## Csharp

#programmeringsspråk

## Eksamen

Ha en solution for alle deloppgaver, men dele i egne

```
// deloppgave a
{
        // kode
}
```

## Strukturer

Break/Continue

- The `continue` statement breaks one iteration (in the loop).
- The `break` statement breaks the loop.

### While-loops

```
while (condition)
{
  // code block to be executed
}
```

### Do/while-loop

```
do
{
  // code block to be executed
}
while (condition);
```

### For-loops

```
for (statement 1; statement 2; statement 3)
{
```

```
  // code block to be executed
}
```

// **Statement 1** is executed (one time) before the execution of the code block.
// **Statement 2** defines the condition for executing the code block.
// **Statement 3** is executed (every time) after the code block has been executed.

*Eksempel*

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

*Loop through array*

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

*Break/Continue*
//The `continue` statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

## If Else

```
if (condition1)
{
  // block of code to be executed if condition1 is True
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and
condition2 is True
}
else
{
  // block of code to be executed if the condition1 is false and
condition2 is False
}
```

Shorthand

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

### Switch / Case

Kan være vanskelig å bruke for områder.

```
int day = 4;
switch (day)
{
  case 6:
    Console.WriteLine("Today is Saturday.");
    break;
  case 7:
    Console.WriteLine("Today is Sunday.");
    break;
  default:
    Console.WriteLine("Looking forward to the Weekend.");
    break;
}
```

// **break**: used to "jump out" of a `switch` statement.

// Ved å fjerne innholdet i en case (også break) vil tilstanden utføre innholdet i neste case.

### Exceptions, Try Catch

```
try
{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e)
{
  Console.WriteLine(e.Message);
  Console.WriteLine("Something went wrong.");
}
finally
{
  Console.WriteLine("The 'try catch' is finished.");
}
```

## Reference Types

Reference types are objects that exist in external memory space. The reference types in C# are as follows:

- `object`
- `string`
- `dynamic`

## Operatorer

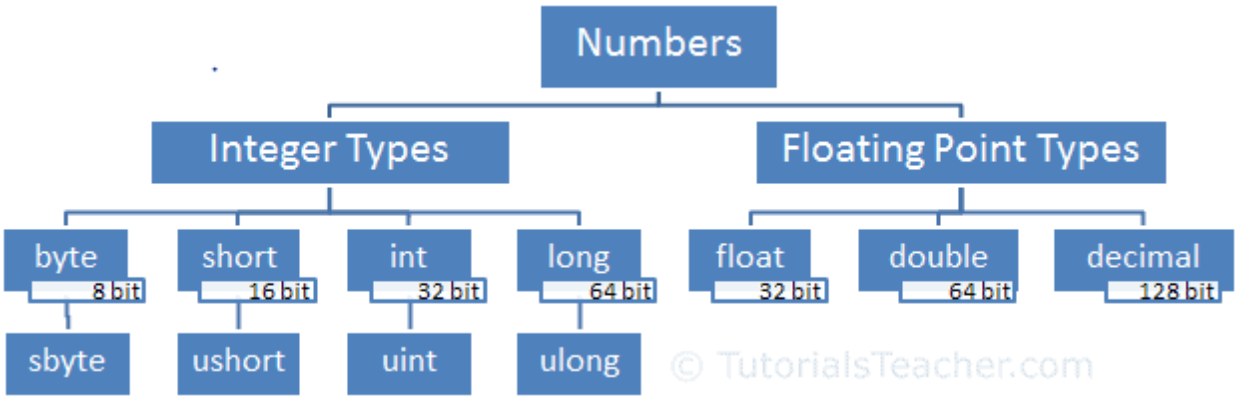
Bilde >

## Operators

| Name | Syntax | Description | Category | Page | Overloadable? | Associativity | Arity |
|---|---|---|---|---|---|---|---|
| Member Access | x.y | Accesses members of an instance, type, namespace, etc. | Primary Operators (These Happen First) | 84 | No | Left-to-Right | 2 |
| Null Conditional Member Access | x?.y | Member access with a null check first | | 284 | No | Left-to-Right | 2 |
| Function Invocation | x(y) | Invoking or calling a method | | 88 | No | Left-to-Right | 2 |
| Aggregate Object Indexing | a[x] | Accessing items in a collection | | 78 | Indexers | Left-to-Right | 2 |
| Null Conditional Indexing | a?[x] | Collection access with a null check first | | 284 | Indexers | Left-to-Right | 2 |
| Postfix Increment | x++ | Adds 1 to a variable (original value returned) | | 58 | Yes | Left-to-Right | 1 |
| Postfix Decrement | x-- | Subtracts 1 from a variable (original value returned) | | 58 | Yes | Left-to-Right | 1 |
| Type Instantiation | new | Creates new objects by invoking a constructor | | 78 | No | Right-to-Left | 1 |
| Typeof | typeof(T) | Produces a Type object from the name of a type | | 280 | No | Right-to-Left | 1 |
| Checked | checked | Switches to a checked context for overflow | | 287 | No | Right-to-Left | 1 |
| Unchecked | unchecked | Switches to an unchecked context for overflow | | 287 | No | Right-to-Left | 1 |
| Default | default(T) | Produces the default value of a given type | | 171 | No | Right-to-Left | 1 |
| Delegate | delegate | Defines delegate types or produces a new empty instance | | 206 | No | Right-to-Left | 1 |
| Sizeof | sizeof(T) | Produces the size of a given type | | 276 | No | Right-to-Left | 1 |
| Pointer Dereference | x->y | Member access through a pointer (unsafe context) | | 266 | No | Left-to-Right | 2 |
| Unary Plus | +x | Produces the same value as the operand | Unary Operators | 45 | Yes | Right-to-Left | 1 |
| Unary Minus/Numeric Negation | -x | Produces the negative of the operand | | 45 | Yes | Right-to-Left | 1 |
| Logical Negation | !x | Produces the Boolean inverse of the operand | | 65 | Yes | Right-to-Left | 1 |
| Bitwise Complement | ~x | Produces the bitwise complement of the operand | | 278 | Yes | Right-to-Left | 1 |
| Prefix Increment | ++x | Adds 1 to a variable (incremented value returned) | | 58 | Yes | Right-to-Left | 1 |
| Prefix Decrement | --x | Subtracts 1 from a variable (decremented value returned) | | 58 | Yes | Right-to-Left | 1 |
| Type Casting | (T)x | Specifies a cast or type conversion | | 55 | User-Defined Conversions | Right-to-Left | 1 |
| Await | await | Awaits asynchronous tasks | | 256 | No | Right-to-Left | 1 |
| Address Of | &x | Produces the address of a variable (unsafe context) | | 266 | No | Right-to-Left | 1 |
| Dereferencing/Indirection | *x | Declare pointer types and dereference pointers | | 266 | No | Right-to-Left | 1 |
| Multiplication | x * y | Multiplies two values | Multiplicative Operators | 43 | Yes | Left-to-Right | 2 |
| Division | x / y | Divides the first value by the second | | 43 | Yes | Left-to-Right | 2 |
| Remainder/Modulus | x % y | Produces the remainder of a division operation | | 44 | Yes | Left-to-Right | 2 |
| Addition | x + y | Adds two values | Additive Operators | 43 | Yes | Left-to-Right | 2 |
| Subtraction | x - y | Subtracts the second value from the first | | 43 | Yes | Left-to-Right | 2 |
| Left Shift | x << y | Bitwise shifts a value a number of bits leftward | Shift Operators | 278 | Yes | Left-to-Right | 2 |
| Right Shift | x >> y | Bitwise shifts a value a number of bits rightward | | 278 | Yes | Left-to-Right | 2 |
| Less Than | x < y | Returns whether x is less than y | Relational and Type Testing Operators | 63 | Yes | Left-to-Right | 2 |
| Greater Than | x > y | Returns whether x is greater than y | | 63 | Yes | Left-to-Right | 2 |
| Less Than Or Equal | x <= y | Returns whether x is less than or equal to y | | 63 | Yes | Left-to-Right | 2 |
| Greater Than Or Equal | x >= y | Returns whether x is greater than or equal to y | | 63 | Yes | Left-to-Right | 2 |
| Is | is | Determines if a value is a certain type or matches a pattern | | 146 | No | Right-to-Left | 1 |
| As | as | Type conversion, returns null where (T)x throws an exception | | 147 | No | Right-to-Left | 1 |
| Equality | x == y | Returns whether x exactly equals y | Equality Operators | 63 | Yes | Left-to-Right | 2 |
| Inequality | x != y | Returns whether x does not equal y | | 63 | Yes | Left-to-Right | 2 |
| Logical AND | x & y | Performs bitwise AND between two values | Logical AND Operator | 278 | Yes | Left-to-Right | 2 |
| Logical XOR | x ^ y | Performs bitwise XOR between two values | Logical XOR Operator | 278 | Yes | Left-to-Right | 2 |
| Logical OR | x \| y | Performs bitwise OR between two values | Logical OR Operator | 278 | Yes | Left-to-Right | 2 |
| Conditional AND | x && y | Returns whether both x and y are true | Conditional AND Operator | 66 | No | Left-to-Right | 2 |
| Conditional OR | x \|\| y | Returns whether either x, y, or both are true | Conditional OR Operator | 66 | No | Left-to-Right | 2 |
| Null Coalescing | x ?? y | Returns x unless it is null, otherwise returns y | Null Coalescing Operator | 283 | No | Right-to-Left | 2 |
| Conditional/Ternary | t ? x : y | If t is true, produces x, otherwise y is produced | Conditional Operator | 67 | No | N/A | 3 |
| Assignment | x = y | Assigns the value of y to the variable x | Assignment and Lambda Operators (These Happen Last) | 46 | No | Right-to-Left | 2 |
| Addition Assignment | x += y | Shorthand for x = x + y | | 47 | Indirectly | Right-to-Left | 2 |
| Subtraction Assignment | x -= y | Shorthand for x = x - y | | 47 | Indirectly | Right-to-Left | 2 |
| Multiplication Assignment | x *= y | Shorthand for x = x * y | | 47 | Indirectly | Right-to-Left | 2 |
| Division Assignment | x /= y | Shorthand for x = x / y | | 47 | Indirectly | Right-to-Left | 2 |
| Remainder Assignment | x %= y | Shorthand for x = x % y | | 47 | Indirectly | Right-to-Left | 2 |
| AND Assignment | x &= y | Shorthand for x = x & y | | 278 | Indirectly | Right-to-Left | 2 |
| OR Assignment | x \|= y | Shorthand for x = x \| y | | 278 | Indirectly | Right-to-Left | 2 |
| XOR Assignment | x ^= y | Shorthand for x = x ^ y | | 278 | Indirectly | Right-to-Left | 2 |
| Left Shift Assignment | x <<= y | Shorthand for x = x << y | | 278 | Indirectly | Right-to-Left | 2 |
| Right Shift Assignment | x >>= y | Shorthand for x = x >> y | | 278 | Indirectly | Right-to-Left | 2 |
| Lambda Declaration | => | Defines a lambda | | 230 | No | Left-to-Right | 2 |

Negering = !

## Datatyper




| Data Type | Size | Description |
|---|---|---|
| `int` | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| `long` | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| `float` | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| `double` | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| `bool` | 1 bit | Stores true or false values |
| `char` | 2 bytes | Stores a single character/letter, surrounded by single quotes |
| `string` | 2 bytes per character | Stores a sequence of characters, surrounded by double quotes |

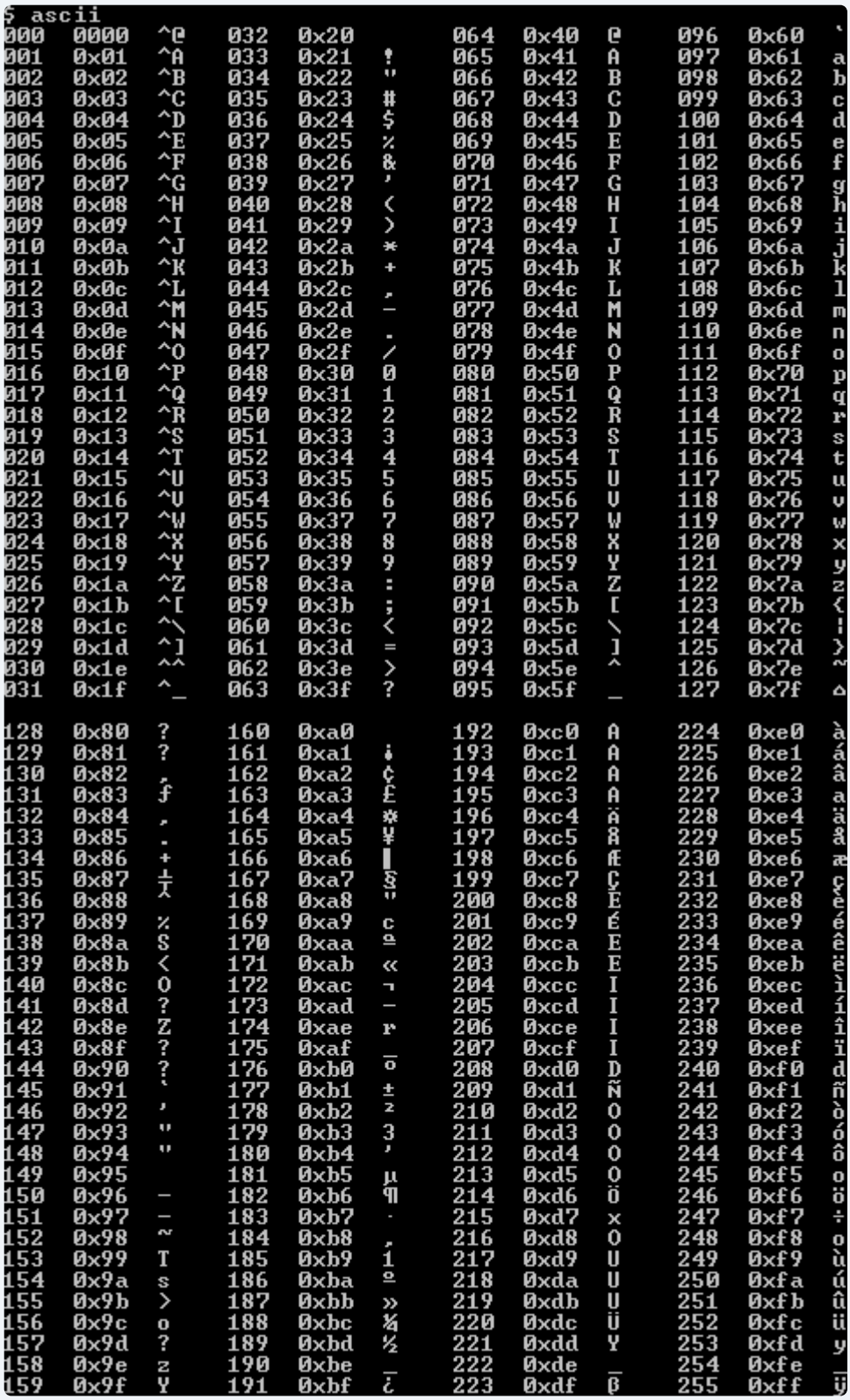
| C# type/keyword | Range | Size |
|---|---|---|
| `sbyte` | -128 to 127 | Signed 8-bit integer |
| `byte` | 0 to 255 | Unsigned 8-bit integer |
| `short` | -32,768 to 32,767 | Signed 16-bit integer |
| `ushort` | 0 to 65,535 | Unsigned 16-bit integer |
| `int` | -2,147,483,648 to 2,147,483,647 | Signed 32-bit integer |
| `uint` | 0 to 4,294,967,295 | Unsigned 32-bit integer |
| `long` | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | Signed 64-bit integer |
| `ulong` | 0 to 18,446,744,073,709,551,615 | Unsigned 64-bit integer |
| `nint` | Depends on platform (computed at runtime) | Signed 32-bit or 64-bit integer |
| `nuint` | Depends on platform (computed at runtime) | Unsigned 32-bit or 64-bit integer |

| Type | Description | Range | Suffix |
|---|---|---|---|
| byte | 8-bit unsigned integer | 0 to 255 | |
| sbyte | 8-bit signed integer | -128 to 127 | |
| short | 16-bit signed integer | -32,768 to 32,767 | |
| ushort | 16-bit unsigned integer | 0 to 65,535 | |

| Type | Description | Range | Suffix |
|---|---|---|---|
| int | 32-bit signed integer | -2,147,483,648<br>to<br>2,147,483,647 | |
| uint | 32-bit unsigned integer | 0 to 4,294,967,295 | u |
| long | 64-bit signed integer | -9,223,372,036,854,775,808<br>to<br>9,223,372,036,854,775,807 | l |
| ulong | 64-bit unsigned integer | 0 to 18,446,744,073,709,551,615 | ul |
| float | 32-bit Single-precision floating point type | -3.402823e38 to 3.402823e38 | f |
| double | 64-bit double-precision floating point type | -1.79769313486232e308 to 1.79769313486232e308 | d |
| decimal | 128-bit decimal type for financial and monetary calculations | (+ or -)1.0 x 10e-28<br>to<br>7.9 x 10e28 | m |
| char | 16-bit single Unicode character | Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode) | |
| bool | 8-bit logical true/false value | True or False | |
| object | Base type of all other types. | | |
| string | A sequence of Unicode characters | | |
| DateTime | Represents date and time | 0:00:00am 1/1/01<br>to<br>11:59:59pm 12/31/9999 | |

```
bool isOpen = true;
byte age = 45;
sbyte temperature = 58;
char grade = 'a';
decimal numberOfAtoms = 1493867940.23m;
double weightOfHippos = 243906.12;
float heightOfGiraffe = 908.32f;
int seaLevel = -24;
uint year = 2023u;
nint pagesInBook = 412;
unint milesToNewYork = 2597;
long circumferenceOfEarth = 25000l;
ulong depthOfOcean = 28000ul;
short tableHeight = 4;
ushort treeBranches = 33;
```

ASCII-tabell >



$ ascii

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 0000 | ^@ | 032 | 0x20 | | 064 | 0x40 | @ | 096 | 0x60 | ` |
| 001 | 0x01 | ^A | 033 | 0x21 | ! | 065 | 0x41 | A | 097 | 0x61 | a |
| 002 | 0x02 | ^B | 034 | 0x22 | " | 066 | 0x42 | B | 098 | 0x62 | b |
| 003 | 0x03 | ^C | 035 | 0x23 | # | 067 | 0x43 | C | 099 | 0x63 | c |
| 004 | 0x04 | ^D | 036 | 0x24 | $ | 068 | 0x44 | D | 100 | 0x64 | d |
| 005 | 0x05 | ^E | 037 | 0x25 | % | 069 | 0x45 | E | 101 | 0x65 | e |
| 006 | 0x06 | ^F | 038 | 0x26 | & | 070 | 0x46 | F | 102 | 0x66 | f |
| 007 | 0x07 | ^G | 039 | 0x27 | ' | 071 | 0x47 | G | 103 | 0x67 | g |
| 008 | 0x08 | ^H | 040 | 0x28 | ( | 072 | 0x48 | H | 104 | 0x68 | h |
| 009 | 0x09 | ^I | 041 | 0x29 | ) | 073 | 0x49 | I | 105 | 0x69 | i |
| 010 | 0x0a | ^J | 042 | 0x2a | * | 074 | 0x4a | J | 106 | 0x6a | j |
| 011 | 0x0b | ^K | 043 | 0x2b | + | 075 | 0x4b | K | 107 | 0x6b | k |
| 012 | 0x0c | ^L | 044 | 0x2c | , | 076 | 0x4c | L | 108 | 0x6c | l |
| 013 | 0x0d | ^M | 045 | 0x2d | - | 077 | 0x4d | M | 109 | 0x6d | m |
| 014 | 0x0e | ^N | 046 | 0x2e | . | 078 | 0x4e | N | 110 | 0x6e | n |
| 015 | 0x0f | ^O | 047 | 0x2f | / | 079 | 0x4f | O | 111 | 0x6f | o |
| 016 | 0x10 | ^P | 048 | 0x30 | 0 | 080 | 0x50 | P | 112 | 0x70 | p |
| 017 | 0x11 | ^Q | 049 | 0x31 | 1 | 081 | 0x51 | Q | 113 | 0x71 | q |
| 018 | 0x12 | ^R | 050 | 0x32 | 2 | 082 | 0x52 | R | 114 | 0x72 | r |
| 019 | 0x13 | ^S | 051 | 0x33 | 3 | 083 | 0x53 | S | 115 | 0x73 | s |
| 020 | 0x14 | ^T | 052 | 0x34 | 4 | 084 | 0x54 | T | 116 | 0x74 | t |
| 021 | 0x15 | ^U | 053 | 0x35 | 5 | 085 | 0x55 | U | 117 | 0x75 | u |
| 022 | 0x16 | ^V | 054 | 0x36 | 6 | 086 | 0x56 | V | 118 | 0x76 | v |
| 023 | 0x17 | ^W | 055 | 0x37 | 7 | 087 | 0x57 | W | 119 | 0x77 | w |
| 024 | 0x18 | ^X | 056 | 0x38 | 8 | 088 | 0x58 | X | 120 | 0x78 | x |
| 025 | 0x19 | ^Y | 057 | 0x39 | 9 | 089 | 0x59 | Y | 121 | 0x79 | y |
| 026 | 0x1a | ^Z | 058 | 0x3a | : | 090 | 0x5a | Z | 122 | 0x7a | z |
| 027 | 0x1b | ^[ | 059 | 0x3b | ; | 091 | 0x5b | [ | 123 | 0x7b | { |
| 028 | 0x1c | ^\ | 060 | 0x3c | < | 092 | 0x5c | \ | 124 | 0x7c | ! |
| 029 | 0x1d | ^] | 061 | 0x3d | = | 093 | 0x5d | ] | 125 | 0x7d | } |
| 030 | 0x1e | ^^ | 062 | 0x3e | > | 094 | 0x5e | ^ | 126 | 0x7e | ~ |
| 031 | 0x1f | ^_ | 063 | 0x3f | ? | 095 | 0x5f | _ | 127 | 0x7f | ⌂ |
| 128 | 0x80 | ? | 160 | 0xa0 | | 192 | 0xc0 | A | 224 | 0xe0 | à |
| 129 | 0x81 | ? | 161 | 0xa1 | ¡ | 193 | 0xc1 | A | 225 | 0xe1 | á |
| 130 | 0x82 | , | 162 | 0xa2 | ¢ | 194 | 0xc2 | A | 226 | 0xe2 | â |
| 131 | 0x83 | ƒ | 163 | 0xa3 | £ | 195 | 0xc3 | A | 227 | 0xe3 | a |
| 132 | 0x84 | „ | 164 | 0xa4 | ¤ | 196 | 0xc4 | Ä | 228 | 0xe4 | ä |
| 133 | 0x85 | . | 165 | 0xa5 | ¥ | 197 | 0xc5 | Å | 229 | 0xe5 | å |
| 134 | 0x86 | + | 166 | 0xa6 | ¦ | 198 | 0xc6 | Æ | 230 | 0xe6 | æ |
| 135 | 0x87 | ‡ | 167 | 0xa7 | § | 199 | 0xc7 | Ç | 231 | 0xe7 | ç |
| 136 | 0x88 | ^ | 168 | 0xa8 | " | 200 | 0xc8 | È | 232 | 0xe8 | è |
| 137 | 0x89 | ‰ | 169 | 0xa9 | c | 201 | 0xc9 | É | 233 | 0xe9 | é |
| 138 | 0x8a | S | 170 | 0xaa | ª | 202 | 0xca | E | 234 | 0xea | ê |
| 139 | 0x8b | < | 171 | 0xab | « | 203 | 0xcb | E | 235 | 0xeb | ë |
| 140 | 0x8c | O | 172 | 0xac | ¬ | 204 | 0xcc | I | 236 | 0xec | ì |
| 141 | 0x8d | ? | 173 | 0xad | - | 205 | 0xcd | I | 237 | 0xed | í |
| 142 | 0x8e | Z | 174 | 0xae | r | 206 | 0xce | I | 238 | 0xee | î |
| 143 | 0x8f | ? | 175 | 0xaf | ¯ | 207 | 0xcf | I | 239 | 0xef | ï |
| 144 | 0x90 | ? | 176 | 0xb0 | ° | 208 | 0xd0 | D | 240 | 0xf0 | d |
| 145 | 0x91 | ` | 177 | 0xb1 | ± | 209 | 0xd1 | Ñ | 241 | 0xf1 | ñ |
| 146 | 0x92 | ' | 178 | 0xb2 | ² | 210 | 0xd2 | O | 242 | 0xf2 | ò |
| 147 | 0x93 | " | 179 | 0xb3 | 3 | 211 | 0xd3 | O | 243 | 0xf3 | ó |
| 148 | 0x94 | " | 180 | 0xb4 | ' | 212 | 0xd4 | O | 244 | 0xf4 | ô |
| 149 | 0x95 | | 181 | 0xb5 | µ | 213 | 0xd5 | O | 245 | 0xf5 | o |
| 150 | 0x96 | - | 182 | 0xb6 | ¶ | 214 | 0xd6 | Ö | 246 | 0xf6 | ö |
| 151 | 0x97 | - | 183 | 0xb7 | · | 215 | 0xd7 | x | 247 | 0xf7 | ÷ |
| 152 | 0x98 | ~ | 184 | 0xb8 | ¸ | 216 | 0xd8 | O | 248 | 0xf8 | o |
| 153 | 0x99 | T | 185 | 0xb9 | 1 | 217 | 0xd9 | U | 249 | 0xf9 | ù |
| 154 | 0x9a | s | 186 | 0xba | º | 218 | 0xda | U | 250 | 0xfa | ú |
| 155 | 0x9b | > | 187 | 0xbb | » | 219 | 0xdb | U | 251 | 0xfb | û |
| 156 | 0x9c | o | 188 | 0xbc | ¼ | 220 | 0xdc | Ü | 252 | 0xfc | ü |
| 157 | 0x9d | ? | 189 | 0xbd | ½ | 221 | 0xdd | Y | 253 | 0xfd | y |
| 158 | 0x9e | z | 190 | 0xbe | ¾ | 222 | 0xde | _ | 254 | 0xfe | _ |
| 159 | 0x9f | Y | 191 | 0xbf | ¿ | 223 | 0xdf | ß | 255 | 0xff | ÿ |

## String

### String interpolation / Formatted strings

```
int i = 42; // Your variable
// Using string interpolation
```

```
Console.WriteLine($"{i}. siffer:");
```

```
Console.WriteLine($"({Ar} + {Aj}j) * ({Br} + {Bj}j) = {C}");
```

Kan også brukes utenom Console.Write

```
string strFormat = String.Format("Hello {0}", name);
String.Format(String format, Object ... args);
```

## Integers (heltall)

### int / int

Heltallsdivisjon (to INT del på hverandre) vil returnere antall hele tall av divisor som går inn i dividenten.

### Convert to int

#### Eksplisitt konvertering (casting)

Konverterer uten hensyn til hva som er bak komma.

```
int varINTEGER = (int)varDOUBLE;
```

#### Math.Convert

Avrunder riktig - men hvis i midten avrunder den ned for partall og opp for oddetall.

```
int varInteger = (int)Math.Round(varDouble);
```

#### Convert.ToInt

```
int tall = Convert.ToInt16(Console.ReadLine());
```

## Convert to Float

```
float.Parse("stringy");
```

```
int val1 = 1;
float val2 = (float)val1;
```

```
int i = 8;
float f = Convert.ToSingle(i);
```

### Random

```
Random r = new Random();
int tall = r.Next(<min//, <maks//); // min =< tall < maks (ekskludert
maks)
Console.WriteLine(tall);

r.NextDouble // mellom 0 og 1
r.Next(10,50)/10.0 // mellom 1,0 og 4,9

int[] tall = new int[10];
tall[0] = r.Next(10,100);
```

### Array / Tabell

```
int [] tab = new int[10];
tab[3] = 11; tab[3] = tab[3] + 1;
Console.WriteLine("Verdi i posisjon 4: " + tab[3].ToString());
```

// En initialisert array av **int** eller **double** vil inneholde kun 0.

```
// seks elementer - tegn
char[] tegnTab = { 'B', 'e', 'r', 'g', 'e', 'n' };
// ti elementer - heltall
int[] tallTab = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
// tre elementer - tekst
string[] tekstTab = { "Bergen", "Oslo", "Trondheim" };
```

### Liste

Kan i mange sammenhenger brukes i stedet for tabell.
Ingen øvre grense. Legger til med Add.

Henter verdier på samme måte som Arrays.

```
List<int> tall = new List<int>();

// Bruk:
tall.Add(2); // posisjon 0
tall.Add(3); // posisjon 1
tall.Add(8); // posisjon 2
int x = tall[2] + 5; // x = 8 + 5
tall.RemoveAt(1); // fjerner 3
```

C# List (with Examples)

Iterate list

```
for (int i = 0; i < albums.Count; i++)
            Console.WriteLine(albums[i]);
```

## Funksjoner og metoder

Funksjoner gjør gjerne en spesifikk oppgave (5-7 linjer?)
Metode er en funksjon i en klasse.
Behøver navn, retur-datatype, argument-datatype og funksjonskroppen.

> I klasse "Program" må man ha "static" foran.

```
// En metode (i Program) defineres ved å skrive:
static void metodenavn(int Tall)
{
        // metodekropp
        return Resultat
}
// En slik metode kan brukes (dette omtales som metodekall) i klassen
ved å skrive:
metodenavn();

// Er man i en annen klasse må man spesifisere klassen
Program.metodenavn(999);
```

> Man bruker "static" når man er i program-klassen? Det er noe med klasser, og hvordan det skal kalles. klasse.funksjon (Console.WriteLine) eller variabel.funksjon (s.Length).

### Metoder med argumenter og *uten returverdi

```
void funksjonsnavn(dt_1 fa_1, …, dt_n fa_n)
{
        // funksjonskropp
}
```

- Forkortelse dt står for datatype og fa står for 'formell argument'.
- Vi ser at en funksjon kan ha flere argumenter og at hver argument har en datatype.
- Formelle argumenter kan brukes i funksjonskroppen som vanlige variabler.

### Metoder med argumenter og *med* returverdi

```
dt_rt metodenavn(dt_1 fa_1, …, dt_n fa_n)
{
        // metodekropp
        return utr;
}
```

- Forkortelse dt står for datatype og fa står for formell argument.
- Forkortelse dt_rt står for 'datatypen til metodens returverdi'.
- Navn utr i linjen med return står for et uttrykk (variabel, litteral, sammensatt operasjon, …) som evalueres til en verdi av datatypen dt_rt.

### «Default» argumenter

- En metodes argumenter (for verdioverføring) kan ha «default»-verdier. Disse brukes i tilfelle brukeren kaller metoden uten å oppgi (alle) metodeargumenter.
- "Default"-verdier kan oppgis i metodedefinisjonen:

```
void foo(int x = 0, int y = 0) { … }
```

### Referanser

- En referanse kan oppfattes som et nytt navn for et (allerede eksisterende) minneområde.
- Det nye navnet kan brukes på samme måte som variabelnavn (sjekke eller endre verdien).
- Dette konseptet kan brukes i C# ved argument-overføring:

    - vil kan ha mer enn ett navn for et minneområde.
    - de forskjellige navn kan ha forskjellige rekkevidder (et som brukes i funksjonen) og et som brukes i koden som kaller funksjonen.
- Nøkkelordet ref (eller out) brukes - når metoden implementeres (metodedefinisjon) og når metoden brukes (metodekall).

Metode som bruker verdioverføring:

```
static int add1(int x, int y)
{
        return x + y;
}
```

Metode som bruker referanseoverføring

```
static int add2(ref int x, ref int y)
{
        return x + y;
}
```

- Metoder har samme effekt (returnerer summen av argumenter), men på to forskjellige måter.
- `ref` sender minneaddressen (64 bits) som argument.
- Skisser minneområdet for å observere forskjeller. Bruk følgende main:

```
static void Main(string[] args)
{
        int a1 = 5;
        int a2 = 10;
        int z = add1(a1, a2); // int x = a1; int y = a2; z = 15;
        int q = add2(ref a1, ref a2); // x ⟷ a1 = 5; y ⟷ a2 = 10;
}
```

### Metode som bruker referanse-return: static void (ref)

Man kan ha en funskjon som ikke returnerer med `return`, men referanse-refererer: Kan returnere flere variabler.

```
int res = 0; // må deklarere mottaker-variabelen hvis man bruker `ref`
add3(ref a, ref b, ref res); // sender a og b, mottar res
```

```
res = add3(ref a, ref b, ref res); // DETTE FUNGERER IKKE

static void add3(ref int x, ref int y, ref int svar)
{
        svar = x + y;
}
```

### Metode som bruker referanse-return: static void (out)

Man kan ha en funskjon som ikke returnerer med `return`, men referanse-refererer: Kan returnere flere variabler.

```
int res; // trenger ikke deklgiarere mottaker-variabelen en verdihvis
man bruker out
add3(ref a, ref b, out res); // sender a og b, mottar res

res = add3(ref a, ref b, out res); // DETTE FUNGERER IKKE

static void add3(ref int x, ref int y, out int svar)
{
        svar = x + y;
}
```

### Hvorfor bruke referanseoverføring?

- Ingen duplisering av verdier (slik som ved verdioverføring).
  - Minnebesparende og (for ikke primitive datatyper som vi skal studere senere) mye raskere.
- Minneområder til argumentene blir endret i metode – det innebærer at vi kan (med visse begrensninger) bruke referanseoverføring i stedet (eller i tillegg til) for returverdier.

## Metodekall i (for eksempel) `Main`

- Verdioverføring:
  - metodedefinisjon `int add1(int x, int y) { … }`
  - metodekall `int z = add1(a1,a2);` for to heltallsvariabler `a1` og `a2`
  - I bakgrunnen skjer det noe som svarer til:
    `int x = a1; og int y = a2;`
- Referanseoverføring:
  - metodedefinisjon `int add2(ref int x, ref int y) { … }`
  - metodekall `int q = add2(ref a1, ref a2);` for to heltallsvariabler `a1` og `a2`
  - I bakgrunnen skjer det noe som svarer til:
    `ref int x = ref a1; og ref int y = ref a2;`
    som ikke er lovlige C# setninger men som vi kan oppfatte som:
    x blir et nytt navn for minneområdet til a1,
    y blir et nytt navn for minneområdet til a2

## Verdioverføring versus referanseoverføring

- Lokale/ikke lokale endringer:
  - Verdioverføring: en ny variabel (lokal til metoden) opprettes og alle manipulasjoner av den forblir lokale til metoden. Argumenter som verdioverføres kalles **inn-argumenter**
  - Referanseoverføring: et nytt navn til en variabel sitt minneområde opprettes. All manipulasjon av minneområdet via referansen merkes også utenfor metoden. Argumenter som referanseoverføres kalles **inn-ut-argumenter**. Slike inn-ut-argumenter kan brukes i stedet for (eller i tillegg til) metodens returverdi (dette vil noen ganger kreve tilpassing av kode)
- Aktuell argument til en metode som referanseoverfører det argumentet (som det er snakk om) må være en variabel – kan du forklare hvorfor!?
  - Hint hva er galt med `add2(ref 3, ref a2);` ?

## `ref` vs `out`

- `out` argument:
  - `out` argument (aktuell-argument-variabel) trenger ikke å være initialisert før den skal brukes i metodekallet
  - `out` argument (formel-argument-variabel) MÅ tilordnes en verdi før den skal (eventuelt) leses og før metoden avsluttes
- `out` argument omtaler vi som ut-argument
- `ref` argument:
  - `ref` argument (aktuell-argument-variabel) må være initialisert før den skal brukes i metodekallet
  - `ref` argument (formell-argument-variabel) kan leses og skrives til uten begrensning (eller ingen av delene)
- `ref` argument omtaler vi som inn-ut-argument
- Tommelregel: bruk `out` om du ikke MÅ bruke `ref`

Eksempel 1 out og med returverdi

```
namespace O3
{
    internal class Program
    {
        static void Main(string[] args)
        {
            double radius = 0;
            double hoyde = 0;
            double volum = 0;

            Introduksjon();

            LesSylinderDimensjoner(out radius, out hoyde);

            volum = SylinderVolum(radius, hoyde);

            Utskrift(radius, hoyde, volum);

            Console.ReadKey(true);
        }

        static void Introduksjon()
        {
            Console.Write("En sylinders volum er gitt ved formel 'V = PI
* r^2 * h'\n");
            Console.Write("Der 'r' er radius og 'h' er høyde. \n");
        }

        static void LesSylinderDimensjoner(out double radius, out double
hoyde)
        {
            Console.Write("Skriv inn radius: ");
            radius = double.Parse(Console.ReadLine());
            Console.Write("Skriv inn høyde: ");
            hoyde = double.Parse(Console.ReadLine());
        }

        static double SylinderVolum(double radius, double hoyde)
        {
            return Math.PI * Math.Pow(radius, 2) * hoyde;
        }

        static void Utskrift(double radius, double hoyde, double volum)
        {
            Console.Write("Volum til en sylinder med radus " +
radius.ToString());
            Console.Write(" og høyde " + hoyde.ToString() + " er lik ");
```

```
            Console.WriteLine(volum.ToString());
        }
    }
}
```

Eksempel 2 out

```
namespace O4
{
    internal class Program
    {
        static void Main(string[] args)
        {

            KompMult(1, 2, 3, -4, out string C);
            Console.WriteLine(C);
        }
        static void KompMult(float Ar, float Aj, float Br, float Bj, out
string C)
        {
            float Cr = Ar * Br - Aj * Bj;
            float Cj = Ar * Bj + Aj * Br;
            if (Cj < 0)
            {
                C = $"{Cr} - {Math.Abs(Cj)}j";
            } else
            {
                C = $"{Cr} + {Cj}j";
            }
        }
    }
}
```

## Filbehandling

Vi kan bruke objekter av følgende datatyper:

- StreamReader – lesing fra filer // StreamWriter – skriving til filer
- Du bør ha using System.IO; i programkoden din dersom du bruker disse i programkoden din

### Skriving

```
// Et passende filbehandlings objekt må deklareres:
StreamWriter sw;
// Et passende fil må oprettes (alternativ åpnes):
sw = File.CreateText("Minfil.txt");
// (Objekt)variabelen sw kan brukes omtrent på samme måte som Console
klassen når vi sender data til skjerm (Write eller WriteLine):
sw.Write(melding);
// NB! All data (inkludert tall) du sender vil bli lagret som tekst - En
åpnet fil bør lukkes etter bruk:
sw.Close();
```

### Lesing

Tilsvarende steg som for «skriving til fil»

```
//Et passende filbehandlings objekt må deklareres:
StreamReader sr;
//Et passende fil må åpnes:
sr = File.OpenText("Minfil.txt");
// Objektvariabelen sr kan brukes omtrent på samme måte som
Consoleklassen når vi leser data fra brukeren (ReadLine):
string data = sr.ReadLine();
// Merknad: mange andre metoder som kan brukes til fil-lesing finnes og
kan brukes.
// Eksempel 1: sr.ReadToEnd();
// Eksempel 2: File.ReadAllLines(…);
// En åpnet fil bør lukkes etter et vi er ferdig med å lese:
sr.Close();
```

Vi kan teste om vi er kommet til filslutt før vi velger å lese videre:

```
StreamReader sr = File.OpenText("Personer.txt");
bool ferdig = sr.EndOfStream;
string navn;
while (!ferdig)
{
navn = sr.ReadLine();
Console.WriteLine(navn);
ferdig = sr.EndOfStream;
}
// Merk at lesekoden forutsetter en bestemt filstruktur - hva vil skje
dersom filen har «ikke kompatibel» filstruktur?
```

Filplassering (mappe) bør som oftest oppgis når vi åpner en fil:

```
StreamReader sr;
sr = File.OpenText("D:\\Arbeid\\info.txt");
eller
sr = File.OpenText(@"D:\Arbeid\info.txt");
```

## SerialPort

Et objekt av datatypen SerialPort må opprettes (med passende argumenter).

- Krever:

```
using System.IO.Ports;
```

- En passende bibliotek-pakke må installeres i prosjekt for VS2022.
  - Bruk Tools→«NuGet Package Manager» til å finne «System.IO.Ports» - Porten må åpnes (bruk try-catch til å håndtere eventuelle problemer).
- Oversikt over aktive porter kan fås ved å bruke

```
string [] allePorter = SerialPort.GetPortNames();
```

```
SerialPort sp = new SerialPort("COM3", 9600);
sp.Open(); if (sp.IsOpen) {
        // Lesing / skriving
}
sp.Close();
```

Kode som leser / skriver (logikken må tilpasses problemstillingen):

```
if (sp.IsOpen)
{
        int antallByteSomKanLesesNaa = sp.BytesToRead;
        // Lesing
        // obs!lesing kan stoppe programmet inntil data er lest
        // logikken til programmet må tilpasses
        char etTegn = Convert.ToChar(sp.ReadChar());
        string enLinje = sp.ReadLine();
        // mange andre – ReadExisting() kan være meget nyttig
        // Skriving
        sp.Write("1234");
```

```
        sp.WriteLine("Bergen");
}
```

## Timer

Komponent av datatypen Timer (finnes i «Toolbox» der vi finner andre komponenter) kan brukes til å få GUI program til å utføre periodisk operasjon. Aktuelle Properties:

- Enabled – må settes til true når Timer skal brukes
- Interval – bestemmer hvor ofte operasjon skal utføres
- Name – navnet til Timer (objekt)variabelen i prosjektet
- Aktuelle hendelser og metoder:
- Tick – hendelse som oppstår når (Interval-definert) tid er gått
- private void t_Tick(object sender, EventArgs e) { … } – metode som utføres når hendelsen Tick har oppstått (når navnet til Timer-objekt-variabel er t)

## Snippets

### Exit GUI

```
Environment.Exit(0);
```

### Alle tall delelig med 9

```
for (int i = 1; i ≤ 100; i++)
{
    if (i % 9 == 0)
    {
        Console.WriteLine(i);

    }
}
```