INTEGERS

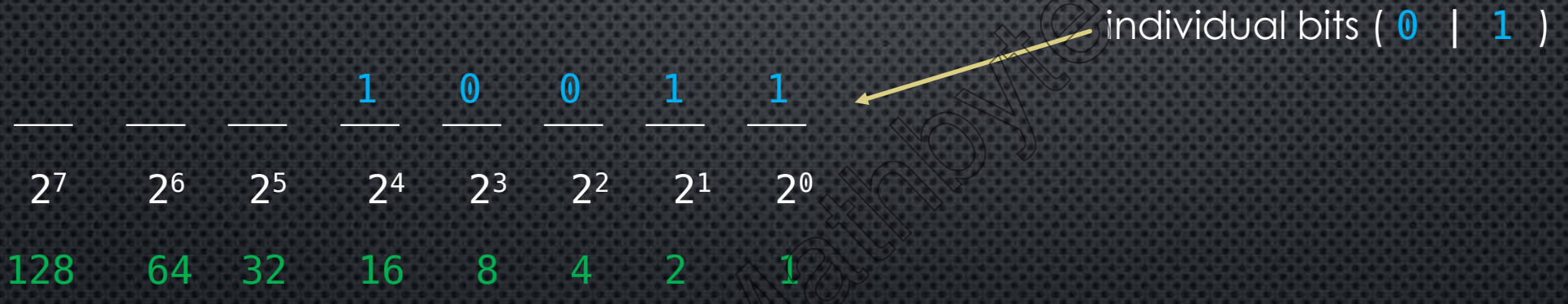
Data Type

The `int` data type

Ex: `0, 10, -100, 100000000`, …

How large can a Python `int` become (positive or negative)?

Integers are represented internally using base-2 (binary) digits, not decimal.



`1 x 16 + 0 x 8 + 0 x 4 + 1 x 2 + 1 x 1` = 16 + 2 + 1 = 19

$(10011)_2 = (19)_{10}$

Representing the decimal number 19 requires **5 bits**

What's the largest (base 10) integer number that can be represented using 8 bits?

Let's assume first that we only care about non-negative integers

1	1	1	1	1	1	1	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$$
$$= 255$$
$$= 2^8 - 1$$

If we care about handling negative integers as well, then 1 bit is reserved to represent the sign of the number, leaving us with only 7 bits for the number itself out of the original 8 bits

The largest number we can represent using 7 bits is $2^7 - 1 = 127$

So, using 8 bits we are able to represent all the integers in the range [-127, 127]

Since 0 does not require a sign, we can squeeze out an extra number,
and we end up with the range [-128, 127]

$$[-2^7, 2^7 - 1]$$

If we want to use 16 bits to store (signed) integers, our range would be:

$2^{(16-1)} = 2^{15} = 32,768$ Range: [-32,768 … 32,767]

Similarly, if we want to use 32 bits to store (signed) integers, our range would be:

$2^{(32-1)} = 2^{31} = 2,147,483,648$ Range: [-2,147,483,648 … 2,147,483,647]

If we had an unsigned integer type, using 32 bits our range would be:

$[0, 2^{32}]$ = [0 … 4,294,967,296]

In a 32-bit OS:

memory spaces (bytes) are limited by their address number → 32 bits

4,294,967,296 bytes of addressable memory

= 4,294,967,296 / 1024 kB = 4,194,304 kB

= 4,194,304 / 1024 MB = 4,096 MB

= 4,096 / 1024 GB = 4 GB

So, how large an integer can be depends on how many bits are used to store the number.

Some languages (such as Java, C, …) provide multiple distinct integer data types that use a fixed number of bits:

Java

`byte`	signed 8-bit numbers	-128, …, 127
`short`	signed 16-bit numbers	-32,768, …, 32,767
`int`	signed 32-bit numbers	-2^{31} …, $2^{31} - 1$
`long`	signed 64-bit numbers	-2^{63} …, $2^{63} - 1$

and more…

Python does not work this way

The `int` object uses a ***variable*** number of bits

Can use 4 bytes (32 bits), 8 bytes (64 bits), 12 bytes (96 bits), etc. Seamless to us

[since `int`s are actually objects, there is a further fixed overhead per integer]

Theoretically limited only by the amount of memory available

Of course, larger numbers will use more memory
and standard operators such as `+`, `*`, etc. will run
slower as numbers get larger