



# Basic Types in Go



bool

uint

float32

int8

uint8/byte

float64

int

int16

uint16

complex64

uint32

int32/rune

complex128

uint64

int64

uintptr

string

# BASIC TYPES

bool	uint	float32
int	uint8/byte	float64
int8	uint16	complex64
int16	uint32	complex128
int32/rune	uint64	string
int64	uintptr	

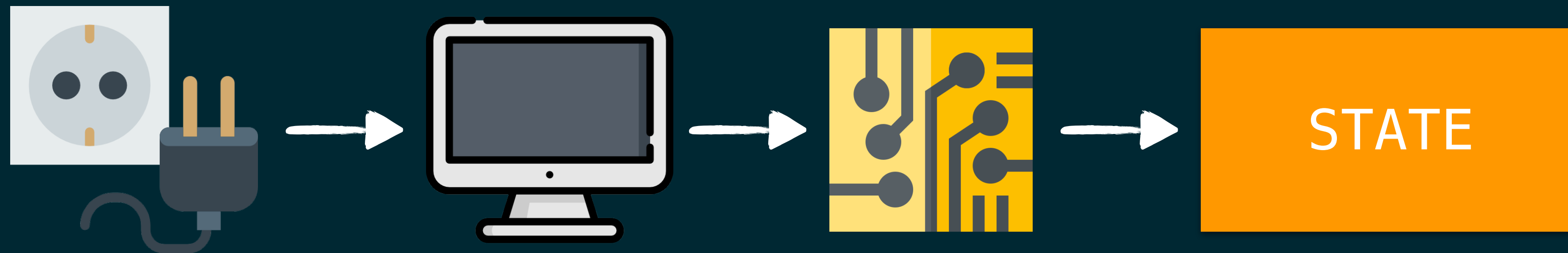




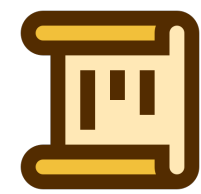
# BOOL VALUES



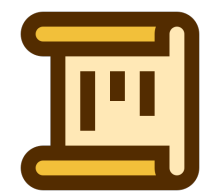
true	1	ON
false	0	OFF



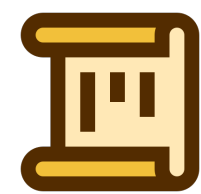
# BASE 2



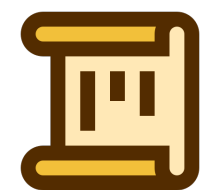
The number of values -  $\{0, 1\}$



## What is a **Byte** - *8 bits*



## What is a bit - one of the values (**0 or 1**)



## How to represent a **Byte**:

$10 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

# BOOL SIZE

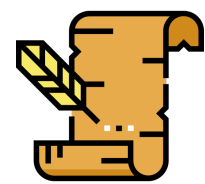


1B

8b

$2 * 2^{8-1} - 1$

0–255



**1 Byte** is the smallest addressable unit (**not 1 bit**)

# BOOL OPERATIONS



## operators & punctuation

+	&	+=	&=	&&	==	!=	(	)
-		-=	=		<	<=	[	]
*	^	*=	^=	<-	>	>=	{	}
/	<<	/=	<<=	++	=	::=	,	;
%	>>	%=	>>=	--	!	...	.	:
			&^=					





# INT VALUES



8

$$-2^7 = -128 \iff 2^7 - 1 = 127$$

16

$$-2^{15} = -32,768 \iff 2^{15} - 1 = 32,767$$

32

$$-2^{31} = -2,147,483,648 \iff 2^{31} - 1 = 2,147,483,647$$

64

$$-2^{63} = -9,223,372,036,854,775,808 \iff 2^{63} - 1 = 9,223,372,036,854,775,807$$

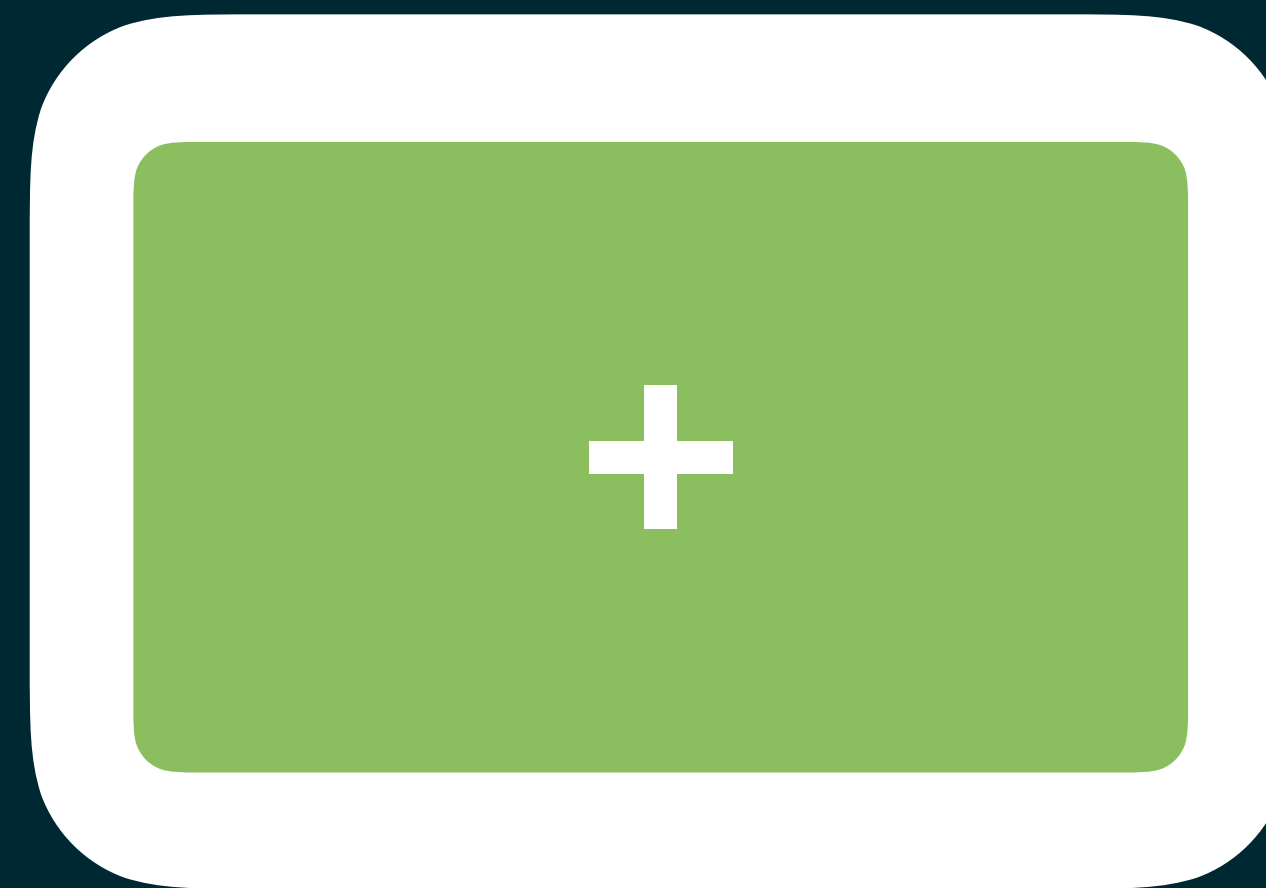


# WHAT IS UINT?

int



uint



# UINT VALUES

8

$$0 \leq 2^7 * 2 - 1 = 255$$

16

$$0 \leq 2^{15} * 2 - 1 = 65,535$$

32

$$0 \leq 2^{31} * 2 - 1 = 4,294,967,295$$

64

$$0 \leq 2^{63} * 2 - 1 = 18,446,744,073,709,551,615$$



# INT/UINT SIZE

int	uint	4/8B	32/64b
int8	uint8	1B	8b
int16	uint16	2B	16b
int32	uint32	4B	32b
int64	uint64	8B	64b



# INT/UINT OPERATIONS



## operators & punctuation

+	&	+=	&=	&&	==	!=	(	)
-		-=	=		<	<=	[	]
*	^	*=	^=	<-	>	>=	{	}
/	<<	/=	<<=	++	=	::=	,	;
%	>>	%=	>>=	--	!	...	.	:
			&^=					





# FLOAT VALUES



32

$0 \leq 2^{31} - 1 = 4,294,967,295$

64

$0 \leq 2^{63} - 1 = 18,446,744,073,709,551,615$

# FLOAT SIZE



float32

4B

32b

float64

8B

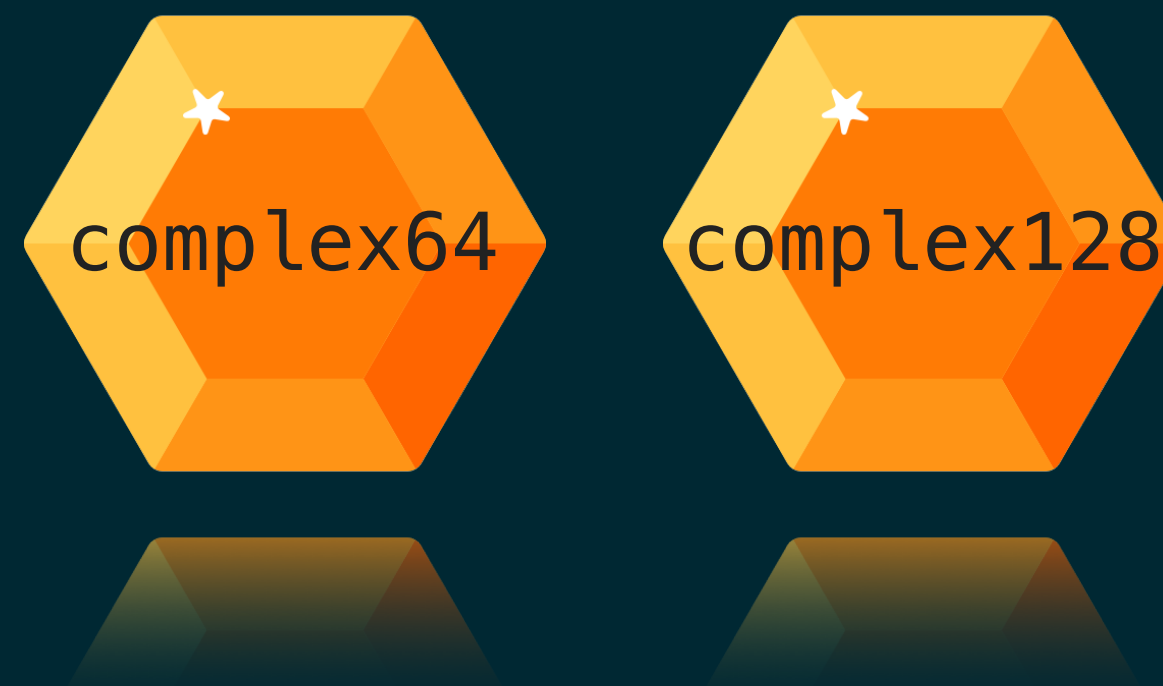
64b

# FLOAT OPERATIONS



## operators & punctuation

+	&	+=	&=	&&	==	!=	(	)
-		-=	=		<	<=	[	]
*	^	*=	^=	<-	>	>=	{	}
/	<<	/=	<<=	++	=	::=	,	;
%	>>	%=	>>=	--	!	...	.	:
			&^=					



# COMPLEX VALUES



64

$0 \leq 2^{63} * 2 - 1 = 18,446,744,073,709,551,615$

128

$0 \leq 2^{127} * 2 - 1 = 3.4028237e+38$

# WHAT IS COMPLEX?



A complex number is a number of form  $a+bi$ , where **a** and **b** are **real numbers** and **i** is the solution of the equation  $x^2=-1$



# COMPLEX SIZE



complex64

4B

32b

complex128

8B

64b

# COMPLEX OPERATIONS



## operators & punctuation

+	&	+=	&=	&&	==	!=	(	)
-		-=	=		<	<=	[	]
*	^	*=	^=	<-	>	>=	{	}
/	<<	/=	<<=	++	=	::=	,	;
%	>>	%=	>>=	--	!	...	.	:
			&^=					



# TYPE ALIASES



Type **A**



Type **B**



Type **A**





# RUNES ARE JUST NUMBERS



`rune` = `int32`

byte



# BYTES ARE JUST NUMBERS

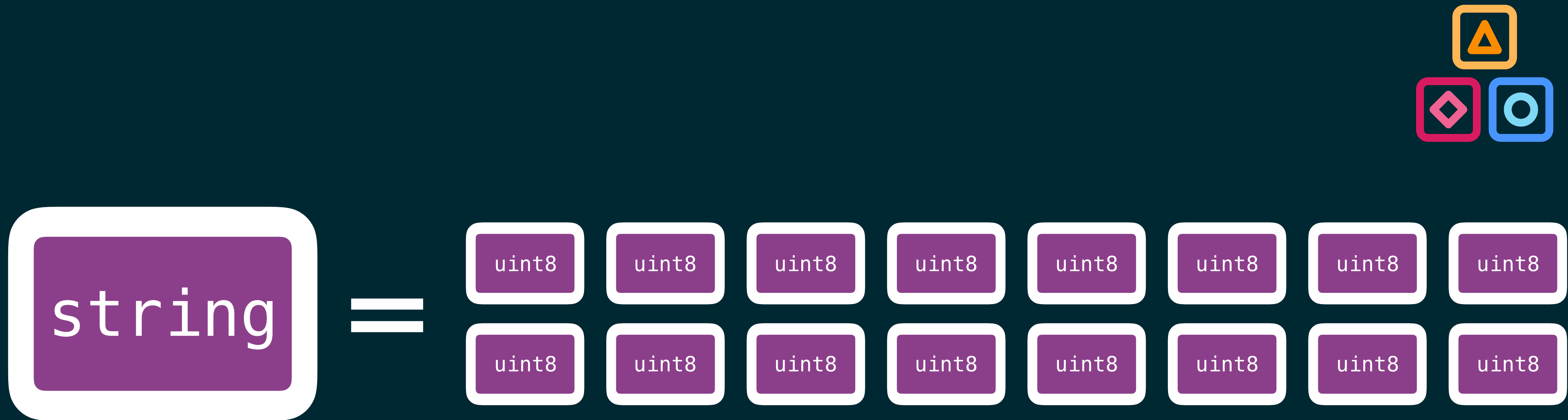


`byte` = `uint8`

string



# STRING = SEQUENCE OF BYTES



# ASCII & UTF8



1B



1-4B





# STRING VALUES



Any UTF-8 sequence



# STRING SIZE



```
type _string struct {  
    elements *byte // 4/8B  
    len int // 4/8B  
}
```

string

8/16B

64/128b

# STRING OPERATIONS



## operators & punctuation

+	&	+=	&=	&&	==	!=	(	)
-		-=	=		<	<=	[	]
*	^	*=	^=	<-	>	>=	{	}
/	<<	/=	<<=	++	=	::=	,	;
%	>>	%=	>>=	--	!	...	.	:
			&^=					

# STRING FACTS

- **Read only** sequence of bytes (`[]byte`)
- 2 styles: **double quote**(interpreted) or **back quote** (raw)
- Strings are **comparable** & can use **comparison operations**
- When comparing, the **underlying bytes** are **compared**
- Destination **string variable** and **source string value** share the same **underlying bytes**
- `[]byte` & `[]rune` interchangeably convert to **string** and vice versa



"string"

`string`

uintptr



# UINTPTR SIZE



uintptr

8/16B

64/128b

# UINTPTR VALUES



32

$-2^{31} = -2,147,483,648 \iff 2^{31} - 1 = 2,147,483,648$

64

$-2^{63} = -9,223,372,036,854,775,808 \iff 2^{63} - 1 = 9,223,372,036,854,775,807$

# UINTPTR OPERATIONS



## operators & punctuation

+	&	+=	&=	&&	==	!=	(	)
-		-=	=		<	<=	[	]
*	^	*=	^=	<-	>	>=	{	}
/	<<	/=	<<=	++	=	::=	,	;
%	>>	%=	>>=	--	!	...	.	:
			&^=					



# UINTPTR FACTS

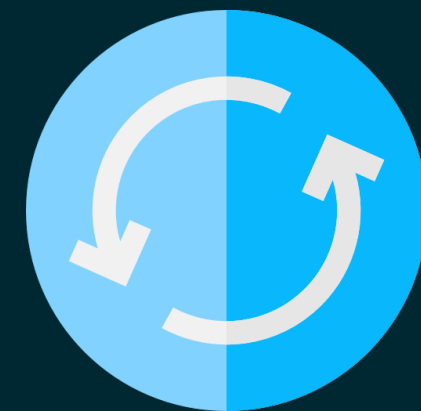


- **Integer** type that is **large enough** to hold **any pointer** bit pattern
- Supports **pointer arithmetic operations**
- A uintptr is an **integer**, not a **reference**
- **Converting unsafe.Pointer** to a **uintptr** creates an **integer value**
- Even if a **uintptr** holds the **address** of some **object**, the **GC will not update** that **uintptr's value** if the object moves

# POINTERS ARE INTERCHANGEABLE



`uintptr`



`unsafe.Pointer`

# WHAT IS UNSAFE POINTER



- **Pointer** represents a **pointer to** an **arbitrary type**
- Allows the program to **read** and **write arbitrary memory**
- A **pointer value** of any type can be converted to a **Pointer** and **vice versa**
- An **uintptr** can be converted to a **Pointer** and **vice versa**

# PLATFORM DEPENDENT TYPES

`int`

`uint`

8/16B

`uintptr`

`string`

64/128b



## QUICK EXERCISE

```
s := []byte("你 好")
if len(s) == 3 {
    fmt.Println("3B")
} else {
    fmt.Println("7B")
}
```

