

BASICS



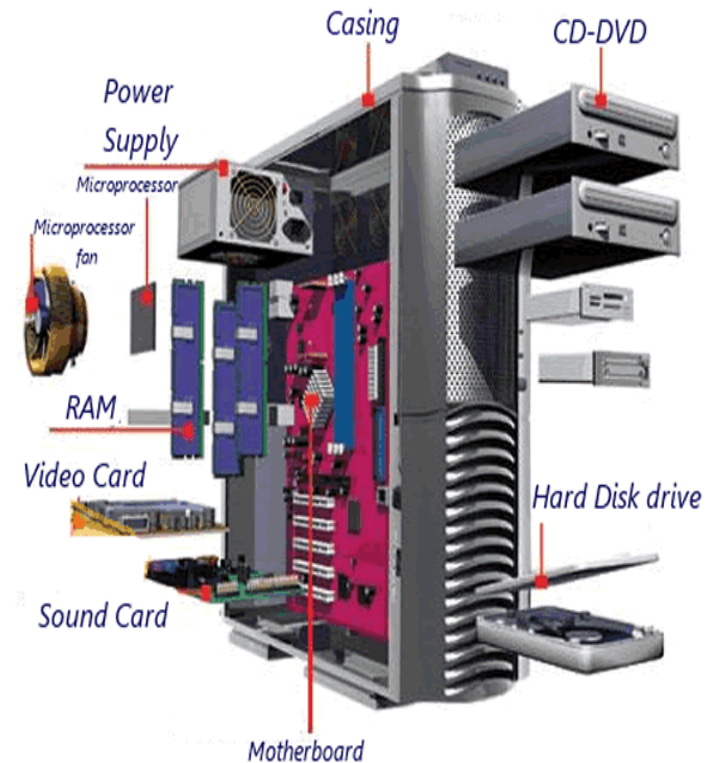
<http://www.flickr.com/photos/oskay/472097903/>

Outline

- Computer Basics
- Programs and Languages
- Introduction to the Idle Shell / Editor
- Our First Program
 - Comments
- Algorithms

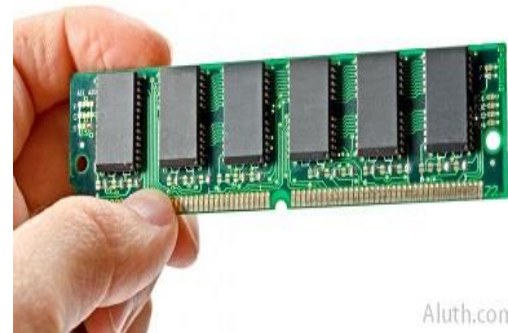
Hardware and Memory

- Most modern computers have similar components including
 - Input devices (keyboard, mouse, touchscreen, etc.)
 - Output devices (display screen, printer, etc.)
 - A processor
 - Two kinds of memory (main memory and auxiliary memory).



Main memory

- Working memory used to store
 - The current program
 - The data the program is using
 - The results of intermediate calculations
- Usually measured in megabytes or gigabytes (e.g. 8GB RAM)
 - RAM is short for *random access memory*
 - A *byte* is a quantity of memory



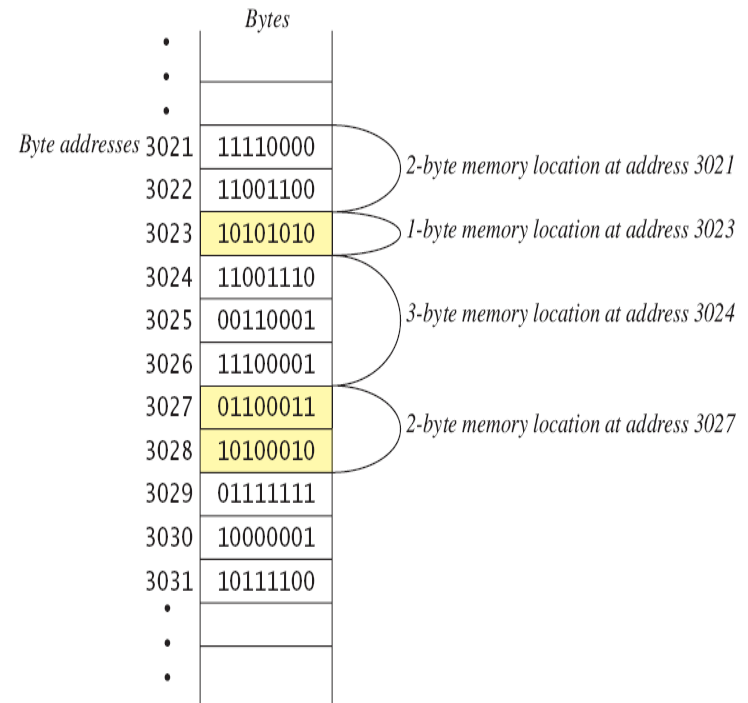
Bits, Bytes, and Addresses

- A *bit* is a digit with a value of either 0 or 1.
- A *byte* consists of 8 bits.
- Each byte in main memory resides at a numbered location called its *address*.



Main Memory

- Data of all kinds (numbers, letters, strings of characters, audio, video, even programs) are encoded and stored using 1s and 0s.
- When more than a single byte is needed, several adjacent bytes are used.
 - The address of the first byte is the address of the unit of bytes.
- When the computer is turned off, main memory is erased (**volatile memory**).



Auxiliary Memory

- Auxiliary memory uses devices such as a hard drive, DVD, USB drive, etc.
- Data (files) need to be “saved” to the auxiliary memory
- Data is still stored in bits and bytes
- When the computer is turned off, this data does not go away (**persistent storage**)

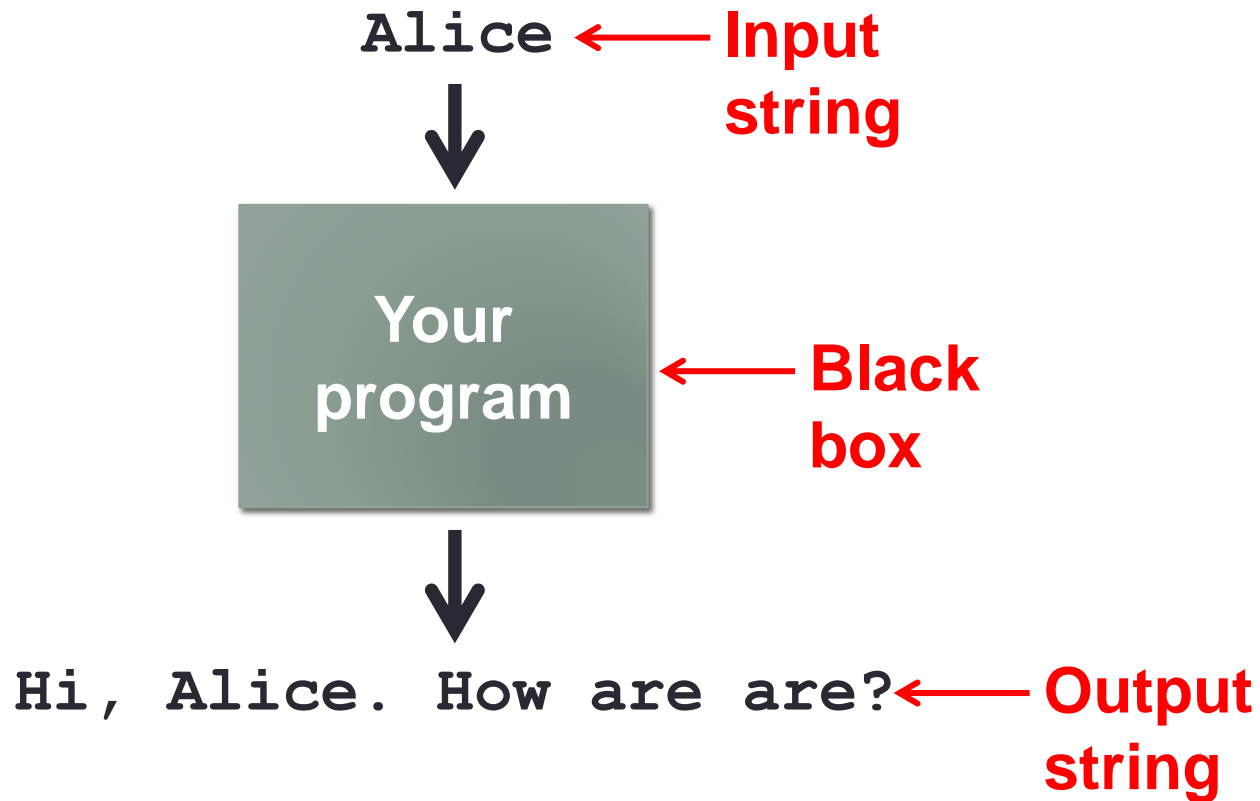


Programs

- A *program* is a set of instructions for a computer to follow.
- We use programs almost daily (email, word processors, video games, bank ATMs, etc.).
- When the computer follows the instructions it is *running* or *executing* the program.



View of Programming from 10,000 Feet

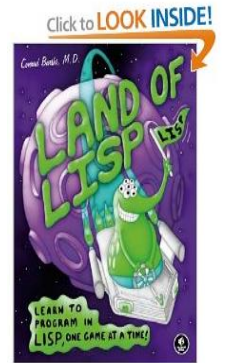
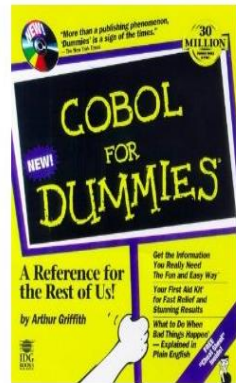
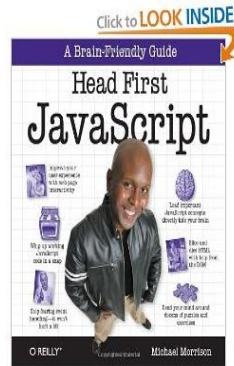
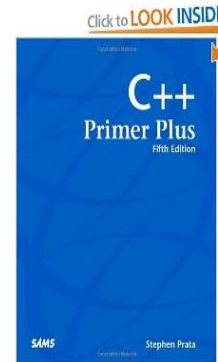
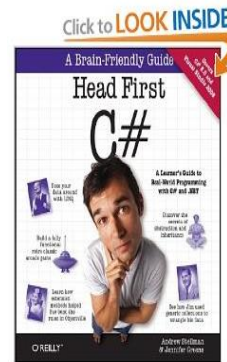
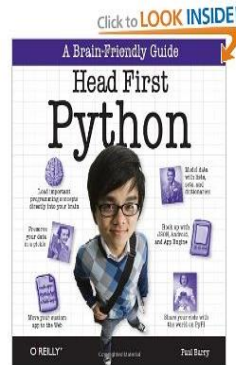
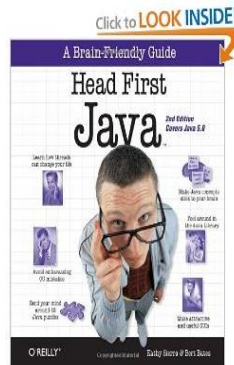


Languages

- **Machine language**
 - Low level, what the hardware understands
 - Tedious and error-prone to write
 - Specific to a particular type of computer
- **Natural language**
 - Imprecise and ambiguous
 - Hard to convert to instructions for the hardware
- **High level programming language**
 - Good balance between the two extremes

Becoming a Programmer: Step 1

Choose a language...



and hundreds more...

Our Choice: Python



- **Advantages**
 - Widely used, modern
 - Freely available, cross-platform
 - Simpler to learn than other languages
- **No perfect single language**
 - You'll learn many other languages
 - C/C++, assembly, Java, C#, JavaScript, PHP...
 - Programming skills translate easily between them



"There are only two kinds of languages: the ones people complain about and the ones nobody uses."

*-Bjarne Stroustrup, father of
C++*

Your First Program



http://www.zazzle.com/baby_girls_first_java_program_hello_world_tshirt-235063563751392326 \$23.95

How Python Works

Source code:

Plain text file created
in some editor
(notepad, vi, TextEdit,
Idle editor, ...) or typed
into the Python shell

```
import dis
def example(x):
    for i in range(x):
        print(2 * i)
```

Example.py



“compiling” `% python Example`

Python bytecode:

Intermediate language
that any device
running Python can
understand (humans
generally ignore this)

```
>>> dis.dis(example)
2      0 SETUP_LOOP                28 (to 30)
      2 LOAD_GLOBAL                 0 (range)
      4 LOAD_FAST                   0 (x)
      6 CALL_FUNCTION               1
      8 GET_ITER
>>   10 FOR_ITER                    16 (to 28)
      12 STORE_FAST                 1 (i)

3      14 LOAD_GLOBAL                 1 (print)
      16 LOAD_CONST                 1 (2)
      18 LOAD_FAST                   1 (i)
      20 BINARY_MULTIPLY
      22 CALL_FUNCTION              1
      24 POP_TOP
      26 JUMP_ABSOLUTE              10
>>   28 POP_BLOCK
>>   30 LOAD_CONST                 0 (None)
      32 RETURN_VALUE
```

“Disassembled” bytecode

How Python Works

Python

bytecode:

Intermediate language
that any device
running Python can
understand (humans
generally ignore this)

“running”

```
>>> dis.dis(example)
2      0 SETUP_LOOP          28 (to 30)
      2 LOAD_GLOBAL           0 (range)
      4 LOAD_FAST             0 (x)
      6 CALL_FUNCTION        1
      8 GET_ITER
  >>  10 FOR_ITER             16 (to 28)
      12 STORE_FAST          1 (i)

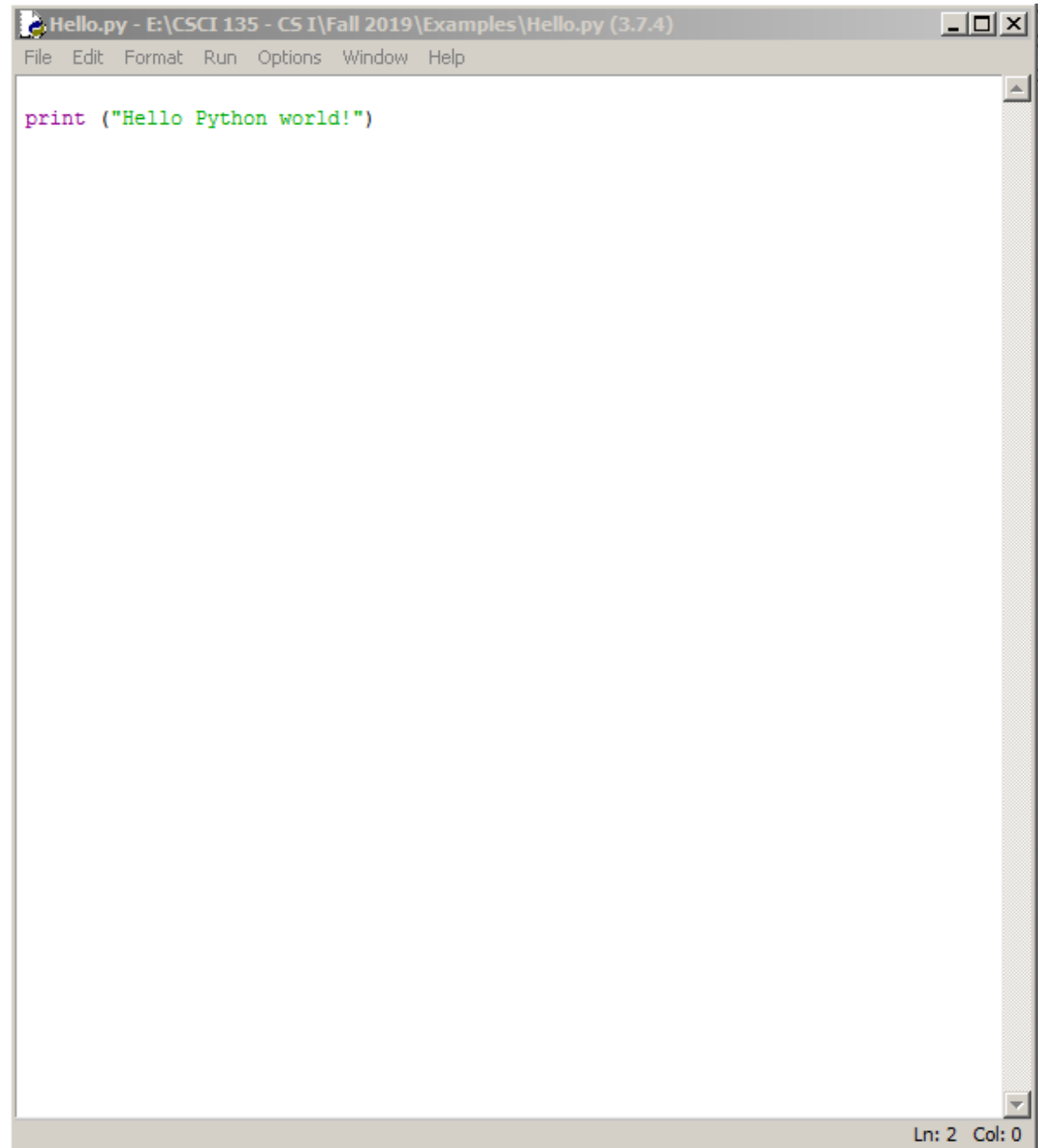
3      14 LOAD_GLOBAL           1 (print)
      16 LOAD_CONST           1 (2)
      18 LOAD_FAST            1 (i)
      20 BINARY_MULTIPLY
      22 CALL_FUNCTION        1
      24 POP_TOP
      26 JUMP_ABSOLUTE       10
  >>  28 POP_BLOCK
  >>  30 LOAD_CONST           0 (None)
      32 RETURN_VALUE
```

“Disassembled” bytecode



```
Administrator: Command Prompt
E:\CSCI 135 - CS I\Fall 2019\Examples>python Hello.py
Hello Python world!
E:\CSCI 135 - CS I\Fall 2019\Examples>
```

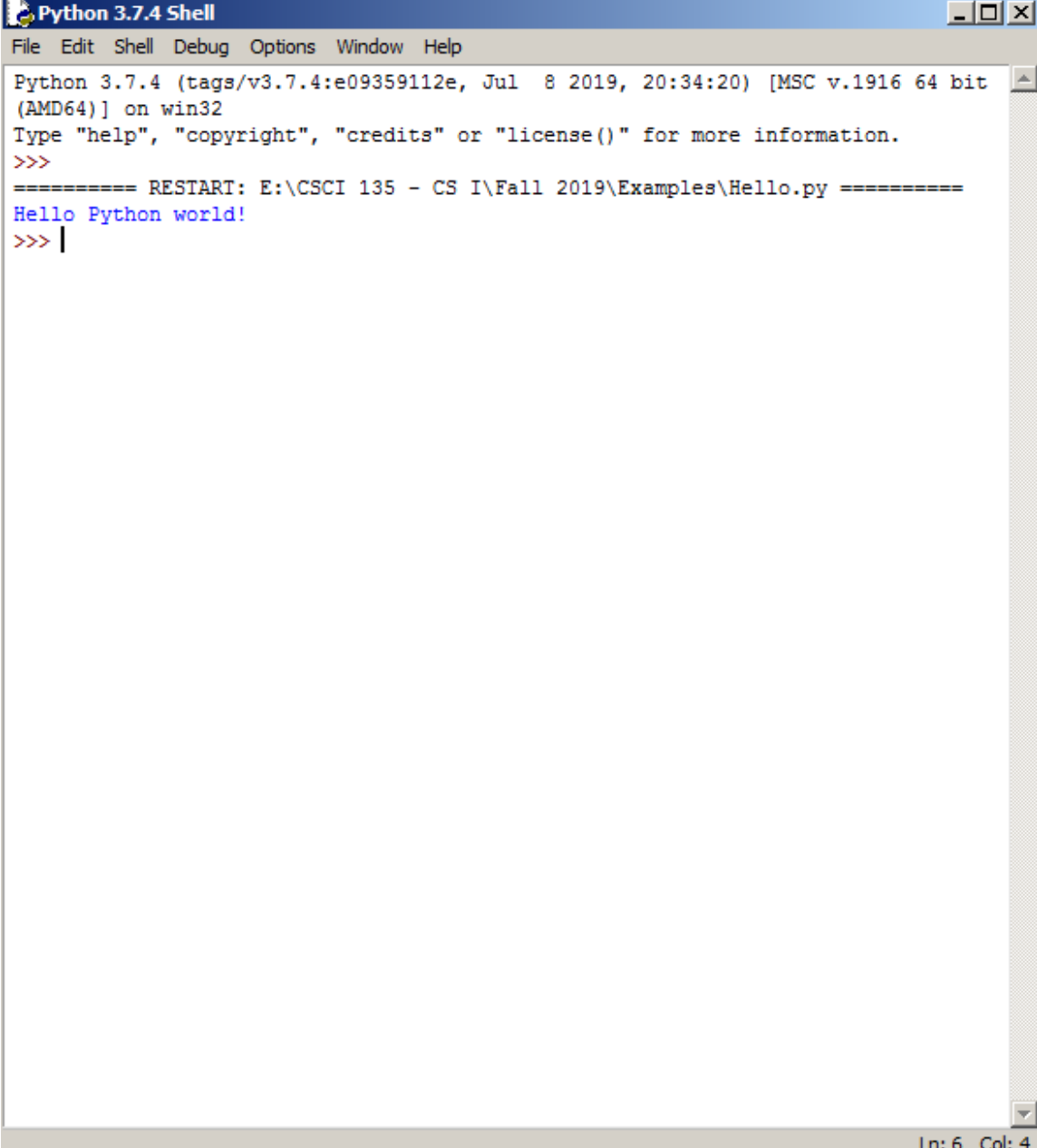
Idle – Python Editor



The image shows a screenshot of the Python Idle editor. The window title is "Hello.py - E:\CSCI 135 - CS I\Fall 2019\Examples\Hello.py (3.7.4)". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains a single line of Python code: `print ("Hello Python world!")`. The status bar at the bottom right indicates "Ln: 2 Col: 0".

```
print ("Hello Python world!")
```


Idle – Python Shell

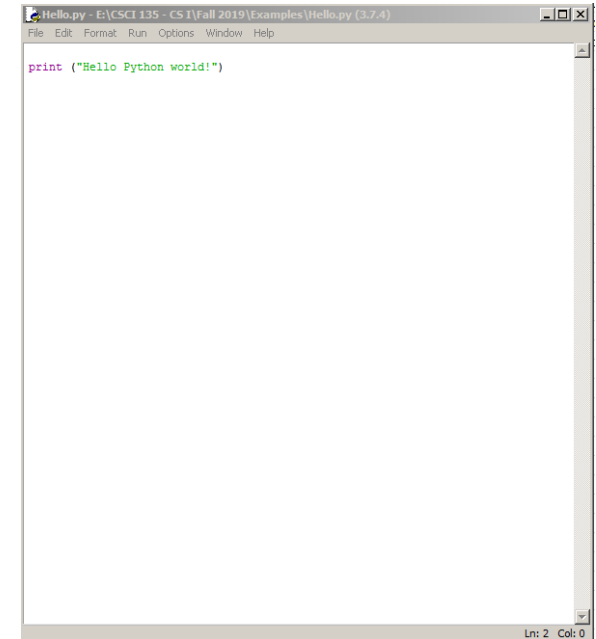


```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:\CSCI 135 - CS I\Fall 2019\Examples\Hello.py =====
Hello Python world!
>>> |
```

Ln: 6 Col: 4

Idle Python Editor

- **Recommended** but not required
- **Free**
- Helpful features:
 - **Syntax highlighting**
 - Run code from editor
- We will use mostly as a **text editor**
 - Ignoring many of its features
- How to install?
 - See course web site, **resources page**
- Can use any text editor / Python editor that you like, though



Anatomy of a Python Program

```
# This is a comment
# Python ignores anything after the # sign
# This is how you should put your name and a description of your
#     code at the beginning of a program

# Name: Michele Van Dyne
# Description: Takes a name as input, prints a greeting to that name,
#     if it is the name of the prof, prints goodbye prof, otherwise just
#     prints goodbye
name = input("Please enter your name: ")
print(f"\nHello, {name}!")
if name == "Michele":
    print("Goodbye, professor")
else:
    print("Goodbye")
```

Some Terminology

OFFICIAL

DEFINITION

- ***Statement*** – an instruction to the computer
- ***Syntax*** – the grammar rules for a programming language
- ***Flow of Control*** – the order in which instructions are executed

Algorithms

- By designing algorithms, programmers provide actions for the computer to perform.
- An *algorithm* describes a means of performing an action.
- Once an algorithm is defined, expressing it in Python (or in another programming language) usually is easy.

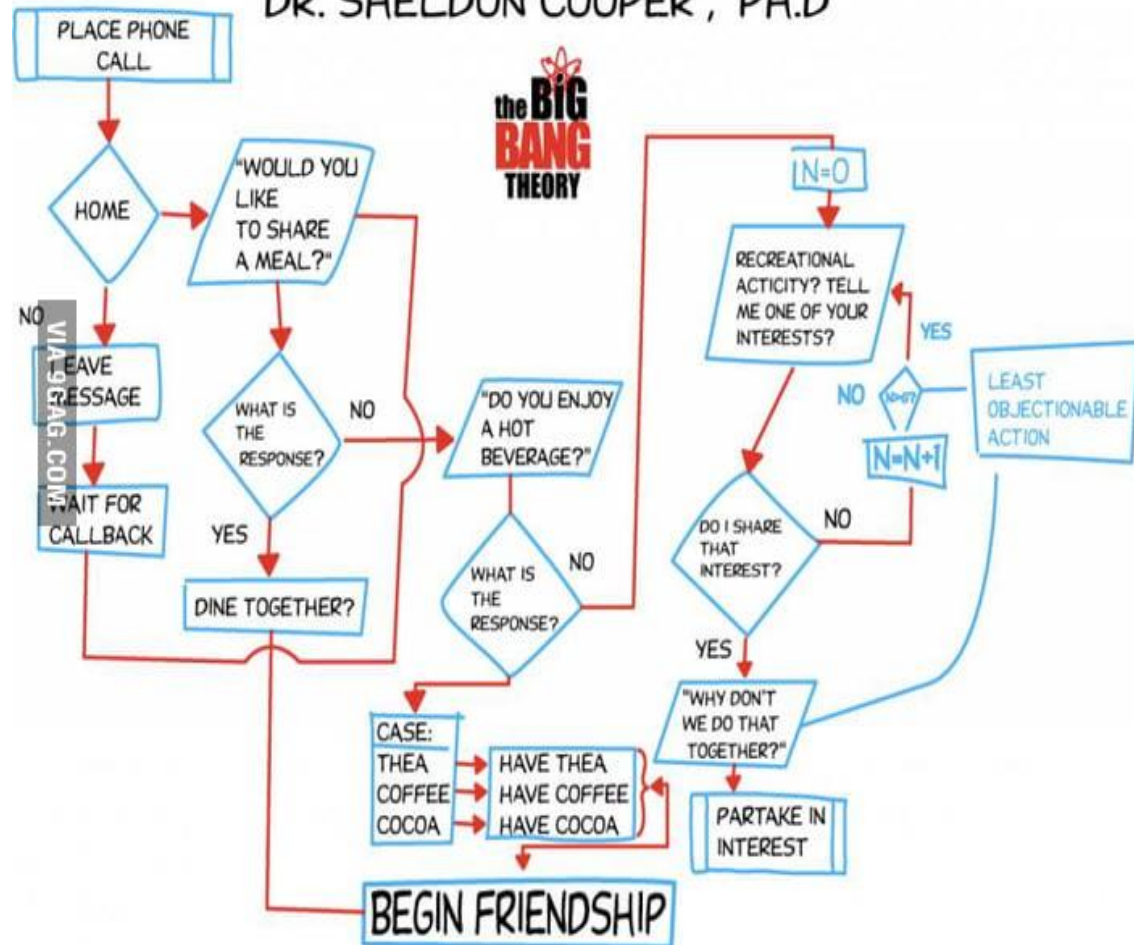


Algorithms

- An algorithm is a set of instructions for solving a problem.
- An algorithm must be expressed completely and precisely.
- Algorithms usually are expressed in English or in *pseudocode*.

THE FRIENDSHIP ALGORITHM

DR. SHELDON COOPER, PH.D



Example: Total Cost of All Items

- Write the number 0 on the whiteboard.
- For each item on the list
 - Add the cost of the item to the number on the whiteboard
 - Replace the number on the whiteboard with the result of this addition.
- Announce that the answer is the number written on the whiteboard.



Summary

- Computer Basics
- Programs and Languages
- Introduction to the Idle Shell / Editor
- Our First Program
 - Comments
- Algorithms

