

A string is a series of characters, such as "hello, world" or "albatross".

The Swift **String** type is used to represent a string.

A computer only understands numbers.

One bit, can be on or off.

0

1

So with one bit, you can represent two values.

If you have 2 bits:

00 => *decimal 0*

01 => *decimal 1*

10 => *decimal 2*

11 => *decimal 3*

With two bits, you can represent four values.

If you have 3 bits:

<i>binary</i>	<i>decimal</i>
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

With three bits, you can represent eight values.

If you have 4 bits:

<i>binary</i>	<i>decimal</i>	<i>hexadecimal</i>
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

With four bits, you can represent sixteen values.

Binary

The base-2 numeral system uses only two values - zero and one.

Number Systems

Base 10

	<div>7</div>	<div>0</div>	<div>3</div>	<div>2</div>
value of column →	10^3 1000	10^2 100	10^1 10	10^0 1
$(7 \times 1000) + (0 \times 100) + (3 \times 10) + (2 \times 1) = 7032$				

“decimal”

Number Systems

Base 2

	<div>1</div>	<div>0</div>	<div>1</div>	<div>1</div>
value of column →	2^3 8	2^2 4	2^1 2	2^0 1

$$(1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) = 11$$

“binary”

Number Systems

Base 16

	A	0	F	2
value of column →	16^3 4096	16^2 256	16^1 16	16^0 1

$$(10 \times 4096) + (0 \times 256) + (15 \times 16) + (2 \times 1) = 41202$$

“hexadecimal”

8 bits
256 values

0-15	16-31	32-47	48-63	64-79	80-95	96-111	112-127
00000000	00010000	00100000	00110000	01000000	01010000	01100000	01110000
00000001	00010001	00100001	00110001	01000001	01010001	01100001	01110001
00000010	00010010	00100010	00110010	01000010	01010010	01100010	01110010
00000011	00010011	00100011	00110011	01000011	01010011	01100011	01110011
00000100	00010100	00100100	00110100	01000100	01010100	01100100	01110100
00000101	00010101	00100101	00110101	01000101	01010101	01100101	01110101
00000110	00010110	00100110	00110110	01000110	01010110	01100110	01110110
00000111	00010111	00100111	00110111	01000111	01010111	01100111	01110111
00001000	00011000	00101000	00111000	01001000	01011000	01101000	01111000
00001001	00011001	00101001	00111001	01001001	01011001	01101001	01111001
00001010	00011010	00101010	00111010	01001010	01011010	01101010	01111010
00001011	00011011	00101011	00111011	01001011	01011011	01101011	01111011
00001100	00011100	00101100	00111100	01001100	01011100	01101100	01111100
00001101	00011101	00101101	00111101	01001101	01011101	01101101	01111101
00001110	00011110	00101110	00111110	01001110	01011110	01101110	01111110
00001111	00011111	00101111	00111111	01001111	01011111	01101111	01111111
128-143	144-159	160-175	176-191	192-207	208-223	224-239	240-255
10000000	10010000	10100000	10110000	11000000	11010000	11100000	11110000
10000001	10010001	10100001	10110001	11000001	11010001	11100001	11110001
10000010	10010010	10100010	10110010	11000010	11010010	11100010	11110010
10000011	10010011	10100011	10110011	11000011	11010011	11100011	11110011
10000100	10010100	10100100	10110100	11000100	11010100	11100100	11110100
10000101	10010101	10100101	10110101	11000101	11010101	11100101	11110101
10000110	10010110	10100110	10110110	11000110	11010110	11100110	11110110
10000111	10010111	10100111	10110111	11000111	11010111	11100111	11110111
10001000	10011000	10101000	10111000	11001000	11011000	11101000	11111000
10001001	10011001	10101001	10111001	11001001	11011001	11101001	11111001
10001010	10011010	10101010	10111010	11001010	11011010	11101010	11111010
10001011	10011011	10101011	10111011	11001011	11011011	11101011	11111011
10001100	10011100	10101100	10111100	11001100	11011100	11101100	11111100
10001101	10011101	10101101	10111101	11001101	11011101	11101101	11111101
10001110	10011110	10101110	10111110	11001110	11011110	11101110	11111110
10001111	10011111	10101111	10111111	11001111	11011111	11101111	11111111

An ***unsigned*** 8-bit integer can store values 0 to 255.

An ***signed*** 8-bit integer can store values -128 to 127.

Unsigned 8-bit number

0 0 1 0 0 1 1 0

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

$$(1 \times 32) + (1 \times 4) + (1 \times 2) = 38$$

(or 26 hexadecimal)

Unsigned 8-bit number

1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

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

$$(1 \times 128) + (1 \times 32) + (1 \times 4) + (1 \times 2) = 166$$

(or A6 hexadecimal)

Signed 8-bit number

two's complement

1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

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

$$(1 \times 32) + (1 \times 4) + (1 \times 2) - (1 \times 128) = -90$$

How to represent a character, using a number.

Decide on a “character encoding” - a table of codes in which you can identify each character with a number. For example:

65 represents 'A'

66 represents 'B'

67 represents 'C'

and so on ...

1960 - ASCII

American Standard Code for Information Interchange

Originally based on telegraph codes, and only used 7-bit values. Some of the characters made sense in the era of teletype machines, but are archaic today - for example Bell, Form Feed, Carriage Return

Originally based on the English alphabet, ASCII encodes 128 specified characters into seven-bit integers. The characters encoded are numbers 0 to 9, lowercase letters a to z, uppercase letters A to Z, basic punctuation symbols, control codes that originated with Teletype machines, and a space.

ASCII

Character set [\[edit \]](#)

Legend:

- Alphabetic
- Control character
- Numeric digit
- Punctuation
- Extended punctuation
- Graphic character
- International
- Undefined

ASCII (1977/1986)																
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1_	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2_	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	006A	006B	006C	006D	006E	006F
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7_	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	007F
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

1987 - Unicode

A modern attempt at consistent encoding and representation of text.

Not US centric. The latest version (9) contains more than 128,000 characters, from 135 scripts.

Unicode has multiple character encodings.

UTF-8 - uses 8-bit byte, and is backward compatible with ASCII.

UTF-16 - uses 16-bit unit, OR two 16 bit values.

String

```
let city = "Tempe"  
var name = "Jane Doe"  
  
var message = "Hello " + name
```

String

```
let count = 17
```

```
var message = "value of count is \ (count) "
```

String interpolation

```
let count = 17  
var message = "value of count is \ (count) "
```

String interpolation is a way to construct a new String value from a mix of constants, variables, literals, and expressions by including their values from inside a string literal.

In this example, the placeholder `\(count)` will be replaced with the actual value of the count constant, when the string interpolation is evaluated.

String

```
let count = 17
```

```
var message = "I  NY"
```

```
var blueHeart = "\u{1F499}"
```