Códigos

Aprenda várias formas de representar dados no computador e suas aplicações em CTFs.

- Dados e códigos
- Código ASCII
- Base64

Dados e códigos

Nos computadores, todos os dados são guardados da mesma forma: em **código binário**. Esse código é como uma sequência de '0's e '1's, e essas sequências são agrupadas e sofrem modificações para representar tudo o que vemos em um computador.

Mas se tudo são apenas zeros e uns, o que diferencia um texto de uma música? A diferença é o modo como esses dados são **representados**.

Para cada tipo de dado existe uma codificação diferente aplicada a ela, e por isso um mesmo dado pode ter várias interpretações dependendo da codificação que você trabalha.

Um exemplo de uma representação de um valor binário é como um número.

Um número em binário pode ser convertido para um número em **decimal**. Por exemplo, `00101010` é o número `42` em decimal.

Uma representação mais comum para um código binário é a base **hexadecimal**, que tem a vantagem de que cada casa pode representar uma sequência de 4 bits. Por exemplo, o mesmo número `00101010` é `2A` em hexadecimal (0010 = 2 e 1010 = A).

Nas próximas seções, você verá outras formas de representar os dados no computador.

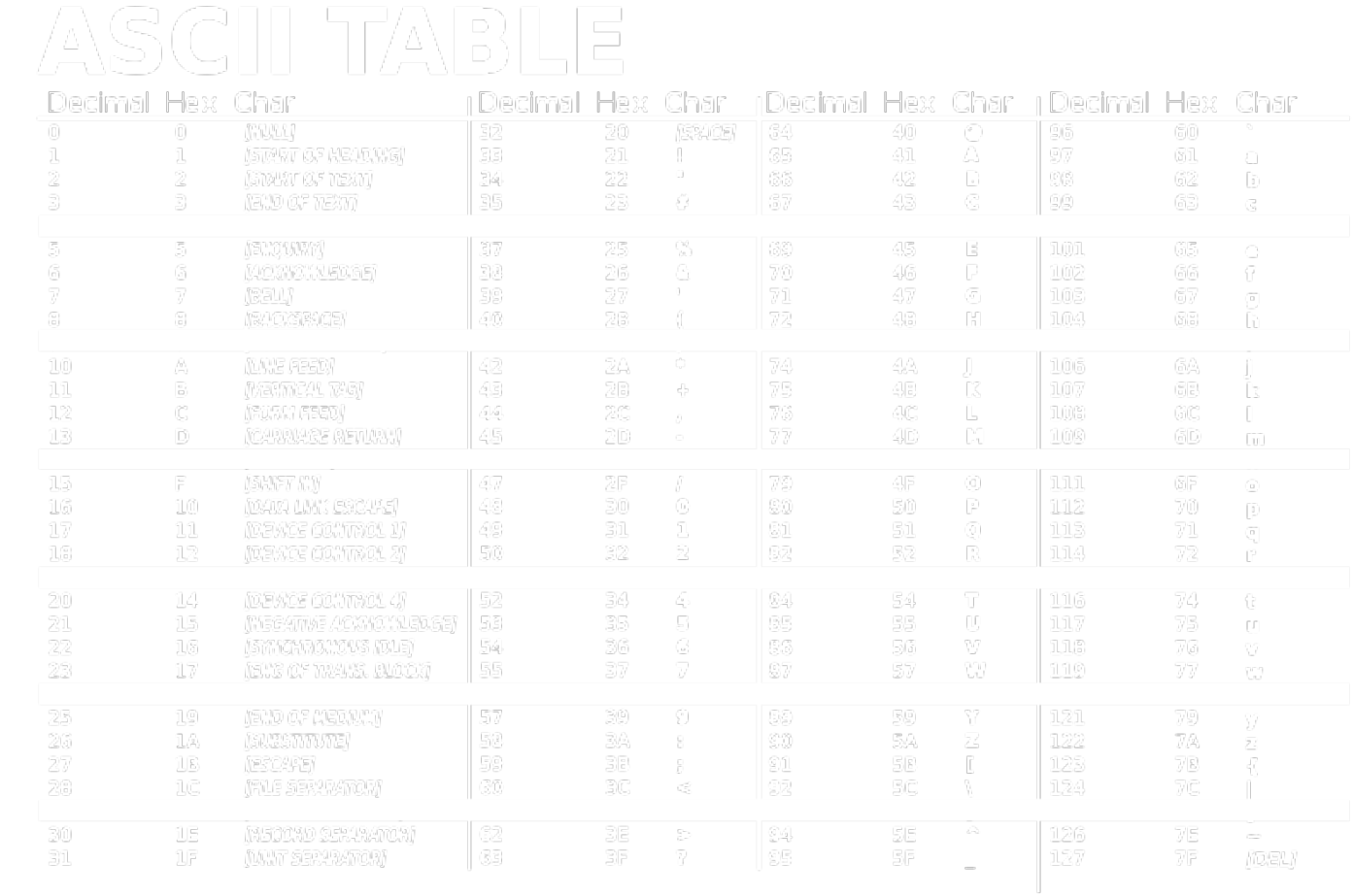
Código ASCII

Nessa seção, vamos apresentar uma das codificações mas famosas para representar texto em um computador: o ASCII.

> O nome ASCII vem do inglês *American Standard Code for Information Interchange* que significa "Código Padrão Americano para o Intercâmbio de Informação"

O ASCII, originalmente baseado no inglês, codifica 128 caracteres específicos com **7 bits**. Como um computador normalmente trabalha na escala de bytes (8 bits), o ASCII é mais frequentemente encontrado numa representação de **8 bits**.

A tabela abaixo mostra todos os caracteres do código ASCII.



Você pode usar essa tabela para decodificar uma sequência de números em binário, decimal ou hexadecimal para ASCII.

ASCII em Python

Uma maneira mais fácil de manipular a codificação ASCII, em vez de usar manualmente uma tabela, é por meio de códigos. Em Python, usamos as funções `ord()` e `chr()` para isso.

A função `ord()` recebe uma string de tamanho 1 e retorna um inteiro que representa o código da letra, se ela for ASCII, devolverá seu código ASCII. Por exemplo, `ord('a')` devolverá `97`.

Já a função `chr()` é o inverso da anterior. Ela recebe um inteiro e devolve o caractere com o respectivo código ASCII. Por exemplo, `chr(97)` devolverá `'a'`.

Exercícios

WeChall: ASCII

WeChall: URL

Referências

ASCII table

Python Built-in Functions

Base64

Agora que você já teve contato com o código ASCII, vamos conhecer o **Base64**, um método para codificar e decodificar dados binários em caracteres ASCII.

Esse método é utilizado frequentemente para transferir dados em meios que só suportam formatos ASCII, por exemplo, enviar anexos por e-mail (que usa o MIME).

O nome base64 origina-se do fato de que esse sistema é constituido de 64 caracteres, representando exatamente 6 bits de dados. Com isso, três bytes de 8 bits podem ser representados por 4 digitos de 6 bits em Base64.

A tabela abaixo mostra a equivalência entre os valores de um conjunto de 6 bits e os caracteres usados para codificação.

Valor	Caractere	Valor	Caractere	Valor	Caractere	Valor	Caractere
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Quando os bits da mensagem original não são multiplos de 6, são adicionados zeros como *padding* . Assim, na mensagem codificada é colocado um `=` para cada dois zeros de *padding*. Aliás, esse `=` é uma forma bem comum de reconhecer um texto codificado em Base64.

Abaixo está um exemplo de um texto codificado em Base64:

```
texto:           M        |        a        |        n
8 bits:   0 1 0 0 1 1 0 1 | 0 1 1 0 0 0 0 1 | 0 1 1 0 1 1 1 0
6 bits:   0 1 0 0 1 1 | 0 1 0 1 1 0 | 0 0 0 1 0 1 | 1 0 1 1 1 0
Valor:         19     |      22     |      5      |      46
texto:         T      |      W      |      F      |      u
(Base64)
```

E se tirarmos o `n`, a codificação vai necessitar de um *padding*:

```
texto:           M        |        a        |
8 bits:   0 1 0 0 1 1 0 1 | 0 1 1 0 0 0 0 1 |
6 bits:   0 1 0 0 1 1 | 0 1 0 1 1 0 | 0 0 0 1 0 0 | 0 0 0 0 0 0
Valor:         19     |      22     |      5      |  (padding)
texto:         T      |      W      |      E      |      =
(Base64)
```

Ferramentas

Para manipular textos em Base64, pode-se usar o comando Unix `base64`.

Por exemplo, se tivermos uma arquivo `sagan.txt` com o texto `The Cosmos is all that is or ever was or ever will be`. Podemos convertê-lo para um arquivo `sagan64.txt` com o comando

```
base64 sagan.txt > sagan64.txt
```

O resultado será o arquivo `sagan64.txt` com o texto `VGhlIENvc21vcyBpcyBhbGwgdGhhdCBpcyBvciBldmVyIHdhcyBvciBldmVyIHdpbGwgYmUK`.

Agora, para decodificar o arquivo `sagan64.txt`, usamos o mesmo comando com a flag `-d`:

```

```

```
base64 -d sagan64. txt
```

Exercícios

OverTheWire: Krypton 0

Decodifique essa mensagem