Operators & Strings

Session Objectives

- Understand what operators are and why they are used
- Explore different types of operators in Python
- Learn about operator precedence and order of execution
- Understand constraints in programming
- Understand string indexing and slicing
- Explore common string methods and operations

Bitwise-Operators

'and' *

bit1	bit2	Result
1	1	1
1	0	0
0	1	0
0	0	0

'or' +

bit1	bit2	Result
1	1	1
1	0	1
0	1	1
0	0	0

'XOR'

bit1	bit2	Result
1	1	0
1	0	1
0	1	1
0	0	0

10 decimal -> Binary

2	10
2	5 - 0
2	2 - 1
2	1 - 0
	0 - 1

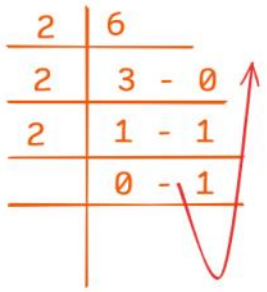
1010

1010

2^3 2^2 2^1 2^0

= (1*2^3)+(0*2^2)+(1*2^1)+(0*2^0)
= 8 + 0 + 2 + 0 = 10

6 decimal -> Binary



0110

0110

2^3 2^2 2^1 2^0

= (0*2^3) + (1*2^2) + (1*2^1) + (0*2^0)
= 0 + 4 + 2 + 0 = 6

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111


```
 1010
&0110
------
 0010   2
```

```
 1010
|0110
------
 1110   14
```

```
 1010
^0110
------
 1100   12
```

NOT | ~X = -(X+1)

0000 - 1010

1111 - 0101 (Flip the bits)

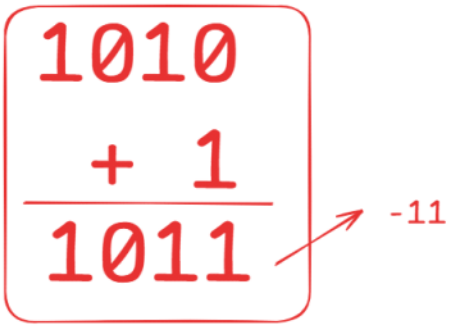
1s Complement

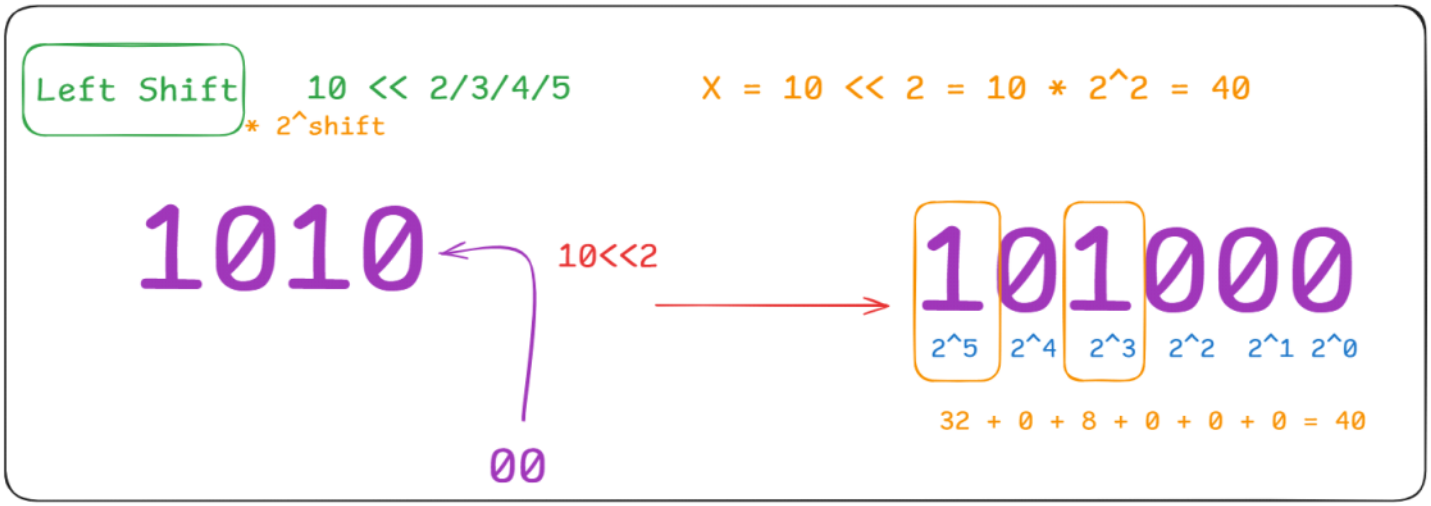
+1

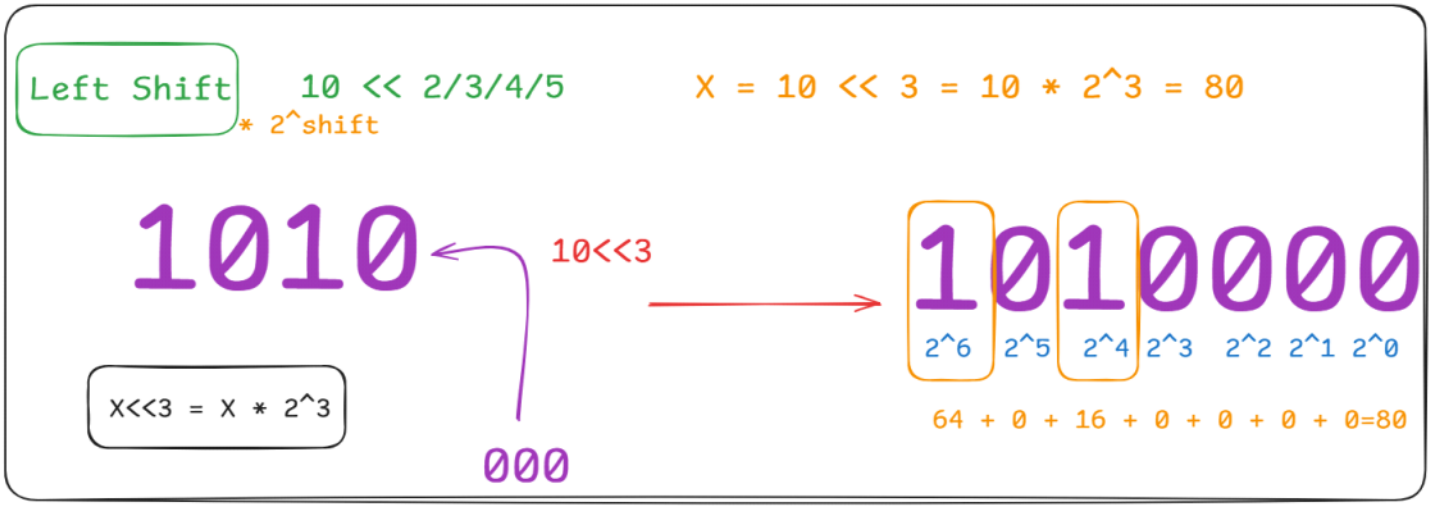
2s Complement

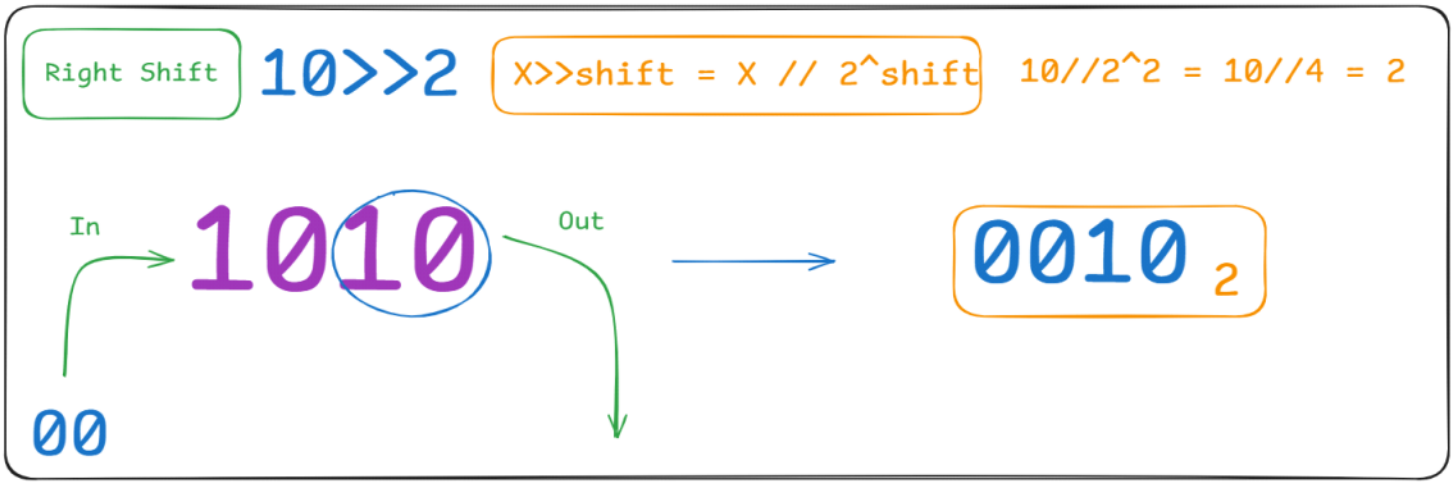
leftmost bit[sign][direction]

1 -> [-ve]
0 -> [+ve]


1010
+ 1
1011
-11


Left Shift
10 << 2/3/4/5
* 2^shift
X = 10 << 2 = 10 * 2^2 = 40
1010
10<<2
00
101000
2^5 2^4 2^3 2^2 2^1 2^0
32 + 0 + 8 + 0 + 0 + 0 = 40


Left Shift
10 << 2/3/4/5
* 2^shift
X = 10 << 3 = 10 * 2^3 = 80
1010
10<<3
X<<3 = X * 2^3
000
1010000
2^6 2^5 2^4 2^3 2^2 2^1 2^0
64 + 0 + 16 + 0 + 0 + 0 + 0=80


Right Shift
10>>2
X>>shift = X // 2^shift
10//2^2 = 10//4 = 2
In
1010
Out
00
0010 2

Bitwise Operators:

- 'AND' = '&' -> 'Both bit with value 1 returns 1 else 0'
- 'OR' = '|' -> 'Both bit with value 0 returns 0 else 1'
- 'XOR' = '^' -> 'ALternative bits return 1 else same bits returns 0'
- 'NOT' = '~' -> '2s Complement to check the sign and evaluate'
- 'Left Shift' = '<<' -> 'Shifts bits to the left'
- 'Right Shift' = '>>' -> 'Shifts bits to the right'

```
# 'AND - &' , 'OR - |' , 'XOR ^'
x = 10
y = 6
print(x&y) # 2
print(x|y) # 14
print(x^y) # 12
```

```
2
14
12
```

```
X = 10
print(~X) # -(X+1) # -(10+1) #-11
```

```
-11
```

```
# Not Logic (~)
X = -10
print(~X) # -(X+1) # -(-10+1) #-(-9) = 9
```

```
9
```

```
# Left Shift
X = 10
print(X<<2) # X * 2^2 = 10 * 4 = 40
```

```
40
```

```
X = 10
print(X<<3) # X * 2^3 = 10 * 8 = 80
```

```
80
```

```
X = 10
print(X<<4) # X * 2^4 = 10 * 16 = 160
```

```
160
```

```
# Right Shift
X = 10
print(X>>2) # X // 2^2 = 10 // 4 = 2
```

```
2
```

```
# Right Shift
X = 10
print(X>>3) # X // 2^3 = 10 // 8 = 1
```

```
1
```

```
# Right Shift
X = 10
print(X>>4) # X // 2^4 = 10 // 16 = 0
```

```
0
```

Assignment Operators:

Assignment Operators: ¶

- '=' , x = 5
- '+=', x+=5 => x = x+5
- '-=', x-=5 => x = x-5
- '* =', x* =5 => x = x*5
- '/=', x/=5 => x = x/5
- '%=', x%=5 => x = x%5
- '**=', x **=5 => x = x **5
- '//=', x//=5 => x = x//5

```
# Note x^5 != x**5
x = 10
y = 11
z = 7
x += y # x = x + y # x = 10 + 11 = 21
print(x) # 21
x -= z # x = x - z # x = 21 - 7 = 14
print(x) # 14
x /= 2 # 14/2 = 7.0[Float]
print(x) #7.0
```

```
21
14
7.0
```

```
x *= x # x = x*x = 7.0 * 7.0 = 49.0
print(x) # 49.0
```

```
49.0
```

```
x %= y # x = x % y => 49.0 % 11 = 5.0
print(x) # 5.0
```

```
5.0
```

```
x //= z # x = x // z => 5.0 // 7 = 0.0
print(x) # 0.0
```

```
0.0
```

```
y **=x # y = y ** x => 11 ** 0.0 => 1.0
print(y) # 1.0
```

```
1.0
```

Membership Operators:

- It Returns Boolean Results
- 'in' : True if the Values is in the sequence else False
- 'not in' : True if the Values is not in the sequence else False

```
print('hello' in 'Hello World') # False [Case-Sensitive]
print('hello' in 'hello World') # True
```

```
False
True
```

```
print('I' in 'India') # True
print('I' in 'America') # False 'I' <> 'i'
```

```
True
False
```

```
print('Mon' in ['Mon','Tue','Wed','Thurs','Fri','Sat','Sun']) # True
print('mon' in ['Mon','Tue','Wed','Thurs','Fri','Sat','Sun']) # False
```

```
True
False
```

```
print('mon' not in ['Mon','Tue','Wed','Thurs','Fri','Sat','Sun']) # True
print('Jan' not in ['Mon','Tue','Wed','Thurs','Fri','Sat','Sun']) # True
```

```
True
True
```

```
# Boolean Conversion :
x = int(True)
print(x) # 1 TinyInt[SQL]
```

```
1
```

```
# Boolean Conversion :
x = int(False)
print(x) # 0 TinyInt[SQL]
```

```
0
```

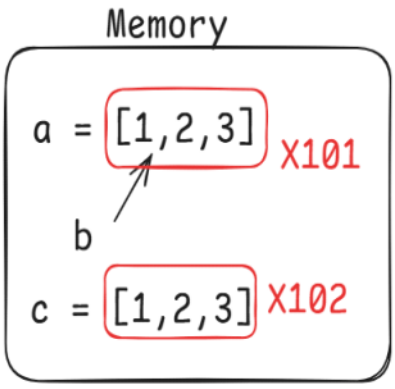
Identity Operators:

- 'is' - Returns True if both the variables refers to the same object (having same memory address)
- 'is not' - Returns True if both the variables refers to the Different object (having same memory address)

```
# '==' [Data Compare] VS is[Data + Address]
a = [1,2,3]
b = a
c = [1,2,3]
print(a==b) # True [Same Data]
print(a is b) # True [Same Address]

print(a==c) # True [Same Data]
print(a is c) # False [Different Address]
```

```
True
True
True
False
```




```
# id -> returning the memory Location
print(id(a)) # Same
print(id(b)) # Same
print(id(c)) # Different
```

```
2091364989248
2091364989248
2091364998464
```

id(var_name) # Address

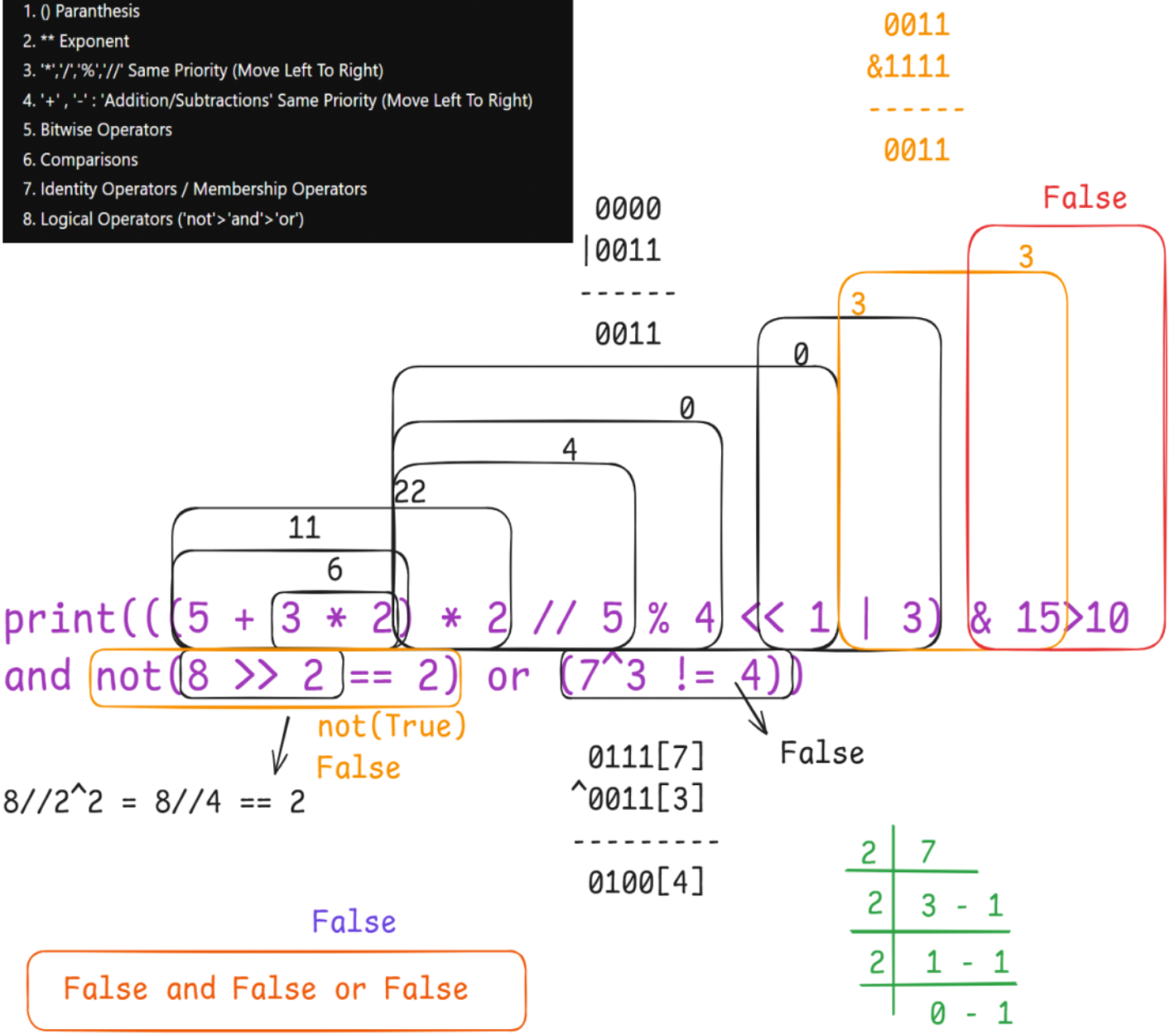
```
print(a is not c) # True ['a' & 'c' is not same]
```

```
True
```

```
10 / 3 = 3.3333
3.333 [floor] [Round Down] -> 3
3.333 [Ceil] [Round Up] -> 4
10 // 3 [Floor Division] -> 3
```

Order Of Operations: (PEMDAS/BODMAS)

1. () Paranthesis
2. ** Exponent
3. '*','/','%','//' Same Priority (Move Left To Right)
4. '+' , '-' : 'Addition/Subtractions' Same Priority (Move Left To Right)
5. Bitwise Operators
6. Comparisons
7. Identity Operators / Membership Operators
8. Logical Operators ('not'>'and'>'or')




```python
# Challenge :
cond1 = (10*3)+((10<<3)*(10%3)) # 30 + ((10*2^3) * 1) -> 30 + (80*1) = 80+30 = 110
cond2 = (5**2)*((3//2)-(10%7)) # (25) * ((1)-(3)) # 25 * -2 = -50
_bool = cond1 > cond2 # True # 110 > -50
print(cond1) # 110
print(cond2) # -50
print(_bool) # True
```

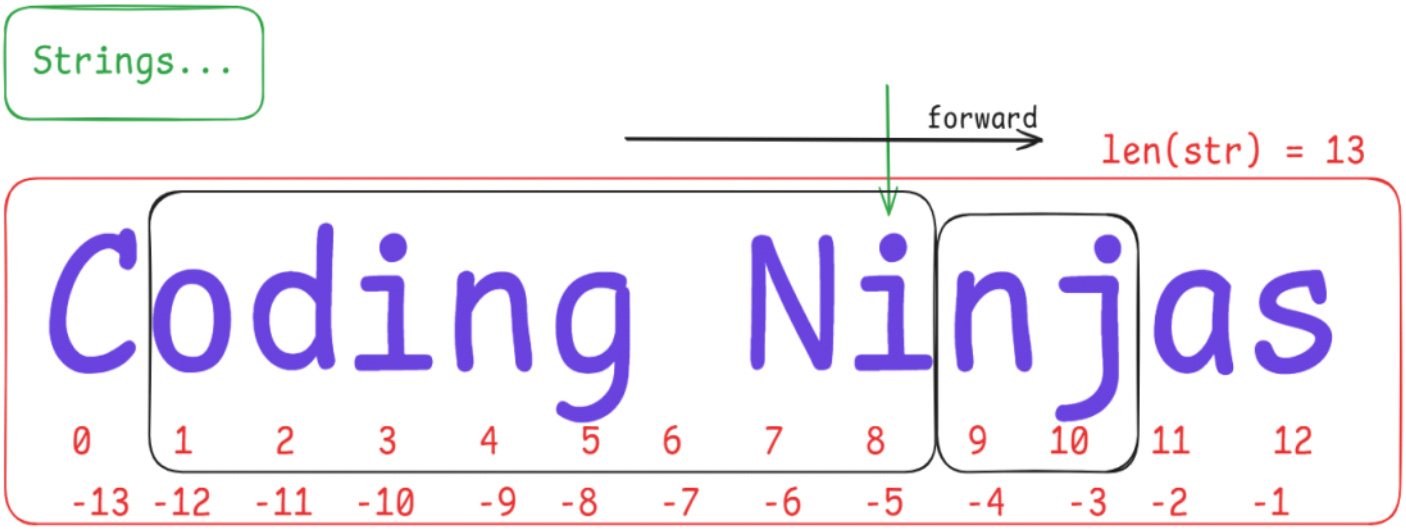
```
110
-50
True
```

```
print(cond1 is cond2) # False [Different Address]
print(cond1 is not cond2) # True
print(id(cond1))
print(id(cond2))
print(id(_bool))
```

```
False
True
140713960224600
2091346884432
140713959095168
```

```
print(((5 + 3 * 2) * 2 // 5 % 4 << 1 | 3) & 15>10 and not(8 >> 2 == 2) or (7^3 != 4))
```

```
False
```



_str[:6] Coding

stop:0

_str[0:12:2]: Cdn ij

_str[-5::-1])

_str[-5:0] ""

start : 5
stop : 0
step : 1

String Comparison → 'a' > 'A' 'a'[97]>'Z'[90]

'A'[65]-'Z'[90] < 'a'[97] - 'z'[122]

ASCII TABLE

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

ASCII = American Standard Code for Information Interchange.'

Java Vs JavaScript

JavaScript > Java

java Vs JavaScript

java > JavaScript

Strings:

- A Sequence of Characters
- '' or "" (Denotes)

```
# Indexing [Python is having a Zero Based Indexing] 0
# Slicing [Start : Stop : Step]
# Indexing[Pos] : +ve[Left To Right] , -ve [Right To Left]
_str = 'Coding Ninjas'
print(_str[0]) # 'C'
print(_str[-1]) # 's'
```

```
C
s
```

```
print(_str[-11]) # 'd'
print(_str[7]) # 'N'
print(_str[-2]) # 'a'
print(_str[-7]) # ' '
```

```
d
N
a
```

```
# IndexError : Out of Range
print(_str[-15])
print(_str[15])
# IndexError: string index out of range
```

```
# Slicing [Start : Stop : Step] # Parameters [Extracting a part of a string]'Substring'
# Start [0] : Default Value
# Stop [length] [Non-Inclusive] #[0,length)
# BETWEEN >=10 AND <=100 [10,100]
# BETWEEN >=10 AND <100 [10,100)
# Step [1] : Linear Positive Step [Direction & Magnitude]
_str = 'Coding Ninjas'
print(_str[:6]) # 'Coding'
print(_str[7:]) # 'Ninjas'
print(_str[7:100]) # 'Ninjas'
print(_str[0:12:2]) # 'Cdn ij'
print(_str[0:12:3]) # 'Ci n'
print(_str[:]) # 'Coding Ninjas'
```

```
Coding
Ninjas
Ninjas
Cdn ij
Ci n
Coding Ninjas
```

```
print(len(_str)) # 13
```

```
13
```

```
print(_str[12]) # 's'
```

```
s
```

```
_str = 'Coding Ninjas'
print(_str[0:15:2]) # 'Cdn ijs'
print(_str[0:20]) # 'Conding Ninjas'
```

```
Cdn ijs
Coding Ninjas
```

```
# Negative Indexing
_str = 'Coding Ninjas'
print(_str[-6:]) # 'Ninjas'
print(_str[::-1]) # [Reverse the str right to left] 'sajniN gnidoC'
```

```
Ninjas
sajniN gnidoC
```

```python
_str = 'Coding Ninjas' # ste1p : 1 [forward]
print(_str[-5:0]) # ' '
```

```

```

```python
_str = 'Coding Ninjas' # step : -1 [reverse]
print(_str[-5:0:-1]) # 'iN gnido'
```

```
iN gnido
```

```python
_str = 'Coding Ninjas' # step : -1 [reverse] [0 - 1] : -1
print(_str[-5::-1]) # 'iN gnidoC'
```

```
iN gnidoC
```

```python
_str = 'Coding Ninjas'
print(_str[-4:-2:-1]) # ' '
```

```

```

```python
_str = 'Coding Ninjas' # step : 1[forward]
print(_str[-5:-1]) # 'inja'
```

```
inja
```

```python
# String Case Methods
# len -> Counts the Length returns 'int' value
_str = 'Python is Awesome 🔥'
print(len(_str)) # 19
```

```
19
```

```python
print(_str.upper()) # Upper Case
```

```
PYTHON IS AWESOME 🔥
```

```python
print(_str.lower()) # Lower Case
```

```
python is awesome 🔥
```

```python
print(_str)
```

```
Python is Awesome 🔥
```

```python
_str = _str.lower()
print(_str)
```

```
python is awesome 🔥
```

```
# .title() # Every First Character of each Word will be Capitalize
print(_str.title())
```

```
Python Is Awesome 🔥
```

```
# .capitalize() # Only First Character of Complete String will be Capitalize
print(_str.capitalize())
```

```
Python is awesome 🔥
```

```
_str = "Python is Awesome" # 17
print(_str[:16]) # [0,16) Exclude[Non-Inclusive]
```

```
Python is Awesom
```

```
_str = "Python is Awesome" # 17
print(_str[:]) # till length including last charaters
```

```
Python is Awesome
```

```
_str = "Python is Awesome" # right to left
print(_str[::-1]) # Reverse Gear
```

```
emosewA si nohtyP
```

```
_str = "Python is Awesome" # right to left
print(_str[:-17:-1])
```

```
emosewA si nohty
```