METU Computer Engineering

CEng 240 – Spring 2021
Week 2

Sinan Kalkan

A Broad Look at Programming and PL, Representation of data in computers

Disclaimer: Figures without reference are from either from "Introduction to programming concepts with case studies in Python" or "Programming with Python for Engineers", which are both co-authored by me.

Previously on CENG240!

METU Computer Engineering

Introduction to the Course

https://ceng240.github.io/

- Objectives
  - This course gives a brief introduction to a working understanding of basic computer organization, data representation, programming language constructs, and algorithmic thinking. It is designed as a first course of programming and supported by laboratory sessions for students outside of the Computer Engineering major.
- Textbook
  - *Programming with Python for Engineers,* by S. Kalkan, O. T. Şehitoğlu and G. Üçoluk. Available at: https://pp4e-book.github.io/
- Course conduct
  - Weekly pre-recorded lectures released before the week.
  - 2-hour live sessions with instructors.
  - Office hours with the assistants.
  - Lab exams.
  - Midterm exam and final exam.

METU Computer Engineering

Previously on CENG240!

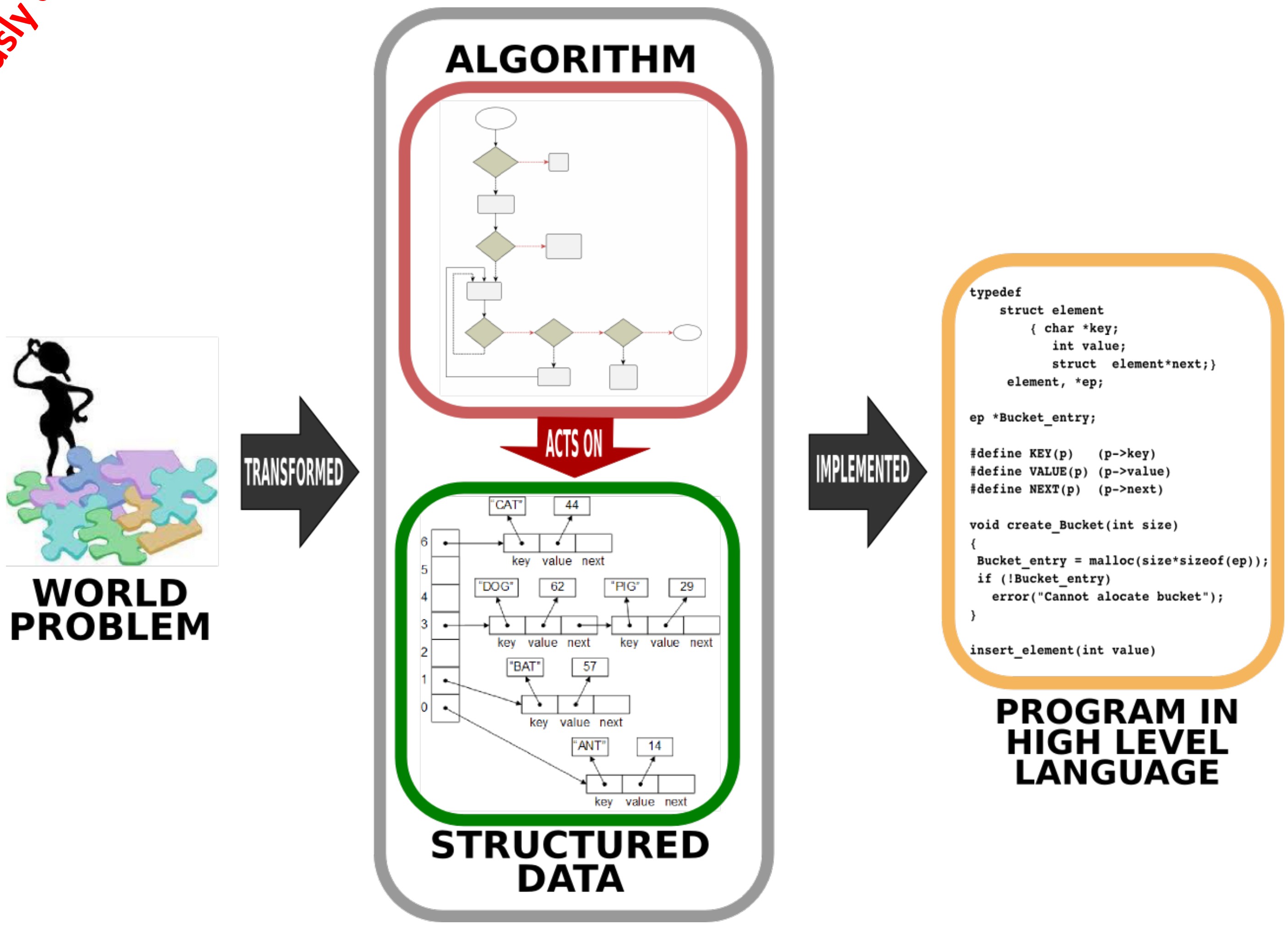
What is a computer?

- **The most common context**: An electronic device that has a 'microprocessor' in it.
  - Binary
- **The broader context**: Any physical entity that can do 'computation'.

https://en.m.wikipedia.org/wiki/File:Computer_from_inside_018.jpg

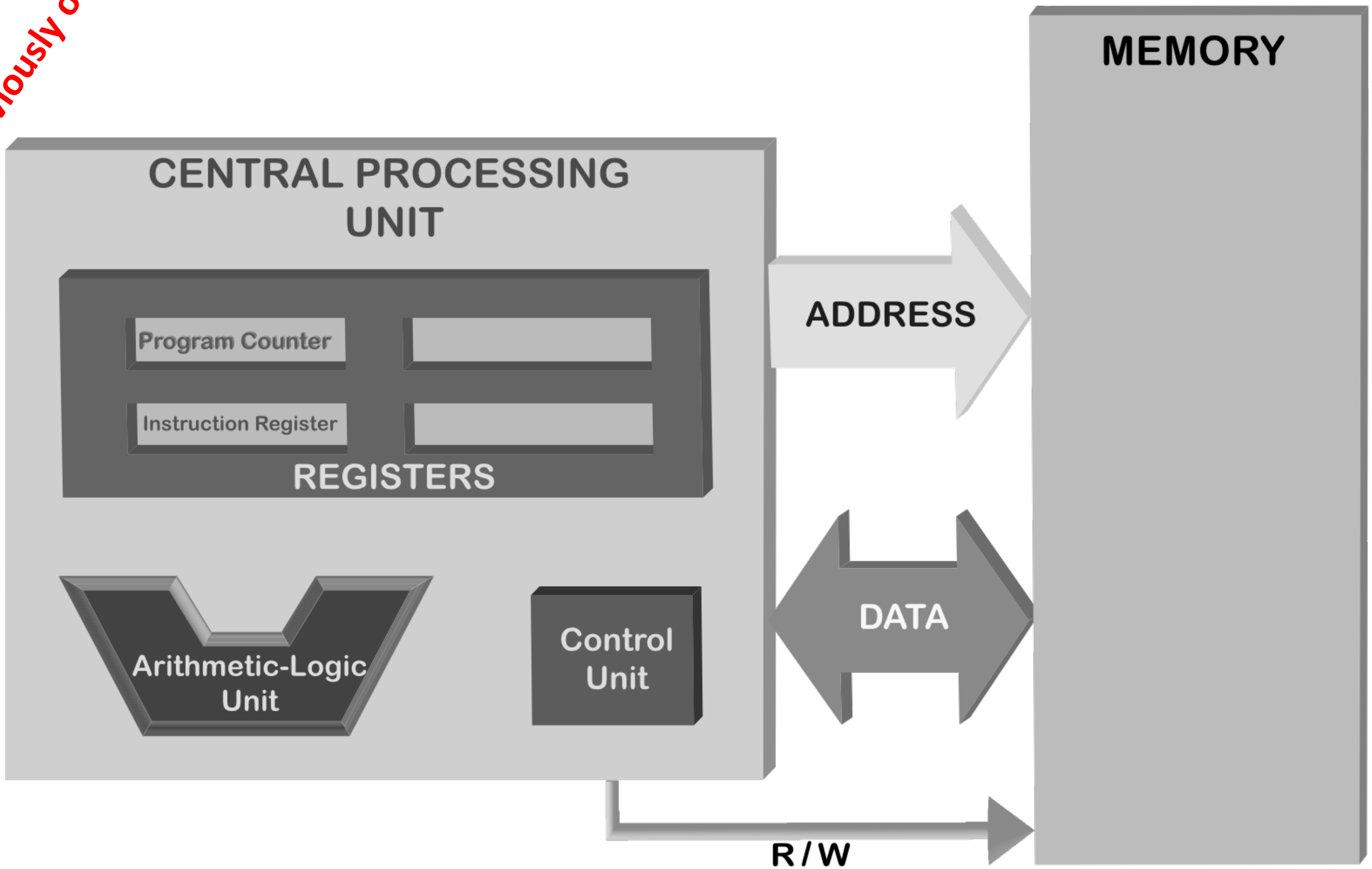
Previously on CENG240!

What is programming?



Previously on CENG240!

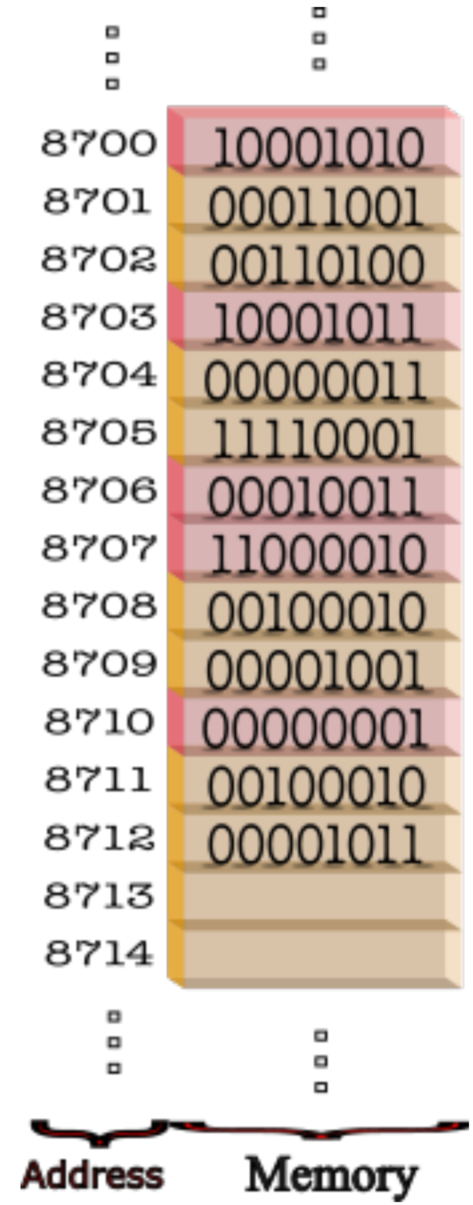
Von Neumann Architecture



Previously on CENG240!

Memory

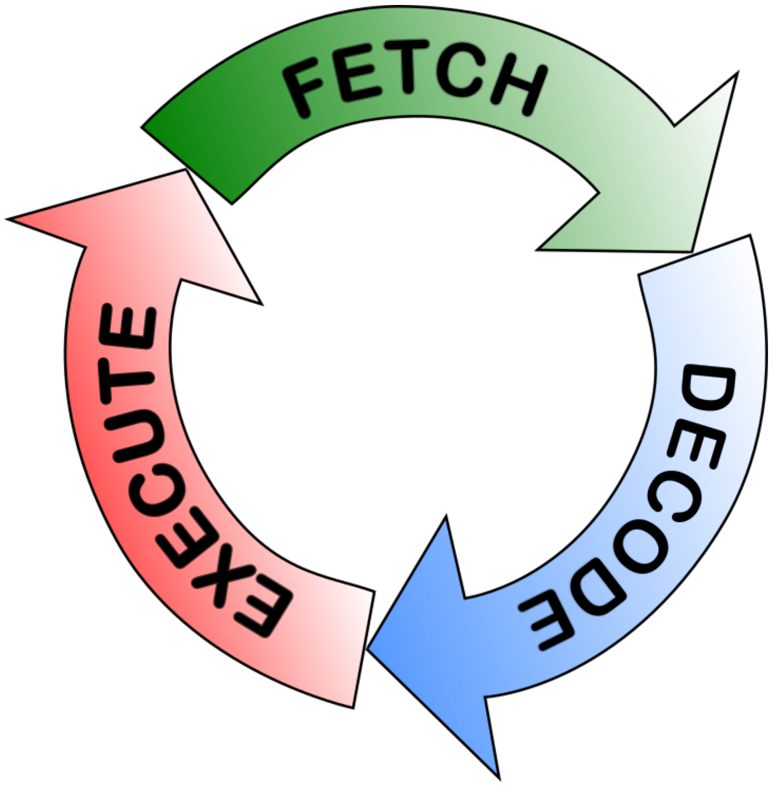
- Random Access Memory (RAM)
- Allows reading and writing operations
- Each access requires an address



Previously on CENG240!

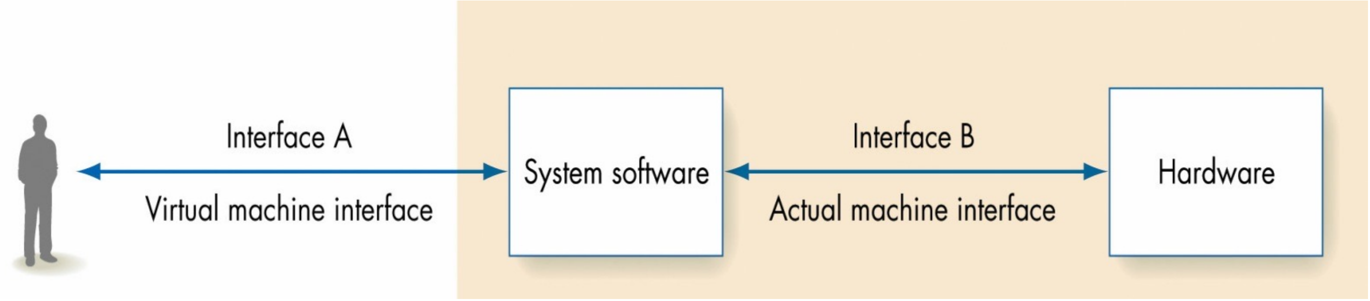
Fetch, decode, execute cycle

- Fetch
  - Retrieve the next instruction from the memory
- Decode
  - Look at the opcode of the instruction and decode what actions should be performed.
- Execute
  - Execute the actions identified in the decode phase.



Previously on CENG240!

OS



From: “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting

- Memory management
- Process management
- Device management
- File management
- Security
- User interface

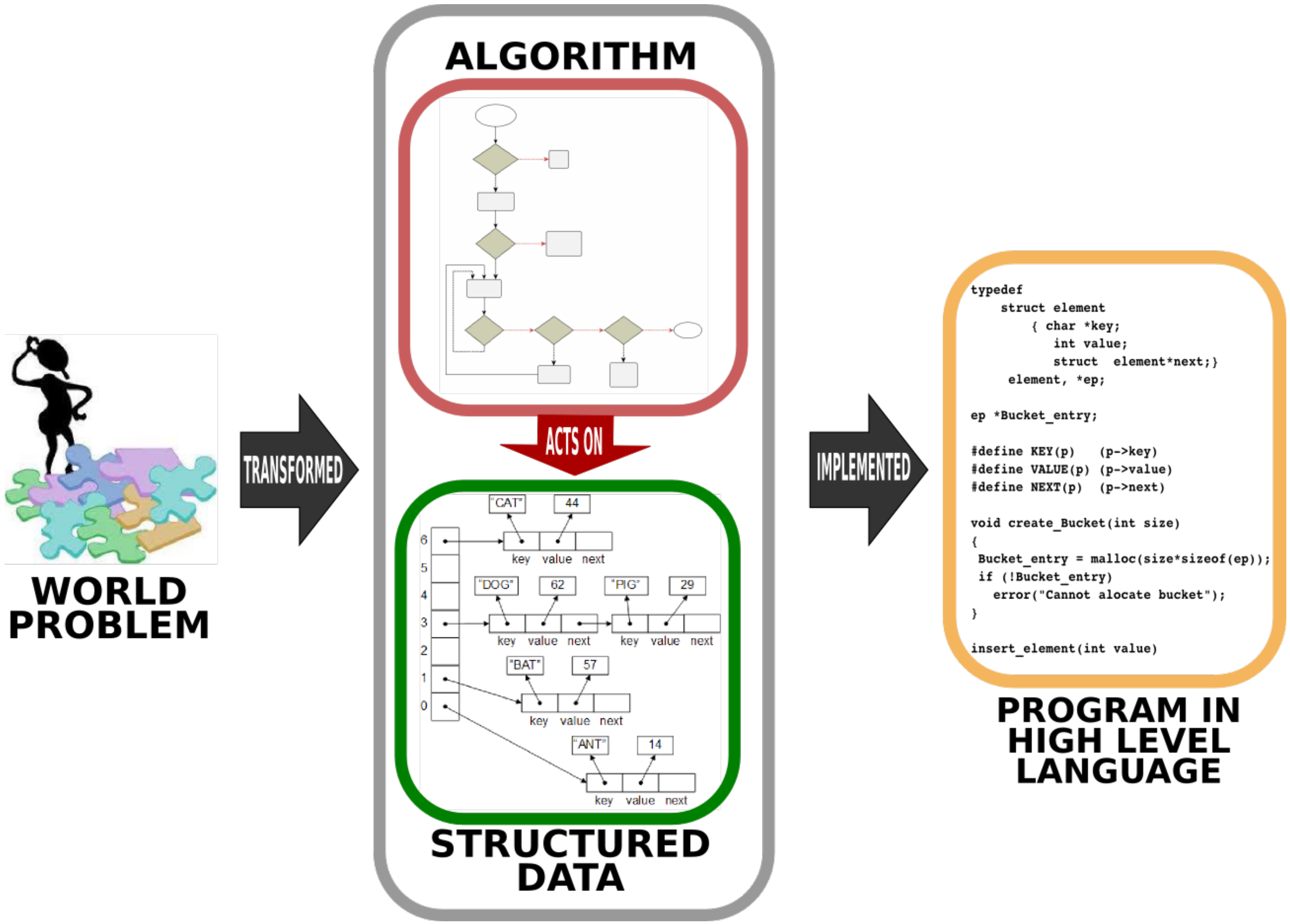
This Week

- A Broad Look at Programming and PL (CH2)
  - Concept of Algorithm, Comparing algorithms, World of PLs, Low-High level PL, Interpreter vs Compiler, Programming Paradigms, Python as a PL

- Representation of data in computers (CH3)
  - Two’s complement representation of integers, IEEE floating-point representation, Information loss with Floating Points, representation of characters, text and Boolean.

Administrative Notes

- Quiz 1 announced!
- Labs starting in two weeks.
- Midterm: 1 June, Tuesday, 17:40


WORLD PROBLEM
TRANSFORMED
ALGORITHM
ACTS ON
"CAT" 44
key value next
"DOG" 62
"PIG" 29
"BAT" 57
"ANT" 14
6 5 4 3 2 1 0
STRUCTURED DATA
IMPLEMENTED
typedef
struct element
{ char *key;
int value;
struct element*next;}
element, *ep;
ep *Bucket_entry;
#define KEY(p) (p->key)
#define VALUE(p) (p->value)
#define NEXT(p) (p->next)
void create_Bucket(int size)
{
Bucket_entry = malloc(size*sizeof(ep));
if (!Bucket_entry)
error("Cannot alocate bucket");
}
insert_element(int value)
PROGRAM IN HIGH LEVEL LANGUAGE

Concept of Algorithm, Comparing algorithms, World of PLs, Low-High level PL, Interpreter vs Compiler, Programming Paradigms, Python as a PL

A BROAD LOOK AT PROGRAMMING AND PL (CH2)

What does ‘algorithm’ mean?

- “A procedure or formula for solving a problem”
- “A set of instructions to be followed to solve a problem”
- “an effective method expressed as a finite list of well-defined instructions for calculating a function”
- “step-by-step procedure for calculations”

A formal definition of algorithm

- "Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, will proceed through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state."

What is an algorithm?

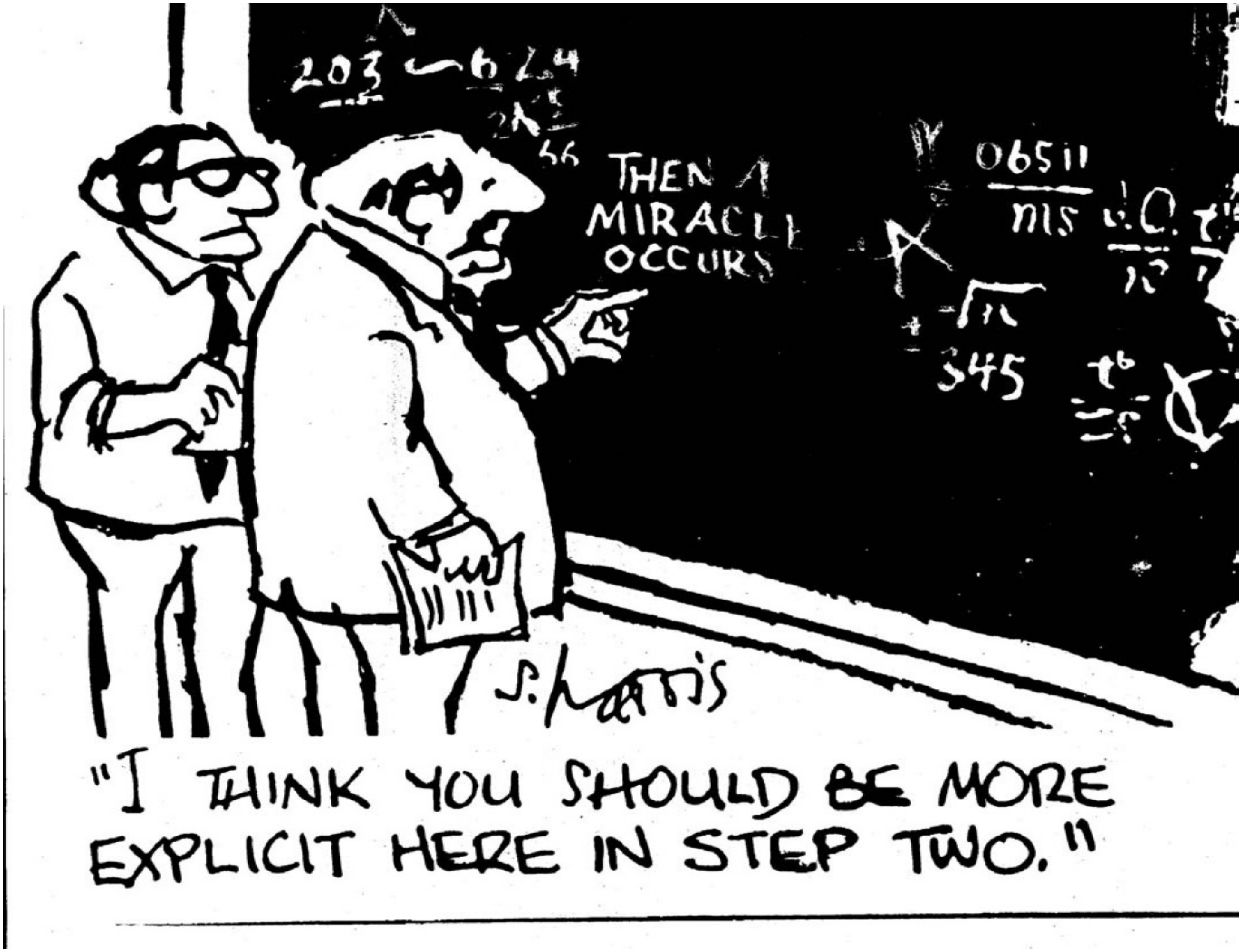
- An algorithm is a list that looks like
  - STEP 1: Do something
  - STEP 2: Do something
  - STEP 3: Do something
  - . .
  - . .
  - . .
  - STEP N: Stop, you are finished

From "Invitation to Computer Science"

Valid Operations in Algorithms

- **Sequential** – simple well-defined task, usually declarative sentence.
- **Conditional**- “ask a question and select the next operation on the basis of the answer to the question – usually an “if-then” or “if then else”
- **Iterative**- “looping” instructions – repeat a set of instructions

From “Invitation to Computer Science”



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

From "Invitation to Computer Science"

An example algorithm from our daily lives

METU Computer Engineering

Algorithm for Shampooing Your Hair

Step	Operation
1	Wet your hair
2	Set the value of *WashCount* to 0
3	Repeat steps 4 through 6 until the value of *WashCount* equals 2
4	Lather your hair
5	Rinse your hair
6	Add 1 to the value of *WashCount*
7	Stop, you have finished shampooing your hair

From "Invitation to Computer Science"

Describing algorithms

Option 1: Use pseudo-code descriptions.

Algorithm. Calculate the average of numbers provided by the user.

Input: N -- the count of numbers
Output: The average of N numbers to be provided

Step 1: Get how many numbers will be provided and store that in N
Step 2: Create a variable named Result with initial value 0
Step 3: Execute the following step N times:
Step 4: Get the next number and add it to Result
Step 5: Divide Result by N to obtain the average

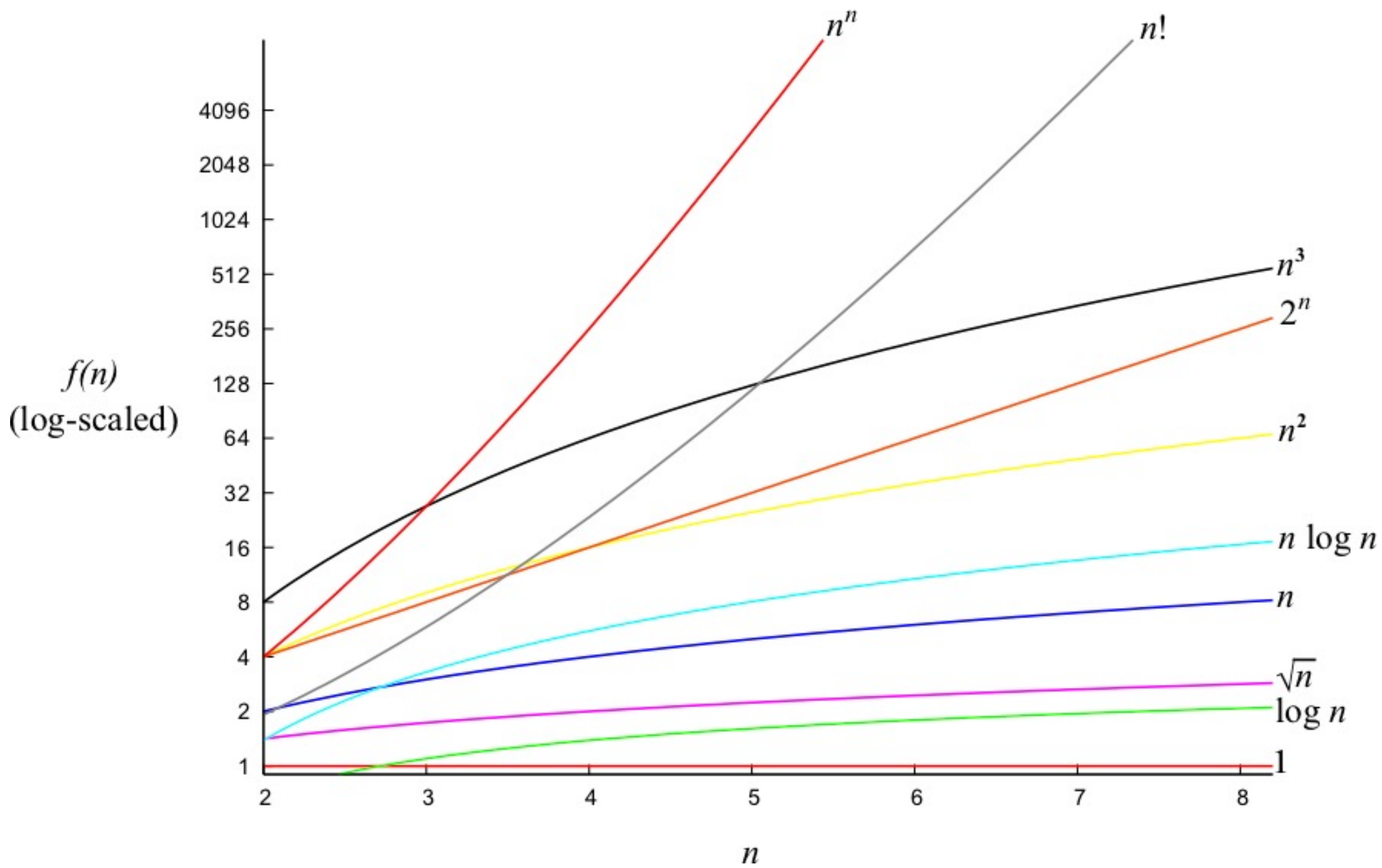
Describing algorithms

Option 2: Use flow-charts.

Comparing Algorithms

- Rougly count the main number of steps in terms of n, the ‘size’ of the problem.
- Example: Guess my number!
  - Random guessing
  - Sweeping from beginning
  - Middle guessing

Comparing Algorithms



The World of Programming Languages

Low-level Languages

Machine Language

```
01010101 01001000
10001001 11100101
10001011 00010101
10110010 00000011
...
```

Assembly Language

```
pushq   %rbp
movq    %rsp, %rbp
movl    alice(%rip), %edx
movl    bob(%rip), %eax
imull   %edx, %eax
movl    %eax, carol(%rip)
...
```

High-level Languages

Compiled & Interpreted Languages (Python, C/C++, ..)

```
int alice = 123;
int bob = 456;
int carol;
main(void)
{
    carol = alice*bob;
}
```

Pseudocode

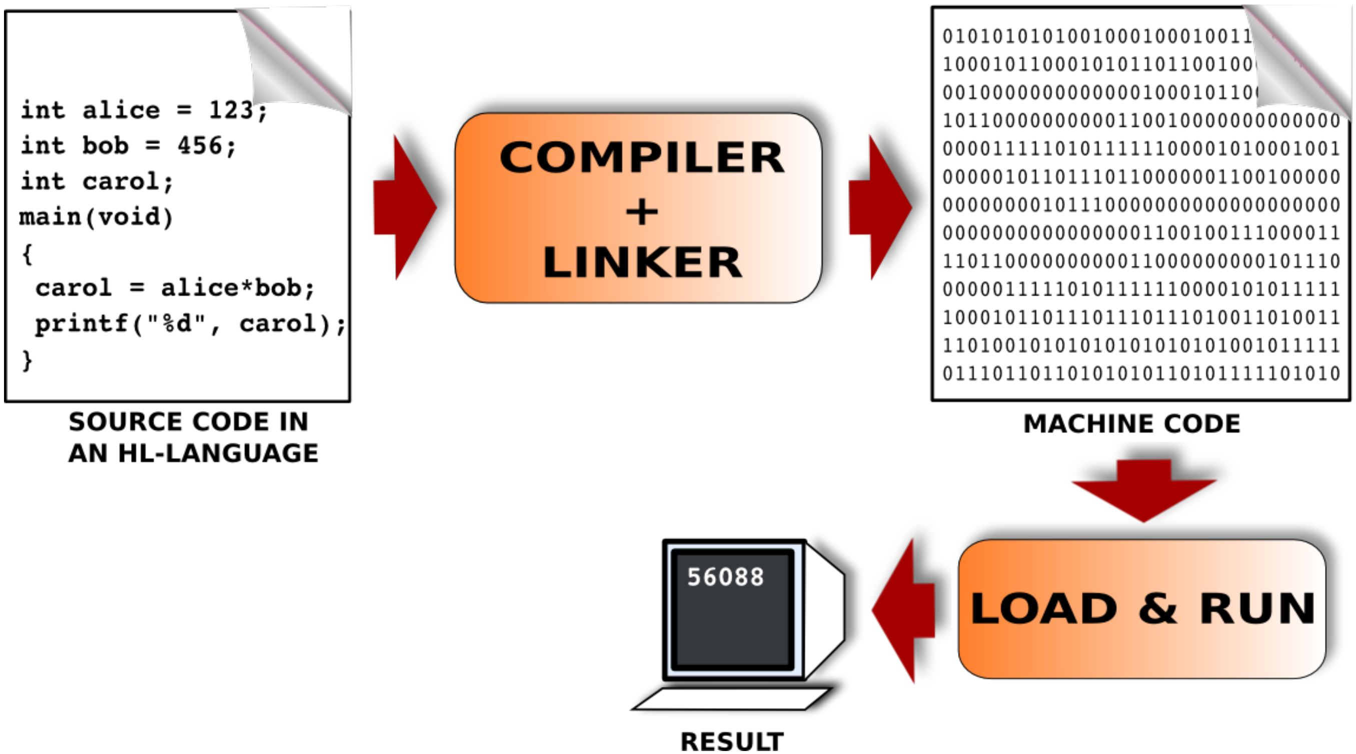
- Initialize alice to 123 and
  bob to 456
- Multiply alice and bob and
store the result into carol

Natural Languages

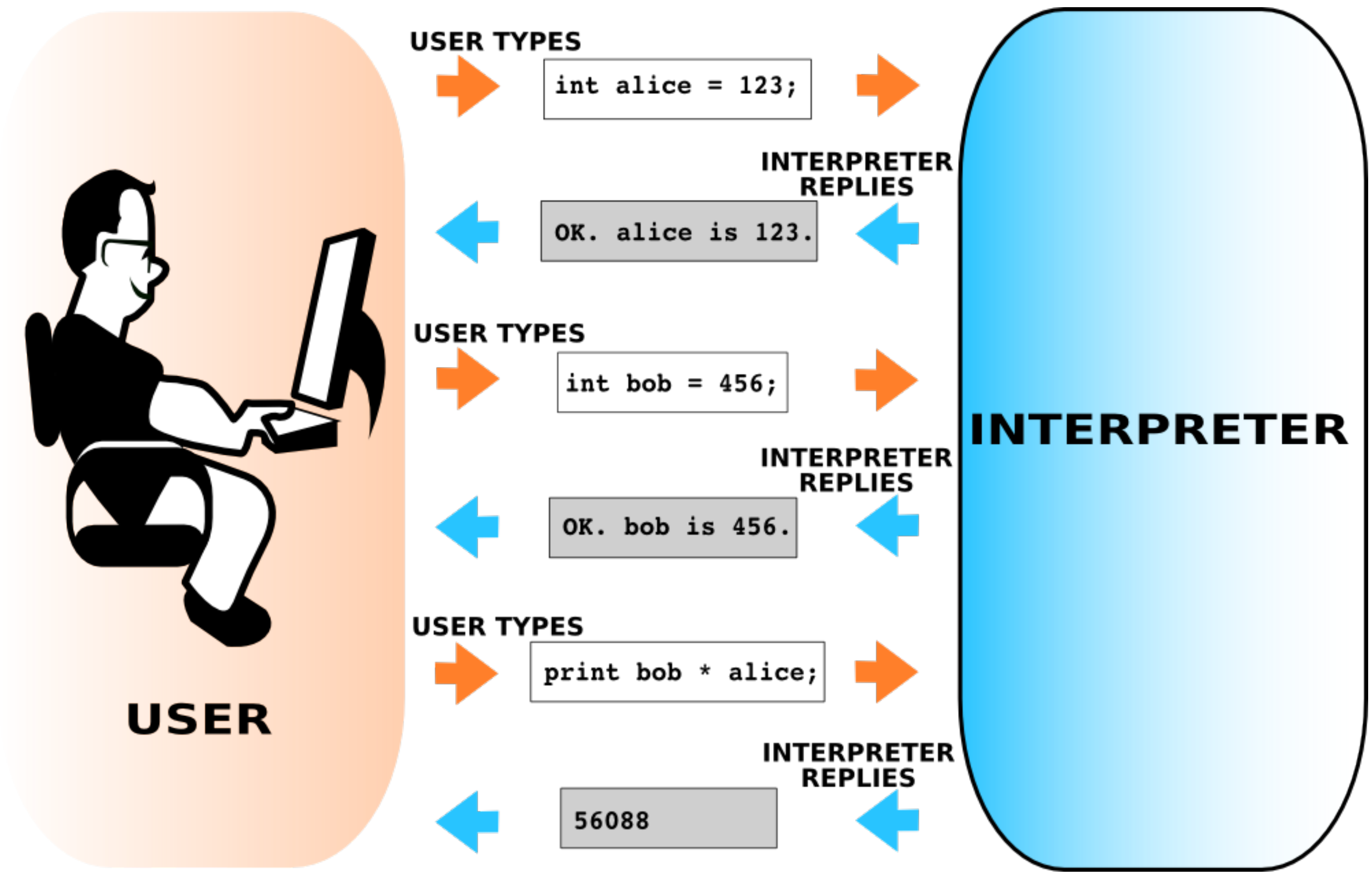
English, Turkish, …

Given two variables called alice and bob with initial values 123 and 456, respectively, multiply them and store the result into another variable called carol.

Interpreter vs. Compiler



Interpreter vs. Compiler



PL Paradigms

- Imperative
- Functional
- Object-oriented
- Logical-declarative
- Concurrent
- Event-driven

MEET python™

Guido van Rossum (1956 -)

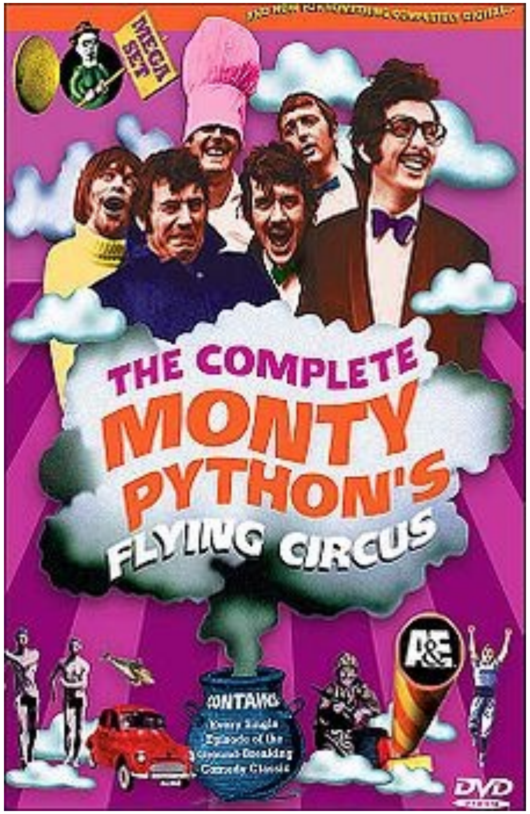
- Zen of Python [https://en.wikipedia.org/wiki/Zen_of_Python]
  - Beautiful is better than ugly.
  - Explicit is better than implicit.
  - Simple is better than complex.
  - Complex is better than complicated.
  - Flat is better than nested.
  - Sparse is better than dense.
  - Readability counts.
  - …

python™

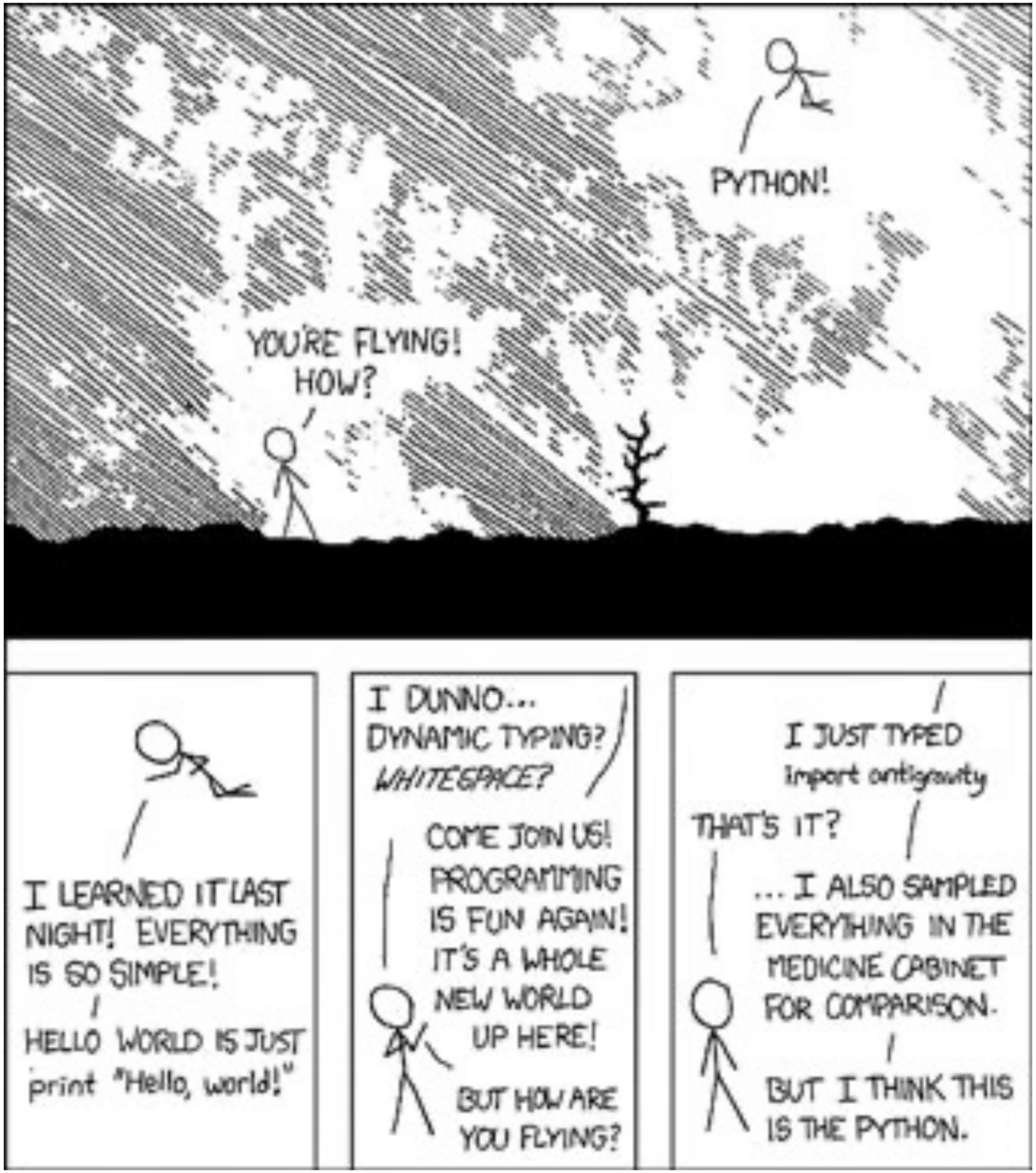
- An interpretive/scripting PL that:
  - Longs for code readability
  - Ease of use, clear syntax
  - Wide range of applications, libraries, tools
- Multiple Paradigms:
  - Functional
  - Imperative
  - Object-oriented

- Started at the end of 1980s.
- V2.0 was released in 2000
  - With a big change in development perspective: Community-based
  - Major changes in the facilities.
- V3.0 was released in 2008
  - Backward-incompatible
  - Some of its features are put into v2.6 and v2.7.

- Where does the name come from?
  - While van Possum was developing Python, he read the scripts of Monty Python’s Flying Circus and thought ‘python’ was “short, unique and mysterious” for the new language [1]
- One goal of Python: “fun to use”
  - The origin of the name is the comedy group “Monty Python”
  - This is reflected in sample codes that are written in Python by the original developers.



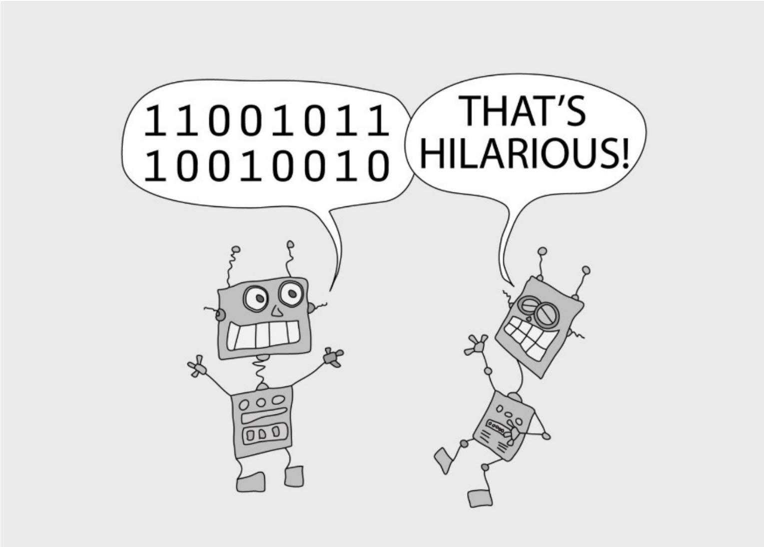
[1] https://docs.python.org/2/faq/general.html#why-is-it-called-python

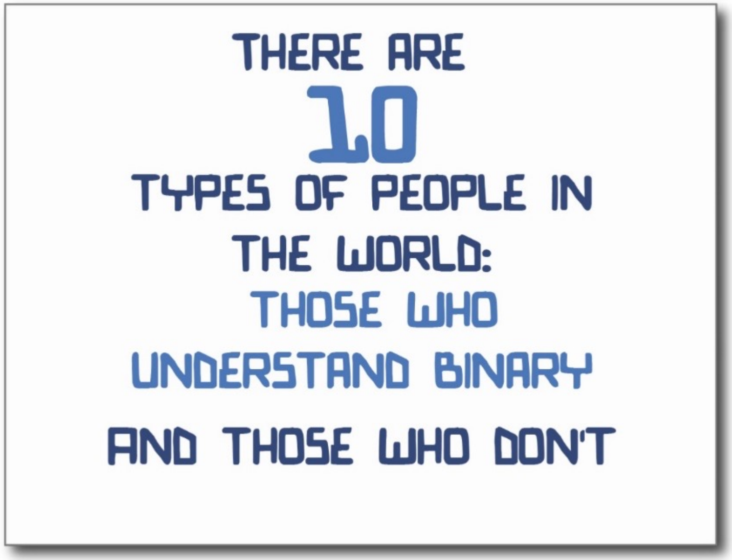

YOU'RE FLYING!
HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST
print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED
import antigravity
THAT'S IT?
...I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

METU Computer Engineering

```
skalkan@divan:~$ python
Python 2.5.2 (r252:60911, Jan 24 2010, 17:44:40)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

METU Computer Engineering





Two's complement representation of integers, IEEE floating-point representation, Information loss with Floating Points, representation of characters, text and Boolean.

REPRESENTATION OF DATA IN COMPUTERS (CH3)

Data Representation

- Based on 1s and 0s
  - So, everything is represented as a set of binary numbers
- We will now see how we can represent:
  - Integers: 3, 1234435, -12945 etc.
  - Floating point numbers: 4.5, 124.3458, -1334.234 etc.
  - Characters: /, &, +, -, A, a, ^, 1, etc.
  - …

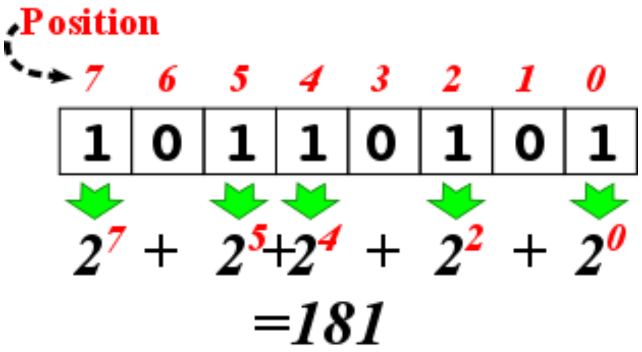
Binary Representation of Numeric Information

- Decimal numbering system
  - Base-10
  - Each position is a power of 10

    $3052 = 3 \times 10^3 + 0 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$

- Binary numbering system
  - Base-2
  - Uses ones and zeros
  - Each position is a power of 2

    $1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$



Decimal-to-binary Conversion

	Dividend		Divisor		Quotient	Remainder
Step 1	19	÷	2	=	9	1
Step 2	9	÷	2	=	4	1
Step 3	4	÷	2	=	2	0
Step 4	2	÷	2	=	1	0
Step 5	1	÷	2	=	0	1

Continue until quotient is zero

The result:

1	0	0	1	1

Binary Representation of Numeric Information (continued)

- Representing integers
  - Decimal integers are converted to binary integers
  - Question: given k bits, what is the value of the largest integer that can be represented?
  - $2^k - 1$
    - Ex: given 4 bits, the largest is 2^4-1 = 15
- Signed integers must also represent the sign (positive or negative) - ***Sign/Magnitude notation***

Binary Representation of Numeric Information (continued)

- Sign/magnitude notation

  1 101 = - 5

  0 101 = + 5
- Problems:
  - Two different representations for 0:
    - 1 000 = -0
    - 0 000 = +0
  - Addition & subtraction require a watch for the sign! Otherwise, you get wrong results:
    - 0 010 (+2) + 1 010 (-2) = 1 100 (-4)

METU Computer Engineering

Arithmetic in Computers is Modular

Let's add two numbers in binary
(Assume that there is no sign bit)

	1	0	1	1
+	1	1	1	0
~~1~~	1	0	0	1

$$
\begin{array}{r}
(11)_{10} \\
+\ (14)_{10} \\
\hline
(9)_{10}
\end{array}
$$

In other words:

- Numbers larger than or equal to 16 (2^4) are discarded in a 4-bit representation.
- Therefore, $11 + 14$ yields 9 in this 4-bit representation.
- This is actually modular arithmetic:

$$11 + 14\ mod\ 16 \equiv 9\ mod\ 16$$

Binary Representation of Numeric Information (continued)

- Two's complement instead of sign-magnitude representation
  - Positive numbers have a leading 0.
    - 5 => 0101
  - The representation for negative numbers is found by subtracting the absolute value from 2^N for an N-bit system:
    - $-5 \Rightarrow 2^4 - 5 = 16 - 5 = (11)_{10} \Rightarrow (1011)_2$
- Advantages:
  - 0 has a single representation: +0 = 0000, -0 = 0000
  - Arithmetic works fine without checking the sign bit:
    - 1011 (-5) + 0110 (6) = 0001 (1)
    - 1011 (-5) + 0011 (3) = 1110 (-2)

Binary Representation of Numeric Information (continued)

- Shortcut to convert from “two’s complement” :
  - If the leading bit is zero, no need to convert.
  - If the leading bit is one, invert the number and add 1.
- What is our range?
  - With 2’s complement we can represent numbers from -2^{N-1} to $2^{N-1} - 1$ using N bits.
  - 8 bits: -128 to +127.
  - 16 bits: -32,768 to +32,767.
  - 32 bits: -2,147,483,648 to +2,147,483,647.

Binary Number	Decimal Value	Value in Two’s Complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Binary Representation of Numeric Information (continued)

- Example:
  - We want to compute: 12 – 6
  - 12 => 01100
  - -6 => -(00110) => (11001)+1 => (11010)
- 12 – 6 =

```
  01100
+ 11010
-----------
  00110 => 6
```

So, addition and subtraction operations are simpler in the Two's Complement representation

Binary Representation of Numeric Information (continued)

- Due to its advantages, two's complement is the most common way to represent integers on computers.

Why does Two's Complement work?

- Inversion and addition of a 1-bit correspond effectively to subtraction from 0 – i.e., negative of a number.
- Negative of a binary number X: $(00...00)_2 - (X)_2$
- Note that $(00...00)_2 = (11...11)_2 + (1)_2$
- In other words:
  - $(00...00)_2 - (X)_2 = \underbrace{(11...11)_2 - (X)_2}_{\text{Inversion}} + (1)_2$.

(i.e., how we find two's complement)

Inversion

Why does Two's Complement work?

- A 2nd perspective:
  - i - j mod 2^N = i + (2^N-j) mod 2^N
  - Example:
    - Consider X and Y are positive numbers.
    - $X + (-Y) = X + (2^N - Y)$
      $= 2^N - (Y - X) = -(Y - X) = X - Y$

Why does Two's Complement work?

- A smart trick used in mechanical calculators
  - To subtract b from a, invert b and add that to a. Then discard the most significant digit.

http://en.wikipedia.org/wiki/Method_of_complements

Binary Representation of Real Numbers

Conversion of the digits after the dot into binary:

- 1st Way:
  - 0.375 → 0x½ + 1x¼ + 1x1/8 → 011

- 2nd Way:

	Fraction		Multiplier		Whole		Fraction
Step 1	0.375	×	2	=	0	.	75
Step 2	0.75	×	2	=	1	.	5
Step 3	0.5	×	2	=	1	.	0

The result:

.	0	1	1

Continue until fraction is zero

Binary Representation of Real Numbers

- Approach 1: Use fixed-point
  - Similar to integers, except that there is a decimal point.
  - E.g.: using 8 bits:

1	1	1	1	1	1	1	1

$= 1\times2^{3} + 1\times2^{2} + 1\times2^{1} + 1\times2^{0} + 1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$

$= 15.9375$

Assumed decimal point

Binary Representation of Real Numbers

- Location of the decimal point changes the value of the number.
  - E.g.: using 8 bits:

1	1	1	1	1	1	1	1

$$= 1\times2^{4} + 1\times2^{3} + 1\times2^{2} + 1\times2^{1} + 1\times2^{0}$$
$$1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} = 31.875$$

Assumed decimal point

Binary Representation of Real Numbers

- Problems with fixed-point:
  - Limited in the maximum and minimum values that can be represented.
  - For instance, using 32-bits, reserving 1-bit for the sign and putting the decimal point after 16 bits from the right, the maximum positive value that can be stored is slightly less than 2^{15}.
  - Allowing larger values gives away from the precision (the decimal part).

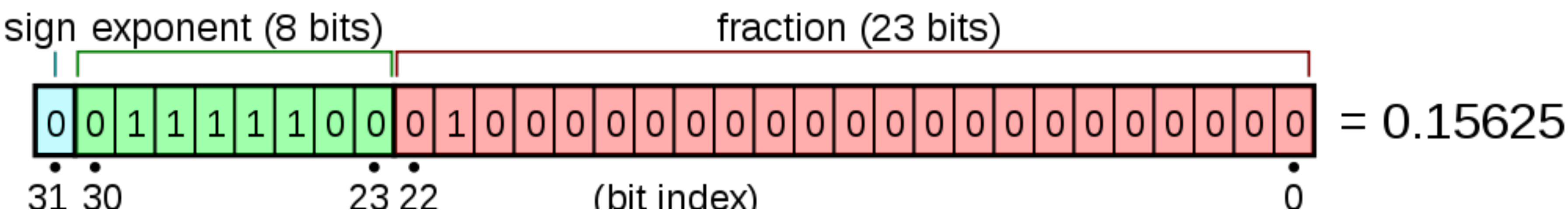
Binary Representation of Real Numbers

- Solution: Use scientific notation: a x 2^b *(or ±M x $B^{\pm E}$)*
  - Example: 5.75
    - 5 → 101
    - 0.75 → ½ + ¼ → $2^{-1}+2^{-2}$ → $(0.11)_2$
    - 5.75 → $(101.11)_2 \times 2^0$
- Number is then normalized so that the first significant digit is immediately to the left of the binary point
  - Example: 1.0111×2^2
- We take and store the mantissa and the exponent.

Binary Representation of Real Numbers

- This needs some standardization for:
  - where to put the decimal point
  - how to represent negative numbers
  - how to represent numbers less than 1

IEEE 32bit Floating-Point Number Representation



$$= (-1)^{\text{sign}}(1.b_{-1}b_{-2}...b_{-23})_2 \times 2^{e-127}$$

$(2 - 2^{-23}) \times 2^{127}$

- M x 2^E
- Exponent (E): 8 bits
  - Add 127 to the exponent value before storing it
  - **E can be 0 to 255 with 127 representing the real zero.**
- Fraction (M - Mantissa): 23 bits
- $2^{128} = 1.70141183 \times 10^{38}$

METU Computer Engineering

IEEE 32bit Floating-Point Number Representation

- Example: 12.375
- The digits before the dot:
  - $(12)_{10} \rightarrow (1100)_2$
- The digits after the dot:
  - 1st Way: 0.375 → 0x½ + 1x¼ + 1x1/8 → 011
  - 2nd Way: Multiply by 2 and get the integer part until 0:
    - 0.375 x 2 = 0.750 = 0 + 0.750
    - 0.750 x 2 = 1.50 = 1 + 0.50
    - 0.50 x 2 = 1.0 = 1 + 0.0
- $(12.375)_{10} = (1100.011)_2$
- NORMALIZE (move the point): $(1100.011)_2 = (1.100011)_2 x2^3$
- Exponent: 3, adding 127 to it, we get 1000 0010
- Fraction: 100011
- Then our number is: 0 10000010 10001100000000000000000

Why add bias to the exponent?

- It helps in comparing the exponents of the same-sign real-numbers without looking out for the sign of the exponent.

Binary Number	Decimal Value	Value in Two's Complement	Value with bias 7
0000	0	0	-7
0001	1	1	-6
0010	2	2	-5
0011	3	3	-4
0100	4	4	-3
0101	5	5	-2
0110	6	6	-1
0111	7	7	0
1000	8	-8	1
1001	9	-7	2
1010	10	-6	3
1011	11	-5	4
1100	12	-4	5
1101	13	-3	6
1110	14	-2	7
1111	15	-1	8

To read more on this:
https://blog.angularindepth.com/the-mechanics-behind-exponent-bias-in-floating-point-9b3185083528

IEEE 32bit Floating-Point Number Representation

- Zero:
  - Exponent: All zeros
  - Fraction: All zeros
  - +0 and -0 are different numbers but they are equal!
- Infinity:
  - Exponent: All ones
  - Fraction: All zeros
- Not a number (NaN):
  - Exponent: All ones
  - Fraction: non-zero fraction.

http://steve.hollasch.net/cgindex/coding/ieeefloat.html

IEEE 32bit Floating-Point Number Representation

- What is the maximum positive IEEE floating point value that can be stored?
  - Just less than 2^{128} $[(2-2^{-23})\times 2^{127}$ to be specific]
  - Why? 2^{128} is reserved for NaN.

- Check out these useful links:
  - http://steve.hollasch.net/cgindex/coding/ieeefloat.html
  - http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html

IEEE 32bit Floating-Point Number Representation

- Now consider 4.1:
  - 4 => $(100)_2$
  - 0.1 =>
    - x 2 = 0.2 = 0 + 0.2
    - x 2 = 0.4 = 0 + 0.4
    - x 2 = 0.8 = 0 + 0.8
    - x 2 = 1.6 = 1 + 0.6
    - x 2 = 1.2 = 1 + 0.2
    - x 2 = 0.4 = 0 + 0.4
    - x 2 = 0.8 = 0 + 0.8
    - ....
- So,
  - Representing a fraction which is a multiple of $1/2^n$ is lossless.
  - Representing a fraction which is not a multiple of $1/2^n$ leads to precision loss.

Representing Real Numbers: Information Loss

```
>>> 0.9375 - 0.9
0.03749999999999998
```

```
>>> 2000.0041 - 2000.0871
-0.0829999999998563

>>> 2.0041 - 2.0871
-0.08299999999999974
```

```
>>> sin(PI)
1.2246467991473532e-16
>>> cos(PI)
-1.0
```

```
>>> A = 1234.567
>>> B = 45.67834
>>> C = 0.0004
>>> AB = A + B
>>> BC = B + C
>>> print (AB+C)
1280.2457399999998

>>> print (A+BC)
1280.2457400000001
```

Representing Real Numbers: Information Loss

- What can you do?

1. It is in your best interest to refrain from using floating points. If it is possible transform the problem to the integer domain.
2. Use the most precise type of floating point provided by your high-level language, some languages provide you with 64 bit or even 128 bit floats, use them.
3. Use less precision floating points only when you are in short of memory.
4. Subtracting two floating points close in value has a potential danger.
5. If you add or subtract two numbers which are magnitude-wise not comparable (one very big the other very small), it is likely that you will lose the proper contribution of the smaller one. Especially when you iterate the operation (repeat it many times), the error will accumulate.
6. You are strongly advised to use well-known, commonly used scientific computing libraries instead of coding floating point algorithms by yourself.

Numbers in Python

- Integers
- Floating point numbers
- Complex numbers

Representing Boolean Values

- The CPU often needs to compare numbers, or data:
  - $3 >^? 4$
  - $125 \ =^? 1000/8$
  - $3 \ \leq^? \ 12345.34545/12324356.0$
- We have the truth values for representing the answers to such comparisons:
  - If correct: TRUE, True, true, T, 1
  - If not correct: FALSE, False, false, F, 0

Binary Representation of Textual Information

- ASCII (American Standard Code for Information Interchange) code set
  - Originally: 7 bits per character; 128 character codes
- Unicode code set
  - 16 bits per character
- UTF-8 (Universal Character Set Transformation Format) code set.
  - Variable number of 8-bits.

Binary Representation of Textual Information (cont'd)

ASCII
7 bits long

Decimal	Binary	Val.
48	00110000	0
49	00110001	1
50	00110010	2
51	00110011	3
52	00110100	4
53	00110101	5
54	00110110	6
55	00110111	7
56	00111000	8
57	00111001	9
58	00111010	:
59	00111011	;
60	00111100	<
61	00111101	=
62	00111110	>
63	00111111	?
64	01000000	@
65	01000001	A
66	01000010	B

Hex.	Unicode	Charac.
0x30	0x0030	0
0x31	0x0031	1
0x32	0x0032	2
0x33	0x0033	3
0x34	0x0034	4
0x35	0x0035	5
0x36	0x0036	6
0x37	0x0037	7
0x38	0x0038	8
0x39	0x0039	9
0x3A	0x003A	:
0x3B	0x003B	;
0x3C	0x003C	<
0x3D	0x003D	=
0x3E	0x003E	>
0x3F	0x003F	?
0x40	0x0040	@
0x41	0x0041	A
0x42	0x0042	B

Unicode
16 bits long

Partial listings only!

UTF-8 Illustrated

Bits	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+007F	0xxxxxxx					
11	U+07FF	110xxxxx	10xxxxxx				
16	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
21	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+3FFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Character		Binary code	Binary UTF-8
$	U+0024	0100100	00100100
¢	U+00A2	00010100010	11000010 10100010
€	U+20AC	0010000010101100	11100010 10000010 10101100
𤭢	U+24B62	000100100101101100010	11110000 10100100 10101101 10100010

How about a text?

- Text in a computer has two alternative representations:

1. A fixed-length number representing the length of the text followed by the binary values of the characters in the text.
   - Ex: “ABC” =>

   00000011 01000001 01000001 01000001 (3 ‘A’ ‘B’ ‘C’)

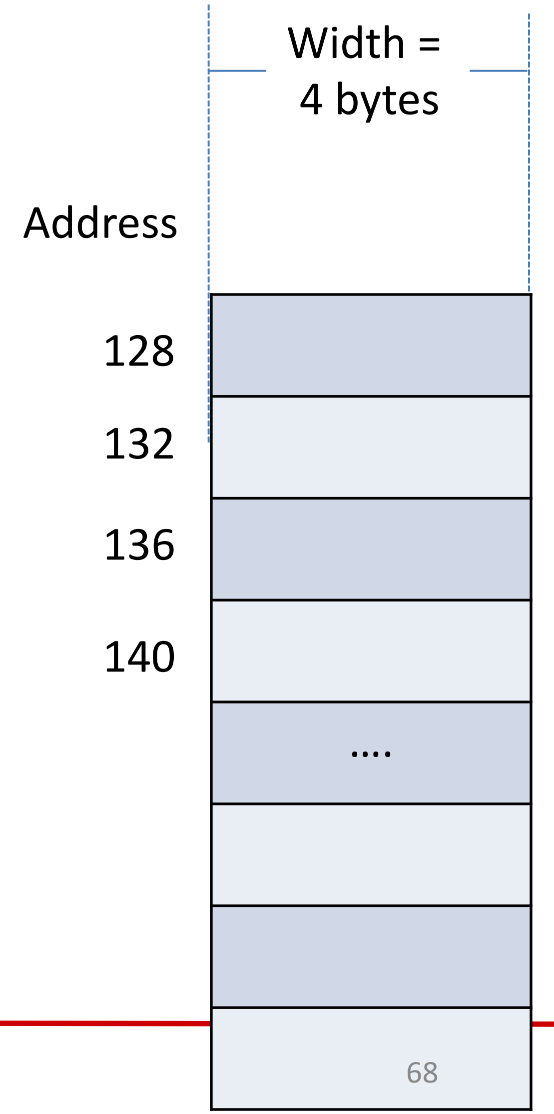
2. Binary values of the characters in the text ended with a unique marker, like “00000000” which has no value in the ASCII table.
   - Ex: “ABC” =>

   01000001 01000001 01000001 00000000 (‘A’ ‘B’ ‘C’ END)

METU Computer Engineering

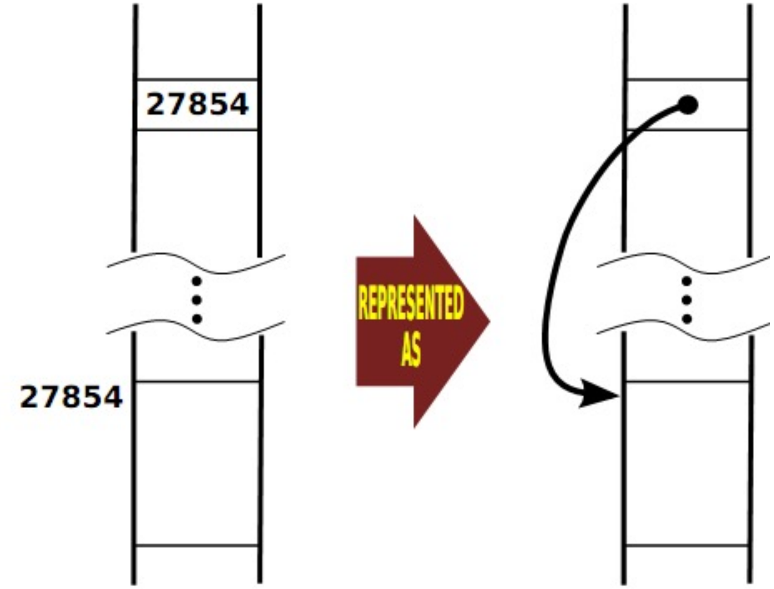
Containers

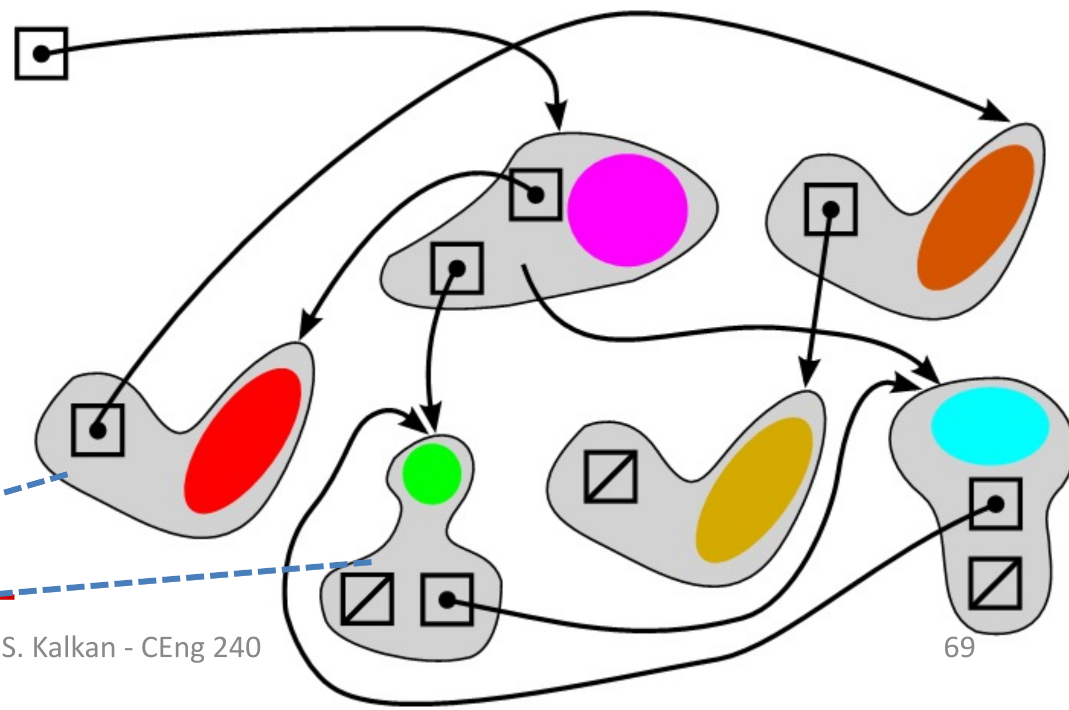
- If you have lots and lots of one type of data (for example, the ages of all the people in Turkey):
  - You can store them into memory consecutively (supported by most PLs)
    - This is called *arrays*.
  - Easy to access an element. Nth element:
    - <Starting-address>+ (N-1)*<Word Width>
    - Ex: 2nd element is at 128 + (2-1) * 4 = 132



Containers

- What if you have to make a lot of deletions and insertions in the middle of an array?
- Then, you have to store your data in blocks/units such that each unit has the starting address of the next unit/block.





Blocks of Data

Final Words: Important Concepts

- The world of programming
  - How we solve problems using computers.
  - Algorithms: What they are, how we write them and how we compare them.
  - The spectrum of programming languages.
  - Pros and cons of low-level and high-level languages.
  - Interpretive vs. compilative approach to programming.
  - Programming paradigms.
- Representation of data
  - Sign-magnitude notation and two’s complement representation for representing integers.
  - The IEEE754 standard for representing real numbers.
  - Precision loss in representing floating point numbers.
  - Representing characters with the ASCII table.
  - Representing truth values.

Final Words: Reading

- The material at the end of the first chapter.

THAT'S ALL FOLKS!
STAY HEALTHY