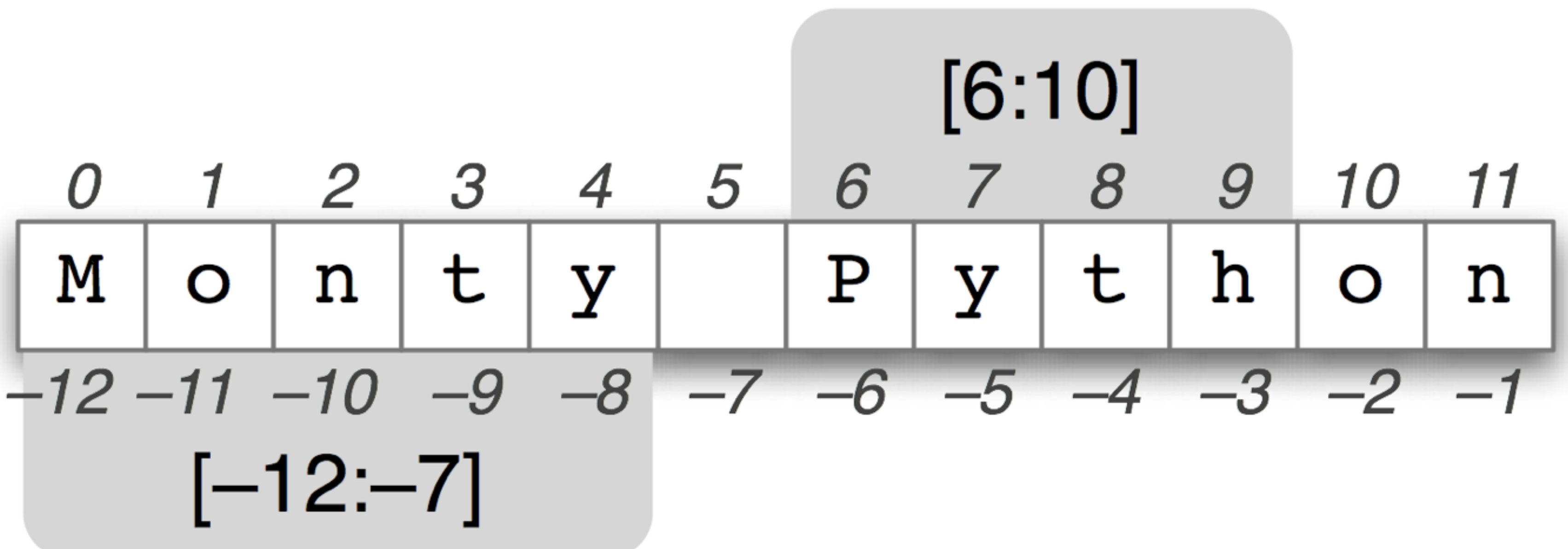
15-112
Fundamentals of Programming

Week 1 - Lecture 5:
Wrapping up 1st week + Intro to strings.



May 20, 2016

On the menu today

Wrap up previous material

- approximate values of floats
- importing modules
- short-circuit evaluation
- conditional (if-else) expression

How does a computer work? (looking under the hood)

Introduction to strings

How does a computer work?

How does a computer work?

1. How does a computer represent data (information)?

2. What are the basic components of computers?

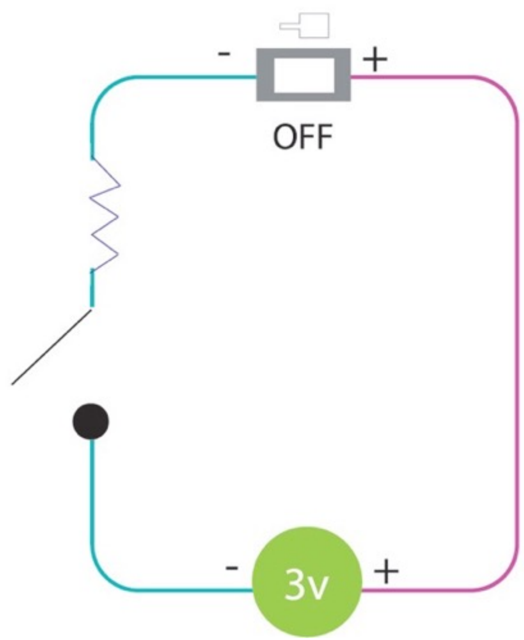
3. How does a computer process information?

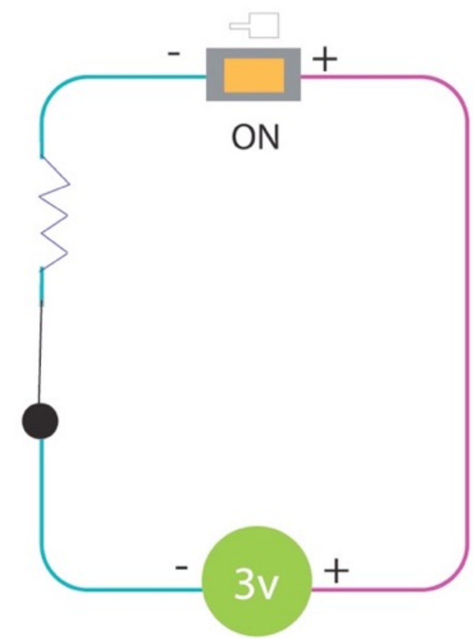
How does a computer represent data?

What is the most basic data/information that can be stored with an electronic device?

What is the most basic (useful) electronic device?

A switch.





On or Off. Is electrical current flowing or not.

How does a computer represent data?

If I am interested in representing *binary* data,
I can do it with a single switch.

Examples:

(Yes or No) (On of Off) (0 or 1) (Apple or Orange)

Why stop at one switch?
What can I do with 2 switches?

	Switch 1	Switch 2
0	Off	Off
1	On	Off
2	Off	On
3	On	On

4 different options:
Can represent 4 different values.

e.g. can represent 0, 1, 2, 3

How does a computer represent data?

Why stop at 2 switches?
What can I do with 3 switches?
What can I do with 300 switches?

With n switches, I can represent 2^n different values.
(To represent n different values, I need $\sim \log_2 n$ switches.)

With 300 switches, I can represent 2^{300} different values.

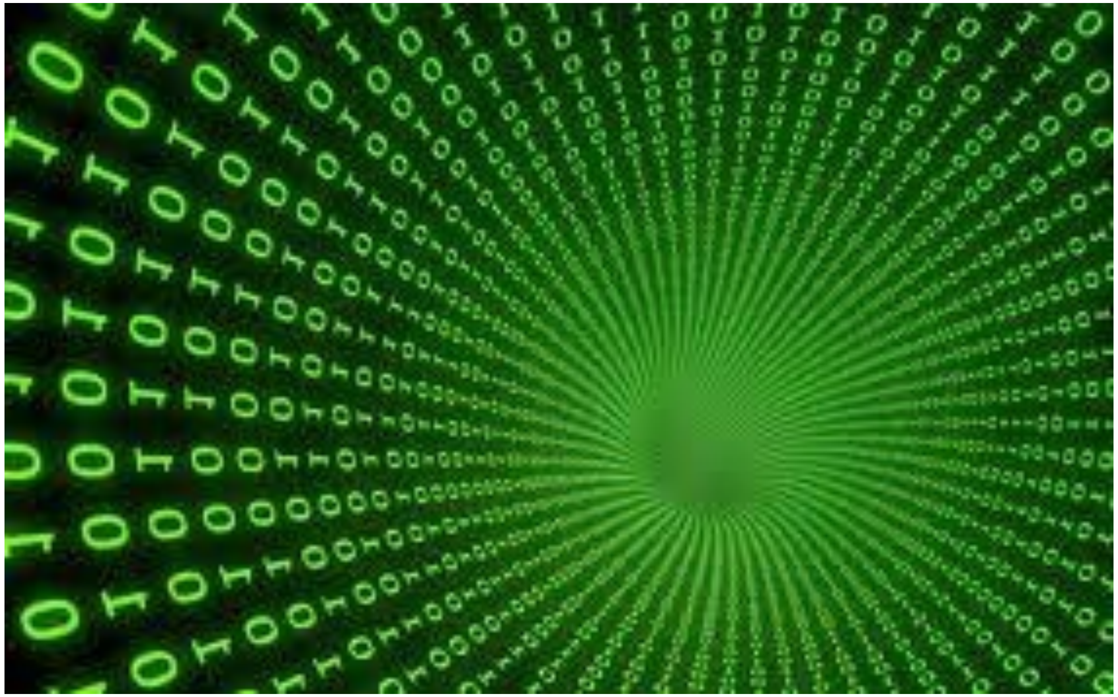
$2^{300} \sim$ number of atoms in the observable universe.

2037035976334486086268445688409378161051468393665936250636140449354381299763336706183397376

No big deal to represent it on paper.
And no big deal to represent it in a computer
(these switches are tiny).

How does a computer represent data?

Have you ever heard the phrase:
"Everything in a computer is just 0s and 1s"



How does a computer represent data?

In computer science:

A switch's state (off or on) is represented by 0 or 1

So all data is a string of 0s and 1s.

A switch is called a bit. A bit represents either 0 or 1.

With enough switches/bits (0s and 1s),
we can represent any kind of informaiton.

How does a computer represent data?

Representing integers with 0s and 1s.

The convention:

Switch (bit) number:	7	6	5	4	3	2	1	0
Values:	1	1	0	1	0	0	1	1

Number represented: $2^7 + 2^6 + 2^4 \;+\; 2^1 + 2^0$

$= 211$

How does a computer represent data?

Representing characters (and text).

The American Standard Code for Information Interchange (ASCII)

ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	(	0010 1000
N	0100 1110	l	0110 1100	)	0010 1001

1 byte = 8 bits

1 kilobyte = 2^{10} bytes (1024 bytes)

1 megabyte = 2^{10} kilobytes

1 gigabyte = 1,000,000,000 bytes

How does a computer work?

1. How does a computer represent data (information)?

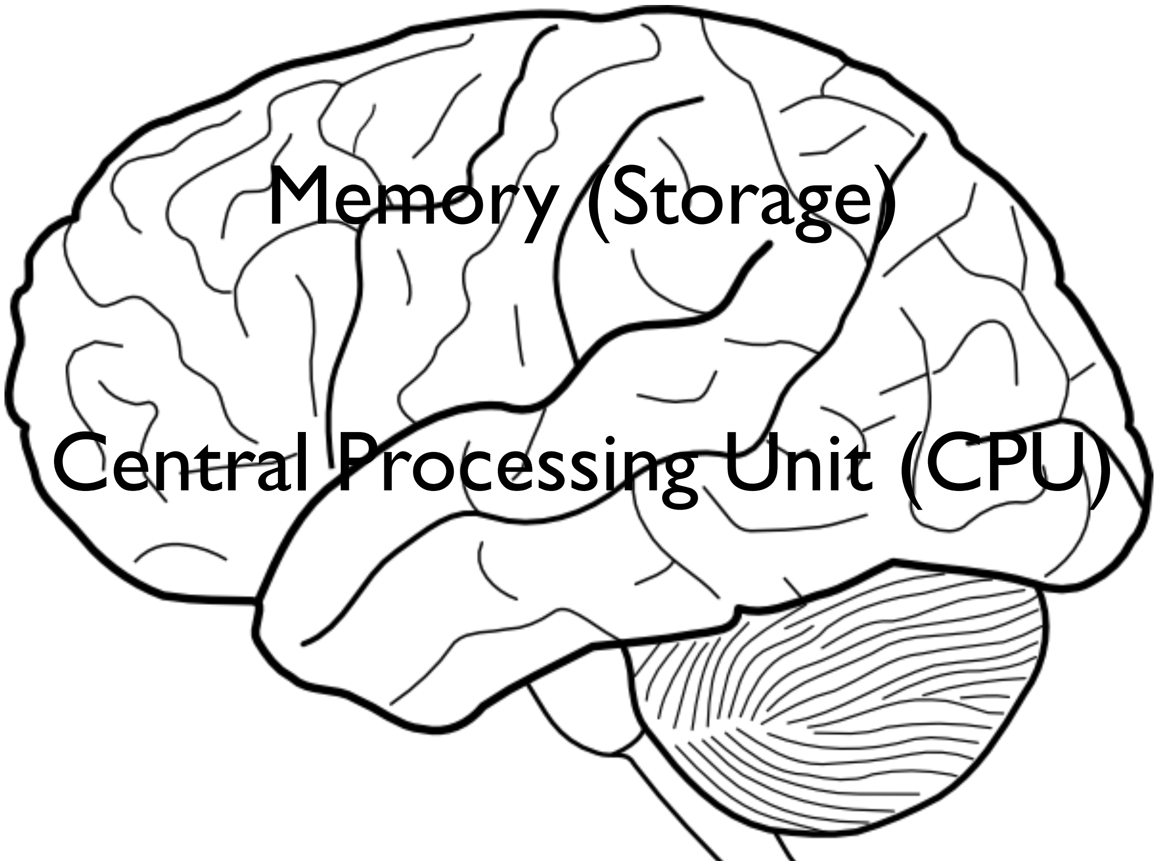
2. What are the basic components of computers?

3. How does a computer process information?

Basic components of computers

3 Main Parts:

Input/Output components



Memory (Storage)

Central Processing Unit (CPU)

Basic components of computers

Input/Output components

Input: keyboard, mouse, microphone.

Output: screen, speakers.

Basic components of computers

3 Main Parts:

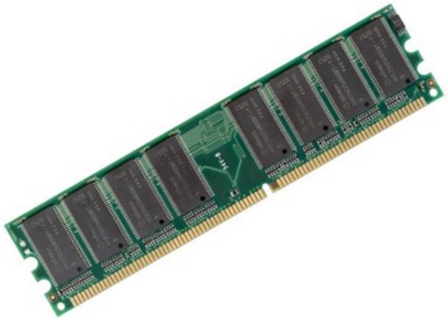
Input/Output components

Memory (Storage)

Central Processing Unit (CPU)

Basic components of computers

Memory (Storage)
2 Main Parts

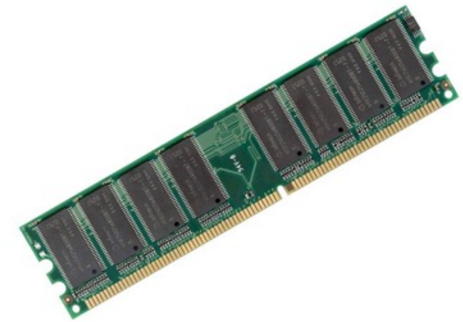


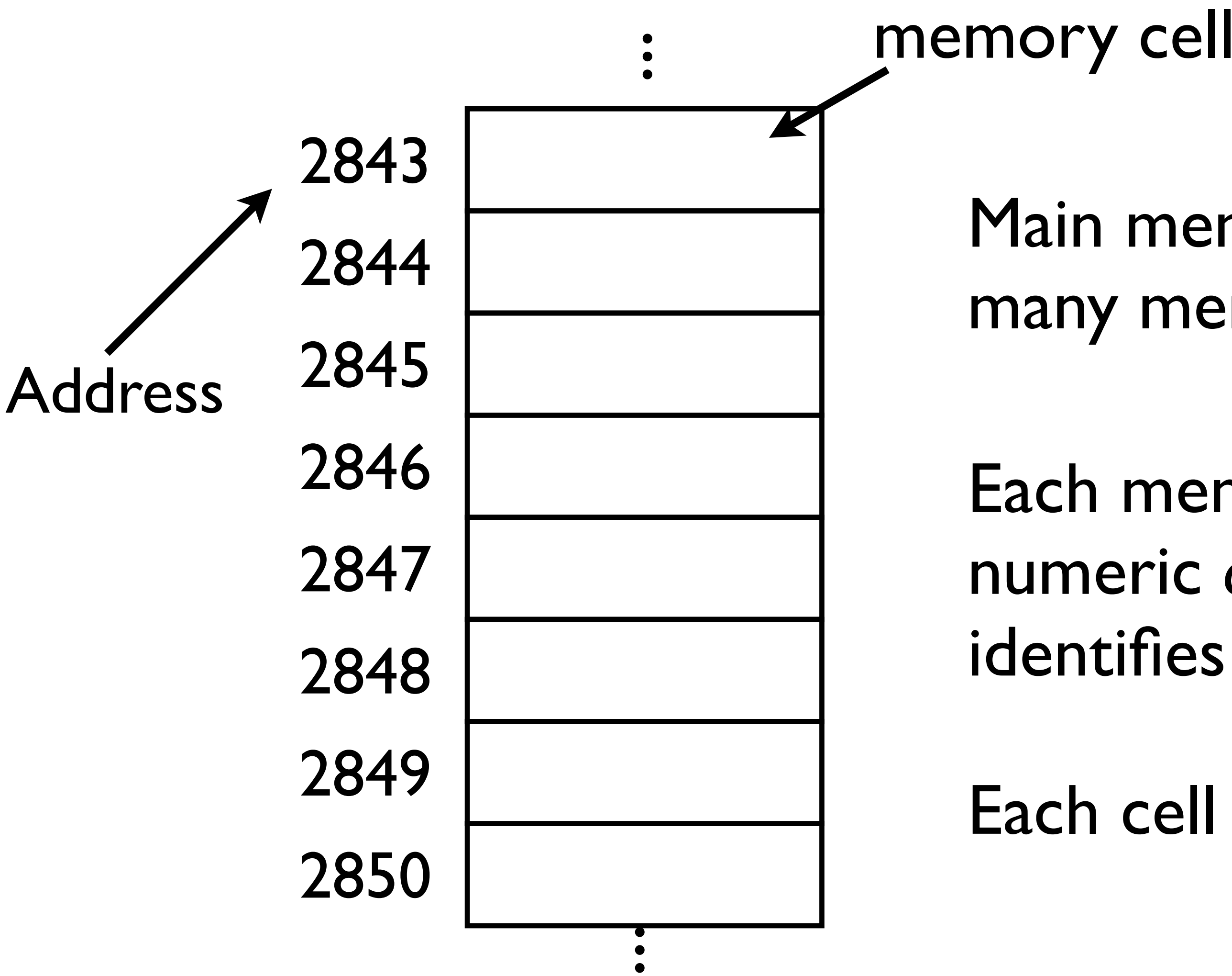
- RAM (Random Access Memory)
  Stores "active" (currently used) data.
  CPU can directly access it.
  When a program terminates, contents are lost.

- Hard drive (and other secondary storage)
  Stores "inactive" data. (e.g. videos you are not watching.)
  CPU does not directly access it.
  Contents are not lost when computer shuts down.
  Access time is much slower compared to RAM.

Basic components of computers

Memory (Storage)
Closer look at RAM (Main memory)





Main memory is divided into many memory locations (cells)

Each memory cell has a numeric *address* which uniquely identifies it.

Each cell contains 1 byte of data.

Basic components of computers

3 Main Parts:

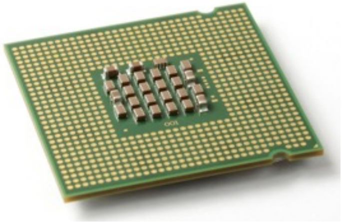
Input/Output components

Memory (Storage)

Central Processing Unit (CPU)

Basic components of computers

Central Processing Unit (CPU)



The "action" part of computer's brain.

Carries out the instructions of a program.

- Arithmetic operations.
- Logical operations.
- input/output operations.

The instructions it understands are very basic:

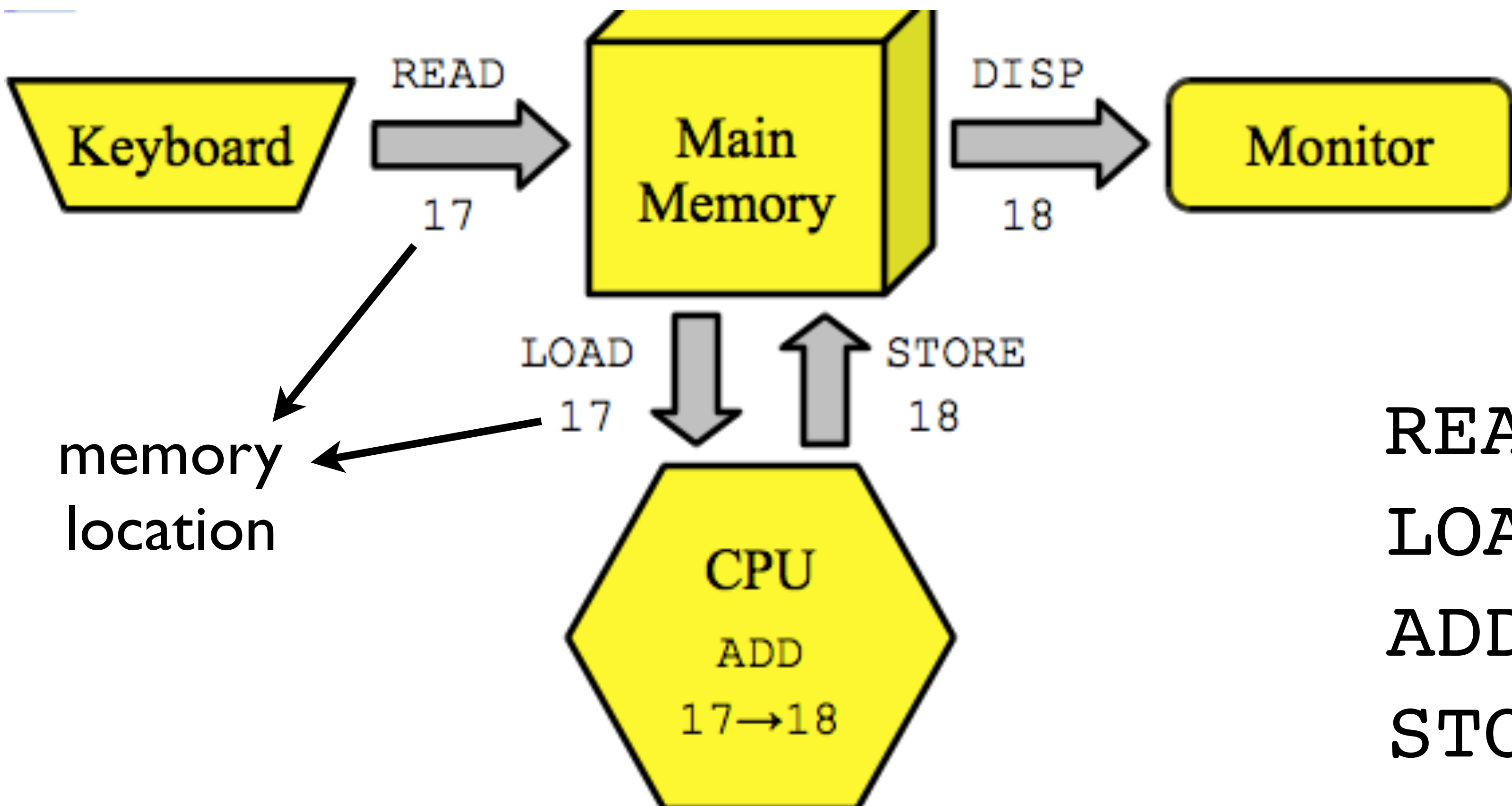
LOAD	ADD	DISP
READ	STORE	

How does a computer work?

1. How does a computer represent data (information)?
2. What are the basic components of computers?
3. How does a computer process information?

How does a computer process information?

Example: Read a number from the keyboard, add 1 to it, then display the new value on the screen.




```
READ 17
LOAD 17
ADD
STORE 18
DISP 18
```

How does a computer process information?

The instructions that the CPU understands is called the machine language.

But CPU can only understand 0s and 1s.
Each instruction is represented by a series of bits.

Previous example: Read a number from the keyboard, add 1 to it, then display the new value on the screen.

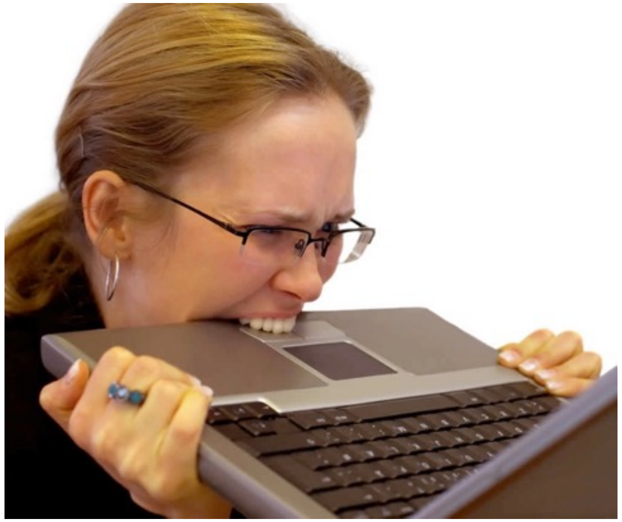
The first 20 bytes of the machine language:

```
01111111 01000101 01001100 01000110 00000001
00000001 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000010 00000000 00000011 00000000
```

MORE THAN 6500 BYTES IN TOTAL!

How do programmers process information?

Surely you don't want to write code in machine language!



- Tedious, confusing, hard to read.

- If you change one bit by accident,
program's behavior will be totally different.

- Errors are hard to find and correct.

How do programmers process information?

High-Level Programming Languages

The idea:

- Develop a language that is a mix of English and math.
  (easy to read, understand, and write)

(One instruction in a high-level language can correspond to hundreds of instructions in machine language.)

The secret to programming/computing

Many layers of *abstraction*.

- We start with electronic switches.
- We abstract away and represent data with 0s and 1s.
- We have machine language (0s and 1s) to tell the computer what to do.
- We abstract away and build/use high-level languages.
- We abstract away and build/use functions and *objects* (more on this later).

This is how large, complicated programs are built!

Introduction to Strings

Builtin Data Types

Python name	Description	Values
NoneType	absence of value	None
bool (boolean)	Boolean values	True, False
int (integer)	integer values	-2^{63} to $2^{63}-1$
long	large integer values	all integers
float	fractional values	e.g. 3.14
complex	complex values	e.g. 1+5j
str (string)	text	e.g. "Hello World!"
list	a list of values	e.g. [2, 5, "hello", "hi"]

...

Introduction to Strings

- String representation in memory

- Built-in string operations

String representation in memory

Every type of data in a computer is represented by numbers (binary numbers)

Each character in a string is a number.

print(ord(“a”))	97
print(chr(97))	a
print(ord(“b”))	98
print(“a” < “b”)	True
print(“a” < “A”)	False
print(“A” < “a”)	True

String representation in memory

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Example

Input: one character

Output: that character capitalized (if it is a letter).

```
def toUpperCaseLetter(c):
  if ((“a” <= c) and (c <= “z”)):
    return chr(ord(c) - (ord(“a”) - ord(“A”)))
  return c
```

Introduction to Strings

- String representation in memory
- Built-in string operations

String gluing

Concatenation

print(“Hello” + “World” + “!”) HelloWorld!

print(“Hello” “World” “!”) HelloWorld!

s = “Hello”

print(s “World” “!”) ERROR

String gluing

Repetition

```
print("SPAM!!!" * 20)

print(20 * "SPAM!!!")

print(20 * "SPAM!!!" * 20)
```

String chopping

Indexing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10

s = “Go Tartans!”

print(s[0]) G

length = len(s) (length stores 11)

print(s[5], s[length-1], s[3]) r ! T

expression that should evaluate to an integer

String chopping

Indexing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

print(s[-1]) !

print(s[-11]) G

print(“Yabadabaduuu!”[5]) a

print(s[len(s)]) INDEX ERROR

String chopping

Slicing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

print(s[3:7])	Tart
print(s[3:len(s)])	Tartans!
print(s[0:len(s)])	Go Tartans!
print(s[3:])	Tartans!
print(s[:1])	G
print(s[:])	Go Tartans!

String chopping

Slicing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

print(s[0:len(s):2])	G atn!
print(s[::])	Go Tartans!
print(s[len(s)-1:0:-1])	!snatraT o
print(s[len(s)-1:-1:-1])	range is empty, so it prints nothing
print(s[::-1])	!snatraT oG WEIRD!

Strings are immutable!!!!!

Slicing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

s[3] = “t”   ERROR

s += “ haha”

print(s)   Go Tartans! haha   # Worked! Why?

s = s[:3] + “t” + s[4:]   effectively same as   s[3] = “t”

print(s)   Go tartans! haha

Example: getMonthName

<u>Input</u>: a number from 1 to 12
<u>Output</u>: first three letters of the corresponding month.

e.g. 1 returns "Jan", 2 returns "Feb", etc...

```
def getMonthName(monthNum):
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"
    pos = (monthNum-1) * 3
    return months[pos:pos+3]
```

Example: indexOf

Input: a character c and a string s

Output: the index of the first occurence of c in s
(return -1 if c is not in s)

```
def indexOf(c, s):
    for index in range(len(s)):
        if (s[index] == c):
            return index
    return -1
```

Example: flipper

<u>**Input**</u>: a string s containing only 0s and 1s
<u>**Output**</u>: s with the 0s and 1s flipped.

Exercise

Example: isPalindrome

Input: a string s
Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome(s):
    return s == s[::-1]
```

Example: isPalindrome

Input: a string s
Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def reverseString(s):
    return s[::-1]
```

```
def isPalindrome(s):
    return s == reverseString(s)
```

This strategy is not recommended.
You create a new string, which is not necessary.

Example: isPalindrome

<u>Input</u>: a string s
<u>Output</u>: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome2(s):
    mid = len(s)//2
    for i in range(mid):
        if (s[i] != s[-1-i]): return False
    return True
```

This is a good way of doing it.

Example: isPalindrome

Input: a string s

Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome2(s):
    mid = len(s)//2
    for i in range(mid):
        if (s[i] != s[len(s)-1-i]): return False
    return True
```

Most programming languages
don't allow negative indices.

Example: isPalindrome

Input: a string s
Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome3(s):
    while (len(s) > 1):
        if (s[0] != s[-1]): return False
        s = s[1:-1]
    return True
```

Even worse than the first one.