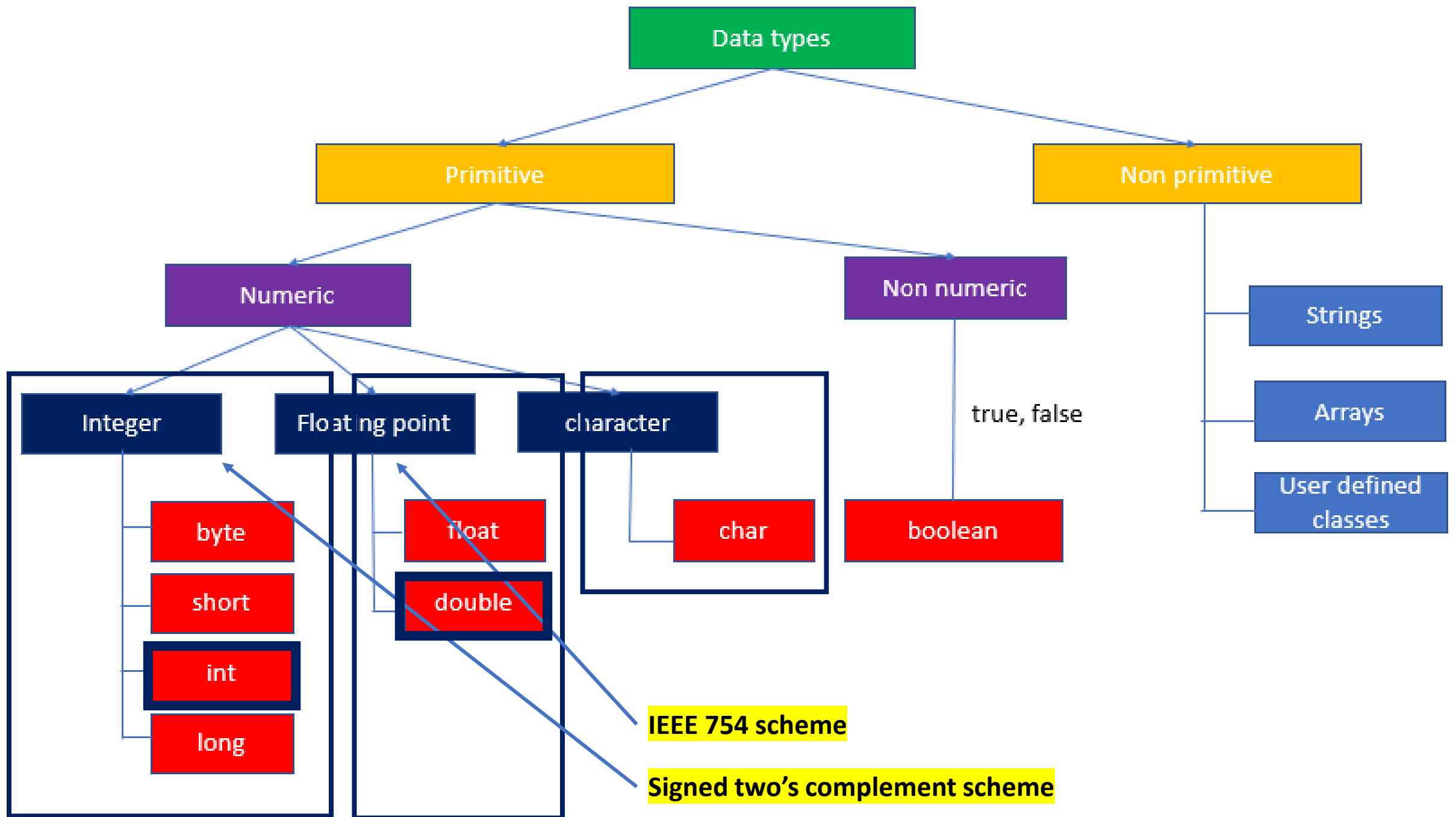


# Chapter 24

## Java data types - char





## Characters (char)

**char** have any but **one character** (letters, numbers, symbols, space...)

**char** is represented using single quotes( 'a', ' ', '8', '&' )

[illegible]

**NOTE:** `char` is a **primitive data type** whereas `String` is a **class** in java.

# String?

## Collection of characters

## A String can have zero or more characters

**String is represented using double quotes( "sure", "a", "" )**

# Note

- Integer(**byte, short, int, long**) data types are **signed primitives** (positive, negative)
- Floating point(**float, double**) data types are **signed primitives** (positive, negative)
- Char data type is a **unsigned primitive data type (only positive)**

# Use case(char)

- To represent **section of a student** (Ex: A, B, C)
- To represent **grade of a student** (Ex: A, B)
- Let's look at an example
- Look at the default value of a char variable(will discuss later)

```
void charDemo1 () {  
    heading("====char demo====");  
    char section = 'A';  
    byte a = 'A'; //how did byte can store a char  
    System.out.println(section);  
    System.out.println(a);  
}
```

# Story behind characters representation



Early days of a computer

1. Computer can represent numbers, but how to deal with characters('a', 'A') 🤖 🤖 🤖`?

2. Let's map each character with a digit (అంకెల భాష )

0 - ' ', a - 1, b - 2, c - 3, d - 4, e - 5, f - 6, g - 7,  
h - 8, i - 9, j - 10, k - 11, l - 12, m - 13, n - 14, o -  
15, P - 16, q - 17, r - 18, s - 19, t - 20, u - 21,  
v - 22, w - 23, x - 24, y - 25, z - 26

I love you - 9012152250251521

010100001001010010100101010101001010100  
110010101010101010010100101001010010101  
101001010101001010101001010100101010010

We need a standard way to encode the characters 🤖



Your lover

























0 - ' ', a - 11, b - 20, c - 13, d - 14, e - 15, f - 16, g - 17,  
h - 18, i - 9, j - 10, k - 1, l - 2, m - 3, n - 4, o - 5,  
P - 6, q - 7, r - 8, s - 19, t - 12, u - 21, v - 22, w - 23,  
x - 24, y - 25, z - 26

I tevo yeu - 9012152250251521

010100001001010010100101010101001010100  
110010101010101010010100101001010010101  
101001010101001010101001010100101010010

# ASCII

- In 1960, American standard association created a 7 bit encoding called ASCII
- ASCII stands for **American Standard Code for Information Interchange**
- 7 bits gives 128 possible values ( 0 to 127 )
- New micro processors wanted to use power of 2, then they made it to 8 bits
- Extended to 8-bit to better utilize 8 bit computer memory organization (parity check)
- 8 bits – 256 ( 0 – 127 are same as ASCII)

0: 	32:	64: @	96: `
1: 	33: !	65: A	97: a
2: 	34: "	66: B	98: b
3: 	35: #	67: C	99: c
4: 	36: \$	68: D	100: d
5: 	37: %	69: E	101: e
6: 	38: &	70: F	102: f
7: 	39: '	71: G	103: g
8: 	40: (	72: H	104: h
9:	41: )	73: I	105: i
10:	42: *	74: J	106: j
11:	43: +	75: K	107: k
12:	44: ,	76: L	108: l
13:	45: -	77: M	109: m
14: 	46: .	78: N	110: n
15: 	47: /	79: O	111: o
16: 	48: 0	80: P	112: p
17: 	49: 1	81: Q	113: q
18: 	50: 2	82: R	114: r
19: 	51: 3	83: S	115: s
20: 	52: 4	84: T	116: t
21: 	53: 5	85: U	117: u
22: 	54: 6	86: V	118: v
23: 	55: 7	87: W	119: w
24: 	56: 8	88: X	120: x
25: 	57: 9	89: Y	121: y
26: 	58: :	90: Z	122: z
27: 	59: ;	91: [	123: {
28:	60: <	92: \	124:
29:	61: =	93: ]	125: }
30:	62: >	94: ^	126: ~
31:	63: ?	95: _	127: 

# Different language standards

- **ASCII** (American Standard Code for Information Interchange) for the **United States**.
- **ISO 8859-1** for Western **European Language**.
- **KOI-8** for **Russian**.
- **GB18030** and **BIG-5** for **Chinese**, and so on.

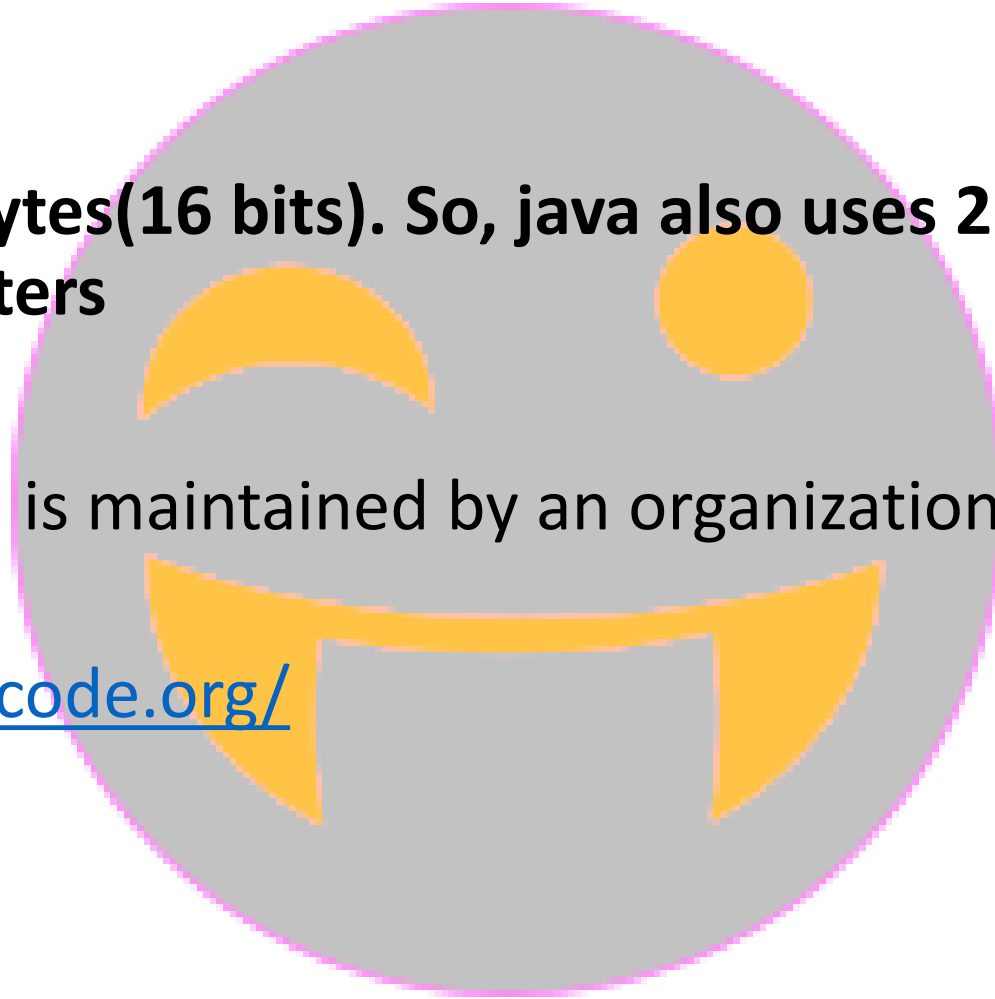
**Issues: Different codes in different language standards**

**Unicode system (Universal code)**



# Unicode

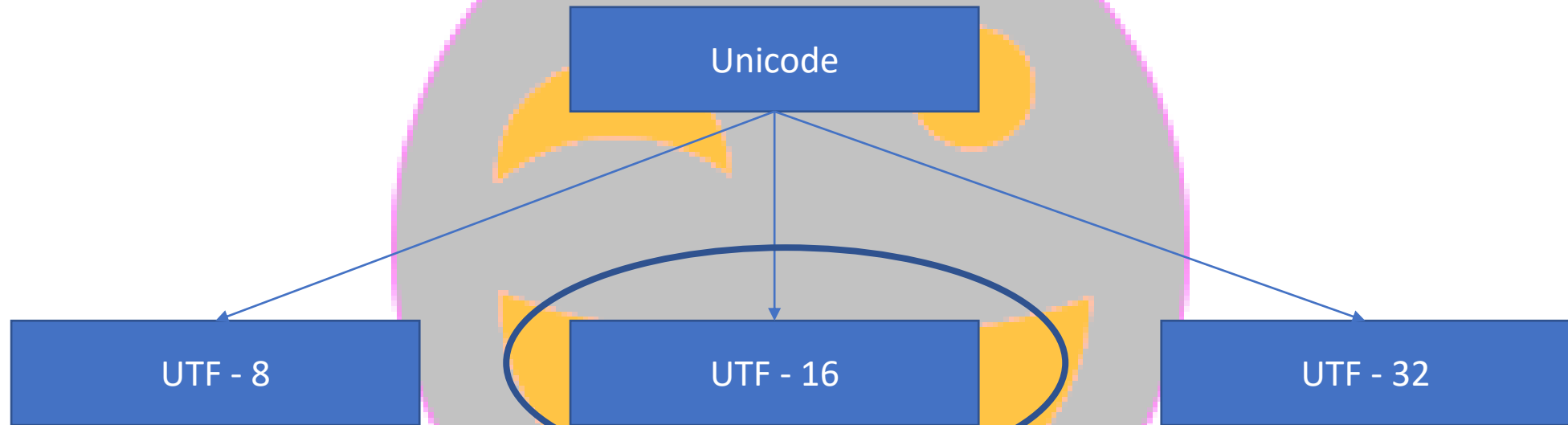
- Unicode uses **2 bytes(16 bits)**. So, java also uses **2 bytes(16 bits)** to **represent characters**
- **0 to 65,536**
- Unicode standard is maintained by an organization called **Unicode consortium**
- <https://home.unicode.org/>



# Character

- Java uses 2 bytes(16 bits) to represent a character
- $-2^{15}$  to  $2^{15}-1$
- ~~$-2^{15}$  to  $2^{15}-1$~~
- It is **16 bit unsigned** primitive data type
- **Range:** 0 to  $2^{16}-1$  (0 to 65535)
- So, essentially **characters are integers internally**
- **Each character** will have a **mapping integer** (<https://unicode-table.com/en/>)
  - For example ,the char literal '**A**' here will be stored as 65 internally
  - char degree = '**A**';
- Mapping will be defined by the **Unicode character set**
- Unicode is a standard specification and it can have **different implementations**

# Implementations (Encodings)



**Range: 0 to 65535, same as Unicode set**

**Unicode Transformation Format**

<https://unicode-table.com/en/>



What values can be put in a char container?

# Integer literal

- For Integer data types(**byte, short, int, long**), we can specify literals in 4 ways

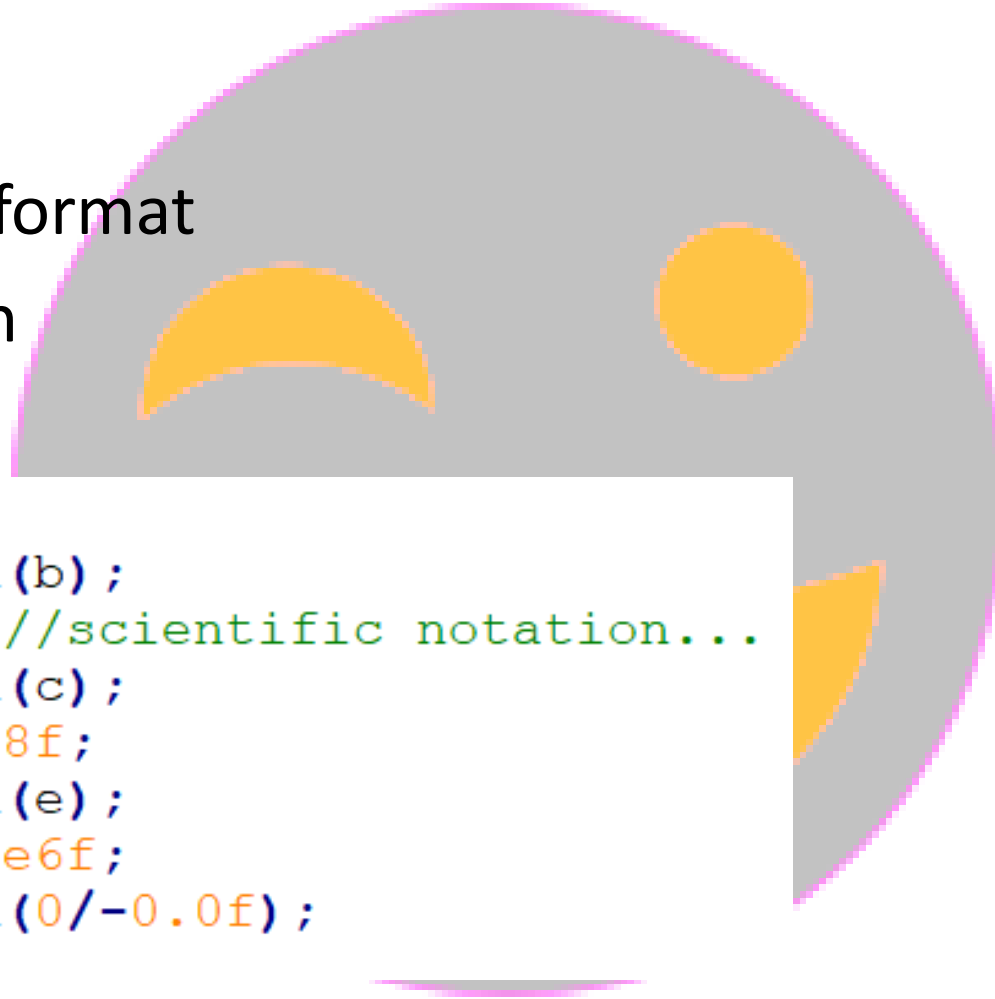
- Decimal literals (Base 10)
- Binary literal(Base 2)
- Octal literals(Base 8)
- Hexa-decimal literals(Base 16)

```
System.out.println("=====int=====");
System.out.println("Default value of int is: "+d.intBasket);
d.intBasket = 99; //Decimal literals
//binary - 2 (base 2) - 0 or 1
System.out.println("decimal value: "+d.intBasket);
d.intBasket = 0b01100011; //binary number: 0b /0B
System.out.println("binary value: "+d.intBasket);
System.out.println(d.intBasket);
d.intBasket = 0143; //octal number
System.out.println("octal value: "+d.intBasket);
d.intBasket = 0X7fffffff; //hexa decimal value
System.out.println("hexa decimal value: "+d.intBasket);
System.out.println("Range");
System.out.println("min value is: "+Integer.MIN_VALUE);
System.out.println("max value is: "+Integer.MAX_VALUE);
```

# A floating numbers can be expressed in two formats

- Decimal number format
- Scientific notation

```
float b = 5.67f;  
System.out.println(b);  
float c = 567e-2f;//scientific notation...  
System.out.println(c);  
float e = 34.776288f;  
System.out.println(e);  
float f = 0.000034e6f;  
System.out.println(0/-0.0f);
```



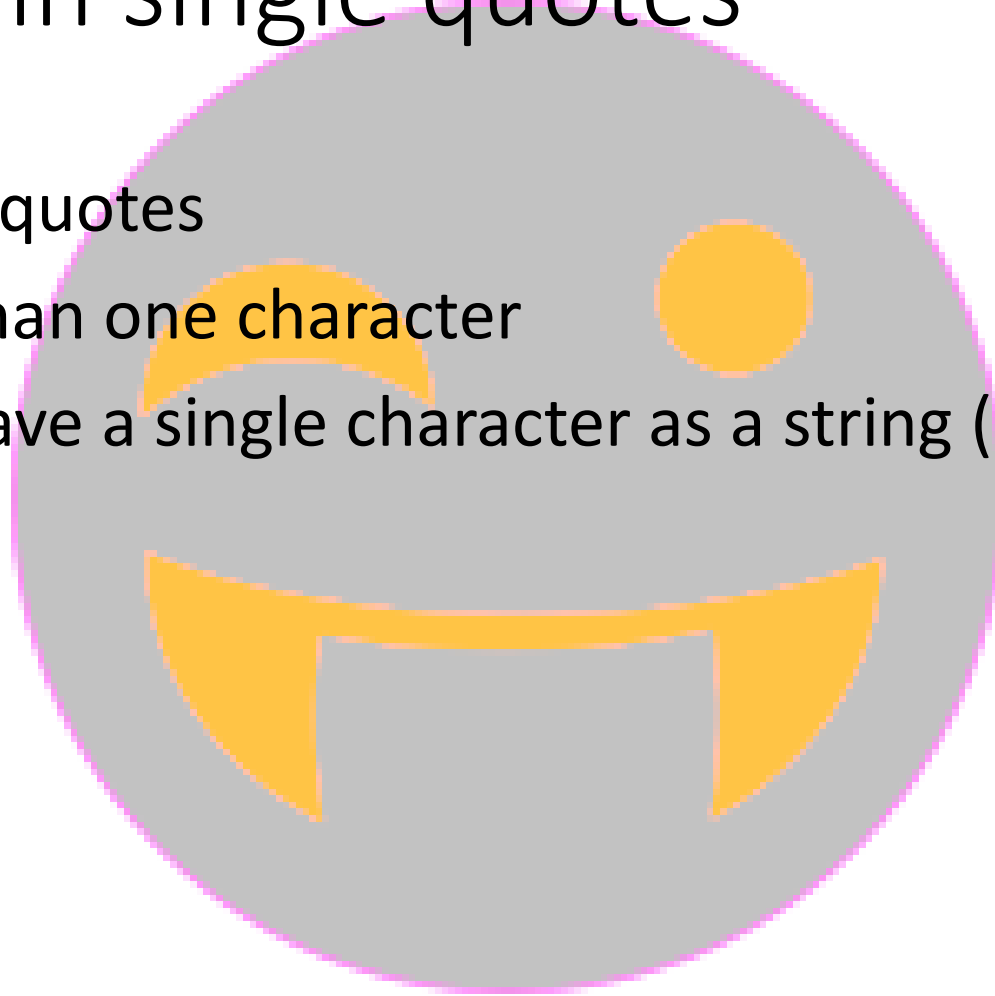
# Char literal can be expressed in 4 ways

- Enclosed in **single quotes**
  - Escape sequence
  - **u**nicode escape sequence
  - Octal escape sequence
- 
- **Special way**



# 1. Enclosing in single quotes

- Can't use double quotes
- Can't use more than one character
- It is possible to have a single character as a string (EX: String name = "A")





## 2. Escape sequence(8) – starts with backslash

Escape Sequence	Description
'\n'	A linefeed
'\r'	A carriage return
'\f'	A form feed
'\b'	A backspace
'\t'	A tab
'\\'	A backslash
'\"'	A double quote
'\''	A single quote

# Descriptions for escape sequences

- **Carriage return** means to return to the beginning of the current line without advancing downward. The name comes from a printer's carriage, as monitors were rare when the name was coined. This is commonly escaped as "`\r`", abbreviated CR, and has ASCII value 13 or 0xD.
- **Linefeed** means to advance downward to the next line; however, it has been repurposed and renamed. Used as "newline", it terminates lines (commonly confused with separating lines). This is commonly escaped as "`\n`", abbreviated LF or NL, and has ASCII value 10 or 0xA. CRLF (but not CRNL) is used for the pair "`\r\n`".
- **Form feed** means advance downward to the next "page". It was commonly used as page separators, but now is also used as section separators. Text editors can use this character when you "insert a page break". This is commonly escaped as "`\f`", abbreviated FF, and has ASCII value 12 or 0xC.

# 3. Unicode escape sequence

- It can be expressed as a Unicode escape sequence as `'\uxxxx'`

Sign	Meaning
<code>\u</code>	denotes the start of the Unicode escape sequence
<code>xxxx</code>	represents exactly four hexadecimal digits. xxxx is the Unicode value for the character.

```
void charDemo() {  
    heading("char");  
    System.out.println(charBasket);  
    charBasket = 'a';  
    System.out.println(charBasket);  
    charBasket = 'F';  
    charBasket = 70;  
    charBasket = '\u0046';  
    System.out.println(charBasket);  
}
```

# Hexadecimal – 16 – (0 to 15)

0	8	0	8	<b>Convert 125 into hexadecimal</b>
1	9	1	9	
2	10	2	A	
3	11	3	B	
4	12	4	C	
5	13	5	D	
6	14	6	E	
7	15	7	F	

Range for the Unicode sequence is : ``\u000`` to ``\uffff``

DEC	HEX	Meaning		DEC	HEX	Meaning	
0	00	NUL	Null	17	11	DC1	Device Control 1
1	01	SOH	Start of Heading	18	12	DC2	Device Control 2
2	02	STX	Start of Text	19	13	DC3	Device Control 3
3	03	ETX	End of Text	20	14	DC4	Device Control 4
4	04	EOT	End of Transmission	21	15	NAK	Negative Ack.
5	05	ENQ	Enquiry	22	16	SYN	Sync. Idle
6	06	ACK	Acknowledgment	23	17	ETB	End of Transmission
7	07	BEL	Bell	24	18	CAN	Cancel
8	08	BS	Back Space '\b'	25	19	EM	End of Medium
9	09	HT	Horizontal Tab '\t'	26	1A	SUB	Substitute
10	0A	LF	Line Feed '\n'	27	1B	ESC	Escape
11	0B	VT	Vertical Feed	28	1C	IS4	File Separator
12	0C	FF	Form Feed 'f'	29	1D	IS3	Group Separator
13	0D	CR	Carriage Return '\r'	30	1E	IS2	Record Separator
14	0E	SO	Shift Out	31	1F	IS1	Unit Separator
15	0F	SI	Shift In				
16	10	DLE	Datalink Escape	127	7F	DEL	Delete

# Octal escape sequence

- It can be expressed as **octal sequence** in the form of '\xxx' where x is an octal digit(0-7)
- Using octal escape sequence, you can represent characters whose Unicode code ranges from **0 to 255 decimal integers only**
- Range for the octal sequence is : '\000' to '\377'
- char a = '\43';
- char b = '\899'; //Error (out of range)

$$\begin{array}{r} 8 \overline{) 255} \\ \underline{240} \phantom{0} \\ 15 \phantom{0} \\ \underline{16} \phantom{0} \\ -1 \phantom{0} \\ 7 \end{array}$$
$$\begin{array}{r} 8 \overline{) 31} \\ \underline{24} \phantom{0} \\ 7 \end{array}$$
$$\begin{array}{r} 8 \overline{) 3} \\ \underline{0} \phantom{0} \\ 3 \end{array}$$

$$\begin{array}{r} 377 \\ \underline{0} \\ +1 \\ 43 \end{array}$$

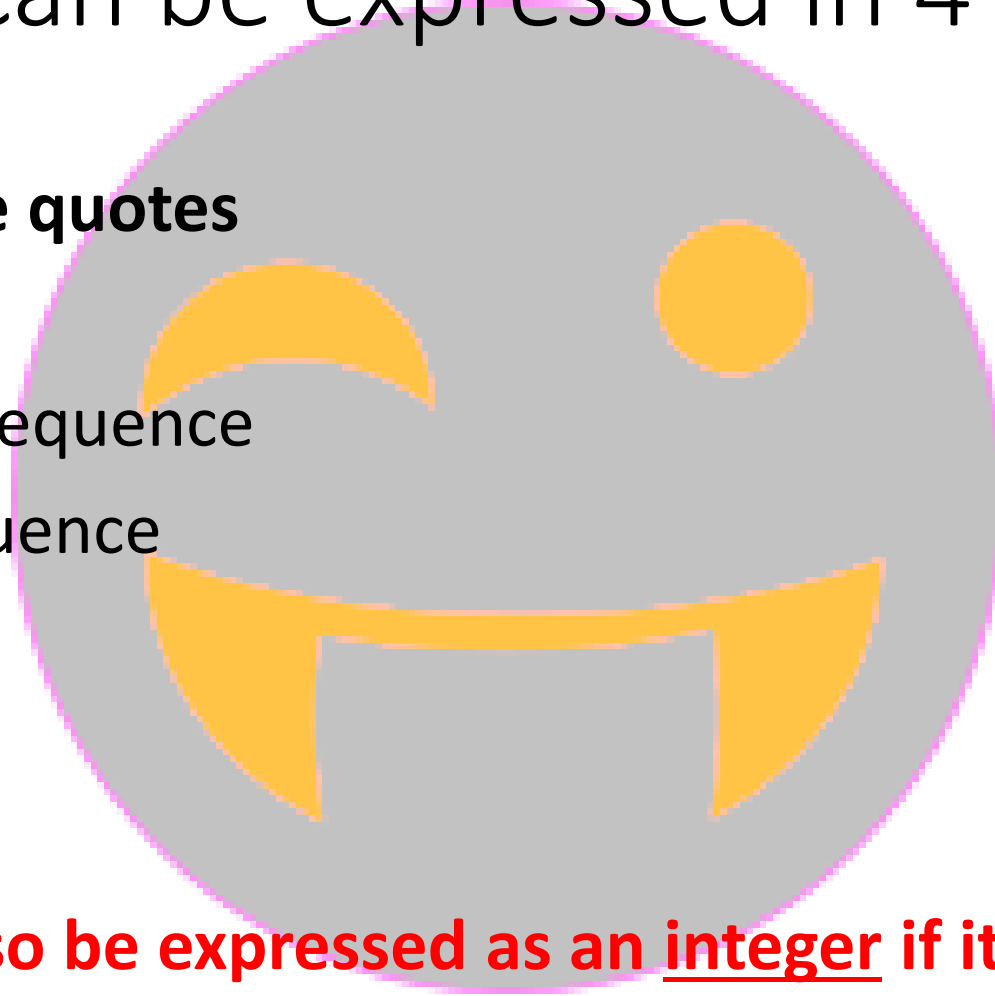
$$3 + 28 = \textcircled{31}$$

# Char literal can be expressed in 4 ways

- Enclosed in **single quotes**
- Escape sequence
- Unicode escape sequence
- Octal escape sequence

- **Special way**

**char literal can also be expressed as an integer if it is within the range(0 to 65536)**



# Note

- char section = 'A' or '\u0042'
- '\u0042' is called Unicode escape sequence (u here stands for **Unicode** and it needs to be in lower case)
- Default value \u0000 represents **null character**
- null character is a **control character** meaning that is not something you can print

```
void charDemo1 () {  
    heading("====char demo====");  
    char section = 'A';  
    byte a = 'A'; //how did byte can store a char  
    System.out.println(section);  
    System.out.println(a);  
}
```



# Note – very important

- You **cannot assign** a value stored in a variable of signed data type(byte, short, int, long) into char as char is an **unsigned data type**.

```
byte k = 113;  
char var3 = 113;//no error  
char var4 = k;//error
```

# What next?

boolean Data type



చిన్న బ్రేక్ చిటికలో వచ్చేస్తా