**Exercises: Data Types and Methods - Exercises**

Problems for exercises and homework for the [“Programming Fundamentals” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

You can check your solutions here: <https://judge.softuni.bg/Contests/206/Data-Types-and-Methods-Exercises>.

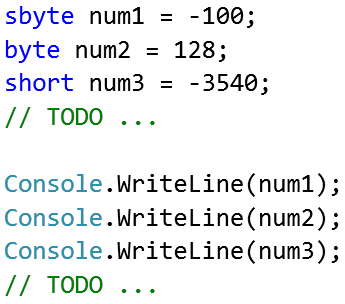
* **Practice Integer Numbers**

Create a new C# project and create a program that **assigns integer values** to **variables**. Be sure that each **value** is stored in the **correct variable type** (try to find the most suitable variable type in order to save memory). Finally, you need to **print** all variables to the console.

|  |  |
| --- | --- |
| **Input** | **Output** |
| -100  128  -3540  64876  2147483648  -1141583228  -1223372036854775808 | -100  128  -3540  64876  2147483648  -1141583228  -1223372036854775808 |

**Hints**

Follow the idea in the code below:



* **Practice Floating Point Numbers**

Create a new C# project and create a program that **assigns floating point values** to **variables**. Be sure that each **value** is stored in the **correct variable type** (try to find the most suitable variable type in order to save memory). Finally, you need to **print** all variables to the console.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3.141592653589793238  1.60217657  7.8184261974584555216535342341 | 3.141592653589793238  1.60217657  7.8184261974584555216535342341 |

**Hints**

Just like at the previous problem, declare several variables of appropriate **floating-point data type**, assign the above listed values and **print** them.

* **Practice Characters and Strings**

Create a new C# project and create a program that **assigns character** and **string values** to **variables**. Be sure that each **value** is stored in the **correct variable**. Finally, you need to **print** all variables to the console.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Software University  B  y  e  I love programming | Software University  B  y  e  I love programming |

**Hints**

Like at the previous problem, declare variables of type **char** or **sting**, assign the above values and **print** them.

* **Variable in Hexadecimal Format**

Write a program that reads a number in **hexadecimal format** (**0x##**) convert it to **decimal format** and prints it.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0xFE | 254 |
| 0x37 | 55 |
| 0x10 | 16 |

**Hints**

* Use [**Convert.ToInt32(HYPERLINK "https://msdn.microsoft.com/en-us/library/1k20k614(v=vs.110).aspx"stringHYPERLINK "https://msdn.microsoft.com/en-us/library/1k20k614(v=vs.110).aspx", 16HYPERLINK "https://msdn.microsoft.com/en-us/library/1k20k614(v=vs.110).aspx")**](https://msdn.microsoft.com/en-us/library/1k20k614(v=vs.110).aspx).
* **Boolean Variable**

Write a program that reads a **string**, converts it to **Boolean** variable and **prints** “**Yes**”if the variable is **true** and “**No**” if the variable is **false**.

|  |  |
| --- | --- |
| **Input** | **Output** |
| True | Yes |
| False | No |

**Hints**

* Use [**Convert.ToBoolean(HYPERLINK "https://msdn.microsoft.com/en-us/library/86hw82a3(v=vs.110).aspx"stringHYPERLINK "https://msdn.microsoft.com/en-us/library/86hw82a3(v=vs.110).aspx")**](https://msdn.microsoft.com/en-us/library/86hw82a3(v=vs.110).aspx).
* **Strings and Objects**

Declare two **string variables** and assign them with “**Hello**” and “**World**”. Declare an **object variable** and assign it with the **concatenation** of the first two variables (mind adding an interval between). Declare a third **string** variable and initialize it with the value of the object variable (you should perform type **casting**).

|  |  |
| --- | --- |
| **Input** | **Output** |
| Hello  World | Hello World |

* **Exchange Variable Values**

Declare two integer variables **a** and **b** and assign them with 5 and 10 and after that **exchange their values** by using some programming logic. Print the variable values before and after the exchange, as shown below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  10 | Before:  a = 5  b = 10  After:  a = 10  b = 5 |

**Hints**

You may use a **temporary variable** to remember the old value of **a**, then assign the value of **b** to **a**, then assign the value of the temporary variable to **b**.

* **Employee Data**

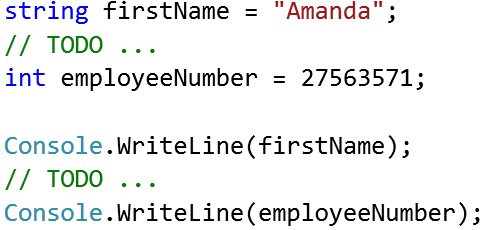
A marketing company wants to keep record of its employees. Each record would have the following characteristics:

* First name
* Last name
* Age (0...100)
* Gender (**m** or **f**)
* Personal ID number (e.g. 8306112507)
* Unique employee number (27560000…27569999)

Declare the **variables** needed to keep the information for a single employee using appropriate primitive data types. Use descriptive names. **Print** the data at the console.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Amanda  Jonson  27  f  8306112507  27563571 | First name: Amanda  Last name: Jonson  Age: 27  Gender: f  Personal ID: 8306112507  Unique Employee number: 27563571 |

**Hints**



* **Reverse Characters**

Write a program to ask the user for **3 letters** and print them in **reversed order**.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| A  B  C | CBA |  | x  Y  z | zYx |  | G  g  n | ngG |

* **Centuries to Nanoseconds**

Write program to enter an integer number of **centuries** and convert it to **years, days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | 1 centuries = 100 years = 36524 days = 876576 hours = 52594560 minutes = 3155673600 seconds = 3155673600000 milliseconds = 3155673600000000 microseconds = 3155673600000000000 nanoseconds |
| 5 | 5 centuries = 500 years = 182621 days = 4382904 hours = 262974240 minutes = 15778454400 seconds = 15778454400000 milliseconds = 15778454400000000 microseconds = 15778454400000000000 nanoseconds |

**Hints**

* Use an appropriate data type for every data conversion. Beware of **overflows**!
* **Convert Speed Units**

Create a program to ask the user for a **distance (in meters)** and the time taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour.

Assume 1 mile = 1609 meters.

**Input**

* On first line you receive – **distance in meters**
* On second – **hours**
* On third – **minutes**
* On fourth – **seconds**

**Output**

Every number in the output should be precise up to 6 digits after the floating point

* On first line – speed in **meters per second** (m/s)
* On second line – speed in **kilometers per hour** (km/h)
* On third line – speed in **miles per hour** (mph)

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1000  1  1  0 | 0.2732241  0.9836066  0.6113155 |  | 10000  0  20  30 | 8.130081  29.26829  18.19036 |  | 200000  2  5  0 | 26.66667  96  59.66439 |

**Hints**

* Search in internet how to convert units.
* The type **float** is big enough for the calculations.
* **Rectangle Properties**

Create a program to calculate rectangle’s **perimeter**, **area** and **diagonal** by given its **width** and **height**.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 10  5 | 30  50  11.1803398874989 |  | 22.1  10.2 | 64.6  225.42  24.3402958075698 |

**Hints**

* Use **Math.Sqrt()** to calculate square root for calculating the diagonal (c2 = a2 + b2). See <http://www.mathopenref.com/rectanglediagonals.html>.
* **Vowel or Digit**

Create a program to check if given symbol is **digit**, **vowel** or any **other symbol**.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| a | vowel |  | 9 | digit |  | g | other |

* **Integer to Hex and Binary**

Create a program to convert a **decimal number** to **hexadecimal** and **binary** number and print it.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 10 | A  1010 |  | 420 | 1A4  110100100 |  | 256 | 100  100000000 |

**Hints**

* Use [**Convert.ToString(number,HYPERLINK "https://msdn.microsoft.com/en-us/library/14kwkz77(v=vs.110).aspx" HYPERLINK "https://msdn.microsoft.com/en-us/library/14kwkz77(v=vs.110).aspx"base)**](https://msdn.microsoft.com/en-us/library/14kwkz77(v=vs.110).aspx)and [**string.HYPERLINK "https://msdn.microsoft.com/en-us/library/ewdd6aed(v=vs.110).aspx"ToUpper()**](https://msdn.microsoft.com/en-us/library/ewdd6aed(v=vs.110).aspx).
* **\* Comparing Floats**

Write a program that **safely compares floating-point numbers** (**double**) with precision **eps** = **0.000001**. Note that we cannot directly compare two floating-point numbers **a** and **b** by **a==b** because of the nature of the floating-point arithmetic. Therefore, we assume two numbers are equal if they are more closely to each other than a fixed constant **eps**. Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Number a** | **Number b** | **Equal (with precision eps=0.000001)** | **Explanation** |
| 5.3 | 6.01 | false | The difference of 0.71 is too big (> eps) |
| 5.00000001 | 5.00000003 | true | The difference 0.00000002 < eps |
| 5.00000005 | 5.00000001 | true | The difference 0.00000004 < eps |
| -0.0000007 | 0.00000007 | true | The difference 0.00000077 < eps |
| -4.999999 | -4.999998 | false | Border case. The difference 0.000001 == eps. We consider the numbers are different. |
| 4.999999 | 4.999998 | false | Border case. The difference 0.000001 == eps. We consider the numbers are different. |

* **Print Part of the ASCII Table**

Find online more information about [ASCII](http://www.ascii-code.com/) (American Standard Code for Information Interchange) and write a program to **prints part of the ASCII table** of characters at the console. On the first line of input you will receive **the char index you should start with** and on the **second line - the index of the last character** you should print.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 60  65 | < = > ? @ A |
| 69  79 | E F G H I J K L M N O |
| 97  104 | a b c d e f g h |
| 40  55 | ( ) \* + , - . / 0 1 2 3 4 5 6 7 |

* **\* Different Integers Size**

Given an input integer, you must determine which primitive data types are capable of properly storing that input.

**Input**

* You receive **N** – integer which can be arbitrarily large or small

**Output**

You must determine if the given primitives are capable of storing it. If yes, then print:

|  |
| --- |
| {N} can fit in:  \* dataType |

If there is more than one appropriate data type, print each one on its own line and order them by size  
(**sbyte < byte < short < ushort < int < uint < long**).

If the number cannot be stored in one of the four aforementioned primitives, print the line:

|  |
| --- |
| {N} can't fit in any type |

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| -150 | -150 can fit in:  \* short  \* int  \* long | 150000 | 150000 can fit in:  \* int  \* uint  \* long |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1500000000 | 1500000000 can fit in:  \* int  \* uint  \* long |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 213333333333333333333333333333333333 | 213333333333333333333333333333333333 can't fit in any type |

**Hints**

* Use the **try … catch** construction.
* **Hello, Name!**

Write a **method** that receives a name as **parameter** and prints on the console. “Hello, <name>!”

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter | Hello, Peter! |

* **Max Method**

Create a method **GetMax(int a, int b)**, that returns maximal of the two numbers. Write a program that reads three numbers from the console and prints the biggest of them. Use the **GetMax(…)** method you just created.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1  2  3 | 3 |  | -100  -101  -102 | -100 |

* **English Name оf The Last Digit**

Write a method that returns the English name of the last digit of a given number.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1024 | four |  | 512 | two |

* **Numbers in Reversed Order**

Write a method that prints the digits of a given decimal number in a reversed order.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 256 | 652 |  | 1.12 | 21.1 |

* **Fibonacci Numbers**

Define a method **Fib(n)** that calculates the nth [Fibonacci number](https://en.wikipedia.org/wiki/Fibonacci_number). Examples:

|  |  |
| --- | --- |
| **n** | **Fib(n)** |
| 0 | 1 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 5 |
| 5 | 8 |
| 6 | 13 |
| 11 | 144 |
| 25 | 121393 |

* **Prime Checker**

Write a Boolean method **IsPrime(n)** that check whether a given integer number **n** is [prime](https://en.wikipedia.org/wiki/Prime_number). Examples:

|  |  |
| --- | --- |
| **n** | **IsPrime(n)** |
| 0 | false |
| 1 | false |
| 2 | true |
| 3 | true |
| 4 | false |
| 5 | true |
| 323 | false |
| 337 | true |
| 6737626471 | true |
| 117342557809 | false |

* **Primes in Given Range**

Write a method that calculates **all prime numbers in given range** and returns them as list of integers:

|  |
| --- |
| static List<int> FindPrimesInRange(startNum, endNum)  {  …  } |

Write a method to **print a list of integers**. Write a program that enters two integer numbers (each at a separate line) and prints all primes in their range, separated by a comma.

**Examples**

|  |  |
| --- | --- |
| **Start and End Number** | **Output** |
| 0  10 | 2, 3, 5, 7 |
| 5  11 | 5, 7, 11 |
| 100  200 | 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199 |
| 250  950 | 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947 |
| 100  50 | *(empty list)* |

* **Master Numbers**

A master number is an integer that holds the following properties:

* Is symmetric (palindrome), e.g. 5, 77, 282, 14341, 9553559.
* Its sum of digits is divisible by 7, e.g. 77, 313, 464, 5225, 37173.
* Holds at least one even digit, e.g. 232, 707, 6886, 87578.

Write a program to **print all master numbers** in the range [1…**n**].

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 600 | 232  383  464  545 |  | 5000 | 232  383  464  545  626  696  707  858  1661  2552  3443  4334 |

**Hints**

* Write 3 utility methods:
* **IsPalindrome(int num)**
* **SumOfDigits(int num)**
* **ContainsEvenDigit(int num)**
* Loop through all numbers in range [1…**n**] and check every number with the helper methods.
* **\* Factorial**

Write a program that calculates and prints the **n!** for any **n** in the range [1…1000].

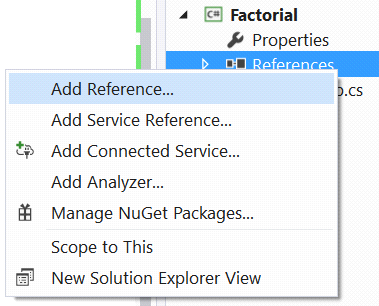
**Examples**

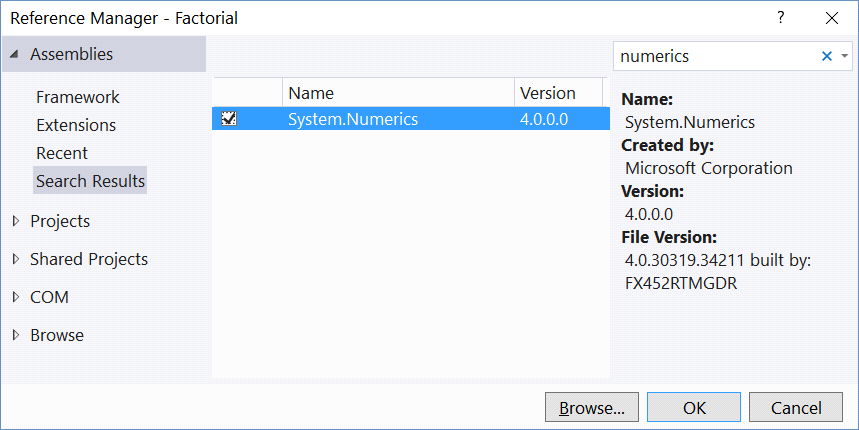
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5 | 120 |  | 100 | 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000 |

**Hints**

Use the class **BigInteger** from the built-in .NET library **System.Numerics.dll**.

* First add reference to **System.Numerics.dll**.





* Import the namespace “**System.Numerics**”:



* Use the type **BigInteger** instead of **long** or **decimal** to keep the factorial value:

