نوع داده بایت (byte)

یکی از انواع داده‌ای که شاید کمتر باهاش آشنا باشی، نوع داده‌ی byte هست.
در واقع byte هیچ فرقی با uint8 نداره؛ هر دو برای نگهداری عددی بین **0 تا 255** استفاده می‌شن.

اما خب شاید بپرسی:

"اگه دقیقاً یکین، پس چرا دو تا اسم مختلف داریم؟"

جوابش ساده‌ست: دلیل این اسم‌گذاری بیشتر **برای افزایش خوانایی کده**. وقتی با داده‌هایی مثل کاراکترها و رشته‌ها کار می‌کنیم، استفاده از byte به جای uint8 معنای بیشتری داره و باعث می‌شه کد واضح‌تر بشه.

چرا byte از 0 تا 255 رو نگه می‌داره؟

چون هر byte در حافظه معادل **8 بیت** هست.
و با 8 بیت می‌تونیم عددی بین 0 تا 255 رو نمایش بدیم.

مثلاً:

0	0	0	0	0	0	0	0

این عدد باینری که معادل عدد 0 بر پایه 10 (دِسیمال) هست کمترین عددیه که با 8 بیت میشه نشون داد

1	1	1	1	1	1	1	1

این عدد باینری که معادل عدد 255 بر پایه 10 (دِسیمال) هست کمترین عددیه که با 8 بیت میشه نشون داد

چطور عدد باینری رو به دسیمال تبدیل می‌کنیم؟

هر رقم در عدد باینری یه "جایگاه" داره. از راست به چپ، جایگاه‌ها 0، 1، 2، ... هستن.

برای تبدیلش به عدد پایه 10 (decimal) فقط کافیه هر رقم رو در $2^{\text{جایگاه}}$ ضرب کنیم و همه رو با هم جمع کنیم.

مثال برای عدد 11111111:

جایگاه	رقم	حاصل ضرب رقم در $2^{\text{جایگاه}}$
0	1	$1 * 2^0 = 1$
1	1	$1 * 2^1 = 2$
2	1	$1 * 2^2 = 4$
3	1	$1 * 2^3 = 8$
4	1	$1 * 2^4 = 16$
5	1	$1 * 2^5 = 32$
6	1	$1 * 2^6 = 64$
7	1	$1 * 2^7 = 128$

مجموع:

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

در واقع به همین روش بود که فهمیدیم عدد 11111111 در باینری معادل عدد 255 در دسیمال هست

حالا اینا چه ربطی به متن و حروف دارن؟

وقتی اولین کامپیوترها ساخته شدن، نیاز بود کاراکترهایی مثل حروف انگلیسی(A تا Z) اعداد (0 تا 9)، علائم نگارشی و چند دستور کنترلی (مثل newline) ذخیره بشن.

جمع این کاراکترها حدوداً 128 تا شد، که به‌راحتی توی یه byte (که میتونه 256 مقدار مختلف یعنی از 0 تا 255 رو داشته باشه) جا می‌گیرن.
به همین خاطر، استانداردی به اسم **ASCII** تعریف شد که هر کاراکتر رو با یه عدد بین 0 تا 127 نمایش می‌داد.

مثلاً:

حرف	معادل عددی در ASCII
A	65
a	97
0	48

برای ذخیره‌ی کلمه‌ی "Hello" فقط به 5 بایت نیاز داریم، چون هر حرفش دقیقاً یه بایت اشغال می‌کنه.

رابطه‌ی byte و string

در زبان Go **کوچک‌ترین نوع داده برای اعداد صحیح بدون علامت** uint8 هست، ولی برای کار با رشته‌ها از همون uint8 با اسم byte استفاده می‌شه.

این باعث می‌شه وقتی رشته‌ای تعریف می‌کنی، در واقع مجموعه‌ای از byteها باشه:

```
var s string = "Hello"
```

در پشت‌صحنه، این رشته همون چیزی هست که ما بهش می‌گیم:

"یه slice ازbyte ها"

یعنی:

```
[]byte{'H', 'e', 'l', 'l', 'o'}
```

برای اثبات این موضوع کافیه از عملگر ایندکس استفاده کنی.

```
var s string = "Hello"
fmt.Println(s[0])
```

خروجی

```
72
```

عدد 72 (که از نوع byte هست) معادل کاراکتر H در سیستم ASCII هستش

جدول ASCII

Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char	Dec	Bin	Char
0	00000000	NUL	22	00010110	SYN	44	00101100	,	66	01000010	B
1	00000001	SOH	23	00010111	ETB	45	00101101	-	67	01000011	C
2	00000010	STX	24	00011000	CAN	46	00101110	.	68	01000100	D
3	00000011	ETX	25	00011001	EM	47	00101111	/	69	01000101	E
4	00000100	EOT	26	00011010	SUB	48	00110000	0	70	01000110	F
5	00000101	ENQ	27	00011011	ESC	49	00110001	1	71	01000111	G
6	00000110	ACK	28	00011100	FS	50	00110010	2	72	01001000	H
7	00000111	BEL	29	00011101	GS	51	00110011	3	73	01001001	I
8	00001000	BS	30	00011110	RS	52	00110100	4	74	01001010	J
9	00001001	TAB	31	00011111	US	53	00110101	5	75	01001011	K
10	00001010	LF	32	00100000	␣	54	00110110	6	76	01001100	L
11	00001011	VT	33	00100001	!	55	00110111	7	77	01001101	M
12	00001100	FF	34	00100010	"	56	00111000	8	78	01001110	N
13	00001101	CR	35	00100011	#	57	00111001	9	79	01001111	O
14	00001110	SO	36	00100100	$	58	00111010	:	80	01010000	P
15	00001111	SI	37	00100101	%	59	00111011	;	81	01010001	Q
16	00010000	DLE	38	00100110	&	60	00111100	<	82	01010010	R
17	00010001	DC1	39	00100111	'	61	00111101	=	83	01010011	S
18	00010010	DC2	40	00101000	(	62	00111110	>	84	01010100	T
19	00010011	DC3	41	00101001	)	63	00111111	?	85	01010101	U
20	00010100	DC4	42	00101010	*	64	01000000	@	86	01010110	V
21	00010101	NAK	43	00101011	+	65	01000001	A	87	01010111	W

جمع‌بندی

- byte فقط یه اسم دیگه برای uint8 هست که در زمینه‌ی کاراکترها و متن‌ها استفاده می‌شه.
- یکی از کاربردهای اصلی نوع تایپ byte اینه که می‌تونیم کاراکترهای ASCII رو ذخیره کنیم.
- در Go رشته‌ها در واقع مجموعه‌ای از byteها هستن، یعنی slice از uint8.

زبان های غیر انگلیسی

ما با **ASCII** شروع کردیم که فقط 128 کاراکتر داشت و برای زبان انگلیسی ساخته شده بود. بعدش نسخه‌ی **Extended ASCII** اومد که 256 تا کاراکتر داشت، ولی بازم برای بقیه زبان‌ها (مثل فارسی، عربی، چینی و...) کافی نبود.

سیستم کدنویسی جهانی Unicode

برای حل این مشکل، استانداردی به اسم **Unicode** ساخته شد.

Unicode چیه؟

Unicode یه سیستم استانداره که **به هر کاراکتر در هر زبان** (فارسی، چینی، روسی، ژاپنی، اموجی و...) یه **کد یکتا** اختصاص میده. به این کدها می‌گن **code points.**

مثلا:

حرف	معادل عددی در Unicode
A	65
پ	1662
☺	128522

اما Unicode چطور در حافظه ذخیره میشه؟

Unicode فقط یه سیستم عددهیه. حالا باید این عددها رو در حافظه ذخیره کنیم. اینجا پای **Encoding** به میون میاد.

مهم‌ترین Encoding که الان همه‌جا استفاده میشه UTF-8:

UTF-8 محبوب‌ترین روش برای ذخیره کاراکترهای Unicode ـه. دلیلش اینه که:

- برای کاراکترهای ASCII فقط 1 بایت استفاده می‌کنه
- برای کاراکترهای غیرلاتین (مثل فارسی، چینی...) بین 2 تا 4 بایت مصرف می‌کنه
- با ASCII سازگاره

مثلاً:

- A با 1 بایت 01000001
- پ با 2 بایت 11011001 10111110
- ☺ با 4 بایت 11110000 10011111 10011000 10101010

در استاندارد Unicode حروف انگلیسی و سایر کاراکترهای استاندارد ASCII **دقیقاً همون عددی رو دارن که در ASCII داشتن**.

چرا این اتفاق افتاده؟

Unicode در واقع یه نسخه‌ی توسعه‌یافته از ASCII محسوب میشه. برای اینکه **سازگاری با سیستم‌های قدیمی** حفظ بشه، تصمیم گرفتن **کدهای 0 تا 127 رو بدون تغییر نگه دارن**.

یعنی چی؟

- اگر یه فایل UTF-8 باشه و فقط حروف انگلیسی داشته باشه **دقیقاً شبیه یه فایل ASCII خواهد بود.**
- این سازگاری یکی از دلایل محبوبیت زیاد UTF-8 هم هست.

چند کاراکتر در Unicode وجود داره؟

تا امروز (سال ۲۰۲۵)، استاندارد Unicode

- **1,114,112 کد پوینت (Code Point)** تعریف کرده.
- ولی فقط حدود **143,859 کاراکتر واقعی** تعریف و استفاده می‌شن (مثل حروف، اعداد، علائم، اموجی و ...).
- باقی کدها **رزرو شده یا هنوز استفاده نشدن**.

Unicode به صورت مداوم در حال به‌روزرسانیه، و در هر نسخه جدید کاراکترهای بیشتری اضافه میشه.

Rune چیست؟

اگر روزی برسه که تمام ظرفیت Unicode پر بشه، باید بتونیم **1,114,112 کاراکتر مختلف** رو ذخیره کنیم.

- برای نمایش این تعداد به صورت باینری، حداقل **3 بایت (21 بیت)** لازمه.
- اما برای سادگی، زبان Go تصمیم گرفته از **4 بایت (32 بیت)** استفاده کنه.
- این ۴ بایت می‌تونستن int32 باشن، اما برای خوانایی بیشتر نوع داده‌ی خاصی به نام rune معرفی شده که **دقیقاً همون int32 هست**، ولی مفهومش رو بهتر منتقل می‌کنه: rune یعنی یه کاراکتر از Unicode

چطور تو Go اینو ذخیره و استفاده می‌کنیم؟

نوع داده‌ی string تو زبان Go در واقع یه **slice از byte** هست ([]byte) ولی با این فرض که اون بایت‌ها به صورت **UTF-8** رمزگذاری شدن.

مثال:

```
s := "سلام"
fmt.Println(s) // خروجی: سلام
fmt.Println(len(s)) // چون تعداد بایت‌ها (نه حروف)؛ UTF-8 برای فارسی از 2 یا 3 بایت استفاده می‌کنه
```

در زبان Go حتی وقتی رشته‌ای شامل کاراکترهای غیرانگلیسی مثل فارسی یا چینی باشه، **به‌صورت byteها (یعنی uint8) ذخیره میشه نه rune.**

حالا بذار کامل و مرحله‌به‌مرحله توضیح بدم تا کامل جا بیفته:

مرحله 1:

string در زبان Go یعنی **یه دنباله از byteها** (نه runeها)

یعنی پشت‌صحنه، این رشته به‌صورت UTF-8 **رمزگذاری (Encoding)** میشه و هر کاراکتر به یک یا چند بایت تبدیل میشه.

مرحله 2:

- Unicode یه استانداردِه که به هر کاراکتر یه عدد (code point) میده.
  - مثلا کاراکتر "س" در Unicode معادل U+0633 (به دسیمال 1587) هست
- UTF-8 یه روش برای **رمزگذاری (Encoding)** این عددهاست به **بایت‌ها**.

بنابراین کاراکتر "س" در UTF-8 به این 2 بایت تبدیل میشه:

```
0xD8 0xB3
```

یعنی داخل رشته سلام، حافظه چیزی شبیه این داره:

```
[0xD8, 0xB3, 0xD9, 0x84, 0xD8, 0xA7, 0xD9, 0x85]
```

پس rune کی استفاده میشه؟

وقتی می‌خوایم یه رشته رو بر اساس کاراکتر بخونیم، نه بایت، از rune استفاده می‌کنیم:

```
for i, r := range s {
    fmt.Println("index: ”, i, “character value: “, r)
}
```

اینجا Go خودش UTF-8 رو decode می‌کنه و هر rune (هر کاراکتر واقعی) رو بهمون میده.

نکته مهم:

در زبان Go وقتی با یک رشته (string) سر و کار داریم و از حلقه‌ی for range استفاده می‌کنیم، زبان Go به‌طور پیش‌فرض هر بار یک **کاراکتر کامل (rune)** رو بهمون می‌ده، نه صرفاً یک بایت!

چرا؟ دلیل این رفتار چیه؟

از اون‌جایی که:

- رشته‌ها در Go به‌صورت دنباله‌ای از بایت‌ها ([]byte) ذخیره می‌شن،
- و بعضی از کاراکترها (مثل حروف فارسی، چینی یا اموجی‌ها) بیشتر از یک بایت هستن (چون UTF-8 هست)

اگه با بایت‌ها کار کنیم، ممکنه یه کاراکتر رو نصفه بخونیم و به هم بریزه!

برای همین Go تصمیم گرفته که توی for range، به‌جای byte با **rune** کار کنه.

یعنی:

- هر بار یک کاراکتر یونیکد کامل (rune) بده
- و همچنین ایندکس i نشون بده این کاراکتر از کدوم بایت توی string شروع شده.

نتیجه نهایی:

حتی وقتی string ما فارسی یا چینیه، باز هم به‌صورت byte های UTF-8 ذخیره میشه، نه به‌صورت rune اما اگه بخوایم با هر کاراکتر واقعی کار کنیم، اون موقع Go خودش اونا رو تبدیل به rune می‌کنه و به ما میده.

تفاوت string و []byte

تا اینجای درس با نوع داده string آشنا شدیم. گفتیم که string در واقع مجموعه‌ای از بایت‌ها هست. حالا می‌خوایم ببینیم فرقش با []byte (یعنی یه آرایه قابل تغییر از بایت‌ها) چیه، چرا string قابل تغییر نیست، و اصلاً کی از کدومش استفاده کنیم؟

string یعنی چی؟

string یعنی یه دنباله (یا لیست) از بایت‌ها که **قابل خوندنه ولی قابل تغییر دادن نیست.**

مثال

```
text := "hello"
```

اینجا text یه رشته‌ست (string) که پشت صحنه **مثل یه لیست از بایت‌ها** ذخیره شده. اما تو اجازه نداری مقدارش رو تغییر بدی:

```
text[0] = 'H' // ⛔ ارور! نمی‌تونی یه حرف داخل string رو عوض کنی
```

[]byte یعنی چی؟

[]byte یعنی یه لیست از بایت‌ها که **کاملاً قابل تغییره.**

مثال

```
data := []byte{'h', 'e', 'l', 'l', 'o'}
data[0] = 'H' // ✅ مشکلی نداره
```

چرا string قابل تغییر (mutable) نیست، ولی byte[] هست؟

بذار یه مثال واقعی بزنم:

فرض کن یه کتاب چاپ شده دستته. تو می‌تونی صفحاتش رو بخونی، ولی نمی‌تونی کلماتش رو تغییر بدی. این میشه string

حالا فرض کن یه دفترچه یادداشت داری که هر وقت خواستی می‌تونی توش چیزی بنویسی یا پاک کنی. این میشه byte[]

stringها به‌طور خاص طراحی شدن که فقط **خواندنی (read-only)** باشن. این باعث می‌شه:

- حافظه کمتر مصرف بشه
- سرعت برنامه بیشتر بشه
- امنیت و پایداری داده‌ها بالا بره (چون مطمئنی کسی رشته‌تو دستکاری نمی‌کنه)

ولی byte[] برای جاهایی استفاده می‌شه که نیاز داریم داده‌ها رو **بیشتر دستکاری کنیم.**

هزینه تبدیل بین string وbyte[]

گاهی اوقات می‌خوای string رو تبدیل به byte[] کنی یا برعکس.

مثال

```
s := "hello"
b := []byte(s) // string → []byte
s2 := string(b) // []byte → string
```

این کار کاملاً شدنیه، ولی حواست باشه که **این تبدیل‌ها ارزون نیستن!**

وقتی تبدیل می‌کنی Go باید:

- حافظه جدید ایجاد کنه
- تمام بایت‌ها رو کپی کنه
- ساختار جدید بسازه

اگه توی یه حلقه یا بارها این تبدیل رو انجام بدی، می‌تونه برنامه رو کند کنه و رم زیادی مصرف کنه. پس فقط وقتی لازمه این کارو انجام بده.

کی از string استفاده کنیم؟ کی از[]byte ؟

نیاز	استفاده از
متن‌هایی که فقط می‌خوای بخونی	string
ذخیره‌سازی ساده کلمات و جملات	string
زمانی که به سرعت و کم‌حافظگی اهمیت می‌دی	string
وقتی می‌خوای بخشی از متن رو تغییر بدی	[]byte
کار با داده‌های باینری (مثل فایل، شبکه، تصویر)	[]byte
رمزگذاری و رمزگشایی	[]byte
وقتی می‌خوای به صورت کاراکتر به کاراکتر چیزی رو ویرایش کنی	[]byte

توابع چاپ از کتابخونه fmt

قبل از اینکه با کتابخونه‌ی strings بیشتر آشنا بشیم، بهتره یه نگاهی عمیق‌تر به توابع چاپ در کتابخونه‌ی fmt بندازیم. به‌طور کلی، سه تابع مهم و پرکاربرد در این کتابخونه برای چاپ مقادیر — مخصوصاً رشته‌ها — عبارتند از:

- fmt.Print
- fmt.Println
- fmt.Printf

در ادامه، این سه تابع رو دقیق‌تر بررسی می‌کنیم:

تابع fmt.Print

با استفاده از این تابع، می‌تونیم به تعداد بی‌نهایت مقدار (مثل رشته، عدد و...) به‌صورت جداگانه وارد کنیم و همه‌شون در خروجی چاپ بشن.

```
fmt.Print(“Hello”, ”2”, “You”)
fmt.Print(“My”, “Friend”)
```

خروجی

```
Hello2YouMyFriend
```

تابع fmt.Println

این تابع هم مثل fmt.Print می‌تونه چندین مقدار رو چاپ کنه، ولی با دو تفاوت مهم:

1. **بین مقادیر فاصله می‌ذاره** (به‌صورت خودکار)
2. **در پایان خروجی، یک کاراکتر \n اضافه می‌کنه** (یعنی بعد از چاپ، به خط بعد می‌ره)

```
fmt.Println(“Hello”, ”2”, “You”)
fmt.Println(“My”, “Friend”)
```

خروجی

```
Hello 2 You
My Friend
```

همون‌طور که دیدی:

- بین "Hello" و 2 و “You” یک فاصله خودکار اضافه شده.
- بین “My” و “Friend” یک فاصله خودکار اضافه شد
- چون fmt.Println خودش \n انتهاش اضافه می‌کنه، چاپ دوم رفت خط بعد.

نکته: اگه بعد از fmt.Println از fmt.Print هم استفاده کنی، باز هم خروجی می‌ره خط بعد چون اون \n هنوز سر جاشه:

```
fmt.Println(“Hello”, ”2”, “You”)
fmt.Println(“My”, “Friend”)
fmt.Print(“Where”, “Are”, “U”, “From”)
```

خروجی

```
Hello 2 You
My Friend
WhereAreUFrom
```

حالا اگه بخوای خودت مشخص کنی که کی بره خط بعد؟

فرقی نمی‌کنه که از fmt.Print استفاده می‌کنی یا fmt.Println یا حتی fmt.Printf کافیه خودت داخل متن از \n استفاده کنی:

به مثال زیر توجه کن

```
fmt.Print(“Hello”, ”2”, “You\n”)
fmt.Print(“My\n”, “Friend”)
```

خروجی

```
Hello 2 You
My
Friend
```

هر جا که \n ظاهر شده خط شکسته میشه و باقی متن به خط بعد منتقل میشه، \n یکی از پرکاربردترین escape sequence هاست

Escape Sequence چیست؟

Escape sequenceها ترکیبی از کاراکترها هستن که با \ (بک‌اسلش) شروع می‌شن و توی رشته‌ها برای انجام یه کار خاص استفاده می‌شن؛ مثلاً رفتن به خط بعد، چاپ کوتیشن، تب، و غیره.

لیست پرکاربردترین Escape Sequence ها

کد	معنی
\n	رفتن به خط جدید
\t	یک فاصله تب (مثل دکمه Tab)
\\	خود علامت \
\"	علامت نقل قول دوتایی (برای چاپ درون متن)
\r	برگشت به اول خط (کم کاربرد تو ویندوز)
\b	بک‌اسپیس (کاراکتر قبلی رو حذف می‌کنه، اغلب نادیده گرفته میشه در ترمینال)

در مثال زیر این escape sequence ها استفاده شده

```
Animal:    "Cat"
Sound:     "Meow"
Info:      Cats love to sleep on \windows\ and chase 'mice'.
FunFact:   They say: "A cat has 9 lives."
Demo:      End
Typo:      Meow!
```

خروجی

```
fmt.Print("Animal:\t\"Cat\"\n")
fmt.Print("Sound:\t\"Meow\"\n")
fmt.Print("Info:\tCats love to sleep on \\windows\\ and chase 'mice'.\n")
fmt.Print("FunFact:\tThey say: \"A cat has 9 lives.\"\n")
fmt.Print("Demo:\tEnd\rMem\n")
fmt.Print("Typo:\tMeoww\b!\n")
```

توضیحات

- \t تب گذاشت (برای جدول‌بندی)
- \" برای نقل‌قول "Cat"
- \\ برای چاپ \windows\
- \r متن قبلی "Start" رو با "End\n" جایگزین کرد
- \b یکی از w ها رو حذف کرد

تابع fmt.Printf

به برنامه زیر توجه کن

```
var n string = "Rose" // name
var h float32 = 1.79  // height
var w float32 = 78.0  // weight
var bmi float32 = w / (h * h)
fmt.Println("Height of", n, "is", h, "m, weight is", w, "kg. So, the BMI is", bmi)
```

خروجی

```
Height of Rose is 1.79 m, weight is 78 kg. So, the BMI is 24.343811
```

در واقع داریم یه جمله‌ای رو تو خروجی چاپ می‌کنیم که داخلش 4 مقدار به‌صورت پارامتری داره وارد می‌شه: اسم، قد، وزن، و شاخص توده بدنی.

میشه گفت انگار یه قالب کلی (template) داریم و این مقادیر فقط می‌رن سر جاشون.

قالب کلی جمله‌مون:

```
Height of "name" is "x.x"m, weight is "x.x"kg. So, the BMI is "x.xxx"
```

کافیه که تو این قالب، اون مقادیر جایگزین بشن تا خروجی همون چیزی بشه که می‌خوایم.

ولی یه مشکلی هست...

این مدل نوشتن، مخصوصاً وقتی تعداد مقادیر زیاد بشه یا بخوای متن دقیق و خوش‌فرم بسازی، دردسر داره.

تابع fmt.Printf خیلی کمکمون می‌کنه چون می‌تونیم یه قالب کلی مشخص کنیم و بعد، مقادیر رو یکی‌یکی بهش بدیم تا خودش اون‌ها رو درست جایگزین کنه.

بازنویسی مثال با fmt.Printf

```
var n string = "Rose" // name
var h float32 = 1.79  // height
var w float32 = 78.0  // weight
var bmi float32 = w / (h * h)
fmt.Printf("Height of %s is %fm, weight is %fkg. So, the BMI is %f", b, h, w, bmi)
```

خروجی

```
Height of Rose is 1.790000m, weight is 78.000000kg. So, the BMI is 24.343811
```

تنها چیزی که باید تو این حالت بدونی، اینه که **هر پارامتر باید با چه format verb چاپ بشه** .
یعنی مثلاً برای رشته از %s استفاده کنی، برای عدد اعشاری از %f و برای عدد صحیح از %d

لیست Format verb های قابل استفاده

قابل استفاده برای تمام انواع داده

Format Verb	توضیح کاربرد	مثال خروجی
%v	مقدار پیش‌فرض (default format)	42, [1 2 3], map[a:1]
%+v	مثل %v ولی با نام فیلدها برای struct	{Name:Ali Age:30}
%#v	نمایش به صورت نحوی (Go-syntax)	[]int{1, 2, 3}
%T	نوع داده	int, []string
%%	علامت %	%

برای اعداد

Format Verb	توضیح	مثال خروجی
%d	عدد دهدهی (Decimal)	123
%b	باینری (Binary)	1111011
%o	اکتال (Octal)	173
%x	هگزادسیمال کوچک	7b
%X	هگزادسیمال بزرگ	7B
%c	کاراکتر یونیکد	A برای 65
%U	یونیکد	U+0041

برای رشته‌ها و بایت‌ها

Format Verb	توضیح	مثال خروجی
%s	رشته معمولی	hello
%q	رشته کوتیشن‌دار (quoted string)	"hello"
%x	نمایش رشته به صورت هگزادسیمال	68656c6c6f
%X	مثل بالا، با حروف بزرگ	68656C6C6F

برای اعداد اعشاری (float)

Format Verb	توضیح	مثال خروجی
%f	نمایش عدد با اعشار معمولی	3.141593
%e	نمای علمی با e	3.141593e+00
%E	نمای علمی با E	3.141593E+00
%g	کوتاه‌ترین بین %f و %e	3.14159 یا 3e+08
%G	مثل %g ولی با E	3.14159 یا 3E+08

برای مقدارهای بولی و اشاره‌گر

Format Verb	توضیح	مثال خروجی
%t	مقدار بولی	true یا false
%p	آدرس حافظه (pointer)	0xc00000e030

نمایش اطلاعات سوله‌های مختلف یک کارخانه

برای هر سوله اطلاعات زیر رو داریم:

- نام سوله
- نام مدیر
- حوزه‌ی فعالیت
- تعداد کارکنان (int)
- تعداد دستگاه‌ها (int)
- دمای محیط (float64)
- فعال یا غیرفعال بودن (bool)

و می‌خوایم اطلاعاتشون رو با fmt.Printf به‌صورت دقیق، فرمت‌شده، و مرتب چاپ کنیم.

```
// Warehouse 1
name1 := "Packaging"
manager1 := "Mr. Ahmadi"
activity1 := "Final product packaging"
workers1 := 24
machines1 := 8
temp1 := 28.6457
active1 := true

// Warehouse 2
name2 := "Injection"
manager2 := "Mrs. Karimi"
activity2 := "Plastic part injection"
workers2 := 15
machines2 := 12
temp2 := 31.0541
active2 := false

fmt.Println("Factory Warehouses Report")
fmt.Println("-----------------------------------------------------------------")
fmt.Printf("%-15s | %-14s | %-25s\n", "Warehouse", "Manager", "Activity")
fmt.Printf("%-15s | %-14s | %-25s\n", name1, manager1, activity1)
fmt.Printf("%-15s | %-14s | %-25s\n", name2, manager2, activity2)

fmt.Println("-----------------------------------------------------------------")
fmt.Printf("%-15s | Workers: %02d | Machines: %02d | Temp: %6.2f°C | Active?
%t\n", name1, workers1, machines1, temp1, active1)
fmt.Printf("%-15s | Workers: %02d | Machines: %02d | Temp: %6.2f°C | Active?
%t\n", name2, workers2, machines2, temp2, active2)
```

خروجی

```
------------------------------------------------------------
Warehouse       | Manager        | Activity
Packaging       | Mr. Ahmadi     | Final product packaging
Injection       | Mrs. Karimi    | Plastic part injection
------------------------------------------------------------
Packaging       | Workers: 24 | Machines: 08 | Temp:  28.65°C | Active? true
Injection       | Workers: 15 | Machines: 12 | Temp:  31.05°C | Active? false
```

نکات آموزشی:

پد کردن (Padding):

- %02d یعنی عدد همیشه با دو رقم نمایش داده می‌شه. اگه یک‌رقمی بود، با صفر پر می‌شه.
  مثلا 08, 09, 12

کنترل دقت اعشار:

- %6.2f یعنی عدد اعشاری با 6 کاراکتر جای می‌گیره که 2 رقم اعشار داره.
  (قبلش فاصله اضافه می‌کنه تا ستون‌ها مرتب بمونن)

ترازبندی رشته‌ها:

- %–20s رشته رو چپ‌چین می‌کنه و اگه کوتاه بود، با فاصله پر می‌کنه.

ترکیب انواع Verb

- %t برای bool
- %s برای رشته
- %d عدد صحیح
- %f اعشاری با نقطه

جمع‌بندی

با سه تابع مهم از کتابخونه‌ی fmt آشنا شدیم که برای چاپ خروجی در زبان Go استفاده می‌شن:

1. fmt.Print
2. fmt.Println
3. fmt.Printf

هر کدوم از این توابع رفتار متفاوتی در نحوه‌ی چاپ خروجی دارن. بسته به شرایط مختلف، ممکنه یکی از اون‌ها مناسب‌تر باشه. حالا بیایم یه جمع‌بندی ساده از تفاوت‌هاشون داشته باشیم:

تفاوت اصلی این سه تابع

- fmt.Print همه‌ی مقادیر رو **پشت سر هم** بدون فاصله چاپ می‌کنه و در پایان هم **خط عوض نمی‌کنه**.
- fmt.Println **بین مقادیر فاصله می‌ذاره** و در انتها هم **می‌ره خط بعد**.
- fmt.Printf کنترل کامل با شماست. قالب رشته‌ای (template) رو خودت تعیین می‌کنی . **نه فاصله می‌ذاره، نه می‌ره خط بعد**، مگر اینکه خودت \n بذاری.

جدول مقایسه‌ی سریع

تابع	بین آرگومان‌ها فاصله می‌ذاره؟	آخرش \n می‌ذاره؟
fmt.Print	نمی‌ذاره	نمی‌ذاره
fmt.Println	می‌ذاره	می‌ذاره
fmt.Printf	خودت فرمت رو کنترل می‌کنی	نمی‌ذاره

توابع پرکاربرد کتابخانه strings

بعد از تسلط به مفاهیم اولیه‌ی نوع داده‌ی string و یاد گرفتن روش‌های مختلف چاپ کردن با توابع fmt.Print, fmt.Println و fmt.Printf حالا وقتشه که بریم سراغ یکی از مهم‌ترین ابزارها برای کار با رشته‌ها

این کتابخونه برای کار با رشته‌ها طراحی شده و به ما اجازه می‌ده که روی stringها انواع عملیات‌های مفید و کاربردی انجام بدیم، مثل:

- پیدا کردن یه زیررشته داخل رشته‌ی دیگه
- تبدیل حروف بزرگ به کوچک یا برعکس
- حذف فاصله‌های اضافی
- چک کردن اینکه آیا رشته‌ای با یه عبارت خاص شروع یا تموم می‌شه
- و کلی کار دیگه که توی پروژه‌های واقعی همیشه به درد می‌خوره

مزیت استفاده از strings

شاید تو بعضی زبان‌های دیگه لازم باشه برای این جور کارها حلقه بنویسی یا خودت دستی چک کنی، ولی Go با کتابخونه‌ی strings خیلی از این کارها رو برات آماده کرده. یعنی فقط کافیه تابع مورد نظر رو صدا بزنی، و رشته‌ات همونطوری که می‌خوای تغییر کنه یا بررسی بشه.

چند تا مثال از کاربردهای strings

- strings.Contains(str, substr) بررسی وجود یه زیررشته داخل رشته
- strings.ToLower(str) تبدیل همه حروف به کوچک
- strings.HasPrefix(str, "start") بررسی اینکه رشته با یه عبارت خاص شروع شده
- strings.ReplaceAll(str, "old", "new") جایگزینی همه‌ی بخش‌های خاص با متن جدید
- strings.TrimSpace(str) حذف فاصله‌های اضافی از ابتدا و انتهای رشته

دسته‌بندی و معرفی توابع پرکاربرد strings

توابع کتابخانه‌ی strings زیادن، ولی بعضی‌هاشون انقدر پرکاربردن که توی اکثر پروژه‌ها دیده می‌شن. برای اینکه بهتر یادشون بگیریم، اومدیم اون‌ها رو به چند **گروه کاربردی** تقسیم کردیم

جستجو و بررسی

تابع	نوع ورودی	نوع خروجی	توضیح	مثال
Contains(s, substr)	string, string	bool	بررسی اینکه substr داخل s وجود دارد	strings.Contains("hello", "ll") → true
HasPrefix(s, prefix)	string, string	bool	بررسی شروع شدن رشته با prefix	strings.HasPrefix("golang", "go") → true
HasSuffix(s, suffix)	string, string	bool	بررسی پایان یافتن رشته با suffix	strings.HasSuffix("file.txt", ".txt") → true
Index(s, substr)	string, string	int	موقعیت اولین وقوع substr در s (نداشته باشه: -1)	strings.Index("banana", "na") → 2
LastIndex(s, substr)	string, string	int	موقعیت آخرین وقوع substr	strings.LastIndex("banana", "na") → 4

جایگزینی و تکرار

تابع	نوع ورودی	نوع خروجی	توضیح	مثال
Replace(s, old, new, n)	string, string, string, int	string	جایگزینی old با new تا حداکثر n بار	strings.Replace("pop", "p", "m", 1) → "mop"
ReplaceAll(s, old, new)	string, string, string	string	جایگزینی تمام oldها با new	strings.ReplaceAll("pop", "p", "m") → "mom"
Repeat(s, count)	string, int	string	تکرار s به تعداد countبار	strings.Repeat("go", 3) → "gogogo"

حذف و برش

تابع	نوع ورودی	نوع خروجی	توضیح	مثال
TrimSpace(s)	string	string	حذف فاصله‌های ابتدا و انتهای رشته	strings.TrimSpace(" hi ") → "hi"
Trim(s, cutset)	string, string	string	حذف تمام کاراکترهای cutset از ابتدا و انتهای رشته	strings.Trim("!!wow!!", "!") → "wow"
TrimPrefix(s, prefix)	string, string	string	حذف prefix از ابتدا (اگه وجود داشته باشه)	strings.TrimPrefix("unfair", "un") → "fair"
TrimSuffix(s, suffix)	string, string	string	حذف suffix از انتها	strings.TrimSuffix("file.txt", ".txt") → "file"

تغییر حروف

تابع	نوع ورودی	نوع خروجی	توضیح	مثال
ToLower(s)	string	string	تبدیل به حروف کوچک	strings.ToLower("Tree") → "tree"
ToUpper(s)	string	string	تبدیل به حروف بزرگ	strings.ToUpper("danger") → "DANGER"
Title(s)	string	string	تبدیل اولین حرف هر کلمه به بزرگ (deprecated)	strings.Title("run away") → "Run Away"

تقسیم و چسباندن رشته‌ها

تابع	نوع ورودی	نوع خروجی	توضیح	مثال
Split(s, sep)	string, string	[]string	تقسیم رشته با جداکننده	strings.Split("O|N", "|") → ["O", “N”]
Join(slice, sep)	[]string, string	string	چسباندن عناصر slice با sep	strings.Join([]string{"O", "N"}, "-") → "O-N"
Fields(s)	string	[]string	جدا کردن کلمات بر اساس فاصله	strings.Fields("Hi There") → ["Hi ", " There"]

نکته نهایی

همه‌ی توابع بالا، رشته‌ها رو تغییر نمی‌دن!
در واقع چون در زبان Go رشته‌ها **immutable** هستن، این توابع همیشه یه رشته‌ی جدید برمی‌گردونن و رشته‌ی اصلی رو تغییر نمی‌دن. پس حواست باشه که نتیجه‌ی توابع رو **دوباره در یه متغیر بریزی یا مستقیم استفاده کنی.**

مثال: دستورالعمل نقاشی خونه

فرض کن یه متن راهنما داریم که توضیح می‌ده چطور دیوارهای خونه رو رنگ کنیم.
این متن چند جمله داره که با نقطه (.) از هم جدا شدن و هر جمله با یه فاصله‌ی اضافی (space) شروع می‌شه.

حالا می‌خوایم چند عملیات روی این متن انجام بدیم:

1. رنگ **gray** مناسب نیست؛ اون رو با **whitesmoke** جایگزین کنیم.
2. هر جا عبارت **white** یا **whitesmoke** اومده، اون رو به حروف بزرگ (uppercase) تبدیل کنیم.
3. جمله‌ها رو با تابع Split از هم جدا کنیم.
4. فاصله‌ی اضافی اول و آخر هر جمله رو با TrimSpace حذف کنیم.
5. در نهایت با Contains بررسی کنیم آیا رنگ‌های **pink** و **yellow** تو متن استفاده شدن یا نه.

```
text := "Paint the living room in gray. Paint the kitchen in yellow. Use white for the ceiling. Do not use pink on any wall."

// 1. جایگزینی رنگ gray با whitesmoke

text = strings.ReplaceAll(text, "gray", "whitesmoke")

// 2. تبدیل white و whitesmoke به حروف بزرگ

text = strings.ReplaceAll(text, "whitesmoke", strings.ToUpper("whitesmoke"))

text = strings.ReplaceAll(text, "white", strings.ToUpper("white"))

// 3. جدا کردن جمله‌ها

sentences := strings.Split(text, ".")

// 4. پاک‌کردن فاصله‌ها از اول و آخر هر جمله

fmt.Println("📋 Sentences after cleaning:")

for _, sentence := range sentences {

	trimmed := strings.TrimSpace(sentence)

	if trimmed != "" {

		fmt.Println("-", trimmed)

	}

}

// 5. بررسی وجود رنگ orange و pink

fmt.Println("\n🔍 Color Checks:")

if strings.Contains(text, "pink") {

	fmt.Println("✅ 'pink' is mentioned in the text.")

} else {

	fmt.Println("❌ 'pink' is not mentioned.")

}

if strings.Contains(text, "orange") {

	fmt.Println("✅ 'orange' is mentioned in the text.")

} else {

	fmt.Println("❌ 'orange' is not mentioned.")

}
```

خروجی

```
📋 Sentences after cleaning:
- Paint the living room in WHITESMOKE
- Paint the kitchen in yellow
- Use WHITE for the ceiling
- Do not use pink on any wall

🔍 Color Checks:
✅ 'pink' is mentioned in the text.
❌ 'orange' is not mentioned.
```

ReplaceAll(text, "gray", "whitesmoke")

تو این مرحله، هرجا که کلمه‌ی "gray" با "whitesmoke" جایگزین می‌کنیم. یعنی رنگ "gray" کاملاً حذف و به "whitesmoke" تغییر داده شده.

ToUpper("whitesmoke")

هر جا کلمه "whitesmoke" و همچنین "white" ظاهر شده بود به حروف بزرگ تبدیل میشه تا در متن برجسته‌تر و قابل توجه‌تر نمایش داده بشه. (این برای تأکید بیشتر روی این رنگ‌هاست).

Split(text, ".")

متن بر اساس نقطه تقسیم میشه؛ یعنی جمله‌ها از هم جدا میشن. این باعث میشه بتونیم هر جمله رو جداگانه بررسی و پردازش کنیم و عملیات بعدی راحت‌تر انجام بشه.

TrimSpace(sentence)

بعد از جدا کردن جملات، معمولاً فاصله‌های اضافی ابتدای هر جمله (که بعد از نقطه ظاهر میشه) حذف میشه تا متن مرتب‌تر و تمیزتر بشه.

Contains(text, "pink") و Contains(text, "orange")

این دو تابع چک می‌کنن آیا کلمات "pink" یا "orange" در متن وجود دارند یا خیر. این قابلیت وقتی کاربرد داره که بخوایم بررسی کنیم که رنگ خاصی استفاده شده یا نه، مثلاً برای فیلتر کردن یا هشدار دادن درباره رنگ‌های ممنوعه.

ترکیب توابع کتابخانه strings

میتونیم توابع کتابخونه strings رو با هم ترکیب کنیم و چندتا کار رو پشت سر هم خیلی راحت انجام بدیم.

مثلاً اول یه متن رو می‌تونیم همه‌اش رو کوچیک کنیم، بعد ببینیم یه کلمه خاص توش هست یا نه، یا حتی کلمه‌ای رو با چیز دیگه‌ای جایگزین کنیم. اینطوری کد تمیزتر و ساده‌تر میشه و فهمیدنش راحت‌تر.

حالا فرض کن می‌خوایم یه برنامه درست کنیم که چک کنه آیا تو متن وارد شده یه کلمه ناسزا هست یا نه. چون ممکنه کاربر اون کلمه رو با حروف بزرگ، کوچیک یا ترکیبی نوشته باشه، ما اول همه متن رو کوچیک می‌کنیم و بعد دنبال اون کلمه می‌گردیم. اینجوری مطمئنیم فرقی نمی‌کنه چطوری کلمه رو نوشته!

```
text := "I can’t believe this shit happened again!"
badWord := "shit"

// بررسی وجود کلمه ناسزا در متن
if strings.Contains(strings.ToLower(text), badWord) {
    fmt.Println("The text contains the bad word.")
} else {
    fmt.Println("The text does not contain the bad word.")
}
```

خروجی

```
The text contains the bad word.
```

- اول متن و کلمه‌ای که می‌خوایم توش بگردیم (مثلاً همون کلمه ناسزا) رو تعریف می‌کنیم.
- بعد با strings.ToLower هر دوتا رو کوچیک می‌کنیم تا فرق بین بزرگ و کوچیک بودن حروف رو نادیده بگیریم.
- بعدش با strings.Contains چک میکنیم که آیا اون کلمه تو متن هست یا نه.
- آخرش هم نتیجه رو چاپ می‌کنیم.

این مثال ساده نشون میده چطور با ترکیب چند تا تابع ساده از strings می‌تونیم کاری مثل جستجو بدون توجه به بزرگ یا کوچیک بودن حروف رو راحت انجام بدیم.

مشکل نگارش متن‌های طولانی و چندخطی

فرض کن می‌خوای یه متن چند خطی یا یه رشته‌ای که داخلش کاراکترهای خاص مثل \n یا \t وجود داره بنویسی. معمولاً ما رشته‌ها رو با کوتیشن دوتایی (") تعریف می‌کنیم. اما اگر رشته شامل خط جدید(newline) ، تب (tab) یا حتی علامت‌های نقل قول باشه، ممکنه کد پیچیده یا ناخوانا بشه و مجبور باشی کاراکترهای escape مثل \n یا \" استفاده کنی.

مثلا در زیر کد مربوط به چاپ دستور پخت کیکو میبینی

```
recipe := "Cake Recipe:\nStep 1: Preheat the oven to 350°F.\nStep 2: \"Mix\" the flour, sugar, and eggs.\nStep 3:\tPour the batter into a pan.\nStep 4: Bake for 30 minutes."

fmt.Println(recipe)
```

خروجی

```
Cake Recipe:

Step 1: Preheat the oven to 350°F.

Step 2: "Mix" the flour, sugar, and eggs.

Step 3:    Pour the batter into a pan.

Step 4: Bake for 30 minutes.
```

دیدیم که خروجی چقدر جذاب و خوانا هست اما وقتی خواستیم برنامشو بنویسیم یکم سخت و پیچیده بود حتی زمانی که بخوایم تغییرش بدیم خوندن کدش کمی سخته

میشه به طور کلی گفت نوشتن رشته با استفاده از “ چنین ایراداتی رو داره

نیاز به استفاده از کاراکترهای escape

- برای درج خط جدید باید از \n استفاده کنیم.
- برای تب باید \t بنویسیم.
- برای درج علامت نقل قول داخل رشته باید از \" استفاده کنیم.

این موارد باعث می‌شود رشته‌ها طولانی‌تر و ناخواناتر شوند.

خوانایی کمتر در رشته‌های چند خطی

- اگر بخواهیم متن چند خطی داشته باشیم، باید هر خط را با \n جدا کنیم که خوانایی متن را کم می‌کند.

خطاهای احتمالی هنگام نوشتن escape ها

- احتمال فراموش کردن escape یا نوشتن اشتباه آن‌ها وجود دارد که منجر به خطا یا خروجی نادرست می‌شود.

عدم امکان نوشتن متن خام (Raw)

- همه کاراکترها مثل \n، \t و ... به صورت escape تفسیر می‌شوند و نمی‌توان متن خام را بدون پردازش نوشت.

در Go می‌توانیم رشته‌ها را با (`) backtick تعریف کنیم که به آن raw string literal می‌گویند. در این حالت:

- رشته دقیقاً همان چیزی است که بین backtick قرار دارد، بدون نیاز به escape کردن
- می‌توانیم متن‌های چند خطی را راحت و خوانا بنویسیم.
- کاراکترهای خاص مثل \n رشته عادی در نظر گرفته می‌شوند، نه به عنوان escape.

بازنویی مثال قبل با `

```
recipe := `Cake Recipe:
Step 1: Preheat the oven to 350°F.
Step 2: "Mix" the flour, sugar, and eggs.
Step 3:      Pour the batter into a pan.
Step 4: Bake for 30 minutes.`
fmt.Println(recipe)
```

خروجی

```
Cake Recipe:
Step 1: Preheat the oven to 350°F.
Step 2: "Mix" the flour, sugar, and eggs.
Step 3:    Pour the batter into a pan.
Step 4: Bake for 30 minutes.
```

خروجی دقیقاً مثل قبله ولی کد خیلی خواناتر و ساده‌تر شده چون:

- نیاز به نوشتن \n و \" و \t نداریم.
- متن کاملاً شبیه به آنچه چاپ می‌شود داخل کد هست.

کی از " و کی از ` استفاده کنیم؟

شرایط و ویژگی	استفاده از کوتیشن دوتایی (")	استفاده از backtick (`)
متن تک خطی ساده	✓	اما معمولاً از "استفاده می‌شود
متن چند خطی	نیاز به \n و escape	مناسب برای متن چند خطی
نیاز به کاراکترهای escape	باید بنویسیم	نیازی به escape نیست
درج متن شامل علامت نقل قول	با \" escape	بدون نیاز به escape
خوانایی و سادگی نوشتن متن طولانی	✗	✓
متن خام بدون پردازش escape	✗	✓
کاربردهای معمول	رشته‌های کوتاه و معمول	متن چند خطی، کدهای HTML/JSON، اسکریپت‌ها

جمع‌بندی

- وقتی رشته شما شامل چند خط یا کاراکترهای خاص باشه که نیاز به escape دارند، استفاده از (`) backtick باعث میشه کد ساده‌تر، تمیزتر و خواناتر بشه.
- درواقع backtick رشته‌ها را به صورت خام (raw) نگه میداره و هیچ کاراکتری رو escape نمیکنه.
- این قابلیت مخصوصاً برای نوشتن متن‌های طولانی، کدهای HTML، JSON یا اسکریپت‌های چند خطی خیلی کاربردیه.

تمرین 1: پیدا کردن ایمیل

برنامه‌ای بنویس که یه متن از ورودی بگیره و بررسی کنه توش آدرس ایمیل وجود داره یا نه.

برای سادگی، فقط بررسی کن که آیا متن شامل کاراکتر @ هست و بعدش . داره یا نه.

ورودی:

```
Please get in touch with me by sending an email to support@example.com
```

خروجی:

```
An email has been found: support@example.com
```

تمرین 2: تمیزکاری جمله برای جستجو ساده تر

جمله ای در برنامه ذخیره شده است، همه کاراکترهای نقطه، ویرگول، علامت تعجب، سؤال و فاصله رو حذف کن، حروف رو کوچیک کن، و جمله رو چاپ کن.

ورودی

```
Yesterday, I bought some apples, oranges, and bananas – they were all fresh and cheap.
```

خروجی

```
yesterday i bought some apples oranges and bananas they were all fresh and cheap
```

تمرین 3: چک کردن آینه ای بودن

برنامه‌ای بنویس که بررسی کنه متن ورودی آینه ای (palindrome) هست یا نه (یعنی از دو طرف خونده بشه فرقی نکنه)

شرط خاص:

- برنامه باید حروف بزرگ و کوچیک رو برابر در نظر بگیره.
- فاصله و علامت‌گذاری‌ها مهم نیستند (برای ساده‌سازی می‌تونی اون‌ها رو پاک کنی یا نادیده بگیری).

ورودی

```
Madam, in Eden, I'm Adam
```

خروجی

```
It is Palindrome.
```

تمرین 4: شمارش تعداد کلمات خاص

برنامه‌ای بنویس که توی یه متن بلند، تعداد تکرار کلمه‌ای مثل "love" رو بشمره.

ویژگی:

- حروف بزرگ و کوچیک مهم نباشه.

ورودی:

```
I love you. Love is everything. LOVE is powerful.
```

خروجی:

```
‘love’ has been repeated for 3 time(s)
```

تمرین 5: سانسور کلمه غیر مجاز

برنامه‌ای بنویس متنی که در برنامه ذخیره شده رو بررسی کنه، دنبال کلمه‌ی shit بگرده و اونو با *** جایگزین کنه. در نهایت، متن جدید رو چاپ کن.

ممکنه در متن کلمه shit به شکل ShIT یا SHIT یا هر ترکیب دیگه‌ای از حروف بزرگ و کوچیک نوشته شده باشه.

برنامه باید به‌صورت غیرحساس به حروف (case-insensitive) عمل کنه و همه‌ی این حالت‌ها رو شناسایی کنه.

نکته:
برنامه فقط در حالتی که بین حروف یه کاراکتر دیگه (مثل s-h-i-t) استفاده شده باشه، نمی‌تونه تشخیص بده، که فعلاً اشکالی نداره. همین که همه‌ی حالت‌های shit با حروف مختلف رو تشخیص بده کافیه.