Bitwise Operators

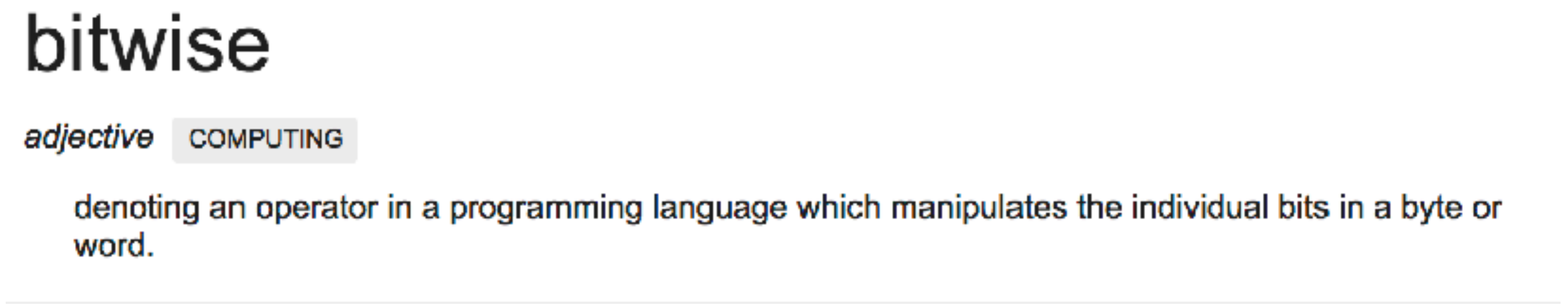
Be able to:

explain the difference between bitwise operations and operations on Boolean variables

perform AND and OR bitwise operations on bytes of data

explain the role of bitwise operations in masks, e.g. subnet masks

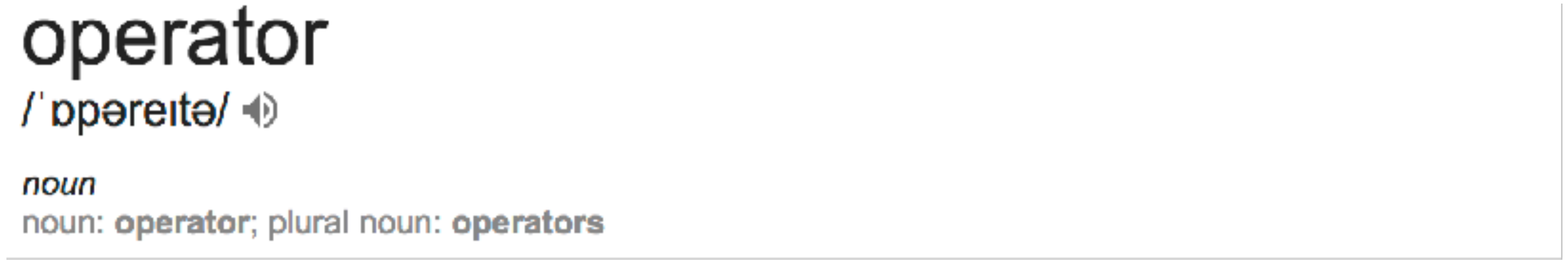
Bitwise operators?



bitwise

adjective COMPUTING

denoting an operator in a programming language which manipulates the individual bits in a byte or word.

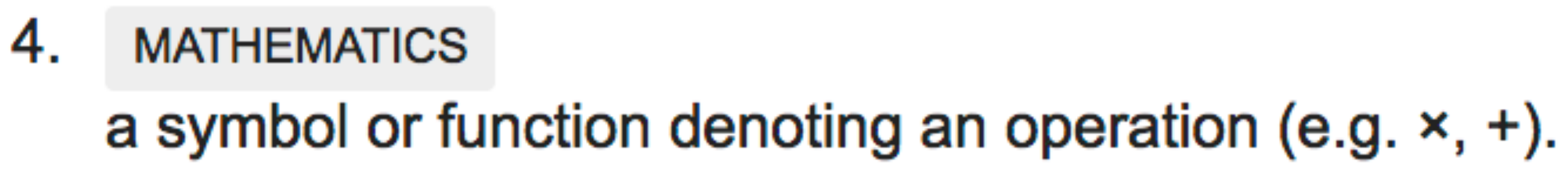


operator

/ˈɒpəreɪtə/

noun

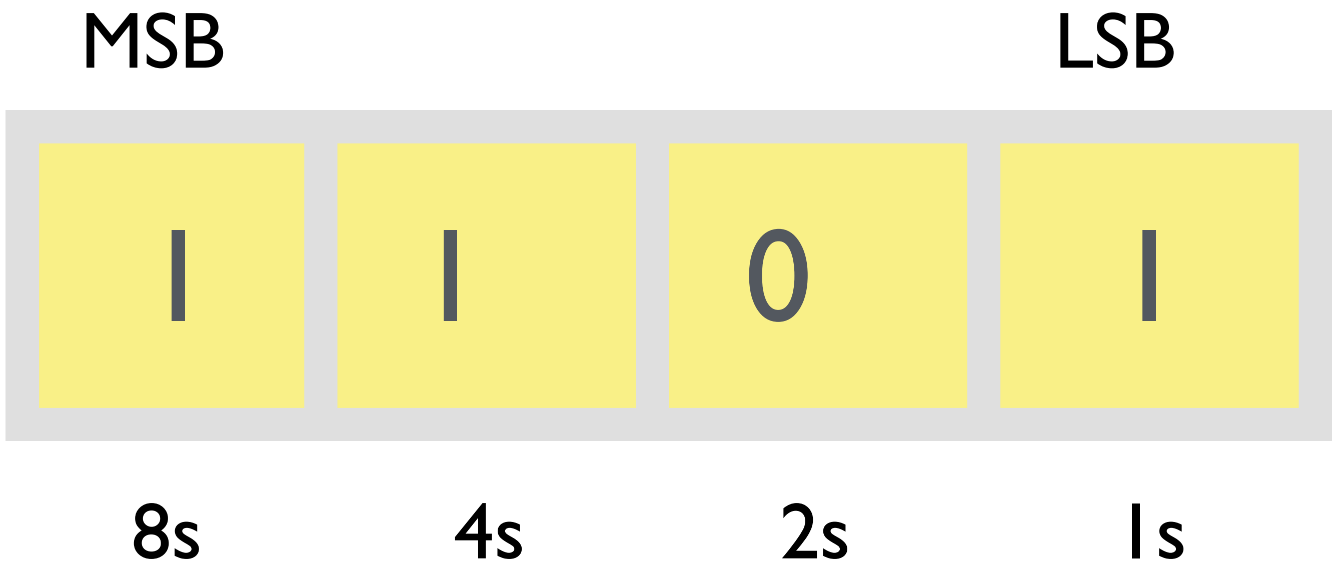
noun: **operator**; plural noun: **operators**



4. MATHEMATICS

a symbol or function denoting an operation (e.g. ×, +).

Binary Number System BASE 2



MSB: most significant bit

LSB: least significant bit

Boolean variables in Python

Python has a data type `bool`

`True or False`

define `playOrNot` function

call `playOrNot` with argument 16 returns `True`

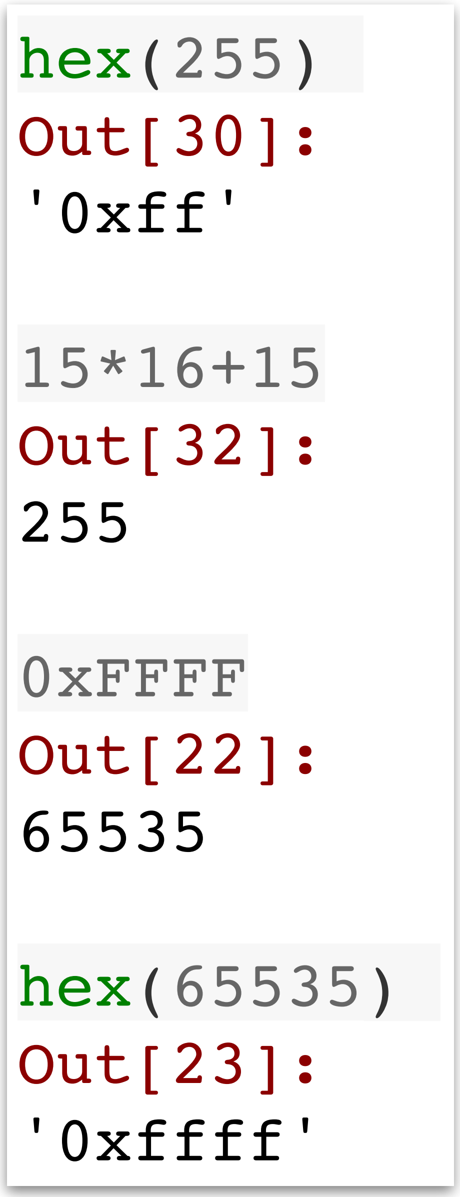
call `playOrNot` with argument 10 returns `False`

set `decision` to what `playOrNot` returns

find type of `decision`

```
def playOrNot (temp):
    play = True
    if temp < 12:
        play = False
    return play

playOrNot(16)
Out[12]:
True

playOrNot(10)
Out[13]:
False

decision = playOrNot(11)

type(decision)
Out[15]:
bool
```

Hex in Python



```
hex(255)
Out[30]:
'0xff'

15*16+15
Out[32]:
255

0xFFFF
Out[22]:
65535

hex(65535)
Out[23]:
'0xffff'
```

`hex` function to convert decimal to hex

`0x` notation for hexadecimal numbers

check if `0xff` is 255

convert from hex to decimal

convert back to hex

Bit-wise operations in Python

x << y

Returns **x** with the bits shifted to the left by **y** places (and new bits on the right-hand-side are zeros).

- Same as multiplying **x** by 2^y.

x >> y

Returns **x** with the bits shifted to the right by **y** places.

- This is the same as dividing **x** by 2^y.

x & y

bitwise and

- and operation, a bit at a time

x | y

bitwise or

- or operation, a bit at a time

Binary Addition

```
      0111
      0111
----------
      ????
```

Bitwise Operation

```
      0111
AND   1111
----------
      ????
```

Bitwise operations - AND

AND	0	1
0	0	0
1	0	1

a = b AND c

a is TRUE if b and c are TRUE
a is FALSE otherwise

e.g.

```
  01101101
& 00111100
  00101100
```

or clearing bits:

```
  01101101
& 11110000  ← Bit mask
  01100000
```

net masks

ww.iplocation.net/subnet-mask

AND &

The & operator compares each binary digit and returns a new integer.

AND	0	1
0	0	0
1	0	1

37	0	0	1	0	0	1	0	1
&23	0	0	0	1	0	1	1	1
	?	?	?	?	?	?	?	?

In Python

```
#AND & in python
res = 0b00100101 & 0b00010111
bin(res)
Out[20]:
'0b101'
```

Bitwise operations - OR

OR	0	1
0	0	1
1	1	1

a = b OR c

a is TRUE if b or c are TRUE

a is FALSE otherwise

e.g.

```
  01101101
| 00111100
  01111101
```

Useful for setting bits:

```
  01101101
| 00001110 <—— Bit mask
  01101111
```

OR |

OR	0	1
0	0	1
1	1	1

37	0	0	1	0	0	1	0	1
23	0	0	0	1	0	1	1	1
	?	?	?	?	?	?	?	?

In Python

```
#OR | in python
res = 0b00100101 | 0b00010111
bin(res)


Out[21]:’0b110111'
```

Bitwise NOT

~01101101 = 10010010

NOT	0	1
	1	0

Truth table

37	0	0	1	0	0	1	0	1
~	?	?	?	?	?	?	?	

Bit shifting

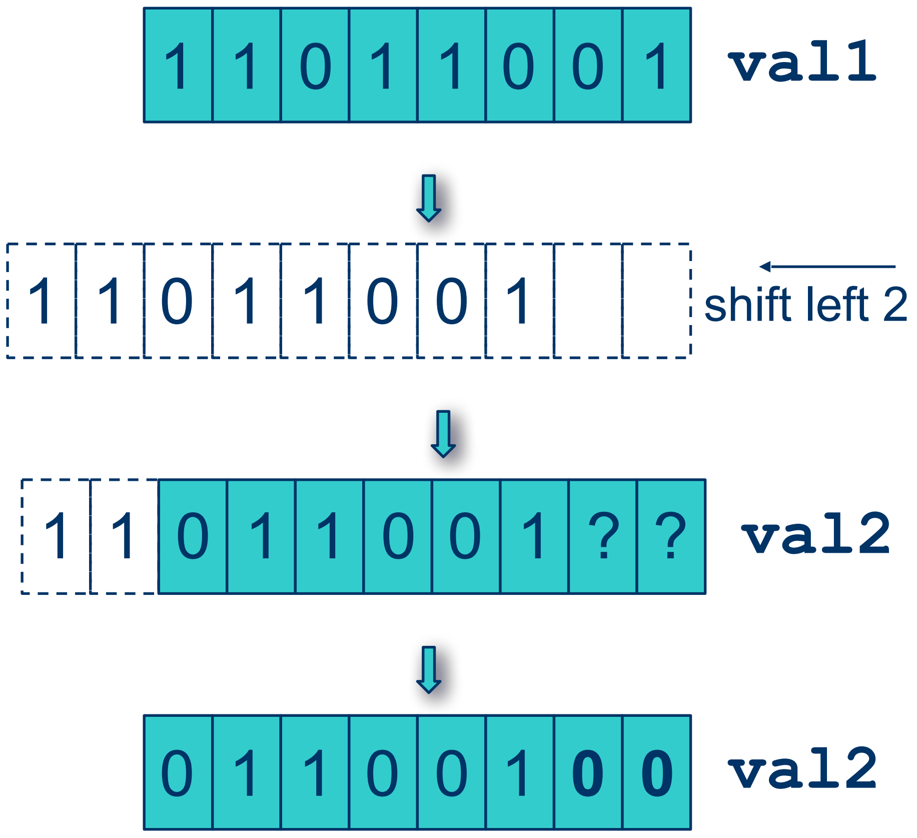
Registers (processor register) – small amount of storage within a digital processor

- More on memory types later
- Have fixed length – 8-bit register, 32-bit register etc

Bit shifting – shift bits out of the register on one end, while other bits are shifted in at the other

Bit-wise shift left

- Shift left operator: <<
  - E.g. **`val2 = val1 << 2;`**
  - Shift bits/digits in **`val1`** left by 2
  - Store in **`val2`**
  - 2 leftmost bits discarded
  - 2 rightmost bits = 0
- Examples show a byte
  - same principle for all operand sizes
- Arithmetic interpretation:



$$x << n = xB^n$$

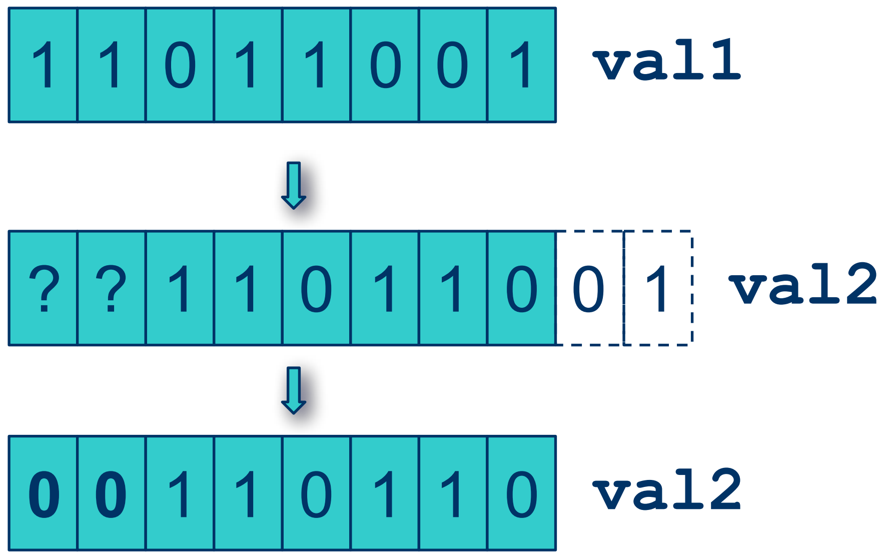
B=**10**: $13_{10} << 2 = 1300_{10} = 13 * \mathbf{10}^{2}$

B=**2**: $101_2 << 3 = 101000_2$

$(5_{10}) << 3 = (40_{10}) = 5 * \mathbf{2}^3$

Bit-wise shift right

- Shift right operator: >>
  - E.g. **`val2 = val1 >> 2;`**
  - Shift bits/digits in **`val1`** right by 2
  - Store in **`val2`**
  - 2 rightmost bits discarded
  - 2 leftmost bits = 0
- Examples show a byte
  - same principle for all operand sizes
- Arithmetic interpretation:



$$x >> n = \frac{x}{B^n}$$

B=**10**: $430_{10} >> 1 = 43_{10} = 430 / \mathbf{10}^1$

B=**2**: $1011_2 >> 2 = 10_2$

$(11_{10}) >> 2 = (2_{10}) = 11 / \mathbf{2}^2$

<u>Note</u>: value of bits discarded = remainder

More examples of binary operation uses

“short-cuts”/alternative ways to do certain operations

- Check if number odd or even

Masks in image processing

Taken from:

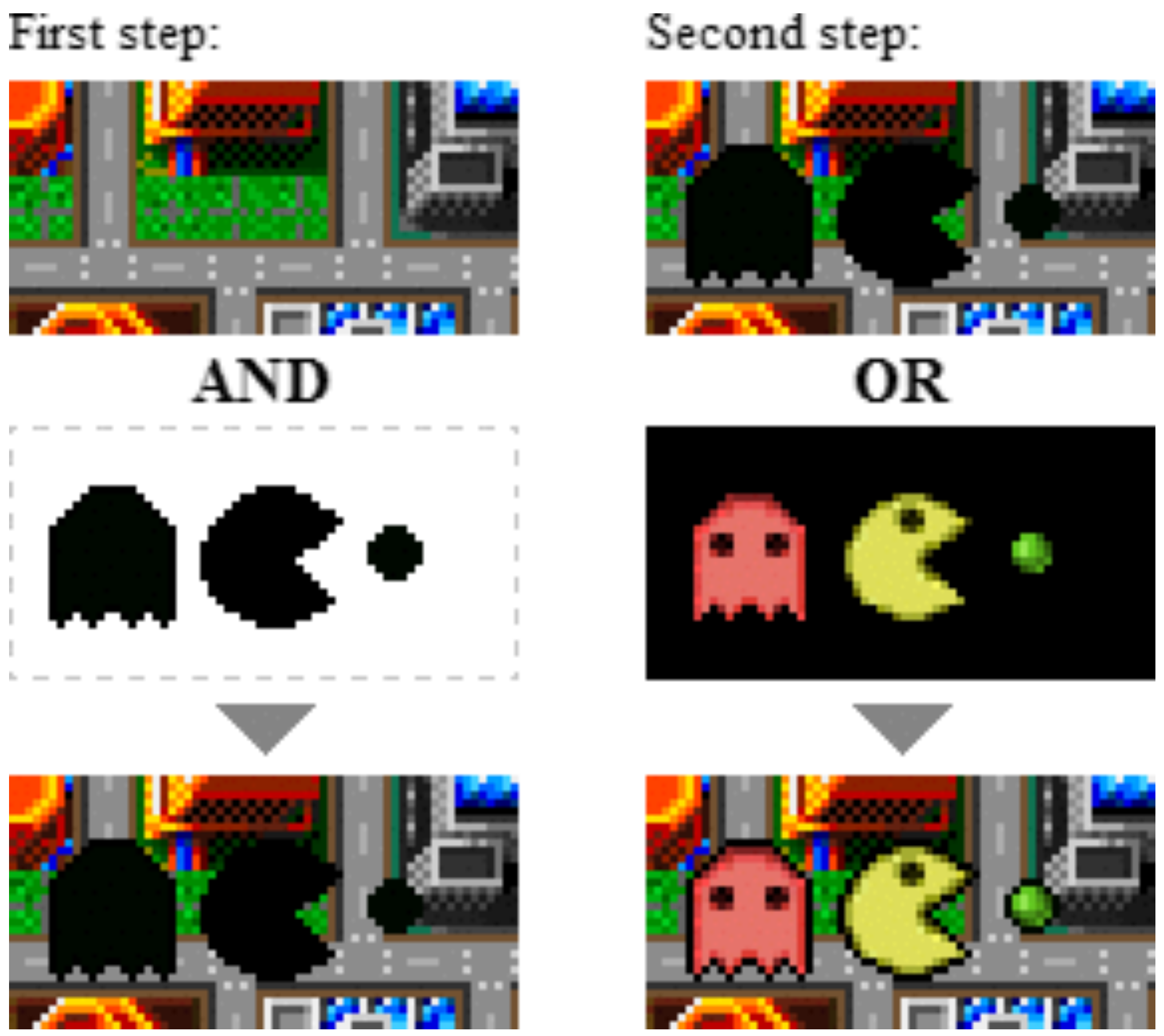
- https://en.wikipedia.org/wiki/Mask_(computing)

First step:

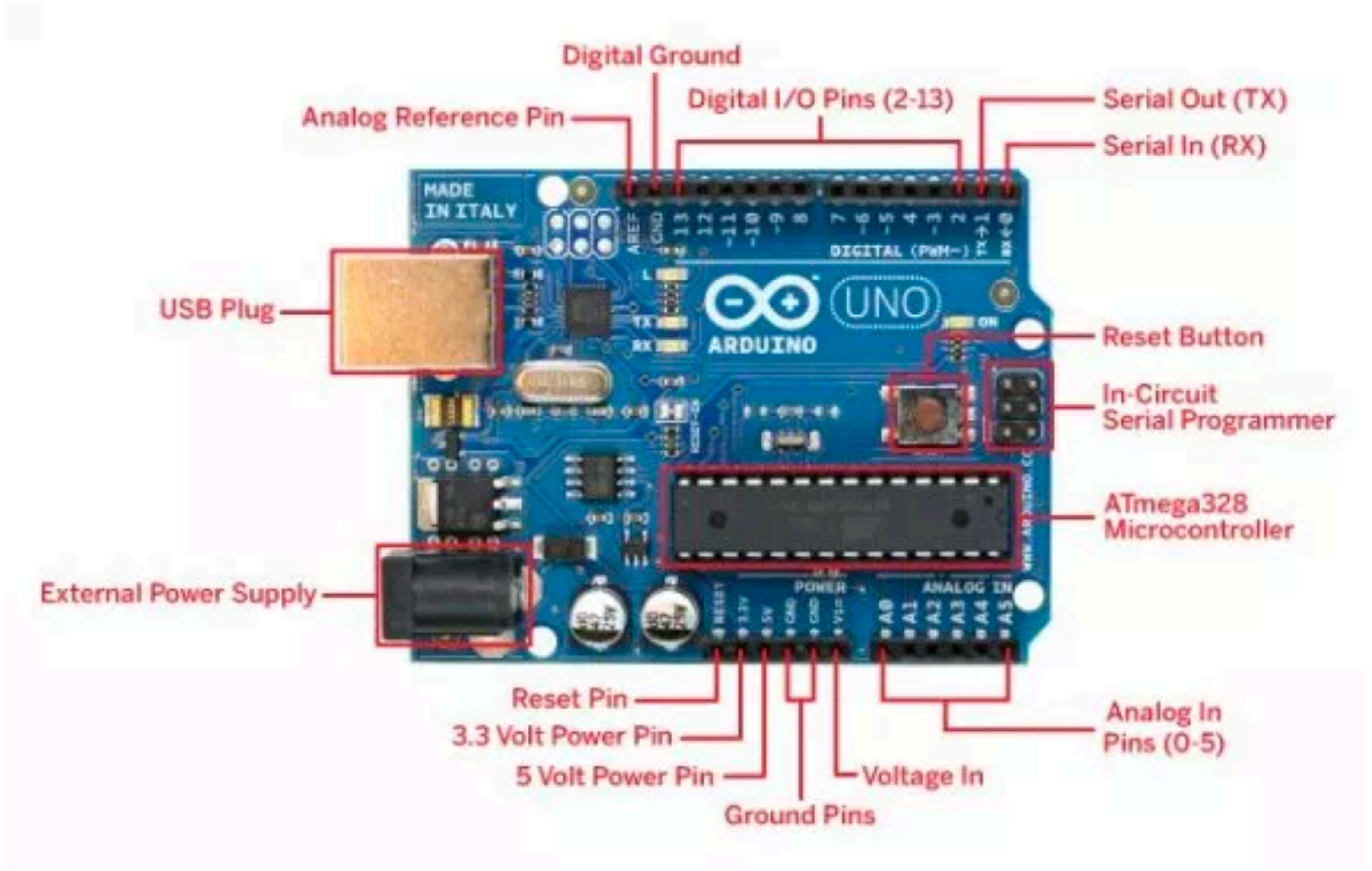
- Mask out the pixels where the characters will go

Second step:

- Mask in the characters



Arduino



Digital Ground
Analog Reference Pin
Digital I/O Pins (2-13)
Serial Out (TX)
Serial In (RX)
USB Plug
Reset Button
In-Circuit Serial Programmer
ATmega328 Microcontroller
External Power Supply
Reset Pin
3.3 Volt Power Pin
5 Volt Power Pin
Ground Pins
Voltage In
Analog In Pins (0-5)

Tutorial 2: Q6 and Q7

Bitwise Operations

1. Determine the results (in hexadecimal) of the following bitwise operations:

    1. 0x96 $\wedge$ 0xf0
    2. 0x96 $\vee$ 0x0f
    3. 0xaa $\otimes$ 0xf0
    4. $\neg$0xa5

2. Consider an IPv4 Internet address 192.168.192.23. If this is a Class C network address, then the rightmost byte is the host address and the other three bytes are the network address.

    1. What subnet mask is required to mask the network address?
    2. What bitwise logic operation will hide (set to 0s) the host address?

    see: https://www.iplocation.net/subnet-mask

Registers & Bitwise Operators

Be able to:

explain the difference between bitwise operations and operations on Boolean variables

perform AND and OR bitwise operations on bytes of data

explain the role of bitwise operations in masks, e.g. subnet masks