Хранение текстов в памяти компьютера

ASCII

Мы уже знаем, что компьютер умеет хранить только двоичные числа. И для записи нечисловой информации (текстов, изображений, видео, документов) прибегают к кодированию.

Самый простой случай кодирования – сопоставление кодов текстовым символам.

Один из самых распространенных форматов такого кодирования – [таблица ASCII](#) (American standard code for information interchange).

Dec	Hex	Char	Cmd
0	00		NUL
1	01	☺	SOH
2	02	☻	STX
3	03	♥	ETX
4	04	♦	EOT
5	05	♣	ENQ
6	06	♠	ACK
7	07	•	BEL
8	08	◘	BS
9	09	○	TAB
10	0A	◙	LF
11	0B	♂	VT
12	0C	♀	FF
13	0D	♪	CR
14	0E	♫	SO
15	0F	☼	SI
16	10	►	DLE
17	11	◄	DC1
18	12	↕	DC2
19	13	‼	DC3
20	14	¶	DC4
21	15	§	NAK
22	16	▬	SYN
23	17	↨	ETB
24	18	↑	CAN
25	19	↓	EM
26	1A	→	SUB
27	1B	←	ESC
28	1C	∟	FS
29	1D	↔	GS
30	1E	▲	RS
31	1F	▼	US

Dec	Hex	Char	Cmd
32	20		(sp)
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	,	
45	2D	-	
46	2E	.	
47	2F	/	
48	30	0	
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	

Dec	Hex	Char	Cmd
64	40	@	
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	
92	5C	\	
93	5D	]	
94	5E	^	
95	5F	_	

Dec	Hex	Char	Cmd
96	60	`	
97	61	a	
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6A	j	
107	6B	k	
108	6C	l	
109	6D	m	
110	6E	n	
111	6F	o	
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	77	w	
120	78	x	
121	79	y	
122	7A	z	
123	7B	{	
124	7C	|	
125	7D	}	
126	7E	~	
127	7F	⌂	DEL

ASCII таблица

Изначально в этой таблице каждому символу был поставлен в соответствие 7-битный код, что позволяло идентифицировать 128 различных символов (128 = 2^7, 2 - потому что двоичный код состоит из двух возможных значений - либо 0, либо 1).

Этого хватало на латинские буквы обоих регистров, знаки препинания и спецсимволы — например, перевод строки или разрыв страницы. Позже код расширили до 1 байта, что позволяло хранить уже 256 различных значений(256 = 2^8, 1 байт = 8 бит): в таблицу помещались буквы второго алфавита (например, кириллица) и дополнительные графические элементы (псевдографика).

В некоторых относительно низкоуровневых языках (например, в C) можно в любой момент перейти от представления строки в памяти к последовательности байтов, начинающейся по какому-либо адресу.

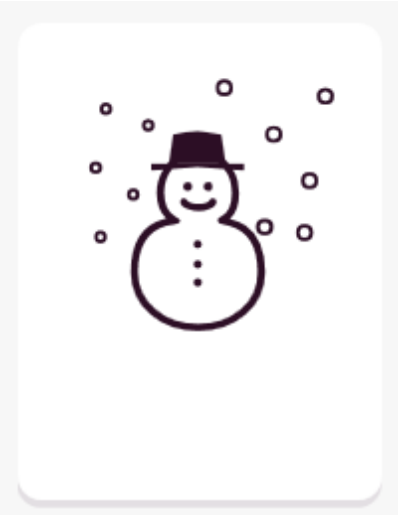
Сейчас однобайтные кодировки отошли на второй план, уступив место Юникоду.

Unicode

Юникод — таблица, которая содержит соответствия между числом и каким-либо знаком, причем количество знаков может быть любым. Это позволяет одновременно использовать любые символы любых алфавитов и дополнительные графические элементы. Кроме того, в Юникоде каждый символ, помимо кода, имеет некоторые свойства: например, буква это или цифра. Это позволяет более гибко работать с текстами.

В Юникод все время добавляются новые элементы, а сам размер этой таблицы не ограничен и будет только расти, поэтому сейчас при хранении в памяти одного юникод-символа может потребоваться от 1 до 8 байт. Отсутствие ограничений привело к тому, что стали появляться символы на все случаи жизни. Например, есть несколько снеговиков.

Вы можете его увидеть, если напишете:



```
print('\u2603')
```

Важно понять, что все строки в Python хранятся именно как последовательность юникод-символов.

Для того, чтобы узнать код какого-либо символа, существует команда **ord()**(англ. order - порядок).

Напишите:

```
print(ord('И'))
```

Зная код, всегда можно получить соответствующий ему символ. Для этого существует функция **chr()** (англ. character - символ).

Напишите:

```
print(chr(1048))
```

Функции **ord()** и **chr()** часто работают в паре. Попробуйте, например, предположить, что будет выведено на экран в результате работы следующей программы:

```
for i in range(26):
  print(chr(ord('A') + i))
```

Настоятельно рекомендуем сначала подумать, и только потом писать этот код :).

UTF-8, или в чем разница между UTF-8 и Unicode

Сейчас, чаще всего, нам попадается на глаза кодировка UTF-8. Чем она отличается от Unicode?

На самом деле, сравнивать UTF-8 и Unicode - это как сравнивать яблоки и апельсины. Кстати, UTF переводится как *Unicode Transformation Format* - формат преобразования Юникода. Хм))

- UTF-8 - это кодировка;
- Unicode - это набор символов.

Что такое набор символов, мы с вами уже разобрали. Для наглядности, еще раз, другими словами, повторим - что такое кодировка:

Кодировка — это алгоритм, который преобразует числа (номера в наборе символов) в двоичный код, понятный машине. Например, последовательность «1 2 3 4» в кодировке UTF-8 запишется как:

```
00000001 00000010 00000011 00000100
```

Разберем пример. Допустим, приложение считает с диска следующую информацию:

```
1101000 1100101 1101100 1101100 1101111
```

Приложение «знает», что эти данные представляют собой строку в формате Unicode, закодированную в кодировке UTF-8, поэтому, на первом этапе, оно преобразует двоичные данные в числа, используя алгоритм UTF-8. В результате получится следующее:

```
104 101 108 108 111
```

Мы с вами уже знаем, что этот набор чисел - не что иное, как unicode-строка. А значит, теперь в роль вступает таблица символов - Unicode, она представит каждое число как символ из таблицы и выведет такой результат:

```
hello
```

Подытожим:

1. Кодировки бывают разные.
2. Самая распространенная - UTF-8.
3. Кодировки - это алгоритмы, которые переводят двоичный код в символы и обратно. Например, слово “ок” с помощью кодировка ASCII будет выглядеть в двоичном формате так - 01101111 01101011.
4. UTF-8 - это кодировка, работающая с бОльшим количеством символов, чем ASCII. Но они совместимы.
5. Unicode - это таблица, по которой подбирается символ с помощью номера, преобразованный из битового(двоичного) формата с помощью алгоритма кодировки (ASCII // UTF-8, и др.)

Практические упражнения

Задание 1. Какая-то там буква

Напишите программу, которая считывает сообщение, затем номер. После этого программа выводит букву из сообщения с таким номером, причем считается, что номера букв отсчитываются с единицы.

Если введенное число не является правильным номером буквы, вывести “ОШИБКА”.

Формат ввода:

В первой строке записано сообщение, во второй - номер буквы.

Формат вывода:

Одна буква или сообщение “ОШИБКА”.

Пример:

Ввод	Вывод
Гендальф 3	н
Ну, привет! -10	ОШИБКА

Примечание:

В этой задаче использование функций chr() и ord() не обязательно.

Задание 2. Цезарь его знает

Как известно, Цезарь тоже пользовался шифрованием сообщений, причем у него был свой способ. Сначала выбирается шаг шифрования (число), а затем все буквы послания заменяются на буквы, отстоящие от них в алфавите на шаг шифрования. Например, при шаге шифрования 3 (таким чаще всего пользовался Цезарь), буква А заменяется на букву Г, а буква Б - на букву Д.

Обратите внимание, что алфавит “зациклен”, то есть при сдвиге буквы Я на шаг 3 получится буква В.

Напишите программу, которая будет зашифровывать послание с помощью шифра Цезаря с заданным шагом.

Формат ввода:

Две строки. Первая содержит шаг шифрования, вторая - послание.

Формат вывода:

Строка с зашифрованным посланием.

Пример:

Ввод	**Вывод**
3 АБВ	ГДЕ
5 На дворе трава, на траве дрова!	Те йзухк чхезе, ту чхезк йхузе!

Примечания:

Символы русского алфавита расположены в стандартной для Python таблице кодировки подряд, то есть номера, выдаваемые функцией ord(symbol), идут подряд. Буква “ё” идет в таблице кодировки отдельно от основного алфавита. При решении задачи считайте, что буквы “ё” в русском алфавите нет.