UNIT 6 · SESSION 7

Bits, Bytes and Encoding

Why a 1 TB drive shows as 931 GB, and how your surname is stored if it has an ñ in it.

2ND SEMESTER · SCHOOL YEAR 2026-2027

Introduction to Computing · Prof. Ryan V. Labrador

Saturday, 26 September 2026 · 2-hour lecture

TODAY

Two hours, four parts

Part 1	**Your Prelim paper, back**	And an honest way to read it that is not just looking at the mark
Part 2	**From bits to terabytes**	The ladder of units, and the 1000 versus 1024 argument
Part 3	**Character encoding**	ASCII, Unicode, UTF-8 — and where ñ and emoji actually live
Part 4	**Everything else**	Images, audio, video, compression, and what it costs to send
After	**Laboratory**	You build a character explorer and publish it as your fifth route

Part 1 is the buffer hour. It is not filler — it is the highest-value hour of revision you will get all semester.

LEARNING OBJECTIVES

By the end of today you will be able to…

1. Sort your own lost marks into three causes, and act on each differently.
2. Convert between bytes, kilobytes, megabytes, gigabytes and terabytes.
3. Explain precisely why a 1 TB drive is reported as 931 GB — and why nobody lied.
4. Distinguish KB from KiB, and Mbps from MB/s, and say why both distinctions matter.
5. Explain what ASCII, Unicode and UTF-8 each are, and how they differ.
6. Work out how many bytes a piece of text takes in UTF-8, including ñ and emoji.

PART 1

Your paper is back

The buffer hour. Read it properly once, and you will not need to read it again.

ERROR ANALYSIS

How to read a marked paper

Not the mark	**Look at what you lost**	The total tells you nothing you can act on. The crosses tell you everything.
Sort them	**Three piles, no exceptions**	Every lost mark goes into exactly one of the three piles below.
Pile 1	**I did not know it**	A content gap. Write the topic on a list. That list is your Midterm revision plan.
Pile 2	**I knew it, I wrote it badly**	Communication. Rewrite that answer properly now, while the question is in front of you.
Pile 3	**I misread the question**	The most fixable pile and the one everyone ignores. Underline the verb next time.

Do the sorting here, in this room, today. It does not happen at home. It has never happened at home.

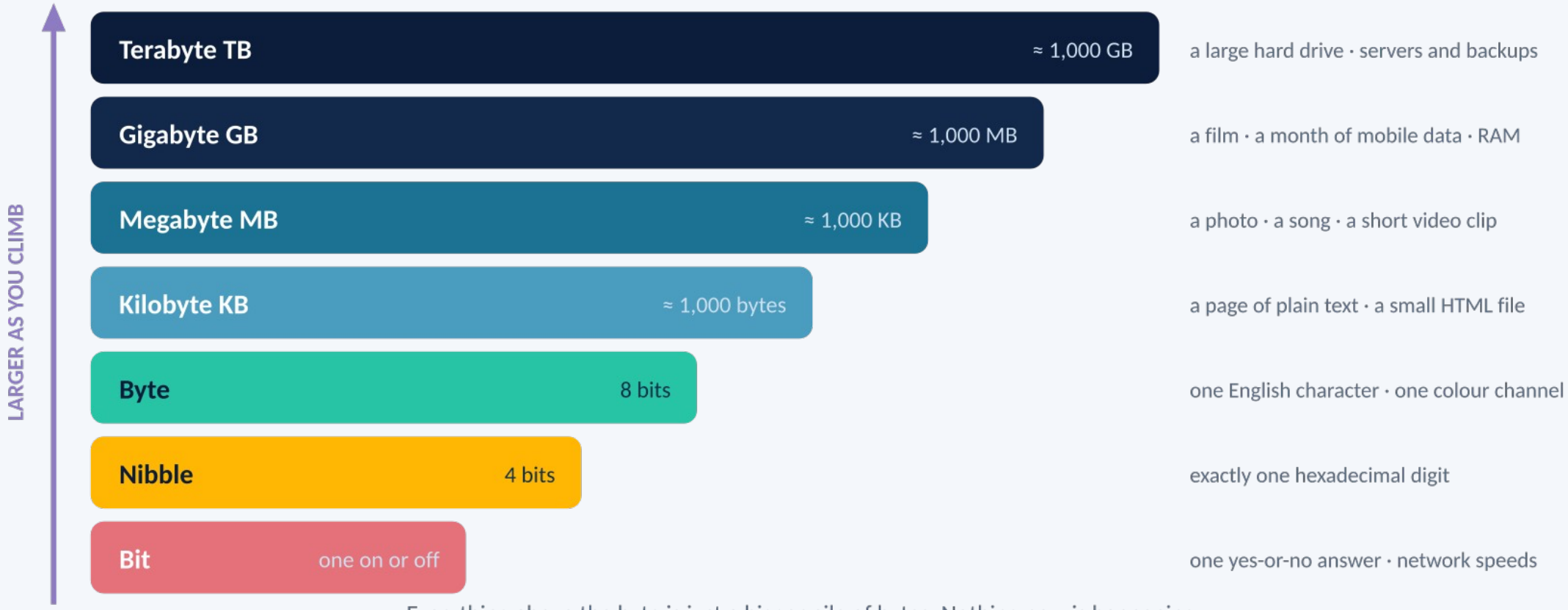
PART 2

From bits to terabytes

Two systems of measurement share the same names. That is the entire problem.

The ladder, from one bit upwards

Climb the ladder: each rung above the byte is about a thousand times the one below it



Everything above the byte is just a bigger pile of bytes. Nothing new is happening.

Two systems, one set of names

PREFIX	DECIMAL — SI	BINARY — IEC	GAP
Kilo	KB = 1,000 bytes	KiB = 1,024 bytes	2.4%
Mega	MB = 1,000 × 1,000	MiB = 1,024 × 1,024	4.9%
Giga	GB = $1{,}000^3$ = 1,000,000,000	GiB = $1{,}024^3$ = 1,073,741,824	7.4%
Tera	TB = $1{,}000^4$	TiB = $1{,}024^4$	10.0%

The gap compounds. At kilobytes it is a rounding error. At terabytes it is a tenth of the drive.

THE FAMOUS ONE

Why a 1 TB drive shows as 931 GB

STEP	THE WORKING	RESULT
What you bought	1 TB, meaning 1,000 × 1,000 × 1,000 × 1,000	1,000,000,000,000 bytes
What one GiB is	1,024 × 1,024 × 1,024	1,073,741,824 bytes
Windows divides	1,000,000,000,000 ÷ 1,073,741,824	**931.32**
What it displays	931 GB — but it means GiB	**931 GiB**
Bytes missing	None. Not one.	**0**

The drive is honest. The operating system is honest. They are using two different units with the same name.

SIDE BY SIDE

SI and IEC, and where the 931 comes from

Two counting systems, one set of names

SI · DECIMAL what drive makers sell, powers of 1,000	
kB	1,000 bytes
MB	$1{,}000^2 = 1{,}000{,}000$
GB	$1{,}000^3 = 1{,}000{,}000{,}000$
TB	$1{,}000^4 = 1{,}000{,}000{,}000{,}000$

IEC · BINARY what Windows counts in, powers of 1,024	
KiB	1,024 bytes
MiB	$1{,}024^2 = 1{,}048{,}576$
GiB	$1{,}024^3 = 1{,}073{,}741{,}824$
TiB	$1{,}024^4 = 1{,}099{,}511{,}627{,}776$

THE 1 TB DRIVE

1,000,000,000,000 ÷ 1,073,741,824 → **931.32**

Windows writes this as 931 GB — it means GiB

Not one byte is missing. The drive maker counted in 1,000s; the operating system divided by 1,024s.

Left column: what you bought. Right column: what the operating system divided by. Both are correct.

IN PRACTICE

Who counts which way

Drive makers	**Decimal, always**	1 TB means a million million bytes. It is the SI meaning and it is not a trick.
Windows	**Binary, labelled GB**	Divides by 1,024 and writes GB. That single labelling choice causes the whole argument.
macOS	**Decimal since 2009**	The same drive reports as about 1 TB. Same hardware, different arithmetic.
Linux	**Both — and it tells you**	df -h counts in 1,024s. df -H counts in 1,000s. Read the flag you typed.
RAM	**Always binary**	Memory is addressed in powers of two, so 8 GB of RAM genuinely is 8 GiB.
Networks	**Decimal, and in BITS**	25 Mbps is 25 million bits per second. Not bytes. Next slide.

THE OTHER TRAP

Megabits are not megabytes

8	Mbps	MB/s	÷ 8
bits in a byte	what the ISP sells	what your download shows	to get from one to the other
the whole conversion	megaBITS per second	megaBYTES per second	25 Mbps ≈ 3.1 MB/s

The maths	**25 ÷ 8 = 3.125**	A 25 Mbps line downloads at about 3 MB per second at its very best — less in practice.
Why it matters	**Nobody is cheating you**	Small b is bits, capital B is bytes. One letter, an eightfold difference in the number.

PART 3

Character encoding

There is nowhere in a computer to put the letter A. There is only somewhere to put a number.

Storage only holds numbers

So we agree on a table

Everybody agrees that A is 65. The machine stores 65. Any other machine that knows the agreement shows an A.

The agreement IS the standard

There is nothing about the number 65 that makes it an A. It is a convention, written down, and kept.

Different tables, broken text

When two machines use different agreements, the text arrives as nonsense. That is the entire history of encoding.

Everything you will ever read on a screen depends on two machines agreeing about a table of numbers.

ASCII

Age	**1960s**	American Standard Code for Information Interchange. Still underneath everything.
Size	**7 bits, 128 codes**	Numbers 0 to 127. It fits inside one byte with a bit to spare.
Contents	**Control codes, then text**	33 control codes, a space, punctuation, the digits, then A–Z and a–z.
Strength	**Everybody agreed**	One table, adopted everywhere. Text written in 1970 still reads correctly today.
Limit	**English, and only English**	No ñ. No é. No Baybayin. No peso sign. Not one character outside American English.

Half the world's writing did not fit. Everything after ASCII is an attempt to fix that.

INSIDE THE TABLE

How ASCII is laid out

RANGE	WHAT IS THERE	EXAMPLE
0 – 31	Control codes, not printable	10 = new line, 13 = carriage return, 9 = tab
32	Space	The most common character in most documents
48 – 57	The digits 0 to 9	The character '0' is 48, not 0. This catches everybody once.
65 – 90	Capital A to Z	A = 65 = 01000001
97 – 122	Lower case a to z	a = 97 = 01100001

A is 65 and a is 97. The difference is 32 — one single bit. Changing case is almost free.

Extended ASCII, and mojibake

One spare bit

ASCII used 7 bits of 8. The spare bit gives 128 more codes — 128 to 255 — and everybody claimed them.

Code pages

One page for Western Europe, another for Cyrillic, another for Greek. Code 241 meant a different character in each one.

Mojibake

Open a file with the wrong page and the text turns to rubbish. A name with ñ arrives as two random symbols and nobody can explain why.

Every one of those pages was a reasonable local decision. Together they were unusable. Hence Unicode.

THE FIX

Unicode

The idea	**One number per character**	Every character in every writing system gets one unique number, forever.
The notation	**A code point**	Written U+ then hex. A is U+0041. ñ is U+00F1. The peso sign is U+20B1.
The size	**Over a million slots**	Every living script, historic scripts, symbols and emoji, with room left over.
Filipino	**Baybayin has a block**	It begins at U+1700. The script has a permanent, standard home.
Important	**It says WHAT, not HOW**	Unicode assigns the numbers. Turning those numbers into bytes is a separate decision.

The last row is the one people miss. Unicode is a list of numbers. UTF-8 is how you write them down.

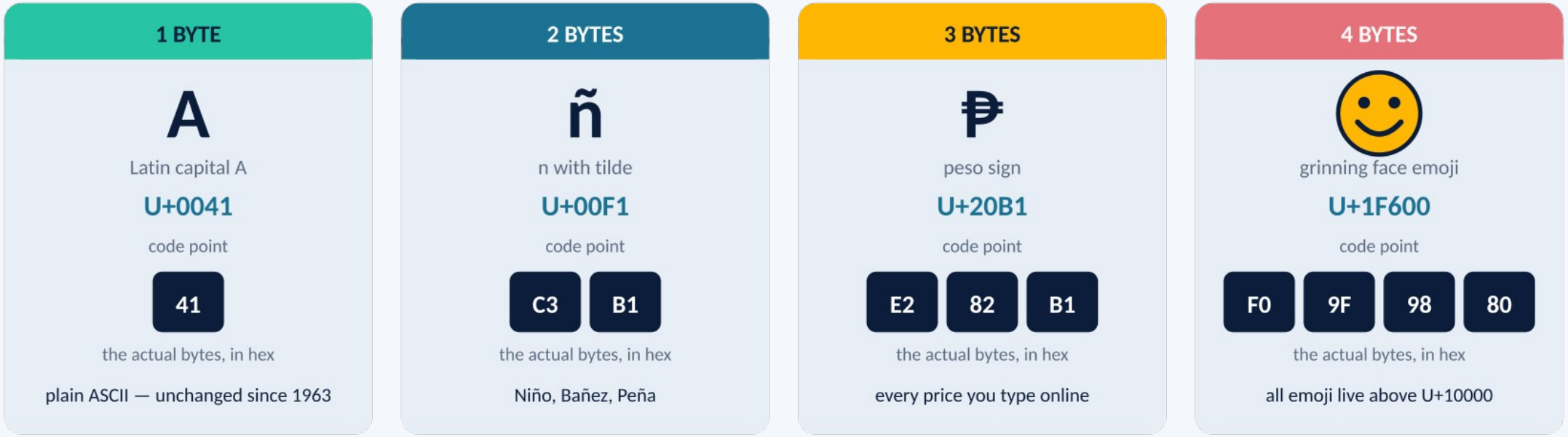
UTF-8 — one to four bytes

CODE POINT RANGE	BYTES USED	EXAMPLES
U+0000 to U+007F	1 byte	A, a, 7, space — the whole of ASCII, unchanged
U+0080 to U+07FF	2 bytes	ñ is U+00F1. é is U+00E9.
U+0800 to U+FFFF	3 bytes	₱ is U+20B1. Baybayin ᜁ is U+1700.
U+10000 and above	4 bytes	😀 is U+1F600 — and every other emoji lives up here.

UTF-8 won because plain English text is byte-for-byte identical to ASCII. Nothing that already existed broke.

How one character becomes bytes

One character in. One, two, three or four bytes out.



The character count never changes. The byte count does. That is the whole of this afternoon's laboratory.

Four characters. Ten bytes between them. The machine never sees a letter — only the row of numbers underneath.

COUNT THEM

Characters are not bytes

THE TEXT	CHARACTERS	BYTES IN UTF-8	WHY
Ilagan	6	6	All ASCII — one byte each
Mabuhay!	8	8	All ASCII, punctuation included
Niño	4	5	ñ costs two bytes, the rest cost one
₱1,500	6	8	The peso sign costs three bytes
Kumusta ka? 😀	13	16	Twelve ASCII characters, then four bytes of emoji

Ask an AI how many bytes a string takes and it will very often give you the character count. Check it.

PART 4

Everything else is numbers too

Pictures, sound and video are the same trick, done at a much larger scale.

A photograph is a grid of numbers

Pixel	**One coloured dot**	A photo is a rectangle of them. There is nothing else in the file.
Colour depth	**24-bit colour**	One byte each for red, green and blue. 16,777,216 possible colours per pixel.
Resolution	**Width × height**	A 12-megapixel photo is about twelve million pixels.
Raw size	**Pixels × 3 bytes**	12,000,000 × 3 = 36,000,000 bytes. About 34 MiB — for one photograph.
What you get	**A few megabytes**	JPEG compression closed that gap. The gap is the reason compression exists.

Unit 5 pays off here: three bytes per pixel is exactly the #RRGGBB you used in the converter lab.

Audio and video, uncompressed

Audio

- Sound is a wave. The machine samples it thousands of times a second.
- CD quality: 44,100 samples every second.
- 16 bits per sample, two channels.
- 44,100 × 16 × 2 ÷ 8 × 60 = 10,584,000
- About 10.6 MB for ONE MINUTE.

Video

- Video is still pictures, 30 or 60 a second, plus the audio.
- 1080p = 1920 × 1080 = 2,073,600 pixels.
- × 3 bytes × 30 frames per second
- = 186,624,000 bytes every second.
- About 178 MiB PER SECOND, uncompressed.

Nobody could store or send that. Everything you watch has been compressed, heavily, before it reached you.

Lossless and lossy

Lossless

- Nothing is discarded. The original comes back exactly.
- ZIP, PNG, FLAC.
- Works by removing repetition, not detail.
- Use for: documents, code, spreadsheets, archives.
- Smaller gains — often two or three times.

Lossy

- Detail is thrown away permanently. It never comes back.
- JPEG, MP3, MP4.
- Discards what human eyes and ears barely register.
- Use for: photographs, music, video.
- Much bigger gains — often ten times or more.

Saving a JPEG re-compresses it. Do that enough times and you can watch the photograph slowly die.

OUT HERE

What a byte actually costs in the provinces

Mobile data	**Sold in buckets**	A megabyte you did not need to send is a megabyte somebody paid for.
Text is tiny	**A page is kilobytes**	Ten thousand characters of plain ASCII is about ten kilobytes. Nothing at all.
Images are not	**A phone photo is megabytes**	One unresized photo can outweigh every word on your entire site.
Video	**Minutes cost hundreds of MB**	This is why a video call fails on a weak provincial link and a chat message does not.
Your site	**Resize before you upload**	A 4,000-pixel photo shown at 400 pixels wastes about 99% of what it cost to send.

A shared free Wi-Fi site or a satellite link has one pipe. Every byte you save is a byte somebody else gets.

HOLD ON TO THIS

Four things from today

	One	Two systems, one name	KB is 1,000 and KiB is 1,024. The drive is honest; so is the operating system.
	Two	Small b is bits	Mbps is not MB/s. Divide by eight, every time, before you complain.
	Three	Unicode says what, UTF-8 says how	A number for every character; one to four bytes to write it down.
	Four	Characters are not bytes	ñ is one character and two bytes. An emoji is one character and four.

If you remember one thing: check whether a number is counting bits or bytes, and in 1000s or 1024s.

AFTER THE BREAK

In the laboratory today

	By hand	Count the bytes	Your group works out five strings on paper before any keyboard is touched.
	Build	/encode	Your motto, one character per row — character, number, eight bits, hex — from a for loop.
	Test	ñ against n	Watch the byte count change while the character count does not.
	Yours	Your name, shifted	Your own full name encoded, and a Caesar shift by your group number.
	AI task	Ask it to count bytes	It very often answers with the character count. Catch it, then check with .encode() yourself.

Type your own surname into it. If it has an ñ, you are about to see exactly what the machine does with you.

WHY THIS UNIT MATTERS

It is the layer under everything else

Networks

Unit 7 is next Saturday. Bandwidth is bits per second, and now you know what a bit is.

The web

Unit 8. Every page you fetch is bytes, and the encoding declaration is a line of HTML.

Security

Unit 9. Passwords, hashes and keys are all bit lengths. Length is the whole argument.

Your own site

Every file you publish costs somebody bytes. You now know how many, and why.

Every job

Storage, bandwidth and encoding turn up in every technical role there is.

Every bill

Data plans, cloud storage and hosting are all priced in the units on slide 7.

Unit 6 is short, arithmetic, and underneath all nine units that follow it.

PROGRAMMING LAB ACTIVITY

This afternoon

Your name, as the machine actually keeps it

What you are building today

Together	**The group page**	An /encode page: your motto one character per row — character, number, eight bits, hex — and the same sentence measured in ASCII and in UTF-8.
On your own	**Your own version — this is the marked one**	Your own full name encoded the same way, plus a Caesar shift by your group number.
To match	**Your page.html twin**	The same thing again, typed out by hand as HTML.

WHERE IT GOES LIVE

`gNN.cic.rmlsystems.net/encode`	`gNN.cic.rmlsystems.net/<yourfolder>/`	`gNN.cic.rmlsystems.net/<yourfolder>/page.html`

Eight identical pages is eight zeros. Your version has to be yours — your data, your words.

NEW THIS WEEK

The Python you have not used before

`for letter in name:`
do this once for every character in turn

`ord("A")`
the number the machine stores — 65

`chr(66)`
the other way round — "B"

`"ñ".encode("utf-8")`
the bytes a character really takes up

at the shell

```
name = "CIC"

for letter in name:
    code = ord(letter)
    print(letter, code, f"{code:08b}", hex(code))
```

It prints: C 67 01000011 0x43 · I 73 01001001 0x49 · C 67 …

TWENTY POINTS

How this is marked

6	**It runs**	Your address loads and there is no traceback on it.
6	**It does this week's thing**	A for loop over a string, with ord and the eight-bit format.
4	**It is yours**	Your own name encoded, and your own shift.
2	**The twin matches**	page.html shows the same thing, by hand.
2	**Worksheet complete**	Including the AI sentence and the twin count.

DUE Sunday 27 September, 11:59pm · the full guide is at labs.cic.rmlsystems.net/guides/

Your name is a number.

Somebody had to decide which number, and then persuade the whole world to agree. That is what a standard is — and this afternoon you build the page that shows it.

Questions?