10. . . の散歩道

10.1 ascii コード

さて、散策したことを振り返りながら家路につこう。

われわれが日常使う数は 10 進数である。しかしコンピュータは内部では 2 進数が使われ、それを人が見て分かりやすいように 16 進数で表すことが多い。ところで、アルファベットの ‘A’ は 16 進数—もしくは 16 進コード—では `0x41` である。その証拠に

[py script]
```
>>> chr(0x41)
'A'
```

と表示されるだろう。ちなみに `0x` は 16 進数であることを示す符丁だ。おや？　もしかして 16. . . の散歩道にでも迷い込んじゃったかな？　ま、いいや。逆に ‘A’ のコードはいくつだろうか。それは `ord()` で知ることができる。

[py script]
```
>>> ord('A')
65
```

あれ？　`0x41` じゃないの？　いいや、合ってるよ。$(65)_{10} = 4 \times 16 + 1 = (41)_{16}$ だから。実は `ord()` は 10 進数の **ascii コード**を返すのだ。ああ、よかった。やっぱり 10. . . の散歩道を歩いていたんだ。

で、何を言いたいかというと、文字は数値にできるということだ。文字が数値になって役立つことに**暗号**がある。文字を別の文字に置き換えても暗号にはなるが、数値であれば計算ができるのだ。だから、たとえば

[py script]
```
>>> chr(65+32)
'a'
```

なんてことができる。こうなる理由は ascii コードの表で、大文字のアルファベットの 32 後ろに小文字のアルファベットが登録されているからである。しかし、この程度のことでは暗号化の恩恵に

あずかれない。単に文字を数値に置き換えるだけの暗号は脆弱（ぜいじゃく）だからだ。

そこで、この道では**公開鍵暗号**について散策してみよう。と言っても、大筋をなぞる程度にね。

では、まずはじめにアルファベットを ascii コードに、また ascii コードをアルファベットに変換するスクリプトを考えよう。こういうときに使う文字列は “Hello, world!” が最適だな。

[py script]

```
>>> def chr2ord(s):
...   for c in s:
...     print(ord(c), end='')
...   print('\n')
...
>>> chr2ord('Hello, world!')
72101108108111443211911111410810033
```

アルファベットは 1 文字ずつ `ord()` によって変換する必要があるので、`s` から `c` を順次取り出している。この記述は以前出会っているね。とくに問題となるところはないだろう。以上。

次は、ここで出力された数字の羅列をアルファベットに変換するプログラムだ。でも、今度は少し考えなくてはならない。なぜなら、アルファベットは必ずしも 2 桁の数字列ではないからだ。実際、`72101...` で始まる数字列を見て分かるように、続く `108108` が文字列 `ll` なのだろう。すると `101 = e` だ。さっき指摘したように、大文字アルファベットの 32 後ろに小文字アルファベットが登録されている。むむ。アルファベットは 26 文字だから `Z` と `a` の間には何かあるのか？

それを知るためには、ascii コードの表を見なくちゃ。

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
	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	:	;	<	=	>	?
	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
	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	[	\	]	^	_
	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
	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	{	|	}	~	
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	

この表は 16 進数の `0x20` から `0x7E` に登録されている文字の一覧である[1]。上段の文字の下の数字が 10 進数の ascii コードである。それは 32 から 126 までの数字なので、`72101...` で始まる数字

[1] `0x20` より前にはエスケープシーケンスなどの特殊文字が登録されている。

列を順に読み出したとき、それが 3 から 9 までの数字ならば 2 桁の数値の先頭、それが 1 ならば 3 桁の数値の先頭、と見ればよいことになる。そのことを考慮してスクリプトは次のようになった。

[py script]

```
>>> def ord2chr(s):
...   str = ''; i = 0
...   while s[i:] != '':
...     if s[i] == '1':
...       str += chr(int(s[i:i+3]))
...       i += 3
...     else:
...       str += chr(int(s[i:i+2]))
...       i += 2
...   print(str)
...
>>> ord2chr('721011081081114432119111111410810033')
Hello, world!
```

やりたいことは、文字列の先頭から順に 2 文字または 3 文字を切り出して、得られた “数値” を文字に直していくことである。そこで最初に、出力する文字列変数 `str` と切り出す位置を示す変数 `i` を用意した。

3 桁の数値にすべき場合は、切り出す先頭が `'1'` のときに限るので、`if s[i] == '1':`を条件としている。このときは `i` から 3 文字分を数値に直し、`chr()` によって文字に変換するのである。`else` 節は 2 文字の場合の処理だ。当然、次の切り出しのために `i` の値は進めておく必要がある。

これを、次に切り出す文字がない場所まで続ければよいので、判定は `while s[i:] != '':`とした。

さっきコード化した数字列を文字列にしてみよう。ちゃんと “Hello, world!” になったね。