!
```

## Remainder

The modulo operator is a slightly less-used operator, which is, however, very handy when we want to check the divisibility of a number, for example. The symbol for the modulo operator is `%`.

```
int remainder = 7 % 2;
System.out.println(remainder); // prints 1
System.out.println(5 % 3); // prints 2
System.out.println(7 % 4); // prints 3
System.out.println(8 % 4); // prints 0
System.out.println(1 % 2); // prints 1
```

If we want to know whether the number given by the user is divisible by four hundred, we check if the remainder is zero after taking the modulo of the number and four hundred.

```
Scanner reader = new Scanner(System.in);
```

```
int number = Integer.valueOf(reader.nextLine());
int remainder = number % 400;

if (remainder == 0) {
    System.out.println("The number " + number + " is divisible by four hund
} else {
    System.out.println("The number " + number + " is not divisible by four
```

Since the modulo is an operation just like other calculations, it can be a part of an expression in a conditional statement.

```
Scanner reader = new Scanner(System.in);

int number = Integer.valueOf(reader.nextLine());

if (number % 400 == 0) {
    System.out.println("The number " + number + " is divisible by four hund
} else {
    System.out.println("The number " + number + " is not divisible by four
}
```

Programming exercise:

## Odd or even

Points 1/1

Write a program that prompts the user for a number and informs us whether it is even or odd.

Sample output

```
Give a number:
2
Number 2 is even.
```

Sample output

Give a number:
7
Number 7 is odd.

Hint: The remainder when dividing by 2 tells us whether the number is even or not. We get the remainder using the `%`-operator. The exercise template contains additional instructions on how to do the checking using the remainder.

Exercise submission instructions

How to see the solution

# Conditional Statements and Comparing Strings

Even though we can compare integers, floating point numbers, and boolean values using two equals signs (`variable1 == variable2`), we cannot compare the equality of strings using two equals signs.

You can try this with the following program:

```
Scanner reader = new Scanner(System.in);

System.out.println("Enter the first string");
String first = reader.nextLine();
System.out.println("Enter the second string");
String second = reader.nextLine();

if (first == second) {
    System.out.println("The strings were the same!");
} else {
    System.out.println("The strings were different!");
}
```

Sample output

```
Enter the first string
same
Enter the second string
same
The strings were different!
```

Sample output

```
Enter the first string
same
Enter the second string
different
The strings were different!
```

This has to do with the internal workings of strings as well as how variable comparison is implemented in Java. In practice, the comparison is affected by how much information a variable can hold — strings can hold a limitless amount of characters, whereas integers, floating-point numbers, and boolean values always contain a single number or value only. Variables that always contain only one number or value can be compared using an equals sign, whereas this doesn't work for variables containing more information. We will return to this topic later in this course.

When comparing strings we use the `equals`-command, which is related to string variables. The command works in the following way:

```
Scanner reader = new Scanner(System.in);

System.out.println("Enter a string");
String input = reader.nextLine();

if (input.equals("a string")) {
    System.out.println("Great! You read the instructions correctly.");
} else {
    System.out.println("Missed the mark!");
}
```

Sample output

```
Enter a string
ok!
Missed the mark!
```

Sample output

```
Enter a string
a string
Great! You read the instructions correctly.
```

The equals command is written after a string by attaching it to the string to be compared with a dot. The command is given a parameter, which is the string that the variable will be compared against. If the string variable is being directly compared with a string, then the string can be placed inside the parentheses of the equals-command within quotation marks. Otherwise, the name of the string variable that holds the string to be compared is placed inside the parentheses.

In the example below the user is prompted for two strings. We first check to see if the provided strings are the same, after which we'll check if the value of either one of the two strings is "two strings".

```
Scanner reader = new Scanner(System.in);

System.out.println("Input two strings");
String first = reader.nextLine();
String second = reader.nextLine();

if (first.equals(second)) {
    System.out.println("The strings were the same!");
} else {
    System.out.println("The strings were different!");
}

if (first.equals("two strings")) {
    System.out.println("Clever!");
}

if (second.equals("two strings")) {
    System.out.println("Sneaky!");
}
```

Sample output

```
Input two strings
hello
world
The strings were different!
```

Sample output

```
Input two strings
two strings
world
The strings were different!
Clever!
```

Sample output

```
Input two strings
same
same
The strings were the same!
```

Programming exercise:

## Password

Points 1/1

Write a program that prompts the user for a password. If the password is "Caput Draconis" the program prints "Welcome!". Otherwise, the program prints "Off with you!"

Sample output

```
Password?
Wattlebird
Off with you!
```

Sample output

Password?
Caput Draconis
Welcome!

Exercise submission instructions

How to see the solution

Programming exercise:

## Same

Points
1/1

Write a program that prompts the user for two strings. If the strings are the same, then the program prints "Same". Otherwise, it prints "Different".

Sample output

Enter the first string:
hello
Enter the second string:
hello
Same

Sample output

Enter the first string:
hello
Enter the second string:
world
Different

Exercise submission instructions

How to see the solution

# Logical Operators

The expression of a conditional statement may consist of multiple parts, in which the logical operators **and** `&&`, **or** `||`, and **not** `!` are used.

- An expression consisting of two expressions combined using the and-operator is true, if and only if both of the combined expressions evaluate to true.
- An expression consisting of two expressions combined using the or-operator is true if either one, or both, of the combined expressions evaluate to true.
- Logical operators are not used for changing the boolean value from true to false, or false to true.

In the next example we combine two individual conditions using `&&`, i.e., the and-operator. The code is used to check if the number in the variable is greater than or equal to 5 and less than or equal to 10. In other words, whether it's within the range of 5-10:

```
System.out.println("Is the number within the range 5-10: ");
int number = 7;

if (number >= 5 && number <= 10) {
    System.out.println("It is! :)");
} else {
    System.out.println("It is not :(")
}
```

Sample output

```
Is the number within the range 5-10:
It is! :)
```

In the next one we provide two conditions using `||`, i.e., the or-operator: is the number less than zero or greater than 100. The condition is fulfilled if the number fulfills either one of the two conditions:

```java
System.out.println("Is the number less than 0 or greater than 100");
int number = 145;

if (number < 0 || number > 100) {
    System.out.println("It is! :)");
} else {
    System.out.println("It is not :(")
}
```

Sample output

```
Is the number less than 0 or greater than 100
It is! :)
```

In this example we flip the result of the expression `number > 4` using `!`, i.e., the not-operator. The not-operator is written in such a way that the expression to be flipped is wrapped in parentheses, and the not-operator is placed before the parentheses.

```java
int number = 7;

if (!(number > 4)) {
    System.out.println("The number is not greater than 4.");
} else {
    System.out.println("The number is greater than 4.")
}
```

Sample output

```
The number is greater than 4.
```

Below is a table showing the operation of expressions containing logical operators.

| number | number > 0 | number < 10 | number > 0 && number < 10 | !(number > 0 && number < 10) | number > 0 \|\| number < 10 |
|---|---|---|---|---|---|
| -1 | false | true | false | true | true |

| number | number > 0 | number < 10 | number > 0 && number < 10 | !(number > 0 && number < 10) | number > 0 \|\| number < 10 |
|---|---|---|---|---|---|
| 0 | false | true | false | true | true |
| 1 | true | true | true | false | true |
| 9 | true | true | true | false | true |
| 10 | true | false | false | true | true |

Programming exercise:

## Checking the age

Points 1/1

Write a program that prompts the user to input their age and checks whether or not it is possible (at least 0 and at most 120). Only use a single `if`-command in your program.

Sample output

```
How old are you? 10
OK
```

Sample output

```
How old are you? 55
OK
```

Sample output

```
How old are you? -3
Impossible!
```

Sample output

```
How old are you? 150
Impossible!
```

Exercise submission instructions

How to see the solution

# Execution Order of Conditional Statements

Let's familiarize ourselves with the execution order of conditional statements through a classic programming exercise.

*'Write a program that prompts the user for a number between one and one hundred, and prints that number. If the number is divisible by three, then print "Fizz" instead of the number. If the number is divisible by five, then print "Buzz" instead of the number. If the number is divisible by both three and five, then print "FizzBuzz" instead of the number.'*

The programmer begins solving the exercise by reading the exercise description and by writing code according to the description. The conditions for execution are presented in a given order by the description, and the initial structure for the program is formed based on that order. The structure is formed based on the following steps:

- Write a program that prompts the user for a number and prints that number.
- If the number is divisible by three, then print "Fizz" instead of the number.
- If the number is divisible by five, then print "Buzz" instead of the number.
- If the number is divisible by both three and five, then print "FizzBuzz" instead of the number.

If-type conditions are easy to implement using `if - else if - else` - conditional statements. The code below was written based on the steps above, but it does not work correctly, which we can see from the example.

```
Scanner reader = new Scanner(System.in);

int number = Integer.valueOf(reader.nextLine());

if (number % 3 == 0) {
    System.out.println("Fizz");
} else if (number % 5 == 0) {
    System.out.println("Buzz");
} else if (number % 3 == 0 && number % 5 == 0) {
    System.out.println("FizzBuzz");
} else {
    System.out.println(number);
}
```

Sample output

```
3
Fizz
```

Sample output

```
4
4
```

Sample output

```
5
Buzz
```

Sample output

```
15
Fizz
```

The problem with the previous approach is that **the parsing of conditional statements stops at the first condition that is true**. E.g., with the value 15 the string "Fizz" is printed, since the number is divisible by three (15 % 3 == 0).

One approach for developing this train of thought would be to first find the **most demanding condition**, and implement it. After that, we would

implement the other conditions. In the example above, the condition "if the number is divisible by both three **and** five" requires two things to happen. Now the train of thought would be:

1. Write a program that reads input from the user.
2. If the number is divisible by both three and five, then print "FizzBuzz" instead of the number.
3. If the number is divisible by three, then print "Fizz" instead of the number.
4. If the number is divisible by five, then print "Buzz" instead of the number.
5. Otherwise the program prints the number given by the user.

Now the problem seems to get solved.

```
Scanner reader = new Scanner(System.in);

int number = Integer.valueOf(reader.nextLine());

if (number % 3 == 0 && number % 5 == 0) {
    System.out.println("FizzBuzz");
} else if (number % 3 == 0) {
    System.out.println("Fizz");
} else if (number % 5 == 0) {
    System.out.println("Buzz");
} else {
    System.out.println(number);
}
```

Sample output

```
2
2
```

Sample output

```
5
Buzz
```

Sample output

```
30
FizzBuzz
```

Programming exercise:

# Leap year

Points
1/1

A year is a leap year if it is divisible by 4. However, if the year is divisible by 100, then it is a leap year only when it is also divisible by 400.

Write a program that reads a year from the user, and checks whether or not it is a leap year.

Sample output

Give a year: 2011
The year is not a leap year.

Sample output

Give a year: 2012
The year is a leap year.

Sample output

Give a year: 1800
The year is not a leap year.

Sample output

Give a year: 2000
The year is a leap year.

Hint 1: The divisibility by a particular number can be checked using the modulo operator, aka `%`, in the following way.

```
int number = 5;

if (number % 5 == 0) {
    System.out.println("The number is divisible by five!");
}

if (number % 6 != 0) {
```

```
    System.out.println("The number is not divisible by six!")
}
```

Sample output

```
The number is divisible by five!
The number is not divisible by six!
```

Hint 2: Think of the problem as a chain of if, else if, else if, ... comparisons, and start building the program from a situation in which you can be certain that the year is not a leap year.

```
Scanner reader = new Scanner(System.in);
int number = Integer.valueOf(reader.nextLine());

if (number % 4 != 0) {
    System.out.println("The year is not a leap year.");
} else if (...) {
    ...
} ...
```

Exercise submission instructions

How to see the solution

Programming exercise:

## Gift tax

Points

1/1

https://www.vero.fi/en/individuals/property/gifts/: *A gift is a transfer of property to another person against no compensation or payment. If the total value of the gifts you receive from the same donor in the course of 3 years is €5,000 or more, you must pay gift tax.*

When a gift is given by a close relative or a family member, the amount of gift tax is determined by the following table (source vero.fi):

| Value of gift | Tax at the lower limit | Tax rate(%) for exceeding part |
|---|---|---|
| 5 000 — 25 000 | 100 | 8 |
| 25 000 — 55 000 | 1 700 | 10 |
| 55 000 — 200 000 | 4 700 | 12 |
| 200 000 — 1 000 000 | 22 100 | 15 |
| 1 000 000 — | 142 100 | 17 |

For example 6000€ gift implies 180€ of gift tax (100 + (6000-5000) * 0.08), and 75000€ gift implies 7100€ of gift tax (4700 + (75000-55000) * 0.12).

Write a program that calculates the gift tax for a gift from a close relative or a family member. This is how the program should work:

Sample output

```
Value of the gift?
3500
No tax!
```

Sample output

```
Value of the gift?
5000
Tax: 100.0
```

Sample output

```
Value of the gift?
27500
Tax: 1950.0
```

Exercise submission instructions

How to see the solution

You have reached the end of this section! Continue to the next section:

→ 7. Programming in our society

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI