Starter

Write down the line of code required to **declare** a variable called `age` to store a person's age. Choose a sensible data type.

Starter

Write down the line of code required to **declare** a variable called age to store a person's age. Choose a sensible data type.

```
int age;
```

Starter

Write down the line of code required to **assign** your age to the variable `age`.

Starter

Write down the line of code required to **assign** your age to the variable `age`.

```
age = 100;
```

Starter

Write down the code required to print `"College age"` if `age` is at least 16.

Starter

Write down the code required to print `"College age"` if age is at least 16.

```
if (age >= 16)
{
  Console.WriteLine("College age");
}
```

Binary

$$10110$$

This number is in binary.

What is binary?

What is this number in base 10?

Binary and Hexadecimal Number Bases

Denary/decimal

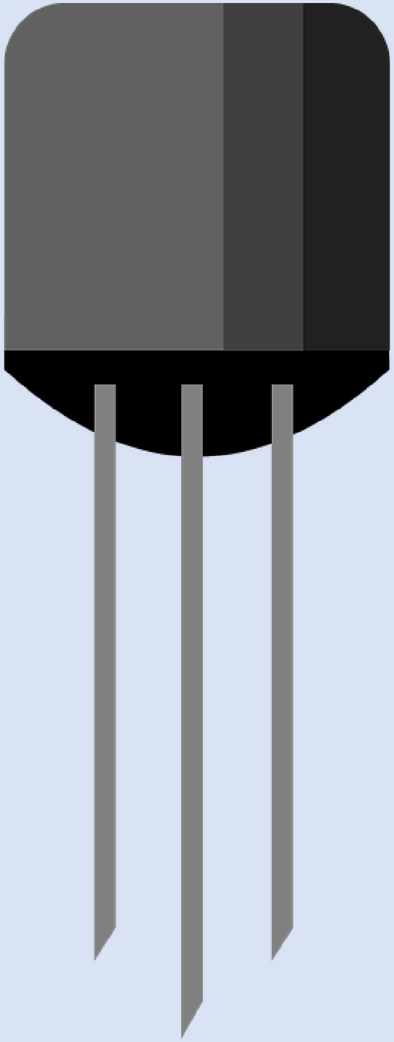
- Denary/decimal (remember both names!) is the number system you are most familiar with
- It is base 10
- Each digit in a number represents a power of 10
- e.g. 123
  - 3 units, 2 tens, 1 hundred

Denary/decimal

- Denary/decimal (remember both names!) is the number system you are most familiar with
- It is base 10
- Each digit in a number represents a power of 10
- e.g. 123
  - 3 units, 2 tens, 1 hundred
- The base of a number system states the number of possible options for each digit
  - Base 10 has 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Representing 10 different digits electronically would be complicated

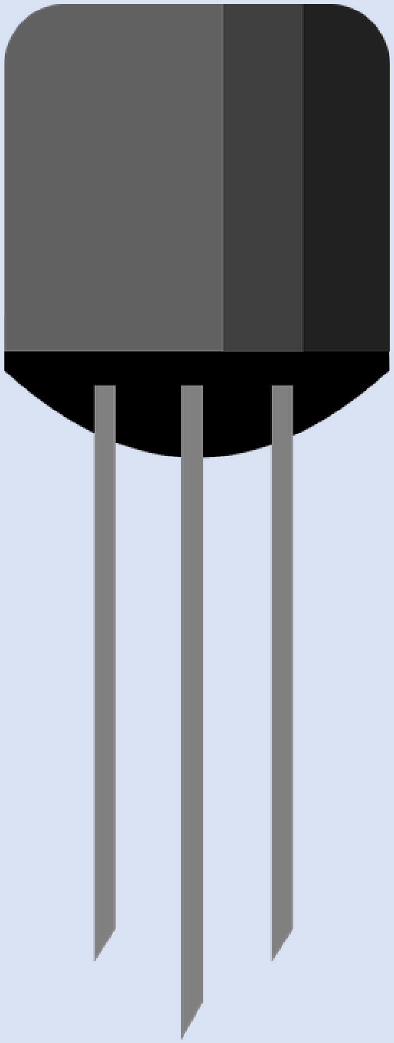
Transistors

- Transistors are tiny electronic components that act as switches with a little extra circuitry



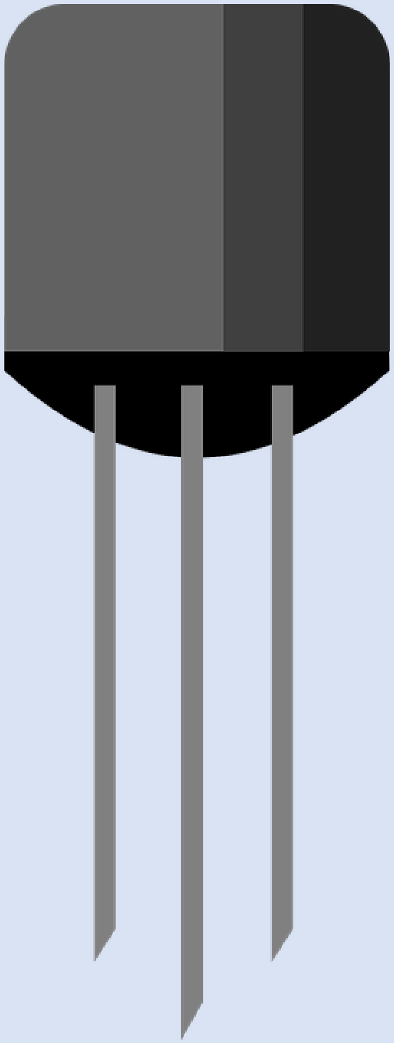
Transistors

- Transistors are tiny electronic components that act as switches with a little extra circuitry
- These switches are either on or off
  - or 1 or 0
  - or “set” or “not set”



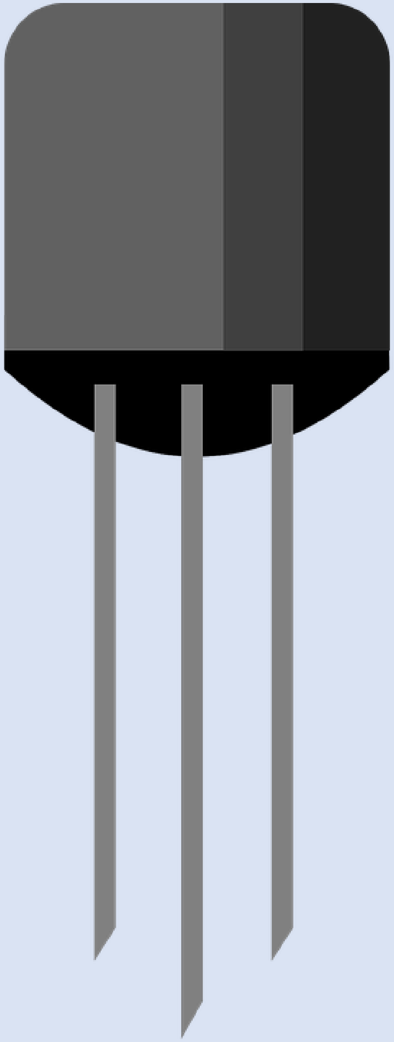
Transistors

- Transistors are tiny electronic components that act as switches with a little extra circuitry
- These switches are either on or off
  - or 1 or 0
  - or “set” or “not set”
- We can use these switches to represent digits, but only if each digit has only two possible options



Transistors

- Transistors are tiny electronic components that act as switches with a little extra circuitry
- These switches are either on or off
  - or 1 or 0
  - or “set” or “not set”
- We can use these switches to represent digits, but only if each digit has only two possible options
- Introducing binary...



Binary

- Binary is **base 2**: each “place” represents a power of 2
  - In 10110, there are 0 units, 1 two, 1 four, 0 eights and 1 sixteen
  - 2 + 4 + 16 = 22
- Each digit is a binary digit, so we call it a “bit”

Converting between binary and denary

- How do we convert from binary to denary?
- How do we convert from denary to binary?
- The trick is to draw little tables
  - Similar to place value in school
- Make notes!

Converting between binary and denary

- How do we convert from binary to denary?
- How do we convert from denary to binary?
- The trick is to draw little tables
  - Similar to place value in school
- Make notes!

How do we count in binary?

Practice

Convert

14

into binary

Practice

Convert

14

into binary

1110

Practice

Convert

101011

into denary

Practice

Convert

101011

into denary

43

Practice

Convert

51

into binary

Practice

Convert

51

into binary

110011

Practice

Convert

1000110

into decimal

Practice

Convert

1000110

into decimal

70

Exercise

Complete the table

Binary	Decimal
11100	
	37
10010	
	49
1001001	
11001100	
	88
	155
100110101	

Exercise

Complete the table

Binary	Decimal
11100	
100101	37
10010	
110001	49
1001001	
11001100	
1011000	88
10011011	155
100110101	

Exercise

Complete the table

Binary	Decimal
11100	28
100101	37
10010	18
110001	49
1001001	73
11001100	204
1011000	88
10011011	155
100110101	309

Bits, nibbles & bytes

- A bit is a single binary digit (e.g. a 0 or a 1)
- A nibble is 4 bits (e.g. 1010)
- A byte is 8 bits (e.g. 10001111)

The picture so far

- We know about denary/decimal, but it isn’t so useful for computers
- We have seen binary, but it’s difficult for humans to read and converting between binary and denary can be cumbersome

What if we had a number system that was easier for humans to read than binary, and easier to convert to/from binary?

Hexadecimal

- Hexadecimal (hex, for short) is the base 16 number system
  - We have 16 digits
- Each digit represents a power of 16
  - $1, 16, 256, \ldots$
- We sometimes use special notation to indicate that a number is in hexadecimal
  - $1A_{16}$ or $1A_h$ or 0x16

Digit	Value
0	0
1	1
…	…
9	9
A	10
B	11
C	12
D	13
E	14
F	15

Converting between hexadecimal and decimal

- How do we convert from hexadecimal to decimal? $0x1A$
- How do we convert from decimal to hexadecimal? 44
- Once again, the trick is drawing little tables.

Converting between hexadecimal and decimal

- How do we convert from hexadecimal to decimal? $0x1A$
- How do we convert from decimal to hexadecimal? 44
- Once again, the trick is drawing little tables.

- How do we count in hex?

Practice

Convert

18

into hexadecimal

Practice

Convert

18

into hexadecimal

12

Practice

Convert

0x2C

into denary

Practice

Convert

0x2C

into denary

44

Practice

Convert

60

into hexadecimal

Practice

Convert

60

into hexadecimal

3C

Practice

Convert

$1AA_h$

into decimal

Practice

Convert

$1AA_h$

into decimal

426_{10}

Exercise

Complete the table

Hexadecimal	Decimal
B	
	14
$1D$	
	26
26	
$12A$	
	200
	654
CAB	

Exercise

Complete the table

Hexadecimal	Decimal
B	
E	14
$1D$	
$1A$	26
26	
$12A$	
$C8$	200
$28E$	654
CAB	

Exercise

Complete the table

Hexadecimal	Decimal
B	11
E	14
$1D$	29
$1A$	26
26	38
$12A$	298
$C8$	200
$28E$	654
CAB	3243

Hexadecimal and binary

- What is the largest value that can be stored with one hexadecimal digit?

Hexadecimal and binary

- What is the largest value that can be stored with one hexadecimal digit?
  - 16

Hexadecimal and binary

- What is the largest value that can be stored with one hexadecimal digit?
  - 16
- So, how many bits are required to store one hex digit?

Hexadecimal and binary

- What is the largest value that can be stored with one hexadecimal digit?
  - 16
- So, how many bits are required to store one hex digit?
  - 4 – a nibble
- Considering each hex digit as a nibble makes converting between hex and binary very manageable

Practice

Convert

0x18

into binary

Practice

Convert

0x18

into binary

11000

Practice

Convert

101111011

into hex

Practice

Convert

101111011

into hex

$17B$

Practice

Convert

$$11FFEE_{16}$$

into binary

Practice

Convert

$11FFEE_{16}$

into binary

0001 0001 1111 1111 1110 1110

Practice

Convert

1111101011001110

into binary

Practice

Convert
1111101011001110
into binary

FACE

Types and Conversions

What is a data type?

Examples

- So far, we have used:
  - `int`
  - `string`
  - `double`
  - `bool`
- Others exist too
  - `float`
  - `char`

Data types

- A **data type** describes the values that a particular variable or constant can take
- It describes how those values are stored
- Data types are used to define operations
  - e.g. + works differently for integers to strings
- Sometimes we need to convert data from one type to another

Expressions and literals

- An expression, much like in maths, can be evaluated to a single value
  - e.g. `4 + 4`
- A literal is a hard-coded value that does not need to be evaluated
  - e.g. `8`
  - e.g. `"Hello, world!"`
- A literal is a type of expression

Decimal numbers

- Decimal numbers are stored as a `float`
- We'll look at what this means, and why the word "float" is relevant in a future lesson
- We can also store decimal numbers as a `double`
  - This is the same as a float, but with *double* the number of bits used to store the number
- In C#, unless specified otherwise, it is assumed that decimal numbers are doubles
  - `3.2` is a `double`
  - `3.2f` is a `float` (appending `f` to the end of a number makes it a `float`)

`char` and `string`

- A `char` is a single character (e.g. a letter, digit or symbol)
  - We use single quotes to write `char` literals (e.g. `'a'`)
- A `string` is a sequence of characters
  - We use double quotes to write `string` literals (e.g. `"apple"`)

`bool` and conditions

- A `bool` (or Boolean) can take one of two values: `true` or `false`
- Conditions (e.g. `name == "Alex"`) evaluate to Booleans
  - They are expressions, just like `4 + 4`

Type conversions

- Widening conversions
  - All integers can also be represented as doubles, but not all doubles can be represented as integers
  - Hence, `int` $\longrightarrow$ `double` is a **widening** conversion
  - These happen implicitly (we don't have to tell C# to do it)
- Narrowing conversions
  - When converting from a type to a more specific type, that is a **narrowing** conversion
  - It is possible to convert `double` $\longrightarrow$ `int`, for example
  - But we have to be explicit about it

Casting

- Casting is the process of manually converting a value from one type to another
- It is not possible to cast to every type
  - e.g. we cannot cast an `int` to a `string`

```
double temperature = 21.6;
int castTemperature = (int) temperature;
```

- What will the value of `castTemperature` be?

Casting

- Casting is the process of manually converting a value from one type to another
- It is not possible to cast to every type
  - e.g. we cannot cast an `int` to a `string`

```
double temperature = 21.6;
int castTemperature = (int) temperature;
```

- What will the value of `castTemperature` be?
- 21 – doubles are truncated when cast to integers

Rounding

```
int roundedDouble = Math.Round(10.5); // 10
```

Rounding

```
int roundedDouble = Math.Round(10.5); // 10

int anotherRoundedDouble = Math.Round(10.5,
MidpointRounding.AwayFromZero); // 11
```

Rounding

```
int roundedDouble = Math.Round(10.5); // 10
```

```
int anotherRoundedDouble = Math.Round(10.5,
MidpointRounding.AwayFromZero); // 11
```

- `Math.Round` defaults to banker's rounding: it will round to the **nearest even number**
- You can use `MidpointRounding.AwayFromZero` to change this

Parsing

- Consider the following strings:
  - `"true"`
  - `"3"`
  - `"false"`
  - `"3.14"`

Parsing

- Consider the following strings:
  - `"true"`
  - `"3"`
  - `"false"`
  - `"3.14"`
- They all represent values in other types
  - But, in their current form (strings), we can't use them like that
  - We can't add `"3.14"` to another number, because it's a string

Parsing

- Consider the following strings:
  - `"true"`
  - `"3"`
  - `"false"`
  - `"3.14"`
- They all represent values in other types
  - But, in their current form (strings), we can't use them like that
  - We can't add `"3.14"` to another number, because it's a string
- **Parsing** is the interpretation of a string into a value that it represents
  - We can parse `"3.14"` (a `string`) to `3.14` (a `double`)

Parsing in practice

- All basic data types have a `.Parse` function
  - This is particularly helpful when consuming user input (e.g. with `Console.ReadLine()`, which always returns a `string`)

```
string boolString = "true";
string doubleString = "-4.5";

bool parsedBool = bool.Parse(boolString);
double parsedDouble = double.Parse(doubleString);
```

Conversion to strings

- Sometimes it's useful to go the other way (to convert to a string)
- All basic data types have a `.ToString` function, but we have to call it on the *value*, not the type itself

```
double pi = 3.14;
string strPi = pi.ToString();
```

Practice

For each of the following, write down how the conversion would be done

- `double` ⟶ `int`
  - Using a cast
- `char` ⟶ `string`
  -
- `string` ⟶ `double`
  -
- `int` ⟶ `double`
  -

Practice

For each of the following, write down how the conversion would be done

- `double` ⟶ `int`
  - Using a cast
- `char` ⟶ `string`
  - Using `.ToString`
- `string` ⟶ `double`
  -
- `int` ⟶ `double`
  -

Practice

For each of the following, write down how the conversion would be done

- `double` ⟶ `int`
  - Using a cast
- `char` ⟶ `string`
  - Using `.ToString`
- `string` ⟶ `double`
  - Using `double.Parse`
- `int` ⟶ `double`
  -

Practice

For each of the following, write down how the conversion would be done

- `double` $\longrightarrow$ `int`
  - Using a cast
- `char` $\longrightarrow$ `string`
  - Using `.ToString`
- `string` $\longrightarrow$ `double`
  - Using `double.Parse`
- `int` $\longrightarrow$ `double`
  - Implicitly – no explicit conversion is required

Extension

- Try typing `Convert.` into your program
- Explore the different options
- Can you figure out what's happening?