Extended Introduction to Computer Science
CS1001.py

Module B Characters and Text Representation: ASCII, Unicode

Instructors: Amir Rubinstein, Michal Kleinbort

Teaching Assistants: Noam Parzanchevsky,

Hussen Abu Hamad, Asaf Cassel, Shaked Dovrat

School of Computer Science

Tel-Aviv University

Fall Semester 2020-21

http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

Lecture 7a: Plan

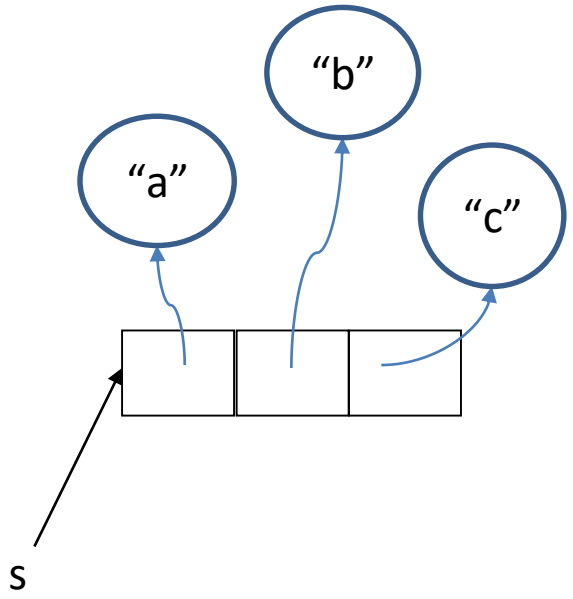
- Data representation (cont.)
  - Representing integers with bits
  - Representing decimal numbers with floating point
  - Representing text and characters

Text (str) Representation

- You saw how int and float are represented using the binary system

- What about str?

```
>>> s = "abc"
>>> hex(id(s))
'0xbe4c00'
>>> hex(id(s[0]))
'0xbd3c00'
>>> hex(id(s[1]))
'0xbdbd00'
>>> hex(id(s[2]))
'0xbc8a20'
```



- But how are single characters represented?

Representation of Characters: ASCII

- The initial encoding scheme for representing characters is called ASCII (American Standard Code for Information Interchange).

- It has 128 characters, represented each by 7 bits (the numbers 0-127)
  - These include 94 printable characters (English letters, numerals, punctuation marks, math operators), space, and 33 invisible control characters (mostly obsolete).

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

(table from Wikipedia. 8 rows/16 columns represented in hex, e.g. 'a' is 0x61, or 97 in decimal representation)

Representation of Characters: Unicode

- With the increased popularity of computers and their usage in multiple languages (mainly those not employing Latin alphabet, but also European languages with characters such as à, á, â, ã, ä, ç, ñ etc), it became clear that ASCII encoding is not expressive enough and should be extended.

- Demand for additional characters (e.g. various symbols that are not punctuation marks) and letters (e.g. Cyrillic, Hebrew, Arabic, Greek), possibly in the same piece of text, led to the 16 bit Unicode (and, along the way, earlier encodings).

- Chinese characters (and additional ones, e.g. Byzantine musical symbols, if you really care) led to 20 bit Unicode.

Fixed Length vs. Variable Length Encodings

- ASCII characters are all encoded using 7 bits per character. This is a fixed-length code.

- Unicode, on the other hand, uses a variable length encoding: Different characters can be encoded using a different number of bits.
  - The 128 "ASCII characters" are encoded using 8 bits (1 byte) per character, and are compatible with the 7 bit ASCII encoding (a leading zero is added).

  - Other characters may have longer encodings. For example, Hebrew letters' encodings are in the range 1488 (א) to 1514 (ת) reflecting the 22+5=27 letters in the Hebrew alphabet.

Unicode in Python

- Python employs Unicode.
  - The built-in function ord returns the Unicode representation of a (single) character (in decimal).
  - The built-in function chr does the opposite.

```
>>> ord(" ") # space
32
>>> ord("a")
97
>>> chr(97)
'a'
>>> ord("א")
1488
>>> bin(ord("א"))
'0b10111010000'
>>> chr(int("10111010000", 2))
'א'
```

Text → Unicode → Bits

```
def text2Unicode(text):
    ''' return a list of ints representing text '''
    lst = []
    for c in text:
        lst += [ord(c)]
    return lst

def text2bits(text):
    ''' return a string of bits representing text '''
    lst = []
    for c in text :
        lst += [bin(ord(c))[2:]]
    return "".join(lst)
```

- Which problem arises in the opposite direction?

Text → Unicode → Bits

```
>>> text2Unicode("נועה קירל שולטטתתת")
[1504, 1493, 1506, 1492, 32, 1511, 1497, 1512, 1500, 32,
1513, 1493, 1500, 1496, 1496, 1496, 1514, 1514, 1514, 1514]
```

```
>>> text2bits("נועה קירל שולטטתתת")
'10111100000101110101011011110001010111010100100000101111001'
11101110110011011110100010111011100100000101111010011011101010
01101110111001011101100010111011000101111010101011
'11010101011110101010111101010'
```

Strings and Sequences in Various Contexts

- Text editing is one of the most popular applications (be it Word, Notepad, Emacs, LATEX, etc.).

- Other than that, character, string, and word operations are important in many other contexts. For example:
  - Linguistics. For example, study letter frequencies across different languages over the same alphabet.

  - Biological sequences. Here the text could be a chromosome, a whole genome, or even a collection of genomes.
    Given that the length of, say, the human genome, is approximately 3 billion letters (A, C, G, T), efficiency may be crucial

  - Musical information retrieval. For example, you may want to whistle or hum into your smartphone, and have it retrieve the most similar piece of music out of some large collection.