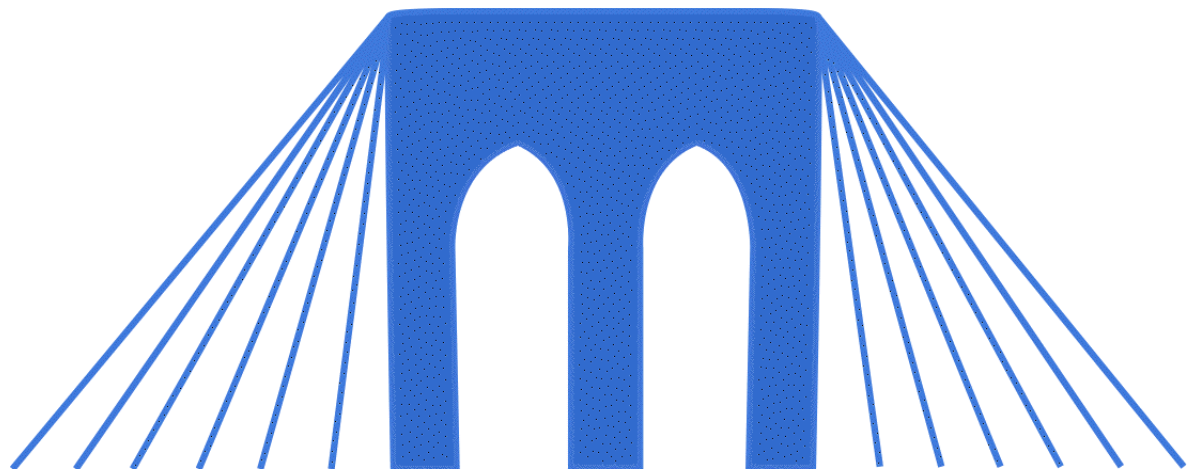
Bridges To Computing



General Information:

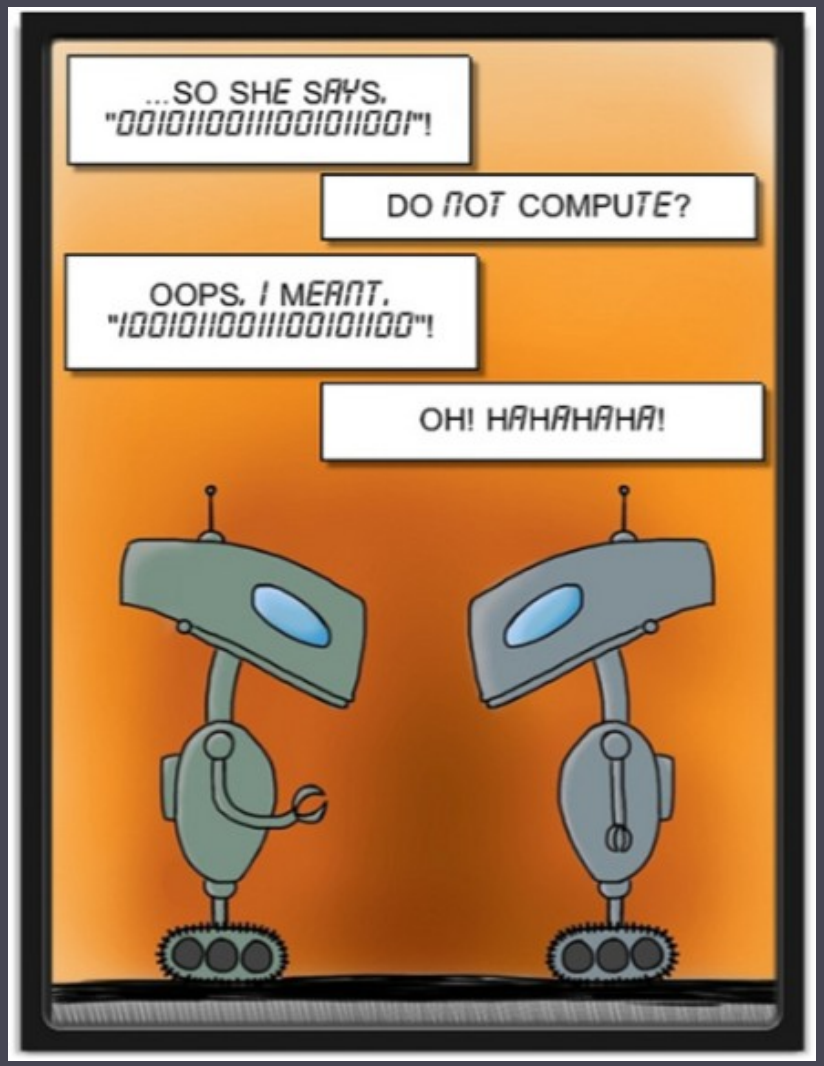
- This document was created for use in the "Bridges to Computing" project of Brooklyn College.
- You are invited and encouraged to use this presentation to promote computer science education in the U.S. and around the world.
- For more information about the Bridges Program, please visit our website at: http://bridges.brooklyn.cuny.edu/

Disclaimers:

- All images in this presentation were created by our Bridges to Computing staff or were found online through open access media sites and are used under the Creative Commons Attribution-Share Alike 3.0 License.
- If you believe an image in this presentation is in fact copyrighted material, never intended for creative commons use, please contact us at http://bridges.brooklyn.cuny.edu/ so that we can remove it from this presentation.
- This document may include links to sites and documents outside of the "Bridges to Computing" domain. The Bridges Program cannot be held responsible for the content of 3rd party sources and sites.

Binary

There are only 10 types of people in the world, those who understand binary and those who don't.



Introduction to Programming Languages

In Binary (ANSI/ANSI/UTF-8) :

01001001 01101110 01110100 01110010 01101111 01100100 01110101 01100011 01110100 01101001 01101111 01101110 00100000
01110100 01101111 00100000 01010000 01110010 01101111 01100111 01110010 01100001 01101101 01101101 01101001 01101110
01100111 00100000 01001100 01100001 01101110 01100111 01110101 01100001 01100111 01100101 01110011 00100000

M. Meyer
Bridges To Computing
2010

Content

1. Languages
    1. Definition & Reliability
    2. Syntax & Semantics
2. Computer Languages
    1. Binary & Assembly
    2. Compilers & Interpreters
    3. Integrated Development Environment (IDE)
    4. Programming vs. Scripting Languages
3. Paradigms
    1. Common Paradigms
        - Imperative
        - Procedural
        - Object-Oriented
    2. Examples

(1.1) Languages

For our purposes: A language is a structured means of information exchange.

- Spoken languages (English, Chinese)
- Written languages (Arabic, Latin)
- Visual languages (ASL, QE)
- Tactile languages (Braille)

Reliable Languages

- Can a painting or piece of music "speak to you"?
- Will it say the same thing to every person?
- Will it say the same thing to you, every time?
- Not all languages are "reliable" means of communication.

(1.2) Syntax & Semantics

- To be useful and reliable, a language must have a well defined syntax and semantics.
- Syntax: The structure and punctuation of the language.
- Semantics: The meaning of objects (words) and combination of objects in the language.

(1.2) Syntax Examples (English)

- Structure: Sentences have a subject (who/what the sentence is about) and a predicate (information about the subject).
- Punctuation: All sentences must end with a full stop ('.' '?' or '!')
  - Example: John was late.

- NOTE: If you get a "syntax error" in a programming language, it is usually a punctuation error.

(1.2) Semantics Examples (English)

- In a language special symbols and combinations of symbols have 'meaning'.
- Example: John was late.
  - John -> A (male) persons name.
  - was -> Past tense state of being.
  - late -> Tardy, deceased (or, in this class, in trouble).

- NOTE: Programming languages have special "reserved words" that have a specific meaning within the language. Examples: int, if, while

(2.1) Computer Languages

- At the lowest level computers are exchanging information in binary format (0,1).
- Combination's of 0's and 1's can have different meanings though, depending on the "encoding scheme" (language) being used.
  - Assembly -> 00001001 -> (Use register 9)
  - ASCI -> 00001001 -> (Tab character)

ASCI/ANSI TABLE

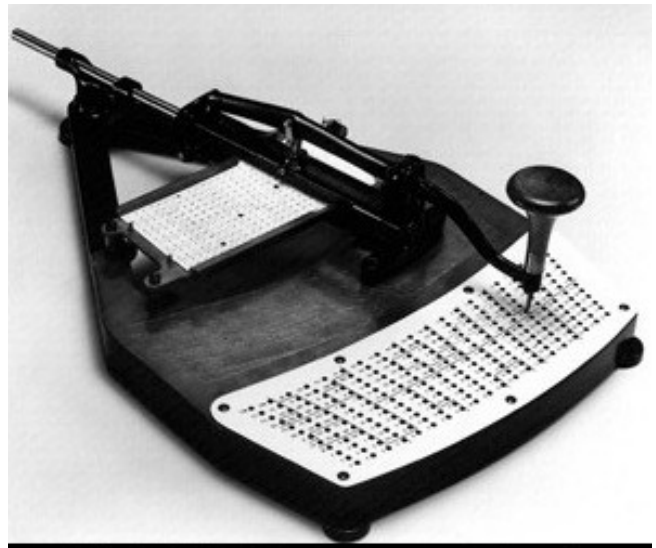
- We can use binary to communicate if we agree on an "encoding".
- An encoding is a like a translation table.
- ASCII & ANSI are two related ways to have 0's and 1's represent letters of the alphabet.

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
				space	0010 0000

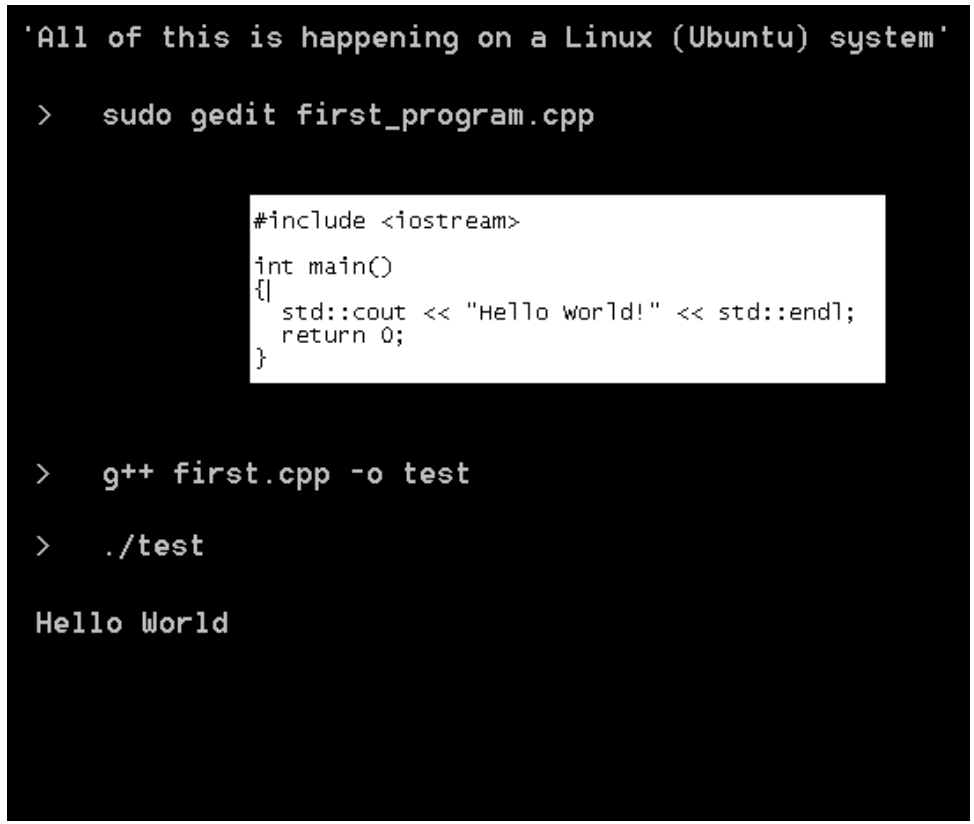
(2.2) Compilers & Interpreters

- Having to write everything out in binary would be both difficult and time consuming (punch-cards). →
- Compilers and Interpreters are programs that take code written in other languages and create the 0's and 1's (the binary) that a machine will actually understand.
- Thank you Admiral Hopper!



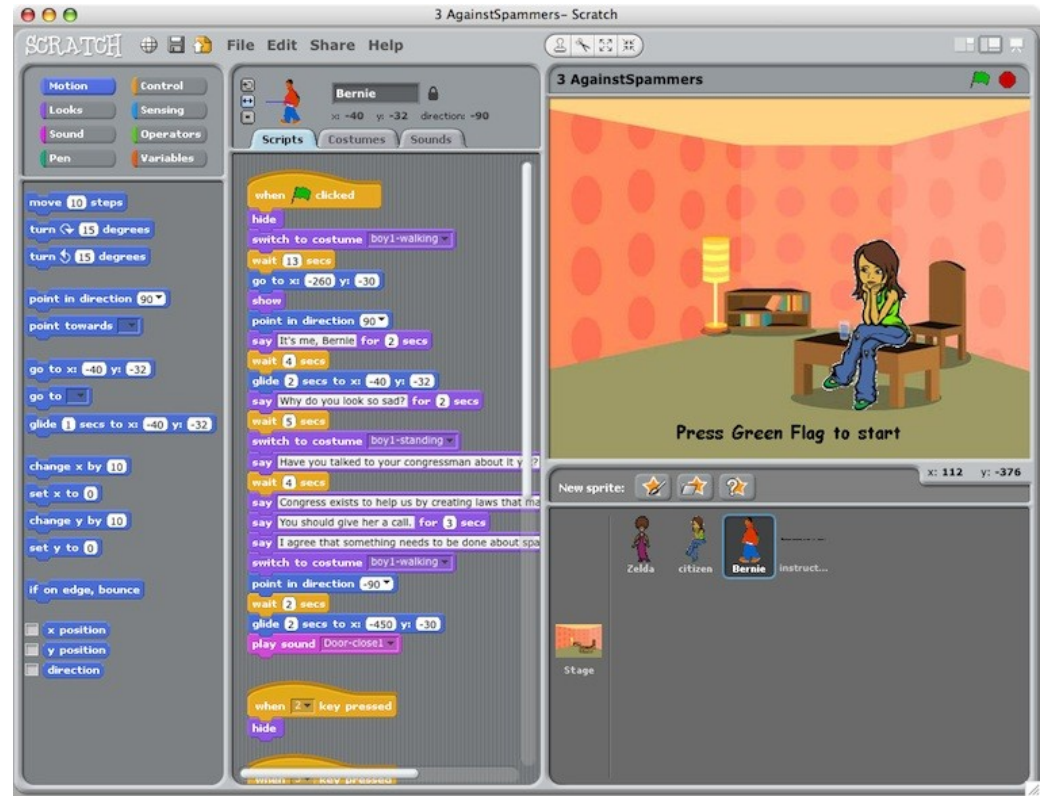
(2.2) High Level Languages

- High level programming languages don't require you to use BINARY (or even know a computer works) in order to write a program.
- High level languages (with compilers and interpreters) allow humans to write using the alpha/numeric characters (abc, 123) that they are already familiar with.
- Examples: C/++/#, Java, Scratch, Robolab, PHP, Python



(2.3) Integrated Development Environment (IDE)

- An IDE is a program that helps you write, run and debug other programs.
- IDE's are very useful, and allow you to do all the separate tasks of creating and testing a program within a single application window.



(2.4) Programming Languages vs. Scripting Languages

- For our purposes a full strength "programming language" (C, C++, Java) is a language that provides some means for the programmer to talk directly to a computers hardware.
  - NOTE: High level programming languages, don't require you to talk to the hardware (but you can if you want too). Low-level programming languages (like Assembly) require you to talk to the hardware.
- A "scripting language" (PHP, Python, JavaScript, Processing, Scratch, Alice) by contrast only allows the programmer to create code that will talk to another existing program or application.
  - NOTE: For us, all scripting languages are high-level languages.

Languages VS. Tasks (1)

- We both speak the same language, therefore, you might think it would be easy for me to get you to complete a certain task or solve a specific problem.
  - Example 1: Give me $50.
  - Example 2: Land a plane.
- In example 1, you are going to want a reason, for 2, you are going to need instructions (if you don't already know how).

Languages VS. Tasks (2)

- Just because you and a computer have a means of communication (a programming language) doesn't mean that it will be easy for you to get the computer to do what you want.
- Computers are dumb!!! Really, really dumb.
- Imaging that you are trying to teach someone to tie their shoes (they don't know how, in fact they don't even know what a shoe is) The only way you can teach them, is by writing a letter (no pictures). Now you can begin to imagine the difficulties that a programmer faces.

(3) Paradigms - 1

- A paradigm is a structured approach to solving a problem.
  - Give me $50 dollars, please.
  - Give me $50 dollars, I will pay you back.
  - Give me $50 dollars, or I will smack you.
- When we are writing a program, we aren't just trying to "communicate" to with the computer, we are trying to get the computer to accomplish a specific task.
  - NOTE: It's no use threatening a computer.
- Programming paradigms are organized approaches to solving the problem of how to get a computer to accomplish a specific task.

(3) Paradigms - 2

- Some spoken/written languages have a very similar syntax and semantics. An example are the "romance languages" of Spanish, Portuguese, French, Italian, Romanian, Catalan. If you know one of these languages already, it is considered much easier to learn one of the others.
- Likewise some programming languages have similar syntax and semantics.
- But more important than the syntax and semantics, when learning a new programming language, is whether or not it belongs to a "programming paradigm" that you already know.

(3) Paradigms -3

- The advantage to learning about programming paradigms is that once you understand a particular paradigm it is easier to learn other programming languages that also use that same paradigm.
- 3-paradigms will be very important to you in this class:
  - Imperative Paradigm
    - A "Smart List"
  - Procedural Paradigm
    - "Making phone calls "
  - Object-Oriented Paradigm
    - A program as set of "interacting objects"

(3) Paradigms -4

- NOTE: There are hundreds of other programming paradigms.
- Some other famous paradigms are:
  - Functional
    - Views program as a mathematical equation
    - Examples: Haskel
  - Logical
    - Views a program as a logic puzzle
    - Examples: Prolog
- But the most popular paradigm is the Imperative one.
  - Views a program as a "smart" list
  - Examples: COBOL, BASIC, Fortran, Ada, C, C++, C#, D, Visual Basic, Java, Python, Perl, PHP, Ruby, Scratch, Robolab, Processing and AS3 to name just a few.

(3.1) Imperative Paradigm

- An Imperative is a command, an obligation, or a requirement.
  - It is a moral imperative that you not cheat.
- In the imperative paradigm, we give the computer a specific set of commands that it must follow.
- The imperative paradigm views a program as a "list" of things the computer must do.
- You might want to remember it as a "smart list".

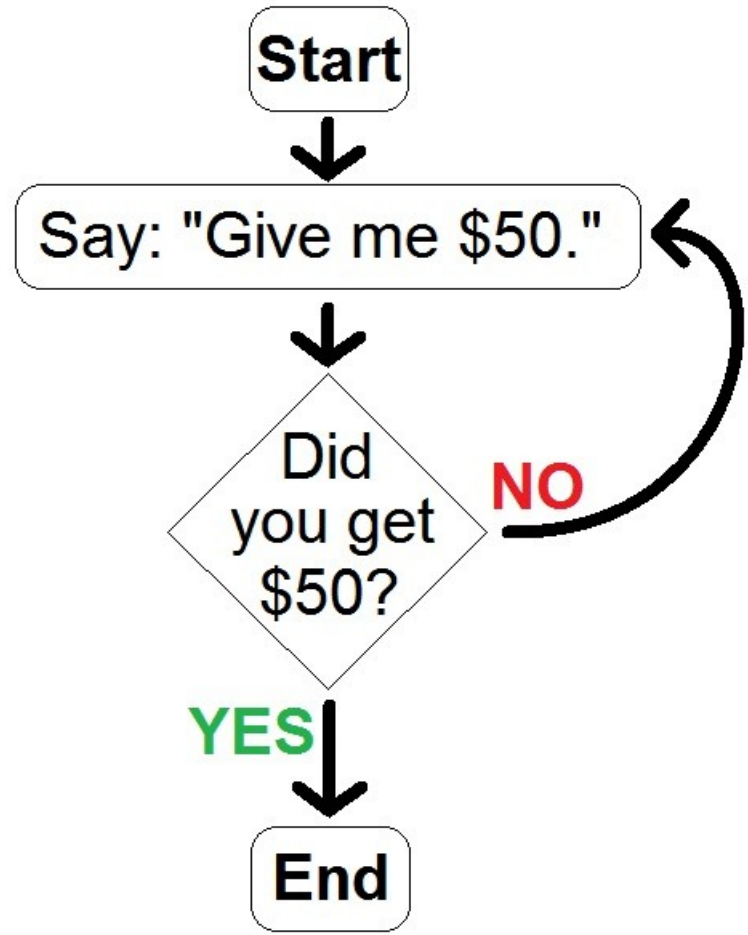
(3.1) "A Smart List"

Imperative languages need 3 things:

1. Sequence -> A predefined order in which to process information.
2. Selection -> The ability to make a choice. The "IF" statement.
3. Repetition -> The ability to repeat an action. The "WHILE" statement.

Pseudo-Code

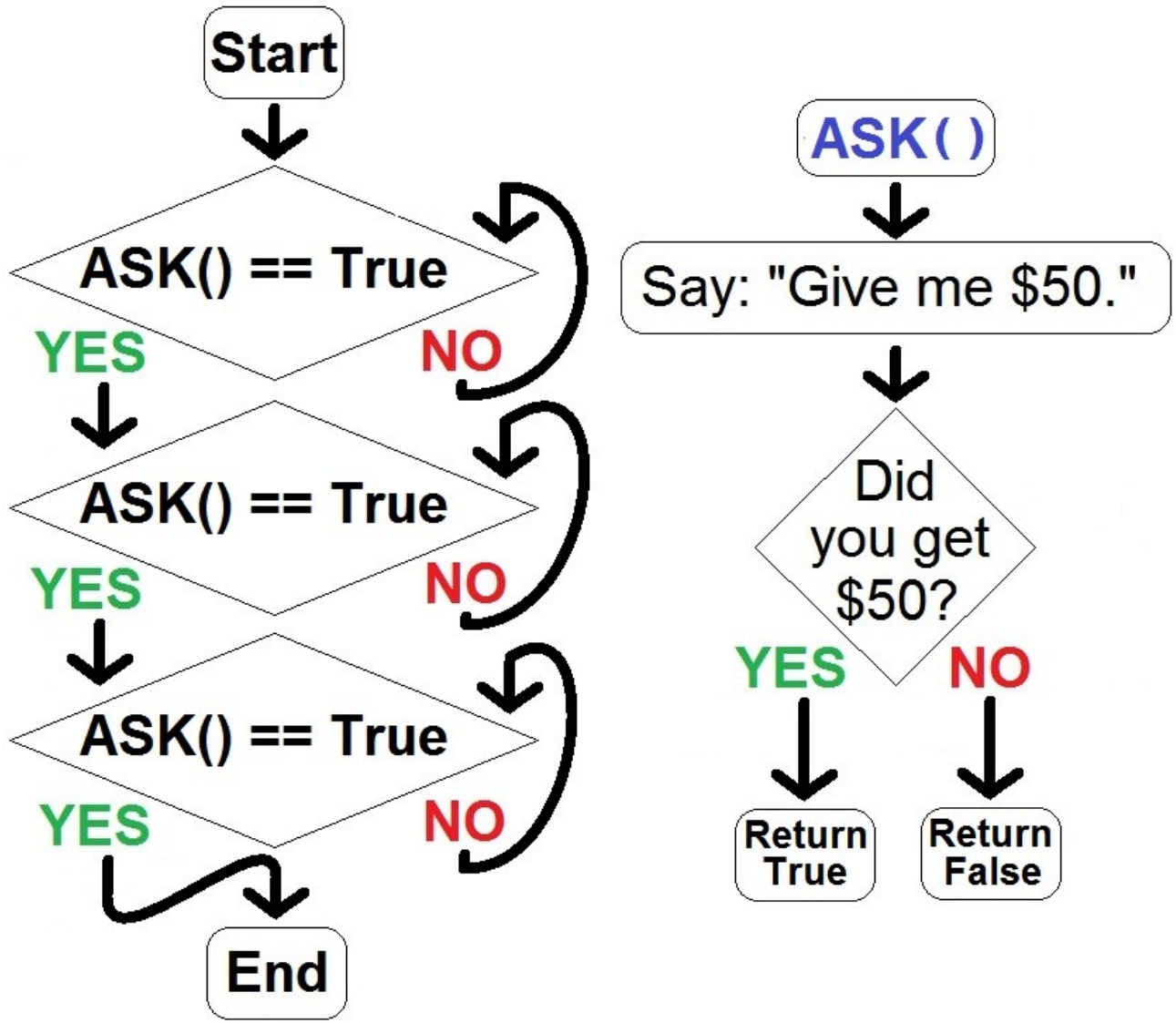
- Pseudo-Code is pretend code that we can use to illustrate a point.
- The Pseudo-Code to the right is an example of Imperative Programming.
- What is the sequence?
- What does the selection and repetition look like?
- Is this an effective program?



(3.2) Procedural Programming

- The procedural programming paradigm is based upon the concept of the "procedure call": the ability to "send a message" to another separate section of a program, and get information back from that section.
- Procedural programming allows us to create sections of code that can be reused over and over. We can give a complex section of code a name and if we need that code in the future, we can simply use the name (instead of retyping the same code, over and over).
- Types of procedural calls:
  - Function Calls (MOST COMMON)
  - Message Passing
  - Event Handlers
  - RPC Calls, REST, etc.

Pseudo-Code


Start
ASK() == True
YES
NO
ASK() == True
YES
NO
ASK() == True
YES
NO
End
ASK( )
Say: "Give me $50."
Did you get $50?
YES
NO
Return True
Return False

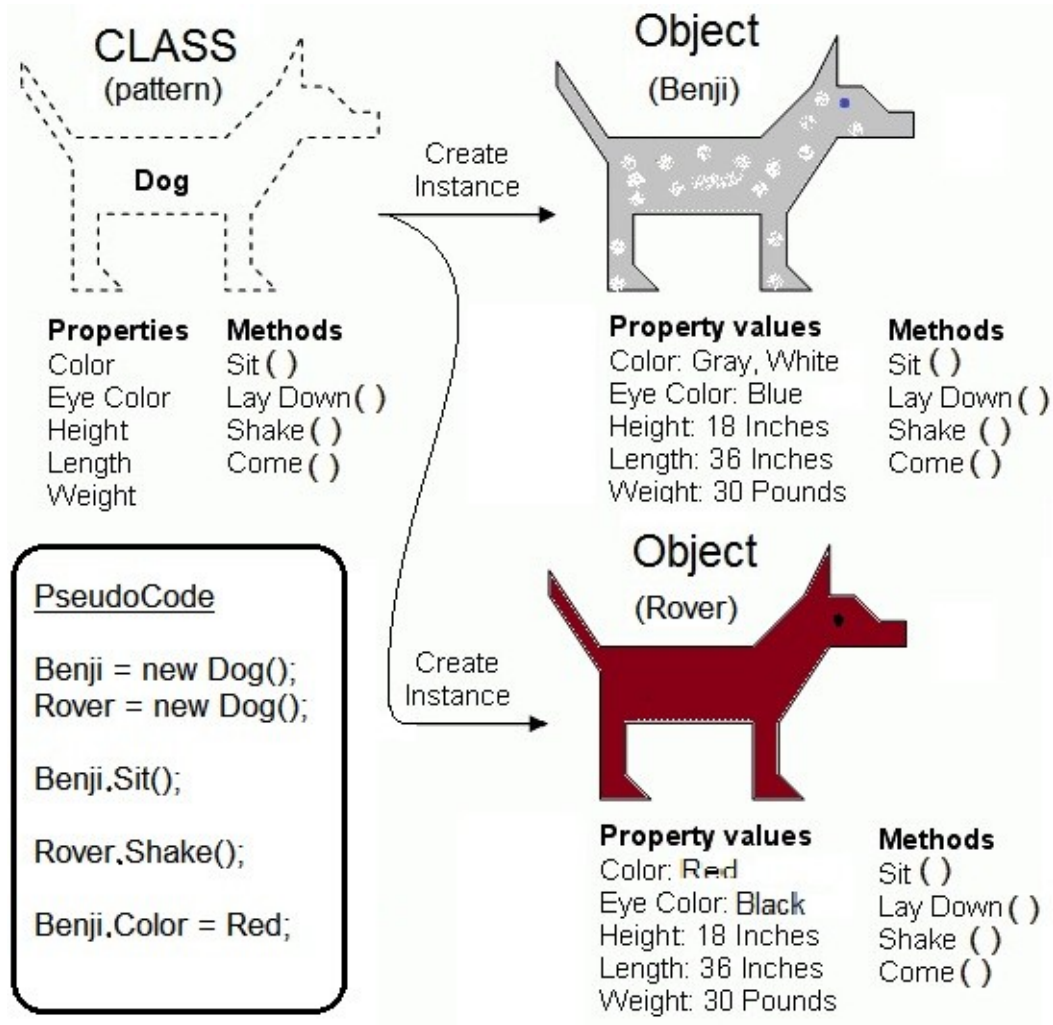
(3.3) Object-Oriented Programming

Basic Concepts:

- Visualize a program as a set of interacting objects (easy to do with games).
- Identify the associated facts and functions of these objects.
  - Properties (Facts): Where is it?
  - Methods (Functions): What can it do?
- Create templates/patterns (classes) for these objects that can be reused.
  - Game-Object -> Enemy ship, and player ship.
  - Bullet Class -> Create LOTS of bullets

OO Programming General Idea (1)

- A class is just a pattern (like a dress pattern).
- A class tells us all about the object, and allows us to create an object based on that pattern.
- We can use a class as many times as we want!!!
- The dot (dot syntax) indicates ownership or belonging.



Review

- What language(s) are you working with in this class? How does that language(s) implement the following:
- Imperative Paradigm ("the smart list")
  - Sequence
  - Selection
  - Repetition
- Procedural Paradigm ("sending messages")
  - What message passing and/or function creating abilities exist.
- Object Oriented Paradigm ("interacting objects")
  - What templates (classes) for objects exist?
  - What are the properties & procedures or those classes?

The End