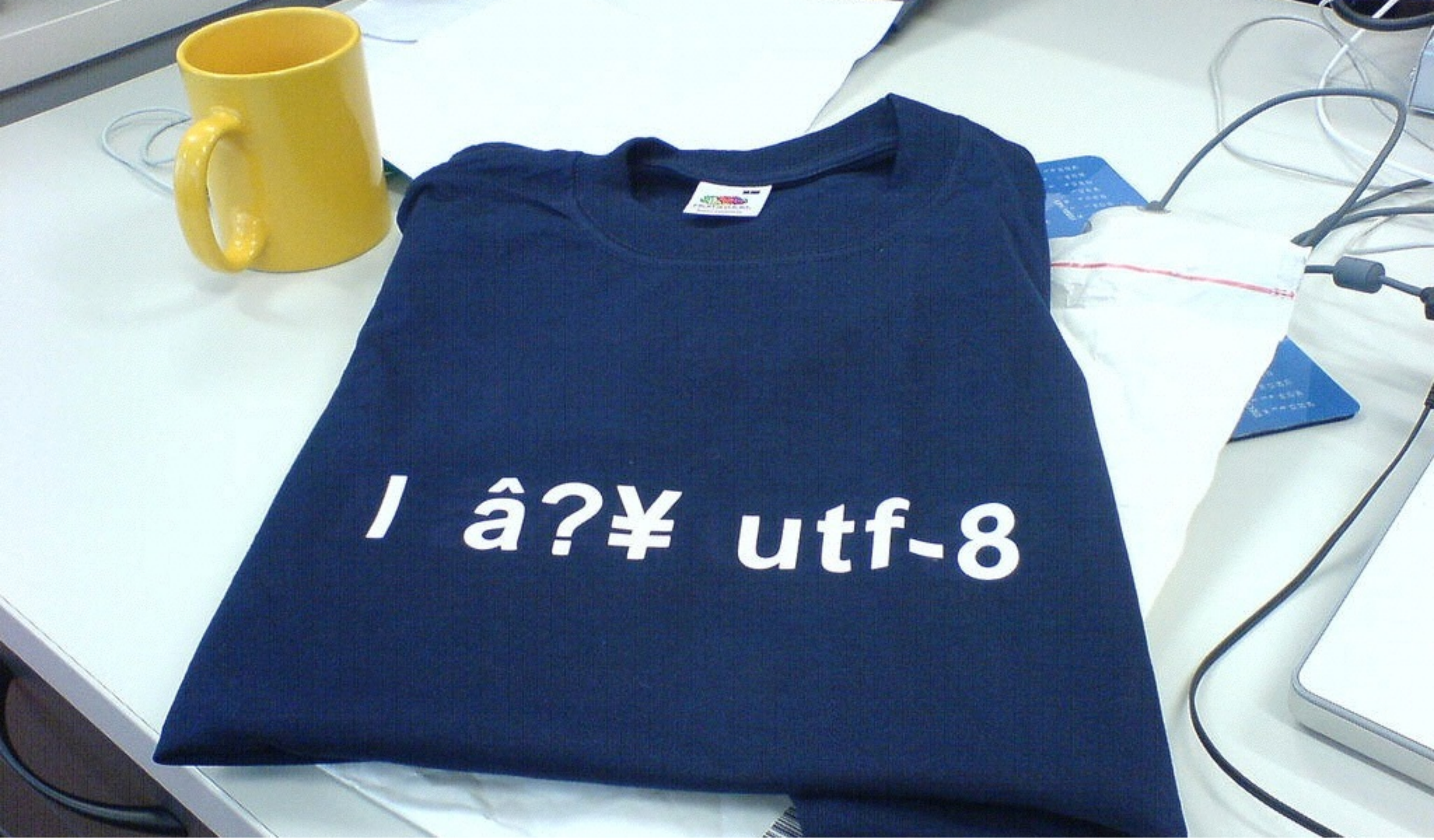
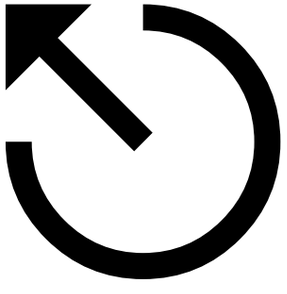


Do you want to print a ☃?

Mark Fowler, Albany Perl Mongers

Perl String Escapes Recap



é

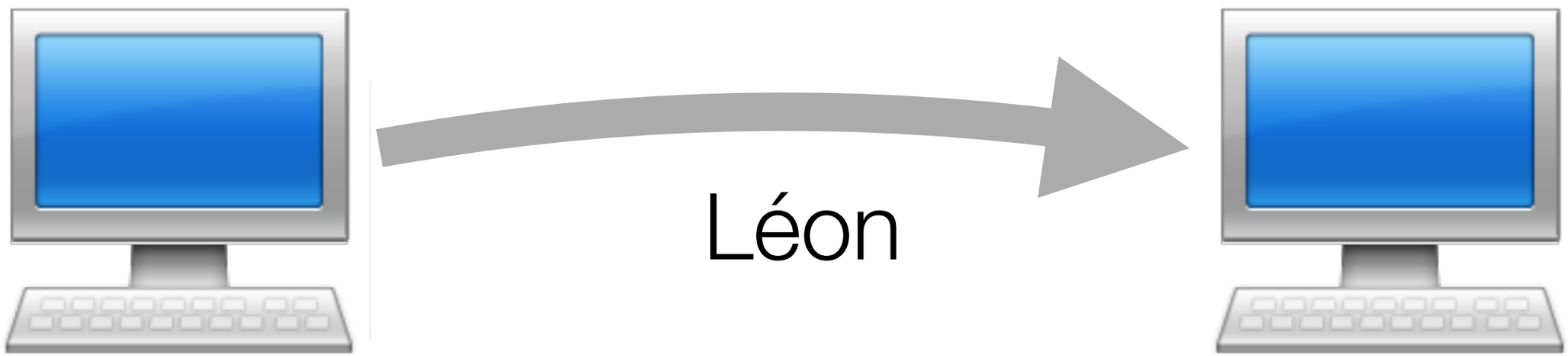
```
# hex
$eacute = "\x{e9}";

# oct (<256 only)
$eacute = "\351";

# name
$eacute = "\N{LATIN SMALL LETTER E WITH ACUTE}";
```

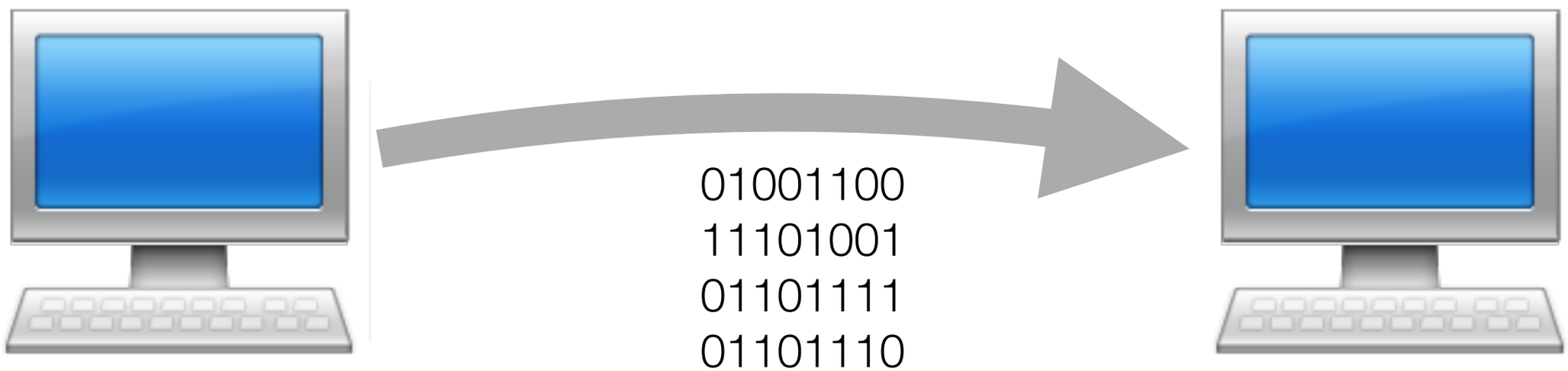
```
# hex
$snowman = "\x{2603}";
```

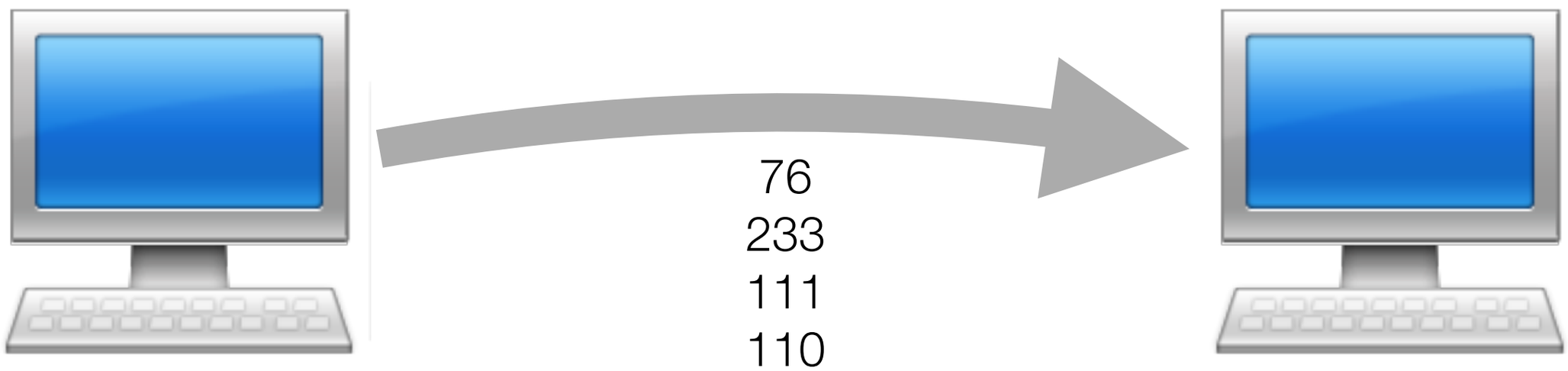
```
# name
$snowman = “\N{SNOWMAN}";
```

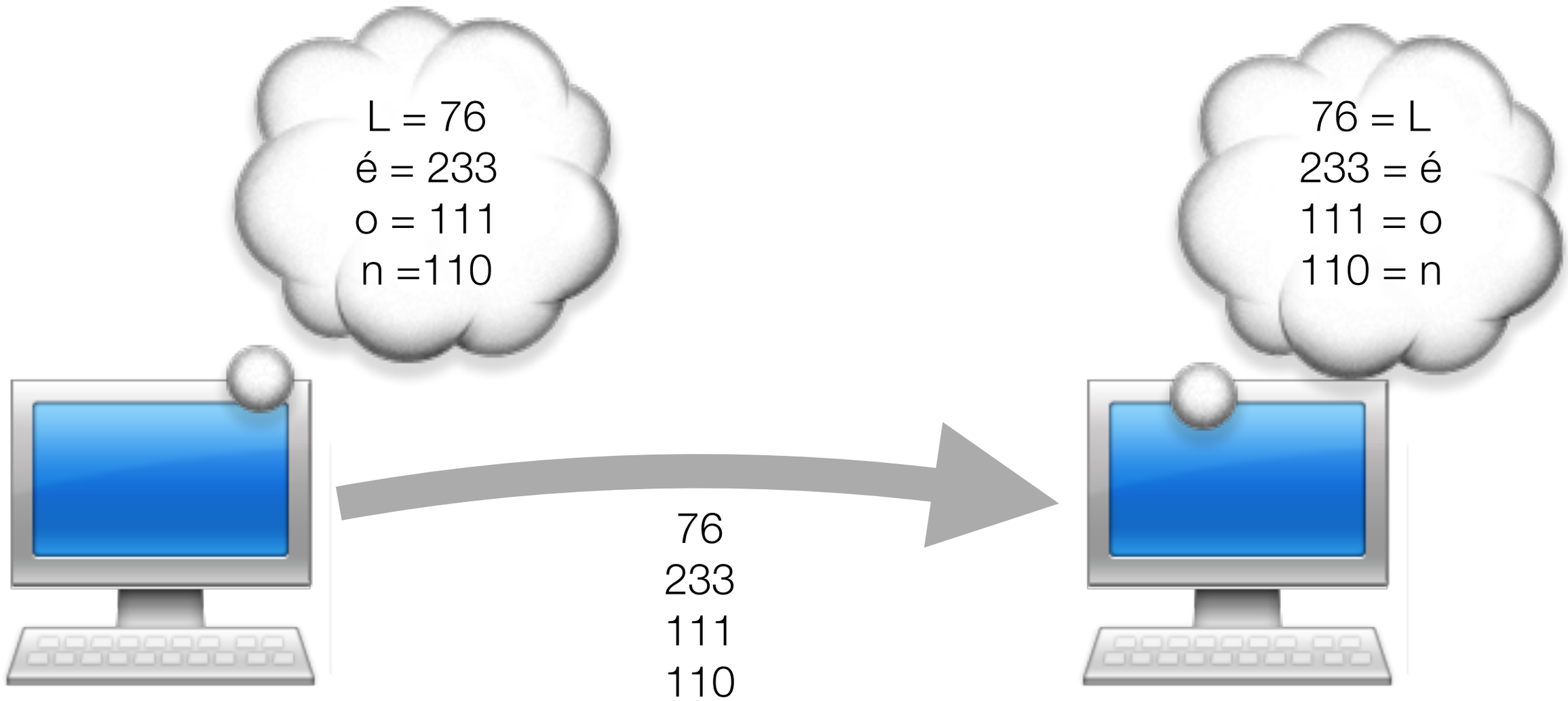
Character Encoding Recap

Léon


Léon


01001100
11101001
01101111
01101110


76
233
111
110


L = 76
é = 233
o = 111
n =110
76
233
111
110
76 = L
233 = é
111 = o
110 = n

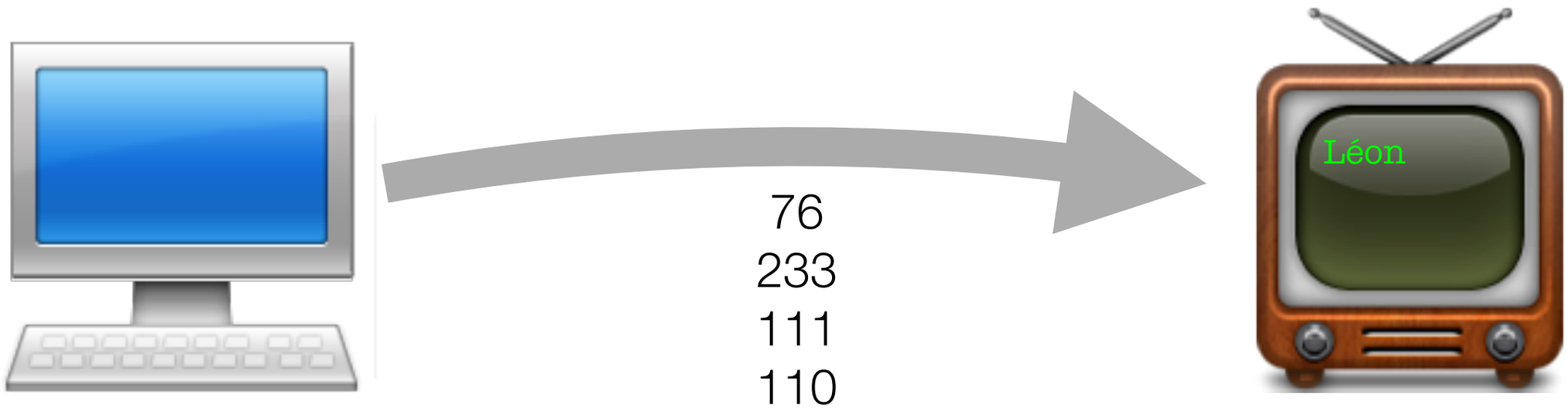
76
233
111
110

76
233
111
110

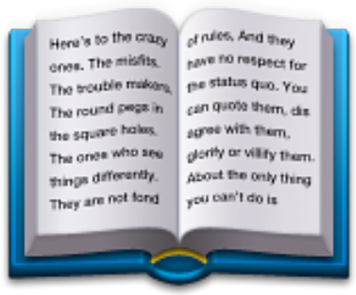
Dear Karla,

Here's to the crazy ones. The misfits. The rebels. The troublemakers. The round pegs in the square holes. The ones who see things differently. They're not fond of rules. And they have no respect for the status quo. You can quote them, disagree with them, glorify or vilify them. About the only thing you can't do is ignore them. Because they change things. They push the human race forward. And while some may see them as the crazy ones, we see geniuses. Because the people who are crazy enough to think they can change the world, are the ones who do.

Take care.
John Appleseed


Léon
76
233
111
110

Terminology



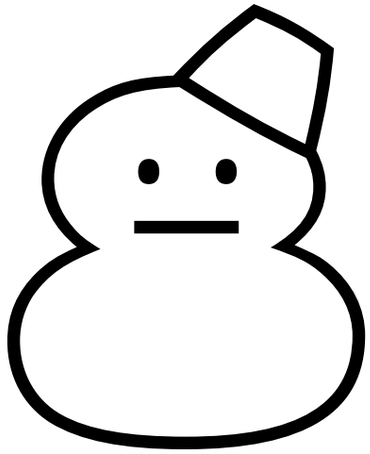
Character

A single letter or symbol

*This is a lie, but a handy one

a

é



Character String

One letter or symbol followed by another, followed by another, and so on…an array of characters

*This is also a lie, but a handy one

anna

elsa

Characters are abstract

Byte

An eight bit number, between 0-255 inclusive

72

144

253

Byte String

A chunk of memory, a series of bytes, that may somehow have characters encoded in it…an array of bytes

76 233

111 110

Bytes are what your code reads

Bytes are what your code prints

Encoding

Going from a Character String to a Byte String

Decoding

Going from a Byte String to a Character String

Character Set Encoding

What bytes are used to represent a character in a byte string

Common Character Sets

ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0 \000 0x00 NUL	1 \001 0x01 SOH	2 \002 0x02 STX	3 \003 0x03 ETX	4 \004 0x04 EOT	5 \005 0x05 ENQ	6 \006 0x06 ACK	7 \007 0x07 BEL	8 \010 0x08 BS	9 \011 0x09 HT	10 \012 0x0A LF	11 \013 0x0B VT	12 \014 0x0C FF	13 \015 0x0D CR	14 \016 0x0E SO	15 \017 0x0F SI	0
1	16 \020 0x10 DLE	17 \021 0x11 DC1	18 \022 0x12 DC2	19 \023 0x13 DC3	20 \024 0x14 DC4	21 \025 0x15 NAK	22 \026 0x16 SYN	23 \027 0x17 ETB	24 \030 0x18 CAN	25 \031 0x19 EM	26 \032 0x1A SUB	27 \033 0x1B ESC	28 \034 0x1C FS	29 \035 0x1D GS	30 \036 0x1E RS	31 \037 0x1F US	1
2	32 \040 0x20 SP	33 \041 0x21 !	34 \042 0x22 "	35 \043 0x23 #	36 \044 0x24 $	37 \045 0x25 %	38 \046 0x26 &	39 \047 0x27 '	40 \050 0x28 (	41 \051 0x29)	42 \052 0x2A *	43 \053 0x2B +	44 \054 0x2C ,	45 \055 0x2D -	46 \056 0x2E .	47 \057 0x2F /	2
3	48 \060 0x30 0	49 \061 0x31 1	50 \062 0x32 2	51 \063 0x33 3	52 \064 0x34 4	53 \065 0x35 5	54 \066 0x36 6	55 \067 0x37 7	56 \070 0x38 8	57 \071 0x39 9	58 \072 0x3A :	59 \073 0x3B ;	60 \074 0x3C <	61 \075 0x3D =	62 \076 0x3E >	63 \077 0x3F ?	3
4	64 \100 0x40 @	65 \101 0x41 A	66 \102 0x42 B	67 \103 0x43 C	68 \104 0x44 D	69 \105 0x45 E	70 \106 0x46 F	71 \107 0x47 G	72 \110 0x48 H	73 \111 0x49 I	74 \112 0x4A J	75 \113 0x4B K	76 \114 0x4C L	77 \115 0x4D M	78 \116 0x4E N	79 \117 0x4F O	4
5	80 \120 0x50 P	81 \121 0x51 Q	82 \122 0x52 R	83 \123 0x53 S	84 \124 0x54 T	85 \125 0x55 U	86 \126 0x56 V	87 \127 0x57 W	88 \130 0x58 X	89 \131 0x59 Y	90 \132 0x5A Z	91 \133 0x5B [	92 \134 0x5C \	93 \135 0x5D]	94 \136 0x5E ^	95 \137 0x5F _	5
6	96 \140 0x60 `	97 \141 0x61 a	98 \142 0x62 b	99 \143 0x63 c	100 \144 0x64 d	101 \145 0x65 e	102 \146 0x66 f	103 \147 0x67 g	104 \150 0x68 h	105 \151 0x69 i	106 \152 0x6A j	107 \153 0x6B k	108 \154 0x6C l	109 \155 0x6D m	110 \156 0x6E n	111 \157 0x6F o	6
7	112 \160 0x70 p	113 \161 0x71 q	114 \162 0x72 r	115 \163 0x73 s	116 \164 0x74 t	117 \165 0x75 u	118 \166 0x76 v	119 \167 0x77 w	120 \170 0x78 x	121 \171 0x79 y	122 \172 0x7A z	123 \173 0x7B {	124 \174 0x7C |	125 \175 0x7D }	126 \176 0x7E ~	127 \177 0x7F DEL	7
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Latin-1

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	€ 20AC		‚ 201A		„ 201E	… 2026	† 2020	‡ 2021		‰ 2030	Š 0160	‹ 2039	Ś 015A	Ť 0164	Ž 017D	Ź 0179
90		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014		™ 2122	š 0161	› 203A	ś 015B	ť 0165	ž 017E	ź 017A
A0	NBSP 00A0	ˇ 02C7	˘ 02D8	Ł 0141	¤ 00A4	Ą 0104	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	Ş 015E	« 00AB	¬ 00AC	- 00AD	® 00AE	Ż 017B
B0	° 00B0	± 00B1	˛ 02DB	ł 0142	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	ą 0105	ş 015F	» 00BB	Ľ 013D	˝ 02DD	ľ 013E	ż 017C
C0	Ŕ 0154	Á 00C1	Â 00C2	Ă 0102	Ä 00C4	Ĺ 0139	Ć 0106	Ç 00C7	Č 010C	É 00C9	Ę 0118	Ë 00CB	Ě 011A	Í 00CD	Î 00CE	Ď 010E
D0	Đ 0110	Ń 0143	Ň 0147	Ó 00D3	Ô 00D4	Ő 0150	Ö 00D6	× 00D7	Ř 0158	Ů 016E	Ú 00DA	Ű 0170	Ü 00DC	Ý 00DD	Ţ 0162	ß 00DF
E0	ŕ 0155	á 00E1	â 00E2	ă 0103	ä 00E4	ĺ 013A	ć 0107	ç 00E7	č 010D	é 00E9	ę 0119	ë 00EB	ě 011B	í 00ED	î 00EE	ď 010F
F0	đ 0111	ń 0144	ň 0148	ó 00F3	ô 00F4	ő 0151	ö 00F6	÷ 00F7	ř 0159	ů 016F	ú 00FA	ű 0171	ü 00FC	ý 00FD	ţ 0163	˙ 02D9

Latin-1

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	€ 20AC		‚ 201A		„ 201E	… 2026	† 2020	‡ 2021		‰ 2030	Š 0160	‹ 2039	Ś 015A	Ť 0164	Ž 017D	Ź 0179
90		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014		™ 2122	š 0161	› 203A	ś 015B	ť 0165	ž 017E	ź 017A
A0	NBSP 00A0	ˇ 02C7	˘ 02D8	Ł 0141	¤ 00A4	Ą 0104	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	Ş 015E	« 00AB	¬ 00AC	- 00AD	® 00AE	Ż 017B
B0	° 00B0	± 00B1	˛ 02DB	ł 0142	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	ą 0105	ş 015F	» 00BB	Ľ 013D	˝ 02DD	ľ 013E	ż 017C
C0	Ŕ 0154	Á 00C1	Â 00C2	Ă 0102	Ä 00C4	Ĺ 0139	Ć 0106	Ç 00C7	Č 010C	É 00C9	Ę 0118	Ë 00CB	Ě 011A	Í 00CD	Î 00CE	Ď 010E
D0	Đ 0110	Ń 0143	Ň 0147	Ó 00D3	Ô 00D4	Ő 0150	Ö 00D6	× 00D7	Ř 0158	Ů 016E	Ú 00DA	Ű 0170	Ü 00DC	Ý 00DD	Ţ 0162	ß 00DF
E0	ŕ 0155	á 00E1	â 00E2	ă 0103	ä 00E4	ĺ 013A	ć 0107	ç 00E7	č 010D	é 00E9	ę 0119	ë 00EB	ě 011B	í 00ED	î 00EE	ď 010F
F0	đ 0111	ń 0144	ň 0148	ó 00F3	ô 00F4	ő 0151	ö 00F6	÷ 00F7	ř 0159	ů 016F	ú 00FA	ű 0171	ü 00FC	ý 00FD	ţ 0163	˙ 02D9

Same
As ASCII

Latin-1

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F
80	€ 20AC		‚ 201A		„ 201E	… 2026	† 2020	‡ 2021		‰ 2030	Š 0160	‹ 2039	Ś 015A	Ť 0164	Ž 017D	Ź 0179
90		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014		™ 2122	š 0161	› 203A	ś 015B	ť 0165	ž 017E	ź 017A
A0	NBSP 00A0	ˇ 02C7	˘ 02D8	Ł 0141	¤ 00A4	Ą 0104	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	Ş 015E	« 00AB	¬ 00AC	- 00AD	® 00AE	Ż 017B
B0	° 00B0	± 00B1	˛ 02DB	ł 0142	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	ą 0105	ş 015F	» 00BB	Ľ 013D	˝ 02DD	ľ 013E	ż 017C
C0	Ŕ 0154	Á 00C1	Â 00C2	Ă 0102	Ä 00C4	Ĺ 0139	Ć 0106	Ç 00C7	Č 010C	É 00C9	Ę 0118	Ë 00CB	Ě 011A	Í 00CD	Î 00CE	Ď 010E
D0	Đ 0110	Ń 0143	Ň 0147	Ó 00D3	Ô 00D4	Ő 0150	Ö 00D6	× 00D7	Ř 0158	Ů 016E	Ú 00DA	Ű 0170	Ü 00DC	Ý 00DD	Ţ 0162	ß 00DF
E0	ŕ 0155	á 00E1	â 00E2	ă 0103	ä 00E4	ĺ 013A	ć 0107	ç 00E7	č 010D	é 00E9	ę 0119	ë 00EB	ě 011B	í 00ED	î 00EE	ď 010F
F0	đ 0111	ń 0144	ň 0148	ó 00F3	ô 00F4	ő 0151	ö 00F6	÷ 00F7	ř 0159	ů 016F	ú 00FA	ű 0171	ü 00FC	ý 00FD	ţ 0163	˙ 02D9

Just in Latin-1

Unicode

1,111,998 Different Characters

109,384 assigned in version 6.0

First 256 are the same as Latin-1

Unicode

1,111,998 Different Characters

109,384 assigned in version 6.0
First 256 are the same as Latin-1

byte != codepoint

Since a single byte can only
map to 256 code points

Unicode

1,111,998 Different Characters

109,384 assigned in version 6.0
First 256 are the same as Latin-1

byte != codepoint

Since a single byte can only
map to 256 code points

multi-byte encodings

UTF-8

```
a  →  \x{61}
F  →  \x{70}

é  →  \x{c3}\x{a9}
☃  →  \x{e2}\x{98}\x{83}
```

UTF-8

```
a  →  \x{61}
F  →  \x{70}

é  →  \x{c3}\x{a9}
☃  →  \x{e2}\x{98}\x{83}
```

UTF-8

```
a  →  \x{61}
F  →  \x{70}

é  →  \x{c3}\x{a9}
☃  →  \x{e2}\x{98}\x{83}
```

Perl Unicode Basics

Meanwhile In Java

```
byte[] bytes = getBytes();
String str = new String(bytes, "UTF-8");
```

Meanwhile In Java

```
byte[] bytes = getBytes();
String str = new String(bytes, "UTF-8");
```

Meanwhile In Java

```
byte[] bytes = getBytes();
String str = new String(bytes, "UTF-8");
```

Meanwhile In Java

```
byte[] bytes = getBytes();
String str = new String(bytes, "UTF-8");
```

Back in Perl: Just Scalars

```
use Encode qw(decode);

my $bytes = getBytes();
$str = decode("utf-8", $bytes);
```

Back in Perl: Just Scalars

```
use Encode qw(decode);

my $bytes = getBytes();
$str = decode("utf-8", $bytes);
```

It’s all up to you

- You must remember which scalars are:
  - Character Strings
  - Byte Strings
- You must remember which characters are:
  - Really Characters
  - Characters containing bytes

Back in Perl: Just Scalars

```
use Encode qw(decode);

my $bytes = getBytes();
$str = decode("utf-8", $bytes);
```

Back in Perl: Just Scalars

```
use Encode qw(encode decode);

my $bytes = getBytes();
$str = decode("utf-8", $bytes);
my $bytes2 = encode("utf-8", $str);
```

Automatic With Filehandles

```
open my $fh1, $filename_to_read;
my $string = decode(scalar(<$fh1>), “utf-8");

open my $fh2, $filename_to_write;
print $fh2, encode(“utf-8”,$string);
```

Assume autodie is in effect, hence no error checking

Automatic With Filehandles

```
open my $fh1, "<:utf-8", $filename_to_read;
my $string = <$fh1>;

open my $fh2, ">:utf-8", $filename_to_write;
print $fh2, $string
```

Assume autodie is in effect, hence no error checking

Automatic With Filehandles

```
open my $fh1, "<:utf-8", $filename_to_read;
my $string = <$fh1>;

open my $fh2, ">:utf-8", $filename_to_write;
print $fh2, $string
```

Assume autodie is in effect, hence no error checking

Automatic With Filehandles

```
binmode $fh1, “utf-8";
binmode $fh2, “utf-8";
```

Assume autodie is in effect, hence no error checking

“Th-th-th-that's all, folks!”

– Porky Pig

As if. What if something goes wrong?

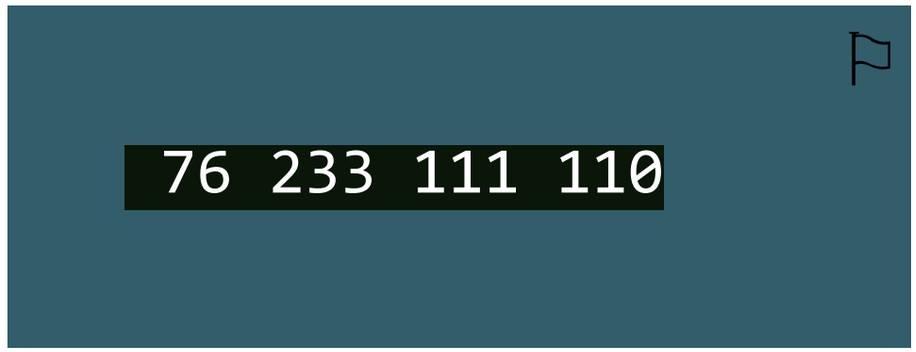
Behind The Curtain

Behind the Curtain

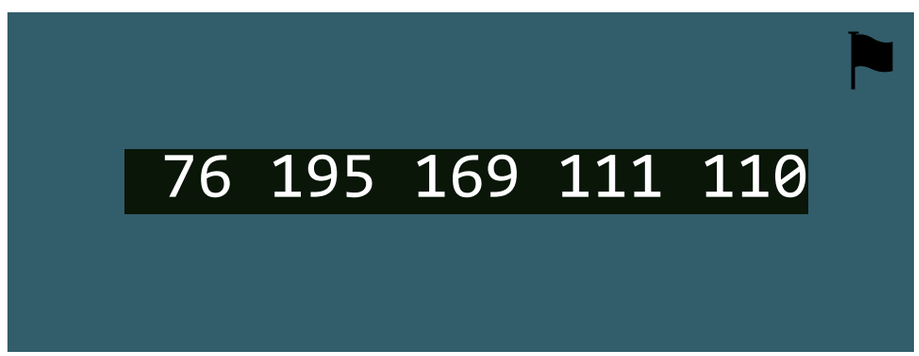
```
76 233 111 110
```

→ Léon

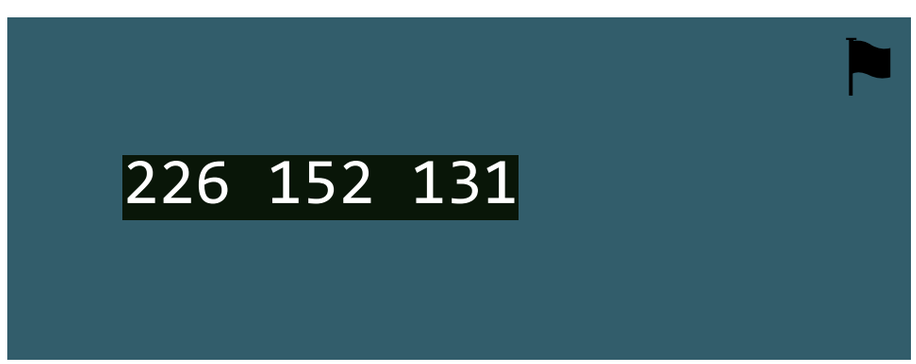
Behind the Curtain



→ Léon



→ Léon



→ ☃

Debugging Perl Unicode

“There’s always another encoding bug, you just haven’t found it yet”

– Fowler’s Rule on Encoding

bytey

bytey

```
Mark@travis:~$ cat /tmp/example.txt
Sven the møøse and Olaf the ☃
Mark@travis:~$ bytey /tmp/example.txt
Sven the m\x{c3}\x{b8}\x{c3}\x{b8}se and Olaf the \x{e2}\x{98}\x{83}
Mark@travis:~$ cat /tmp/example.txt | bytey
Sven the m\x{c3}\x{b8}\x{c3}\x{b8}se and Olaf the \x{e2}\x{98}\x{83}
Mark@travis:~$
```

http://tinyurl.com/getbytey

*yes, I know Sven is a reindeer

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
    s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
        colored(
            join(
              "",
              map {
                   '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split//, $1)
            ),
            'yellow',
       )
    }eg;
    print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
    s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
        colored(
            join(
              "",
              map {
                   '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split//, $1)
            ),
            'yellow',
       )
    }eg;
    print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
     s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
         colored(
             join(
               "",
               map {
                    '\\x{'. sprintf('%02x',ord($_)) . '}'
               } split//, $1)
             ),
             'yellow',
        )
     }eg;
     print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
     s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
         colored(
             join(
              "",
              map {
                   '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split//, $1)
             ),
             'yellow',
        )
     }eg;
     print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
    s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
        colored(
            join(
              "",
              map {
                  '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split(//, $1)
            ),
            'yellow',
        )
    }eg;
    print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
    s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
        colored(
            join(
              "",
              map {
                  '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split//, $1)
            ),
            'yellow',
        )
    }eg;
    print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
    s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
        colored(
            join(
              "",
              map {
                  '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split//, $1)
            ),
            'yellow',
        )
    }eg;
    print
}
```

bytey

```
#!/usr/bin/env perl -s

use strict;
use warnings;

use Term::ANSIColor qw(colored);

$::r = $::r ? "" : '(?![\\x{1b}]\\[\\d+m)';

while (<>) {
     s{$::r([^\x{0a}\x{20}\x{21}-\x{7e}]+)}{
         colored(
             join(
              "",
              map {
                   '\\x{'. sprintf('%02x',ord($_)) . '}'
              } split(//, $1)
             ),
             'yellow',
        )
     }eg;
     print
}
```

Devel::Peek

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "L\x{e9}on"'
SV = PV(0x7fd812004280) at 0x7fd812011368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK)
  PV = 0x7fd811c0dde0 "L\351on"\0
  CUR = 4
  LEN = 16
```

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "L\x{e9}on"'
SV = PV(0x7fd812004280) at 0x7fd812011368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK)
  PV = 0x7fd811c0dde0 "L\351on"\0
  CUR = 4
  LEN = 16
```

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "L\x{e9}on"'
SV = PV(0x7fd812004280) at 0x7fd812011368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK)
  PV = 0x7fd811c0dde0 "L\351on"\0
  CUR = 4
  LEN = 16
```

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "\N{SNOWMAN}"'
SV = PV(0x7f91399b49c0) at 0x7f9139811368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK,UTF8)
  PV = 0x7f913940dde0 “\342\230\203"\0
       [UTF8 "\x{2603}"]
  CUR = 3
  LEN = 16
```

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "\N{SNOWMAN}"'
SV = PV(0x7f91399b49c0) at 0x7f9139811368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK,UTF8)
  PV = 0x7f913940dde0 “\342\230\203"\0
       [UTF8 "\x{2603}"]
  CUR = 3
  LEN = 16
```

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "\N{SNOWMAN}"'
SV = PV(0x7f91399b49c0) at 0x7f9139811368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK,UTF8)
  PV = 0x7f913940dde0 “\342\230\203"\0
       [UTF8 "\x{2603}"]
  CUR = 3
  LEN = 16
```

Devel::Peek

```
perl -e 'use Devel::Peek; Dump "\N{SNOWMAN}"'
SV = PV(0x7f91399b49c0) at 0x7f91[illegible]1368
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK,UTF8)
  PV = 0x7f913940dde0 “\342\230\203"\0
       [UTF8 "\x{2603}"]
  CUR = 3
  LEN = 16
```

Test::utf8

Test::utf8

```
# check the string is good

is_valid_string($string);   # check the string is valid
is_sane_utf8($string);      # check not double encoded

# check the string has certain attributes

is_flagged_utf8($string1);   # has utf8 flag set
is_within_ascii($string2);   # only has ascii chars in it
isnt_within_ascii($string3); # has chars outside the ascii range
is_within_latin_1($string4); # only has latin-1 chars in it
isnt_within_ascii($string5); # has chars outside the latin-1 range
```

Test::utf8

```
# check the string is good

is_valid_string($string);   # check the string is valid
is_sane_utf8($string);      # check not double encoded

# check the string has certain attributes

is_flagged_utf8($string1);   # has utf8 flag set
is_within_ascii($string2);   # only has ascii chars in it
isnt_within_ascii($string3); # has chars outside the ascii range
is_within_latin_1($string4); # only has latin-1 chars in it
isnt_within_ascii($string5); # has chars outside the latin-1 range
```

Test::utf8

```
# check the string is good

is_valid_string($string);   # check the string is valid
is_sane_utf8($string);      # check not double encoded

# check the string has certain attributes

is_flagged_utf8($string1);   # has utf8 flag set
is_within_ascii($string2);   # only has ascii chars in it
isnt_within_ascii($string3); # has chars outside the ascii range
is_within_latin_1($string4); # only has latin-1 chars in it
isnt_within_ascii($string5); # has chars outside the latin-1 range
```

Test::utf8

```
# check the string is good

is_valid_string($string);   # check the string is valid
is_sane_utf8($string);      # check not double encoded

# check the string has certain attributes

is_flagged_utf8($string1);   # has utf8 flag set
is_within_ascii($string2);   # only has ascii chars in it
isnt_within_ascii($string3); # has chars outside the ascii range
is_within_latin_1($string4); # only has latin-1 chars in it
isnt_within_ascii($string5); # has chars outside the latin-1 range
```

Test::utf8

```
# This will pass as it's a normal latin-1 string
is_sane_utf8("Hello L\x{e9}on");

# this will fail because the \x{c3}\x{a9} looks like the
# utf8 byte sequence for e-acute
my $string = "Hello L\x{c3}\x{a9}on";
is_sane_utf8($string);

# this will pass because the utf8 is correctly interpreted as utf8
Encode::_utf8_on($string)
is_sane_utf8($string);
```

Common Problems

Raw Printing a non-ASCII Unicode Character

Raw printing Unicode

Léon & ☃

Raw printing Unicode

“send byte \x{4c}”

Léon & 

Raw printing Unicode

Léon & ☃

“send byte \x{e9}”

Raw printing Unicode

Léon & 

“send byte …?”

Raw printing Unicode

Léon & 

“Ah screw it send utf-8 byte sequence \x{e2}\x{98}\x{83}”

Raw printing Unicode

Wide character in print at -e line 1.

Léon & 

“Ah screw it send utf-8 byte sequence \x{e2}\x{98}\x{83}”

Accidental Unicode Character in Byte String

UTF-8 Upgrade (good)

```
perl -e 'use Devel::Peek; Dump "L\x{e9}on" . "\N{SNOWMAN}"'
SV = PV(0x7fa428804290) at 0x7fa428811320
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK,UTF8)
  PV = 0x7fa428505b60 “L\303\251on\342\230\203"\0
       [UTF8 "L\x{e9}on\x{2603}"]
  CUR = 8
  LEN = 16
```

```
MarkOtravis:~$ perl -E 'binmode STDOUT, 'utf8'; say "L\x{e9}on" . "\N{SNOWMAN}"'
Léon☃
MarkOtravis:~$ perl -E 'binmode STDOUT, 'utf8'; say "L\x{e9}on" . "\N{SNOWMAN}"' | bytey
L\x{c3}\x{a9}on\x{e2}\x{98}\x{83}
```

UTF-8 Upgrade (bad)

```
perl -e 'use Devel::Peek; Dump "L\x{c3}\x{a9}on" . "\N{SNOWMAN}"'
SV = PV(0x7fc072004290) at 0x7fc072011320
  REFCNT = 1
  FLAGS = (POK,READONLY,pPOK,UTF8)
  PV = 0x7fc071c0dc10 “L\303\203\302\251on\342\230\203"\0
       [UTF8 "L\x{c3}\x{a9}on\x{2603}"]
  CUR = 10
  LEN = 16
```

```
Mark@travis:~$ perl -E 'binmode STDOUT, 'utf8'; say "L\x{c3}\x{a9}on" . "\N{SNOWMAN}"'
LÃ©on☃
Mark@travis:~$ perl -E 'binmode STDOUT, 'utf8'; say "L\x{c3}\x{a9}on" . "\N{SNOWMAN}"' | bytey
L\x{c3}\x{83}\x{c2}\x{a9}on\x{e2}\x{98}\x{83}
```

“The Unicode Bug”

The Unicode Bug

```
my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```

```
Mark@travis:~$ perl /tmp/bug.pl | bytey
\x{c9}
\x{e9}
```

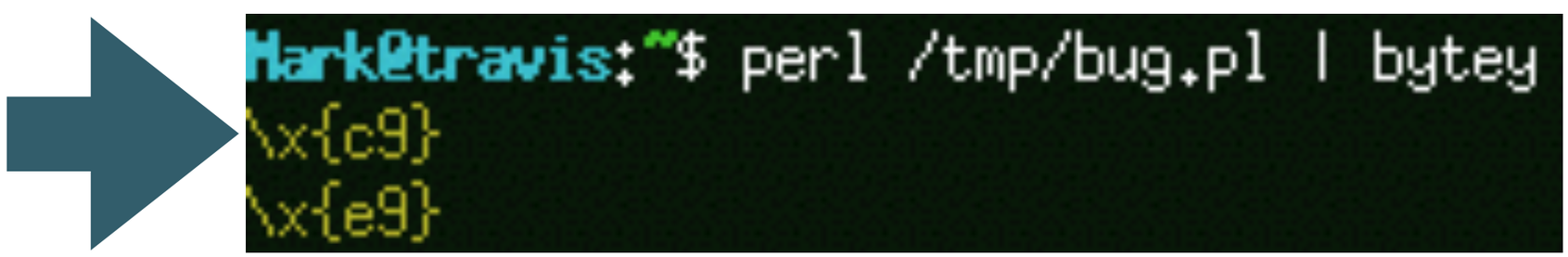
The Unicode Bug

```
my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```

```
Mark@travis:~$ perl /tmp/bug.pl | bytey
\x{c9}
\x{e9}
```

The Unicode Bug

```
my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```



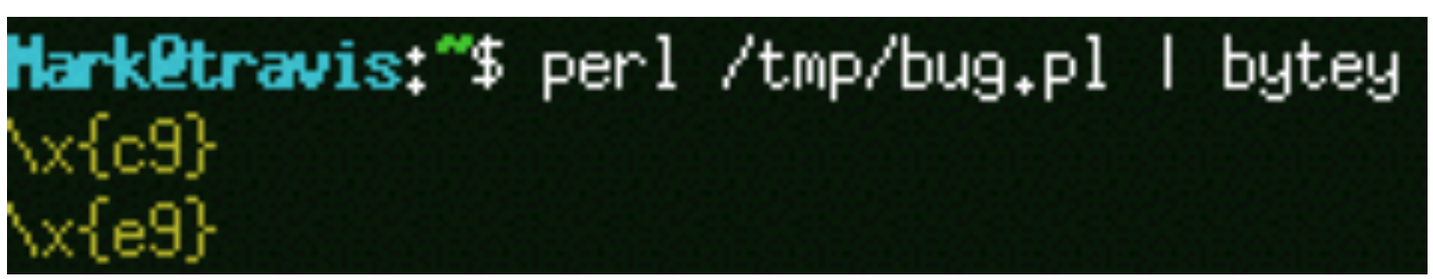
The Unicode Bug

```
my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```

```
Mark@travis:~$ perl /tmp/bug.pl | bytey
\x{c9}
\x{e9}
```

The Unicode Bug

```
my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```



The Unicode Bug

128-255 = ???

The Unicode Bug

```
use feature 'unicode_strings';

my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```

The Unicode Bug

```
use 5.12.0;

my $string = "\x{c9}";
print lc $string, "\n";

$string .= "\N{SNOWMAN}";
chop $string;
print lc $string, "\n";
```

XS routines loses the UTF-8 flag

XS and UTF-8

- Bad XS code copies SV's PV but not UTF-8 flag
  - Flip flag with Encode::_utf8_on
  - or Re-do the decode again
  - or use Data::Structure::Util in anger
  - or PATCH THE DARN XS CODE

Re-encode a byte string a second time

Re-decode a character string a second time

“Doctor, it hurts when I do this.”

“Don't do that.”

– Dr. Kronkheit and His Only Living Patient, Smith & Dale, 1920s

Tricks and Tips

Unicode in Perl Source Code

Unicode In Source Files

```
#!/usr/bin/perl

use 5.18.0;
use warnings;

say length "Léon";

{
  use utf8;
  say length "Léon";
}
```

Unicode In Source Files

```
Mark@travis:~$ bytey /tmp/unicode.txt
#!/usr/bin/perl

use 5.18.0;
use warnings;

say length "L\x{c3}\x{a9}on";

{
    use utf8;
    say length "L\x{c3}\x{a9}on";
}
Mark@travis:~$
```

Unicode In Source Files

```
#!/usr/bin/perl

use 5.18.0;
use warnings;

say length "Léon";          # prints “5”

{
   use utf8;
   say length "Léon";       # prints “4”
}
```

Unicode Regex

Matching Digits With a Regex

/^\d+$/

Matching Digits With a Regex

```
perl -E 'say "100" =~ /^\d+$/ ? "yes" : "no"';
yes

perl -E 'say "abc" =~ /^\d+$/ ? "yes" : "no"';
no
```

Matching Digits With a Regex

```
perl -E 'say "100" =~ /^\d+$/ ? "yes" : "no"';
yes

perl -E 'say "abc" =~ /^\d+$/ ? "yes" : "no"';
no
```

Matching Digits With a Regex

```
perl -E 'say "100" =~ /^\d+$/ ? "yes" : "no"';
yes

perl -E 'say "abc" =~ /^\d+$/ ? "yes" : "no"';
no
```

Matching Digits With a Regex

```
perl -E 'say "\x{09EA}" =~ /^\d+$/ ? "yes" : "no"';
```

Matching Digits With a Regex

```
perl -E 'say "\x{09EA}" =~ /^\d+$/ ? "yes" : "no"';
yes
```

09EA BENGALI DIGIT FOUR

Matching Digits With a Regex

```
perl -E 'say "\x{09EA}" =~ /^\d+$/a ? "yes" : "no"';
no
```

Matching Digits With a Regex

/a

Make \d, \s, \w match only in ASCII range

Matching A-Z With a Regex

```
/^[a-z]+$/i
```

Matching A-Z With a Regex

```
perl -E 'say "\x{212A}" =~ /^[a-z]+$/i ? "yes":"no"';
```

Matching A-Z With a Regex

```
perl -E 'say "\x{212A}" =~ /^[a-z]+$/i ? "yes":"no"';
yes
```

212A KELVIN SIGN

Matching A-Z With a Regex

```
perl -E 'say "\x{212A}" =~ /^[a-z]+$/iaa ? "yes":"no"';
no
```

212A KELVIN SIGN

Matching A-Z With a Regex

```
/aa

Make \d, \s, \w match only in ASCII range
Make ASCII/non-ASCII never match
```

Data::Structure::Utils

Handy Dandy Utility Functions

```
 has_utf8($var)
   Returns $var if the utf8 flag is enabled for $var or any scalar
   that a data structure passed in $var contains.

_utf8_off($var)
   Recursively disables the utf8 flag on all scalars within $var.

_utf8_on($var)
   Recursively enables the utf8 flag on all scalars within $var.

utf8_on($var)
   This routine performs a "sv_utf8_upgrade" on each scalar string in
   the passed data structure that does not have the utf8 flag turned
   on.

utf8_off($var)
   This routine performs a "sv_utf8_downgrade" on each scalar string
   in the passed data structure that has the utf8 flag turned on.
```

Converting Anything To ASCII

Converting Anything To ASCII

```
use Text::Unidecode;
say unidecode("北京");
```

```
Bei Jing
```

Normalized Form

md5 bytes only

```
my $code = md5sum( $passwd_str );
```

md5 bytes only

```
my $passwd_bytes = encode("utf-8", $passwd_str);
my $code = md5sum( $passwd_bytes );
```

When é isn’t é

é → \x{e9}

When é isn’t é

é → \x{65} \x{0301}

When é isn’t é

“LATIN SMALL LETTER E”

é → \x{65} \x{0301}

“COMBINING ACUTE ACCENT”

md5 bytes only

```
use Unicode::Normalize;
$passwd_str = NFC($string);
my $passwd_bytes = encode("utf-8", $passwd_str);
my $code = md5sum( $passwd_bytes );
```