Python Fundamentals day 18

Today's Agenda

- String comparison by ignoring cases
- Ascii Table
- Conversion to lowercase
- upper( ) and lower( )
- join( )

String Comparison by ignoring cases

We had seen different cases of comparing two strings except the above case. Let us now learn how to compare two strings by ignoring their cases.

In order to compare two strings without considering their cases, one must understand **ASCII table** as python is case sensitive.

ASCII is the acronym for

the American Standard Code for Information Interchange.

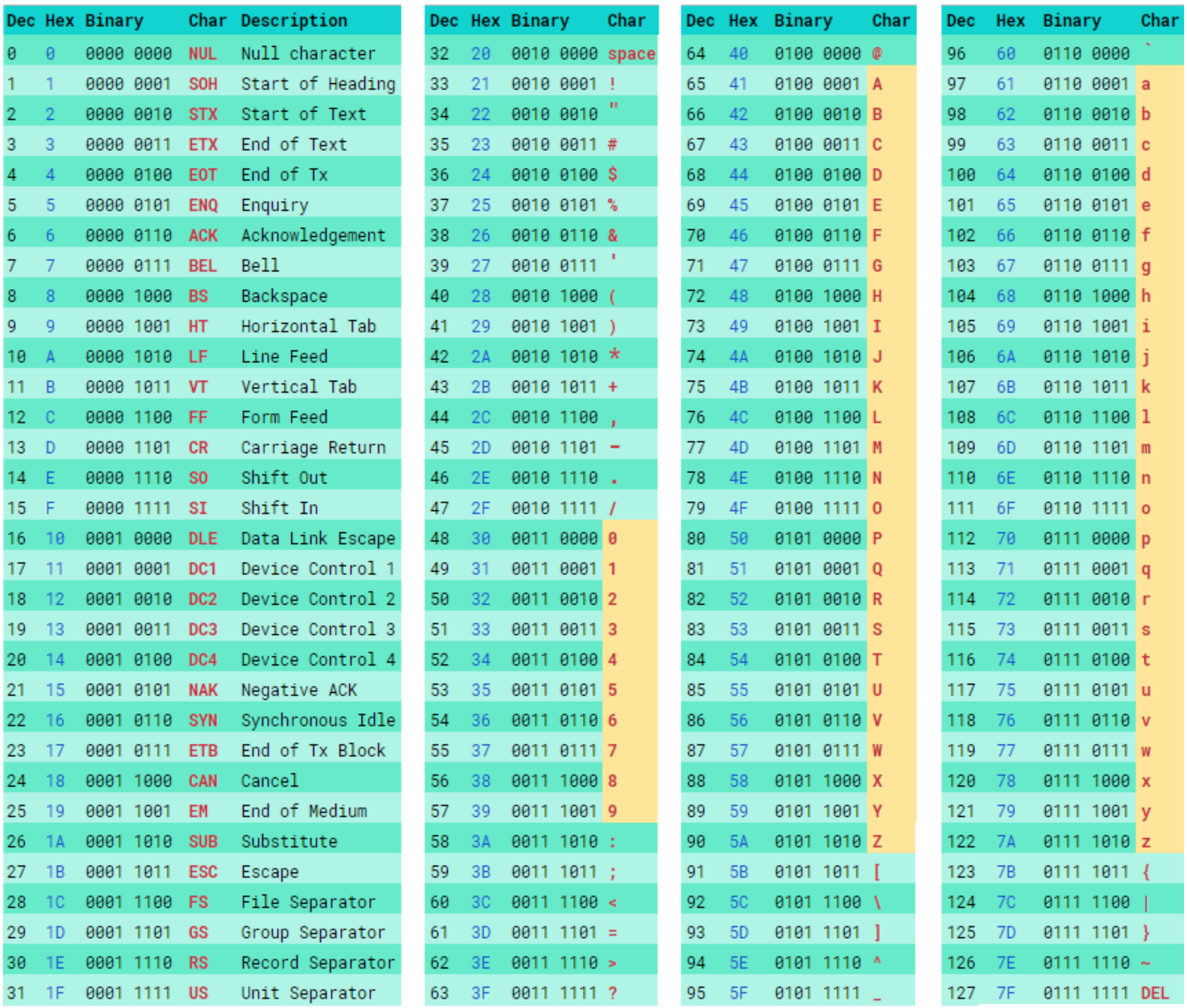
It is a code for representing 128 English characters as numbers, with each letter assigned a number from 0 to 127.

For example, the ASCII code for uppercase *M* is 77 and lowercase m is 109. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Using the range of lowercase alphabets i;e 97 to 122, we can check whether it is a uppercase or lowercase alphabet and perform the conversion.

ASCII TABLE

Have a look at the ASCII TABLE shown below:



Dec	Hex	Binary	Char	Description	Dec	Hex	Binary	Char	Dec	Hex	Binary	Char	Dec	Hex	Binary	Char
0	0	0000 0000	NUL	Null character	32	20	0010 0000	space	64	40	0100 0000	@	96	60	0110 0000	`
1	1	0000 0001	SOH	Start of Heading	33	21	0010 0001	!	65	41	0100 0001	A	97	61	0110 0001	a
2	2	0000 0010	STX	Start of Text	34	22	0010 0010	"	66	42	0100 0010	B	98	62	0110 0010	b
3	3	0000 0011	ETX	End of Text	35	23	0010 0011	#	67	43	0100 0011	C	99	63	0110 0011	c
4	4	0000 0100	EOT	End of Tx	36	24	0010 0100	$	68	44	0100 0100	D	100	64	0110 0100	d
5	5	0000 0101	ENQ	Enquiry	37	25	0010 0101	%	69	45	0100 0101	E	101	65	0110 0101	e
6	6	0000 0110	ACK	Acknowledgement	38	26	0010 0110	&	70	46	0100 0110	F	102	66	0110 0110	f
7	7	0000 0111	BEL	Bell	39	27	0010 0111	'	71	47	0100 0111	G	103	67	0110 0111	g
8	8	0000 1000	BS	Backspace	40	28	0010 1000	(	72	48	0100 1000	H	104	68	0110 1000	h
9	9	0000 1001	HT	Horizontal Tab	41	29	0010 1001	)	73	49	0100 1001	I	105	69	0110 1001	i
10	A	0000 1010	LF	Line Feed	42	2A	0010 1010	*	74	4A	0100 1010	J	106	6A	0110 1010	j
11	B	0000 1011	VT	Vertical Tab	43	2B	0010 1011	+	75	4B	0100 1011	K	107	6B	0110 1011	k
12	C	0000 1100	FF	Form Feed	44	2C	0010 1100	,	76	4C	0100 1100	L	108	6C	0110 1100	l
13	D	0000 1101	CR	Carriage Return	45	2D	0010 1101	-	77	4D	0100 1101	M	109	6D	0110 1101	m
14	E	0000 1110	SO	Shift Out	46	2E	0010 1110	.	78	4E	0100 1110	N	110	6E	0110 1110	n
15	F	0000 1111	SI	Shift In	47	2F	0010 1111	/	79	4F	0100 1111	O	111	6F	0110 1111	o
16	10	0001 0000	DLE	Data Link Escape	48	30	0011 0000	0	80	50	0101 0000	P	112	70	0111 0000	p
17	11	0001 0001	DC1	Device Control 1	49	31	0011 0001	1	81	51	0101 0001	Q	113	71	0111 0001	q
18	12	0001 0010	DC2	Device Control 2	50	32	0011 0010	2	82	52	0101 0010	R	114	72	0111 0010	r
19	13	0001 0011	DC3	Device Control 3	51	33	0011 0011	3	83	53	0101 0011	S	115	73	0111 0011	s
20	14	0001 0100	DC4	Device Control 4	52	34	0011 0100	4	84	54	0101 0100	T	116	74	0111 0100	t
21	15	0001 0101	NAK	Negative ACK	53	35	0011 0101	5	85	55	0101 0101	U	117	75	0111 0101	u
22	16	0001 0110	SYN	Synchronous Idle	54	36	0011 0110	6	86	56	0101 0110	V	118	76	0111 0110	v
23	17	0001 0111	ETB	End of Tx Block	55	37	0011 0111	7	87	57	0101 0111	W	119	77	0111 0111	w
24	18	0001 1000	CAN	Cancel	56	38	0011 1000	8	88	58	0101 1000	X	120	78	0111 1000	x
25	19	0001 1001	EM	End of Medium	57	39	0011 1001	9	89	59	0101 1001	Y	121	79	0111 1001	y
26	1A	0001 1010	SUB	Substitute	58	3A	0011 1010	:	90	5A	0101 1010	Z	122	7A	0111 1010	z
27	1B	0001 1011	ESC	Escape	59	3B	0011 1011	;	91	5B	0101 1011	[	123	7B	0111 1011	{
28	1C	0001 1100	FS	File Separator	60	3C	0011 1100	<	92	5C	0101 1100	\	124	7C	0111 1100	|
29	1D	0001 1101	GS	Group Separator	61	3D	0011 1101	=	93	5D	0101 1101	]	125	7D	0111 1101	}
30	1E	0001 1110	RS	Record Separator	62	3E	0011 1110	>	94	5E	0101 1110	^	126	7E	0111 1110	~
31	1F	0001 1111	US	Unit Separator	63	3F	0011 1111	?	95	5F	0101 1111	_	127	7F	0111 1111	DEL

Let us understand the conversion from uppercase to lowercase and vice-versa.

Consider characters a and A whose ascii values are 97 and 65.

Simple addition and subtraction will help us perform the conversion.

If we subtract 32 from 97, we get 65 which is the ascii value of A.

Char	Value
a	97
↓	− 32
A	65

Note: Uppercase alphabets fall in the range of 65 to 90
Lowercase alphabets fall in the range of 97 to 122.

String Conversion to lowercase

Let us understand the above conversion with the help of codes:

CODE:

```
c = "a"
print(ord(c))  #ord() gives ascii value of a stroed in c
print(ord(c) - 32) # performs 97(ascii value of) - 32 and gives 65
print(chr(ord(c) - 32)) # gives character whose ascii value is 65
```

OUTPUT:

```
97
65
A
```

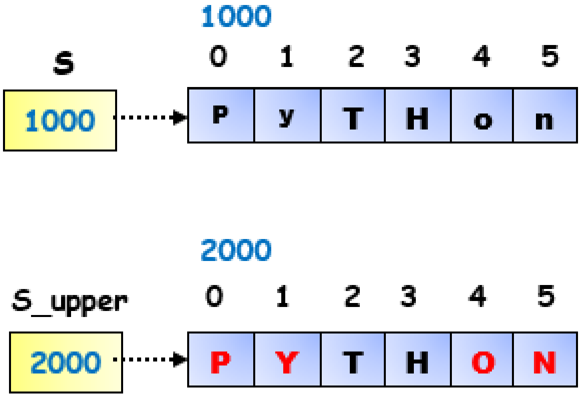
Explanation: In the above code we used two functions ord() & chr().
ord() converts a character to an integer and gives its ascii value.
chr() converts an integer to a character based on its ascii value.
Above code performs the conversion of lowercase a to uppercase A.

Let us now try writing logic for the conversion of an entire string to Uppercase.

CODE:

```
s = input("Enter a String\n")
s_upper = ""
for i in s:
    if ord(i) >= 97 and ord(i) <= 122:
        s_upper +=  chr(ord(i)-32)
    else:
        s_upper += i

print(s)
print(s_upper)
```



OUTPUT:

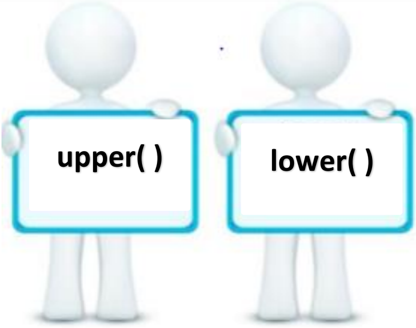
```
Enter a String
pyTHon
pyTHon
PYTHON
```

Explanation: In the above code we are first checking if each character inside the entered string falls in the range of (97,122) which simply means it is a lowercase character if true. If the condition gets evaluated to true, we performing the conversion from **lowercase to uppercase by subtracting 32** to the ascii value of character and storing it in a new string which is s_upper. If in case condition fails, then the character inside string is not a lowercase and do not require any conversion and hence we are simply appending the original character from string s to string s_upper. Thus, by subtracting 32 to any lowercase character we get the ascii value of its uppercase character.

The output we got using the above lines of code can also be achieved in just one line by calling a built-in function upper().
Also vice-versa can be performed by making use of built-in function lower().

upper() and lower()

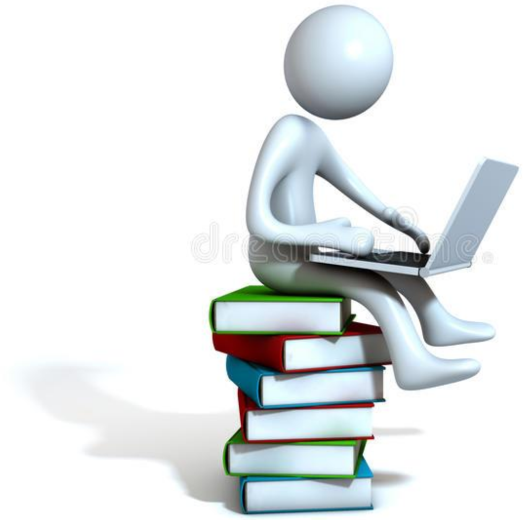
upper() function on a string converts all characters to uppercase.
lower() function on a string converts all characters to lowercase().



CODE:

```
s = input("Enter a String\n")
s_lower = s.lower()

print(s)
print(s_lower)
```



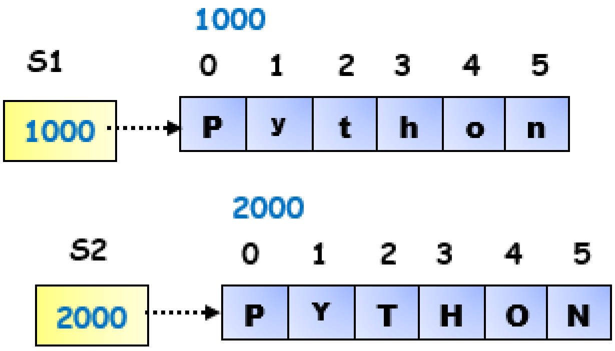
OUTPUT:

```
Enter a String
PYTHON
PYTHON
python
```

After getting to know all this, let us now compare two strings by ignoring their cases.

CODE:

```
s1 = "python"
s2 = "PYTHON"
if s1.upper() == s2.upper():
    print("String values are equal")
else:
    print("String values are unequal")
```



CODE:

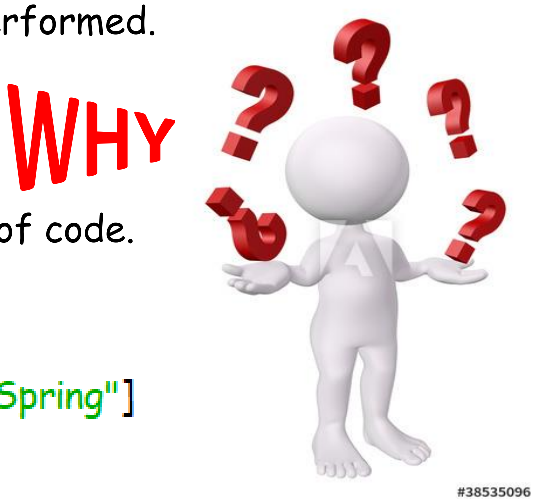
```
s1 = "python"
s2 = "PYTHON"
if s1.lower() == s2.lower():
    print("String values are equal")
else:
    print("String values are unequal")
```

OUTPUT:

```
String Values are equal
```

upper( )

We have seen how to perform concatenation between strings using + operator but this approach is not efficient when concatenation between multiple strings has to be performed.



Why you wonder???

Let us understand this with the help of code.

CODE:

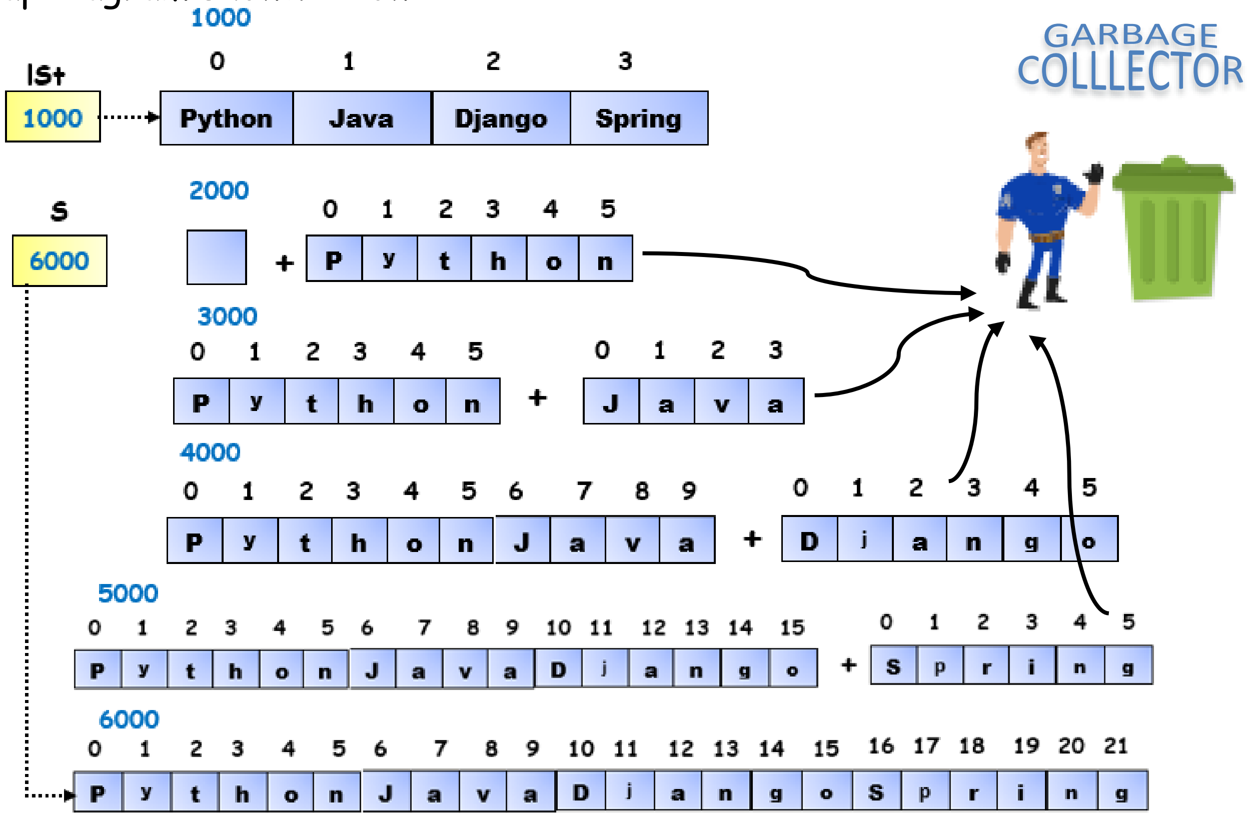
```
lst = ["Python" , "Java" , "Django" , "Spring"]
s = ""
for i in lst:
    s += i
print(s)
```

OUTPUT:

```
PythonJavaDjangoSpring
```

Explanation: In the above code, we have first created a list which now gets allocated memory with some address and has four string values, Python,Java,Django,Spring with lst as reference pointing to it. In the next line we are creating an empty string object with s as reference to it. Inside the for loop to the empty string s, we are attaching the value of i. The first value of i is Python and when concatenation happens, a new string object gets created which consists of added result of an empty string and Python. We are now storing it back in s which means s is now pointing to this newly created string object with address 3000 and hence the previous objects to which s was pointing to, will become garbage object and is collected by garbage Collector. In the second iteration, i takes the value Java which is now concatenated with Python String object having address i 3000. Since we are performing concatenation, a new string object gets created having concatenated result of Python and java with address 4000. The added result is now stored back into s and hence s is now pointing to this string object whose address is 4000. In the next iteration, i gets the value Django which is now concatenated with PythonJava. Since we are performing concatenation, a new string object gets created which consists of added result of PythonJavaDjango with 5000 as address. Similarly, in the last iteration i gets the value Spring which is now concatenated with PythonJavaDjango. Since we are performing concatenation, a new string object gets created which consists of added result of PythonJavaDjangoSpring with 6000 as address and this value is stored back into s which means s is now pointing to this string object with address 6000 and all the other objects who do not have any reference become garbage object as their reference count is zero and hence, are collected by garbage collector.

This way, when we use + operator to perform multiple concatenation operations, memory is not utilised efficiently as multiple string objects are allocated and deallocated memory multiple times and memory gets wasted which directly affects the performance of the software. Understand the above scenario with the help of memory map diagram shown below:



Wondering, is there any better approach to concatenate multiple strings?

The better approach is using **join() function**.

join( ) in python joins each element of an iterable (Such as list, tuple and strings) and returns the concatenated string.

CODE:

```
lst = ["Python" , "Java" , "Django" , "Spring"]
s = "".join(lst)

print(s)
```

OUTPUT:

```
PythonJavaDjangoSpring
```

Explanation: join( ) in the above code takes all the elements present in the list and joins them into one string. Have a look at the memory map diagram shown below where in one shot join( ) concatenates all the strings and stores them into s.

lst: 1000 → 1000

0	1	2	3
Python	Java	Django	Spring

s: 2000 → 2000

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
P	y	t	h	o	n	J	a	v	a	D	j	a	n	g	o	S	p	r	i	n	g