CMPT 120
How computers run programs

Summer 2012

Instructor: Hassan Khosravi

How Computers Represent Information

- All information that is stored and manipulated with a computer is represented in binary
  - with zeros and ones.

- Why just zeros and ones?
  - Computer's memory is a whole bunch of tiny rechargeable batteries (capacitors).
  - discharged (0) or charged (1).
    - It's easy for the computer to look at one of these capacitors and decide if it's charged or not.
  - This could be done to represent digits from 0 to 9
    - difficult to distinguish ten different levels of charge in a capacitor
    - hard to make sure a capacitor doesn't discharge a little to drop from a 7 to a 6

How Computers Represent Information

- A single piece of storage that can store a zero or one is called a bit.
- Bits are often grouped. It's common to divide a computer's memory into eight-bit groups called bytes
  - 00100111 and 11110110
- Number of bits or bytes quickly becomes large

Prefix	Symbol	Factor
(no prefix)		$2^0 = 1$
kilo-	k	$2^{10} = 1024 \approx 10^3$
mega-	M	$2^{20} = 1048576 \approx 10^6$
giga-	G	$2^{30} = 1073741824 \approx 10^9$
tera-	T	$2^{40} = 1099511627776 \approx 10^{12}$

- For example, "12 megabytes" is
  - 12×2^{20} bytes = 12,582,912 bytes = 12582912×8 bits = 100,663,296 bits
- Note that values are approximations
  - Kilo is 1000 here it is 1024

Unsigned Integers

- Consider the number 157
  - $157 = (1 \times 10^2) + (5 \times 10^1) + (7 \times 10^0)$.

- Applying the same logic, there is a counting system with bits, binary or base 2 arithmetic
- The rightmost bit will be the number of 1s(2^0), the next will be the number of 2s (2^1), then 4s (2^2), 8s (2^3), 16s (2^4), and so on.

- $1001_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 8 + 1$

- $10011101_2 = (1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 128 + 16 + 8 + 4 + 1 = 157_{10}$.

binary	decimal	binary	decimal
1111	15	0111	7
1110	14	0110	6
1101	13	0101	5
1100	12	0100	4
1011	11	0011	3
1010	10	0010	2
1001	9	0001	1
1000	8	0000	0

- The computer can do operations like addition and subtraction on binary integers the same way you do with decimal numbers
  - Keep in mind that $1 + 1 = 2_{10} = 10_2$

$$
\begin{array}{r}
1010 \\
+\,0100 \\
\hline
1110
\end{array}
\qquad\qquad
\begin{array}{r}
{\scriptstyle 1} \\
1011 \\
+\,0010 \\
\hline
1101
\end{array}
\qquad\qquad
\begin{array}{r}
{\scriptstyle 1}{\scriptstyle 1} \\
1101 \\
+\;\;0101 \\
\hline
10010
\end{array}
$$

Positive and Negative Integers

- One easy way to think of this is to have the left most bit as the sign
  - (0 = positive, 1 = negative)
  - With four bits
    - 0 111 would 7
    - 1111 would be -7
- Pros:
  - Its easy for the human eye to understand
  - It's easy to tell if the value is negative: if the first bit is 1, it's negative.
  - For positive numbers the values are the same as the unsigned representation.
- Cons
  - Addition and subtraction does not work as before
  - The value 0 has two representations 1000 and 0000.

two's complement notation

- To convert a positive value to a negative value in two's complement, you first flip all of the bits (convert 0s to 1s and 1s to 0s) and then add one.
- For example to show -5
  - Start with the positive version: 0101
  - Flip all of the bits: 1010
  - Add one: 1011

binary	decimal	binary	decimal
1111	−1	0111	7
1110	−2	0110	6
1101	−3	0101	5
1100	−4	0100	4
1011	−5	0011	3
1010	−6	0010	2
1001	−7	0001	1
1000	−8	0000	0

  - With 4bits using two's complement we can show -8, 7

Pros and cons of two's complement

- Pros
  - It's easy to tell if the value is negative: if the first bit is 1, it's negative.
  - For positive numbers the values are the same as the unsigned representation.
  - Addition and subtraction works the same unsigned method
  - The value 0 now has 1 representations 0000
- Cons
  - Not as easy for humans to see

Examples of two's complement

- -6 +4 with 4 digits
  - Start with 6 → 0110
  - Complement → 1001
  - Add 1 → 1010

```
   1 0 1 0
+  0 1 0 0
  ----------
   1 1 1 0
```

- What value is 1110?
  - Take one away → 1101
  - Complement →0010 → which is 2

Examples of two's complement

- −3 + 5 = 2
  - Start with 3 → 0011
  - Complement → 1100
  - Add 1 → 1101

```
   1 1 0 1
+  0 1 0 1
  ----------
 1 0 0 1 0
```

  - We only have 4 bits of memory for values -8 to 7 so we ignore last carried one

- 3 – 4

```
   1 0 0 1 1
-    0 1 0 0
   ----------
    1 1 1 1
```

- What is 1111
  - Take one away → 1110
  - Complement → 0001

I-clicker

- A: I feel comfortable with binary values and mathematical operations on them
- B: I was following the class and got the basics, I need to practice some more to be comfortable with it
- I had difficulty in understanding binary values. I need to go over the theory again.
- D: I didn't understand binary values and operators on them at all

Characters

- A character is a single letter, digit or punctuation
  - Storing characters is as easy as storing unsigned integers. For a byte (8 bits) in the computer's memory, there are $2^8 = 256$ different unsigned numbers
  - Assign each possible character a number and translate the numbers to characters.
  - The character set used by almost all modern computers, when dealing with English and other western languages, is called ASCII
    - T =84
    - $= 36
    - Number 4 as a string = 52
      - Why not give numbers their own value?

ASCII code

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

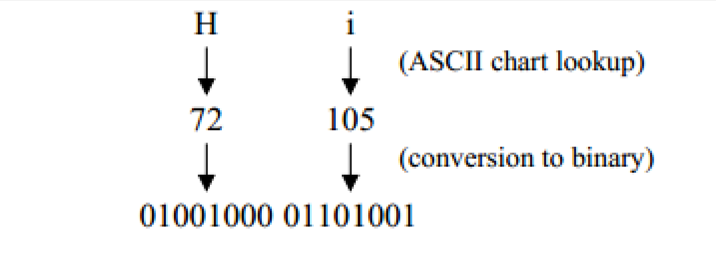
Extended ASCII codes

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	^{n}
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	^{2}
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Strings

- A string is a collection of several characters.
  - Some strings are "Jasper", "742", and "bhay-gn-flay-vn".
  - The particular character set that is used by almost all modern computers, when dealing with English and other western languages, is called ASCII



  - The binary is the same as 18537 how does the computer know whether this is “hi” or 18537?
  - The programming language should take care of that.

Unicode

- With only one byte per character, we can only store 256 different characters in our strings
  - But gets quite hard with languages like Chinese and Japanese

- The Unicode character set was created to overcome this limitation. Unicode can represent up to 2^{32} characters.

- Read topic 2.6 from introduction to computing science and programming

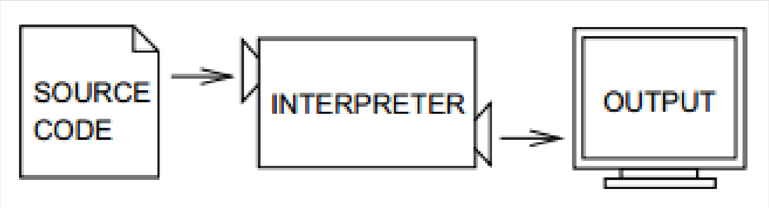
The Python programming language

- The programming language we will use in this course is Python.
- Python is an example of a high-level language;
  - Other high-level languages are C, C++, Perl, and Java.
  - Much easier to program
  - Less time to read and write
  - More likely to be correct
  - Portable
- Low-level languages, sometimes referred to as “machine languages" or “assembly languages”
  - Only used for a few specialized applications.
- Computers can only execute programs written in low level . Programs written in high level have to be processed before then can be run.

- Two kinds of programs process high-level languages into low-level languages:

Interpreters

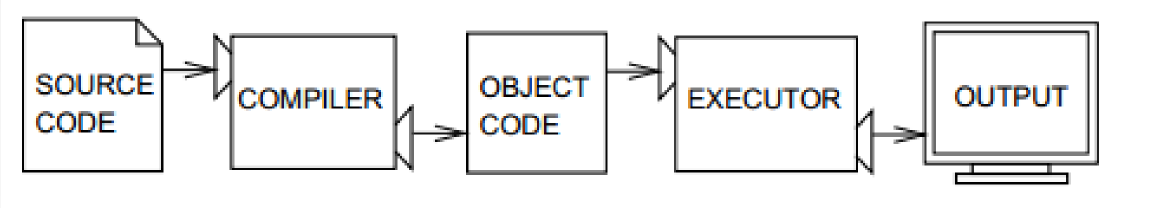
- An interpreter reads a high-level program and executes it,
- It processes the program a little at a time, alternately reading lines and performing computations.



- Python is interpreted

Compiler

- A compiler reads the program and translates it completely before the program starts running.
- In this case, the high-level program is called the source code, and the translated program is called the object code or the executable



- Read chapter 1 from how to think like a computer scientist