

CMPT 120

Intro to CS & Programming I

WEEK 2 (Jan. 13-17)

JÉRÉMIE O. LUMBROSO

Lecture 3: Programming Languages

<http://www.sfu.ca/~jlumbros/Courses/CMPT120/>

Recall Important Distinction

- **Algorithms:** how to think about/deconstruct problems in terms appropriate for computers
- **Programming language:** to express *algorithms* in a way understandable by computers
- **Pseudocode:** a “programming language for people”, a intermediate way to express things like a computer would understand them, but in a less formal way more appropriate for people

From Turing to Python

SMALL HISTORY OF PROGRAMMING



IMPORTANT

This part of the lecture is to give you some insight on the history of computers languages. It is to put the choice of Python in perspective.

None of this needs be known by heart!

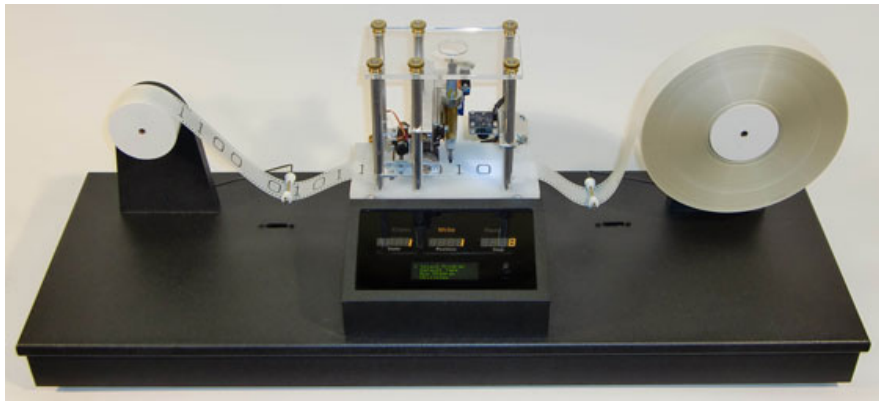


Turing Machines (1936)

Concept by Alan Turing, gives definition of **what can be computed**



... an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings. (Turing 1948, p. 61)



- Machine is a theoretical model
- It has a **tape** for input/output
- It has a **table of rules**, determines action to take
- It has a **current state** (immediate memory)
- **Finding the first 1:** “In state 768, if I read a ‘0’ on the tape, I move the head left and remain in state 768; if I read a ‘1’, I do not move the head but set the state to 456.”

All programs ever written for today’s computers can be translated to work on a Turing Machine.

Turing-Completeness

- All modern programming languages are said to be **Turing complete**, which means that they can do no more and no less than a Turing machine
- **Consequence:** no programming language is **more “powerful”** than another
- **What changes?**
 - The **efficiency**: how fast it runs & how much battery it uses
 - The **expressivity**
 - how easy it is to translate your ideas into code?
 - how little you have to write to do what you want?
 - does the language avoid making repetitions?

Types of Programming Languages

Low-level

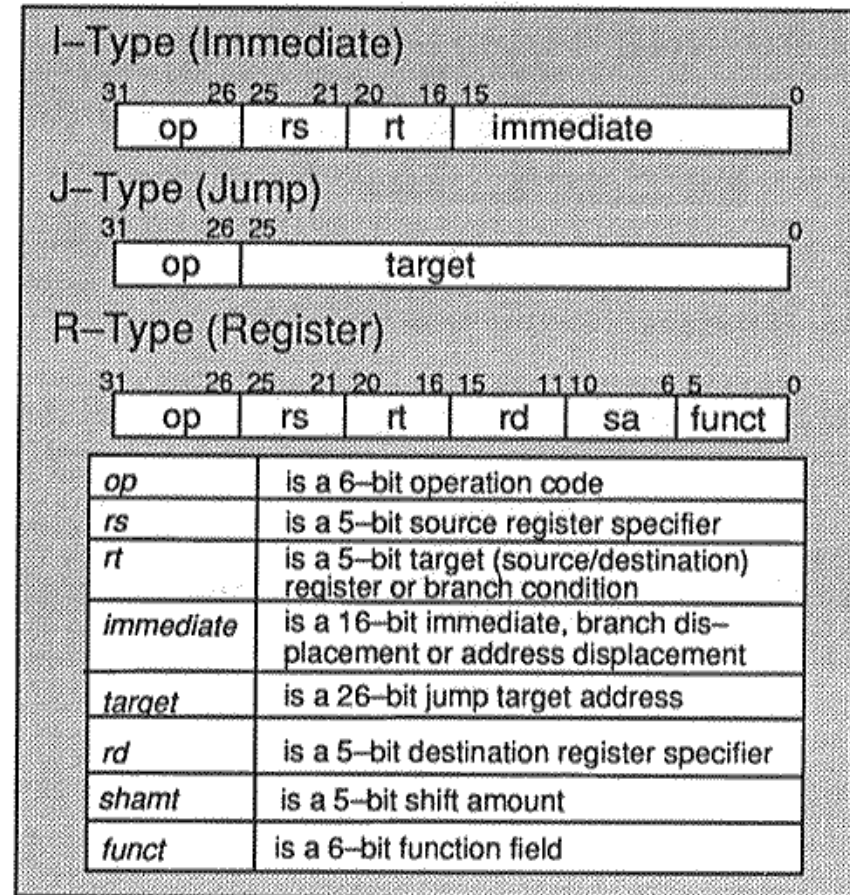
- **Machine (or Assembly) Languages:** directly executed by machine; one per different type of CPU (computer processing unit)

High-level

- **Compiled Languages:** translated (“compiled”) to assembly languages
- **Interpreted Languages:** no compilation; instead of talking directly to machine, runs on top of an interpreter

Assembly (or Machine) Language

- Instructions directly written for the CPU
- Everything must be taken care when writing program
- Uses **registers** (immediate memory = state) and **stack** (written memory = tape)
- Code written by programmer directly converted to binary



add \$6, \$7, \$8 becomes (in MIPS)

0b000000 00111 01000 00110 000000 100000 **or** **0x**00E83020

Why not use Machine-Language?

Advantages

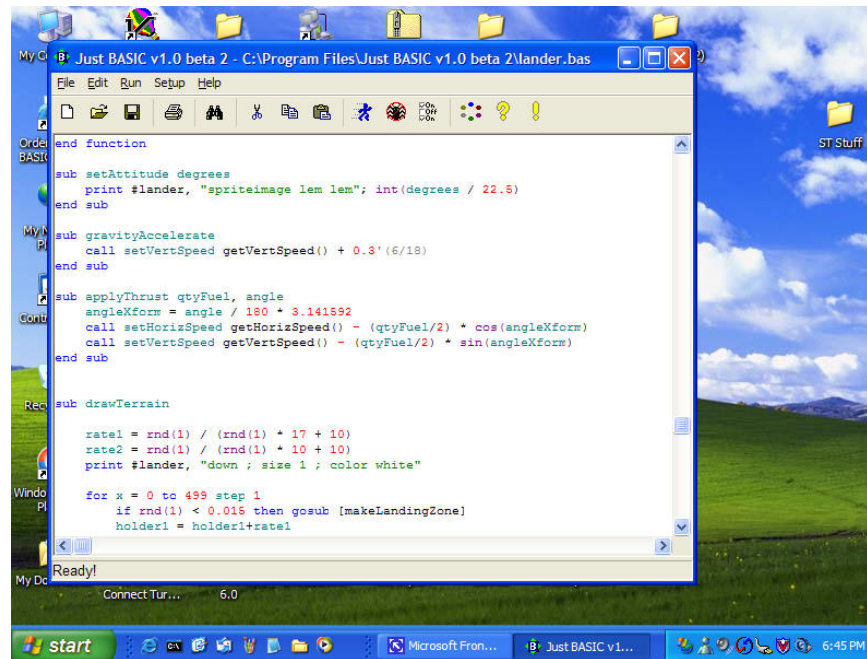
- **VERY FAST** (no way to go faster)

Disadvantages

- Code depends on the CPU
 - One given type of CPU changes frequently
 - Many types of CPU exist (ARM, Intel, etc.)
- Incomprehensible
- Hard to fix when problem (“**debug**”), and to update
- Not very expressive: every simple action requires LOTS of code

BASIC (1965)

- Language for beginners (*Beginner's All-purpose Symbolic Instruction Code*)
- Has been in existence through many different iterations (BASIC, QuickBasic, Virtual Basic, VBA, Dark Basic, Just BASIC, etc.)
- Used very much in Finance



```
end function
sub setAttitude degrees
  print #lander, "spriteimage lem lem"; int(degrees / 22.5)
end sub

sub gravityAccelerate
  call setVertSpeed getVertSpeed() + 0.3 * (6/18)
end sub

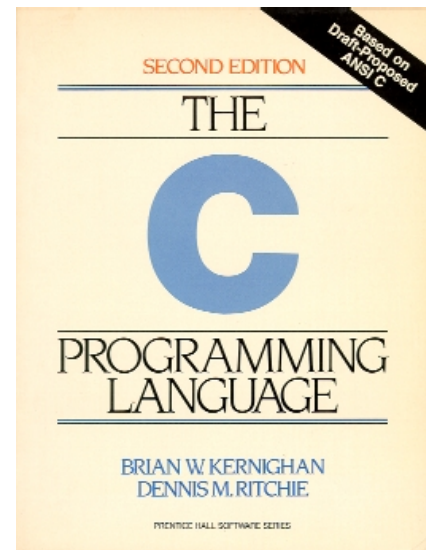
sub applyThrust qtyFuel, angle
  angleXform = angle / 180 + 3.141592
  call setHorizSpeed getHorizSpeed() - (qtyFuel/2) * cos(angleXform)
  call setVertSpeed getVertSpeed() - (qtyFuel/2) * sin(angleXform)
end sub

sub drawTerrain
  rate1 = rnd(1) / (rnd(1) * 17 + 10)
  rate2 = rnd(1) / (rnd(1) * 10 + 10)
  print #lander, "down ; size 1 ; color white"

  for x = 0 to 499 step 1
    if rnd(1) < 0.015 then gosub [makeLandingZone]
    holder1 = holder1 + rate1
  
```

C (1972)

- Created by Kernighan and Ritchie at AT&T Bell Labs for use developing the Unix operating system
- **Very** efficient: translates very directly to machine language
- Most reliably used programming language



```
#include <stdio>

// Recursive version
int recfib(int n) {
    if (n < 2)
        return n;
    else
        return recfib(n-1) + recfib(n-2);
}

void main() {
    printf("%d\n", fib(10));
}
```

```
// Iterative version
int iterfib(int n) {
    int first = 0, second = 1;

    int tmp;
    while (n-- > 0) {
        tmp = first+second;
        first = second;
        second = tmp;
    }
    return first;
}
```

Machine Language vs. High-Level

Which one do you understand more?

```
char tab[] = "hello\0" ;
int i = 0 ;

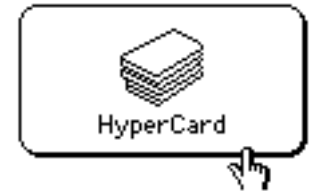
void main() {
    while (tab[i] !=0) {
        tab[i] = tab[i] - 0x20 ;
        i++ ;
    }
    printf("%s\n", &tab);
    exit(0);
}
```



```
.data
tab: .asciiz "hello\0"
.text
start:
    lui $4, tab >> 16
    ori $4, $4, tab & 0xFFFF
    xor $5, $5, $5
    xor $6, $6, $6
label_1 :
    add $7, $4, $5
    lb $8, ($7)
    beq $8, $0, label_2
    addi $6, $8, -0x20
    sb $6, ($7)
    addi $5, $5, 1
    j label_1
label_2 :
    ori $4, $6, 0
    ori $2, $0, 1
    syscall
    ori $2, $0, 10
    syscall
```



HyperCard (1987)

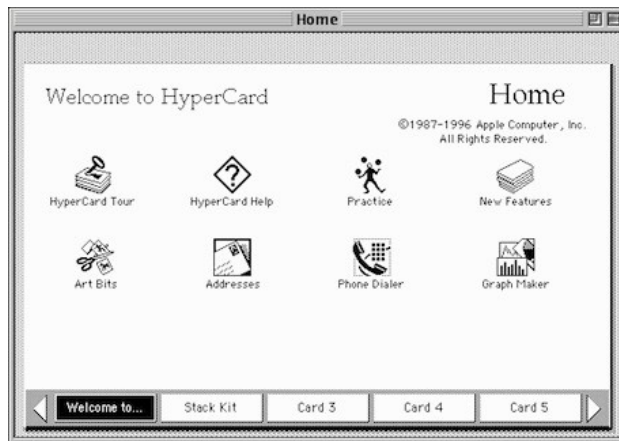


- Created by Apple for Macintosh
- Introduced concept of “HyperLink” and is ancestor/inspiration for web
- Syntax of the language is “user friendly” (it sounds like English)
- Was used to create *Myst*, one of the most popular games that popularized the CD drive for PCs



```
on mouseUp
  ask "What is your name?"
  put first word of it into firstname
  answer "Hello Mr. " & lastname

  answer "Do you have prior programming training?" with "yes" or "no"
  if it contains "no" then
    push card
    go to card "introtoprogramming"
  else
    push card
    go to card "morecomplicatedstuff"
  end if
end mouseUp
```



Java (1995)



- Object-oriented: programs are organized in **classes** which can be copied and reused
- One of the most used interpreted languages
- Interpreted: you can write a program once, and run it on any computer with the JVM (Java Virtual Machine)

```
public class fibo
{
    public static void main(String args[])
    {
        int N = Integer.parseInt(args[0]);
        System.out.println(fib(N));
    }
    public static int recfib(int n)
    {
        if (n < 2) return(1);
        return( recfib(n-2) + recfib(n-1) );
    }
}
```

JavaScript (1995)

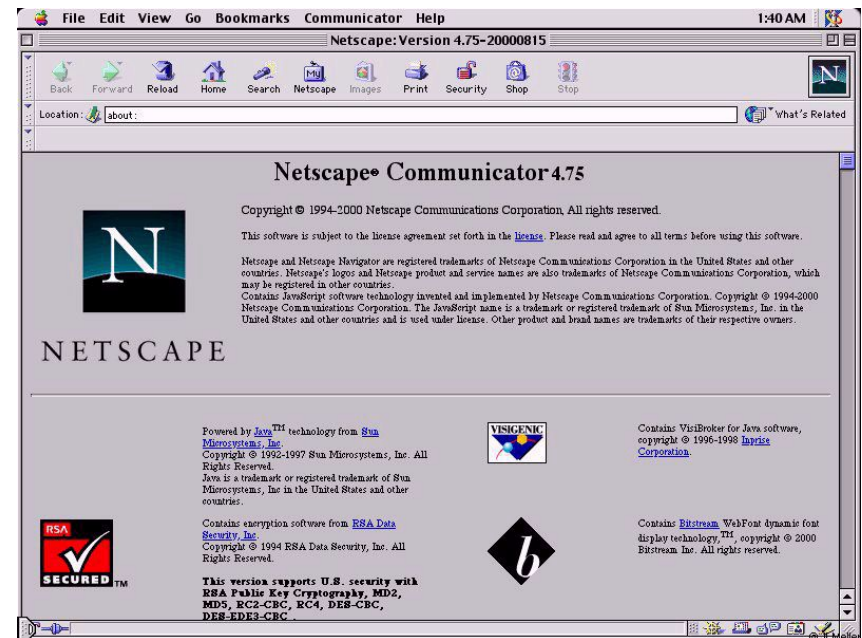


- Not to be confused with Java
- Interpreted language created for Netscape (old browser)
- Runs in webpages
- Looks like Java or C (with braces and semicolons)
- Most powerful feature is to manipulate the DOM (Document Object Model) which is the layout of web pages

```
function getLink() {
    temp = muresources[choice];
    temp2 = "<TITLE>Custom Links</TITLE><H3>" +
        document.m.pername.value +
        ", here are some links for you</H3><P>" +
        temp;
}

function writeIt() {
    diswin = window.open();
    diswin.document.open();
    diswin.document.write(temp2);
    diswin.document.close();
}

function doAll() {
    getLink();
    writeIt();
}
```

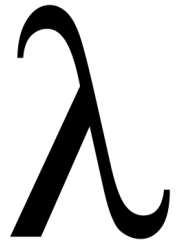


LISP (1958) and Scheme (1975)

A “functional” language: more mathematical

– variables cannot be changed (like in math)

– functions can be manipulated like numbers, text



- Well-known for being “mess of parentheses”
- Used until recently by MIT for CS introduction
- For a long time, was central to Artificial Intelligence research

;; Building a list of squares from 0 to 9:

```
(define (list-of-squares n)
  (let myloop ((i n) (res '()))
    (if (< i 0)
        res
        (myloop (- i 1)
                 (cons (* i i) res))))))
```

```
(list-of-squares 9)
; ==> (0 1 4 9 16 25 36 49 64 81)
```

python™ (1991)

- Invented by Guido van Rossum
- Became popular in mid-2000s when Google started using it
- Interpreted but reasonably efficient
- **Indentation matters!!!** (improves readability)

Recursive version

```
def recfib(n):  
    if n < 2:  
        return n  
    else:  
        return recfib(n - 1) + recfib(n - 2)  
  
print "10th Fibonacci number is %d" % recfib(10)
```



Machine Language vs. Interpreted

The same program in machine language, and in Python.
Both produce an error: **which is easiest to figure out?**

Machine language program

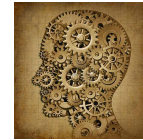
```
Segmentation Fault
```

Python program

```
Traceback (most recent call last):  
  File "/Users/admin/Documents/myprogram.py", line 8, in <module>  
    provokeDivideByZero()  
  File "/Users/admin/Documents/myprogram.py", line 5, in provokeDivideByZero  
    print invert(0)  
  File "/Users/admin/Documents/myprogram.py", line 2, in invert  
    return 1/x  
ZeroDivisionError: integer division or modulo by zero
```

Pacing and Understanding

Your impression of this brief history of programming languages?



A

I knew most of this stuff, *yawn*

B

Wow, this is interesting, and I followed most of it

C

Not sure I followed, but I got a good sense of general ideas

D

I don't think this was useful for me

E

It has dawned on me that you can see who has answered what, and I should probably answer stuff that is flattering to you; so to summarize: **“YOU ARE BRILLIANT!!! THIS WAS GREAT!!! NOW I KNOW ALL!!!”**

First steps in Python

PYTHON?!

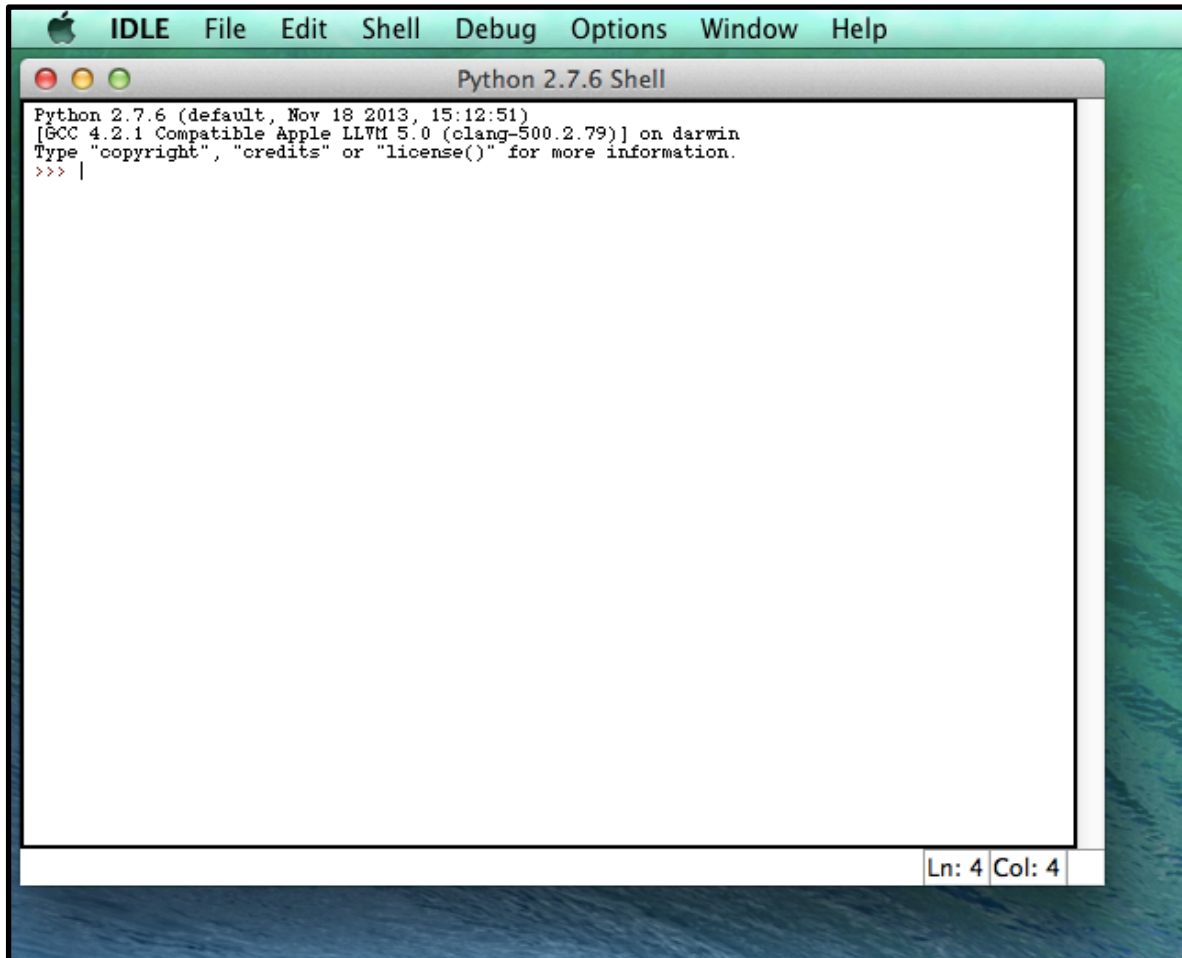
Starting with Python

- Discover the IDLE environment
- How to create and run a Python program
- What is the Python shell




This Python won't eat you, promise!

IDLE: a Python Environment



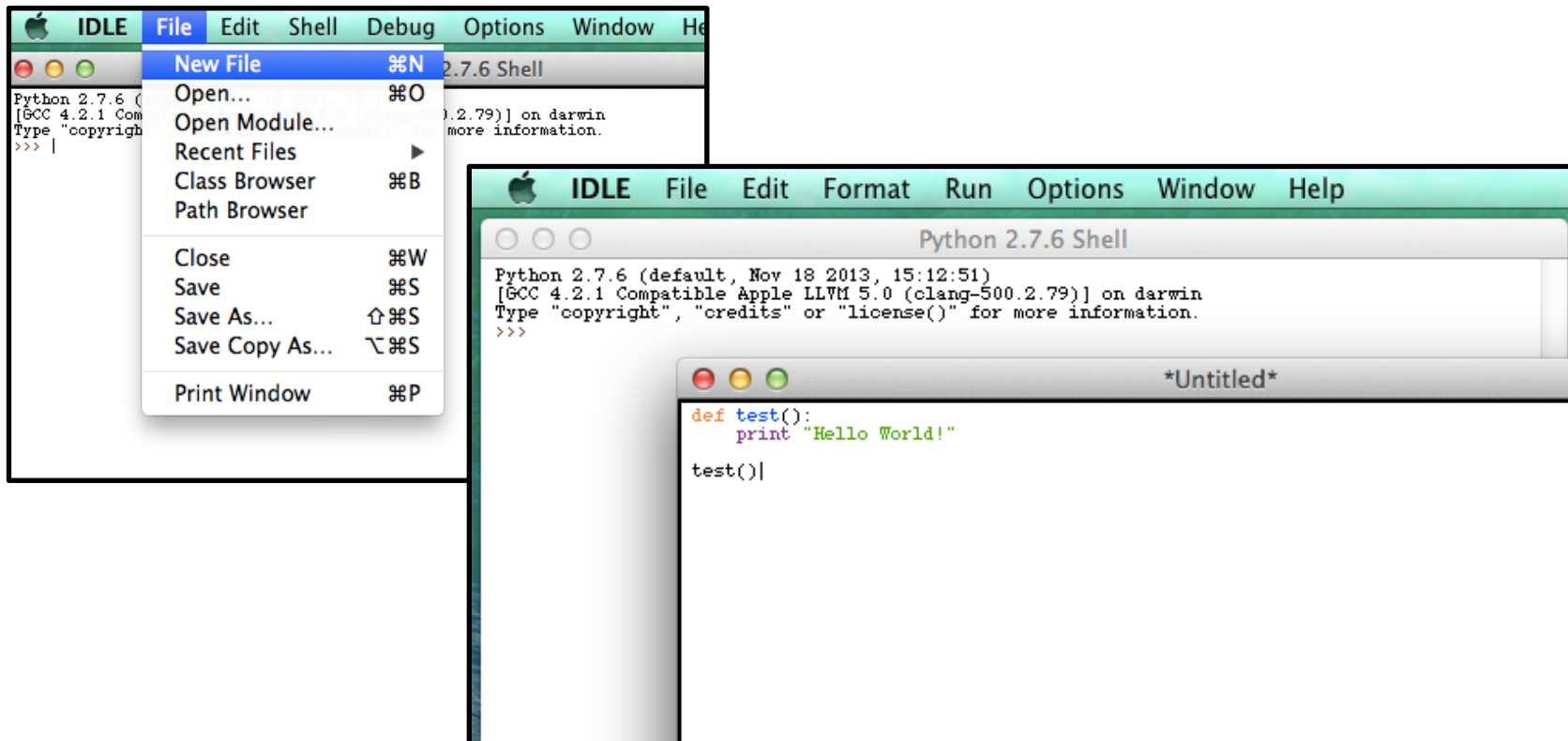
```
Python 2.7.6 (default, Nov 18 2013, 15:12:51)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Ln: 4 Col: 4

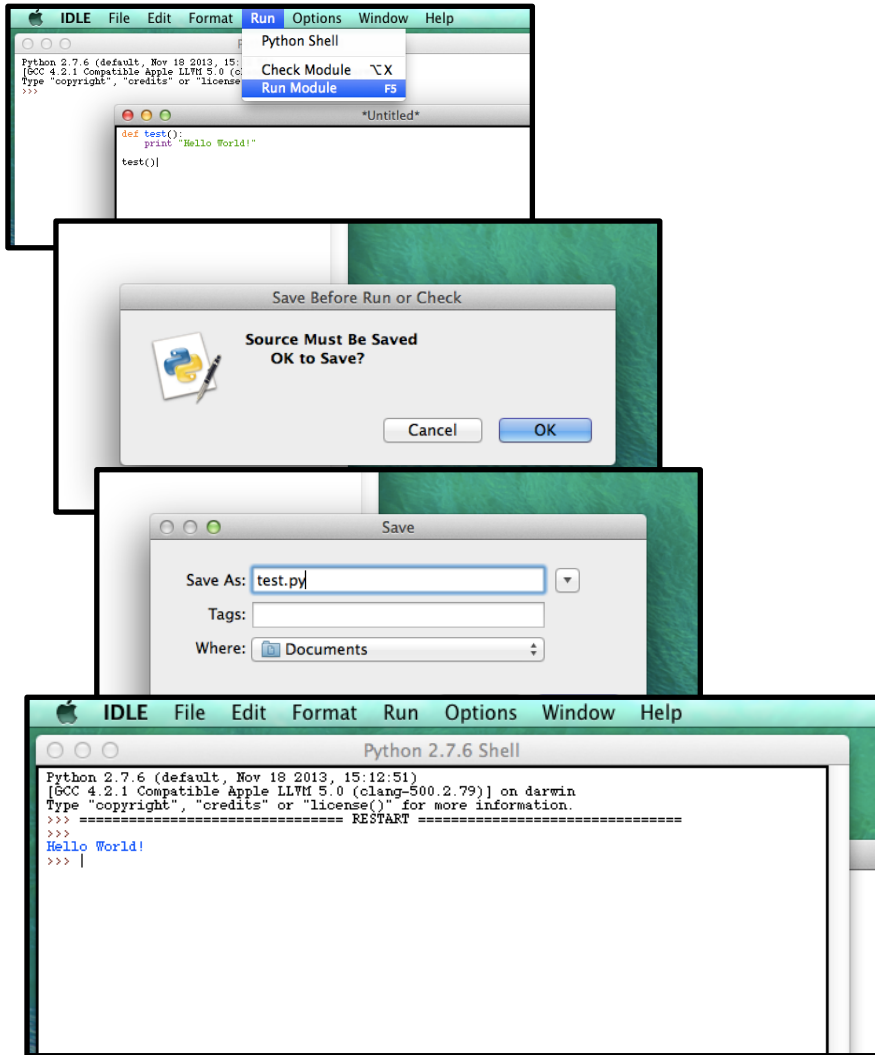
- When you first start IDLE, you get the **Python prompt**
- This prompt allows you to try Python code out and get an **immediate result**
- In particular, **results of expression will be printed out (even if you don't ask them to be printed out)**
- This is not normal  language behavior, and only happens in the shell

IDLE: a Python Environment

When I want to write code, I create a new file



IDLE: a Python Environment



To run my code, I go to Run > Run Module, or I press F5

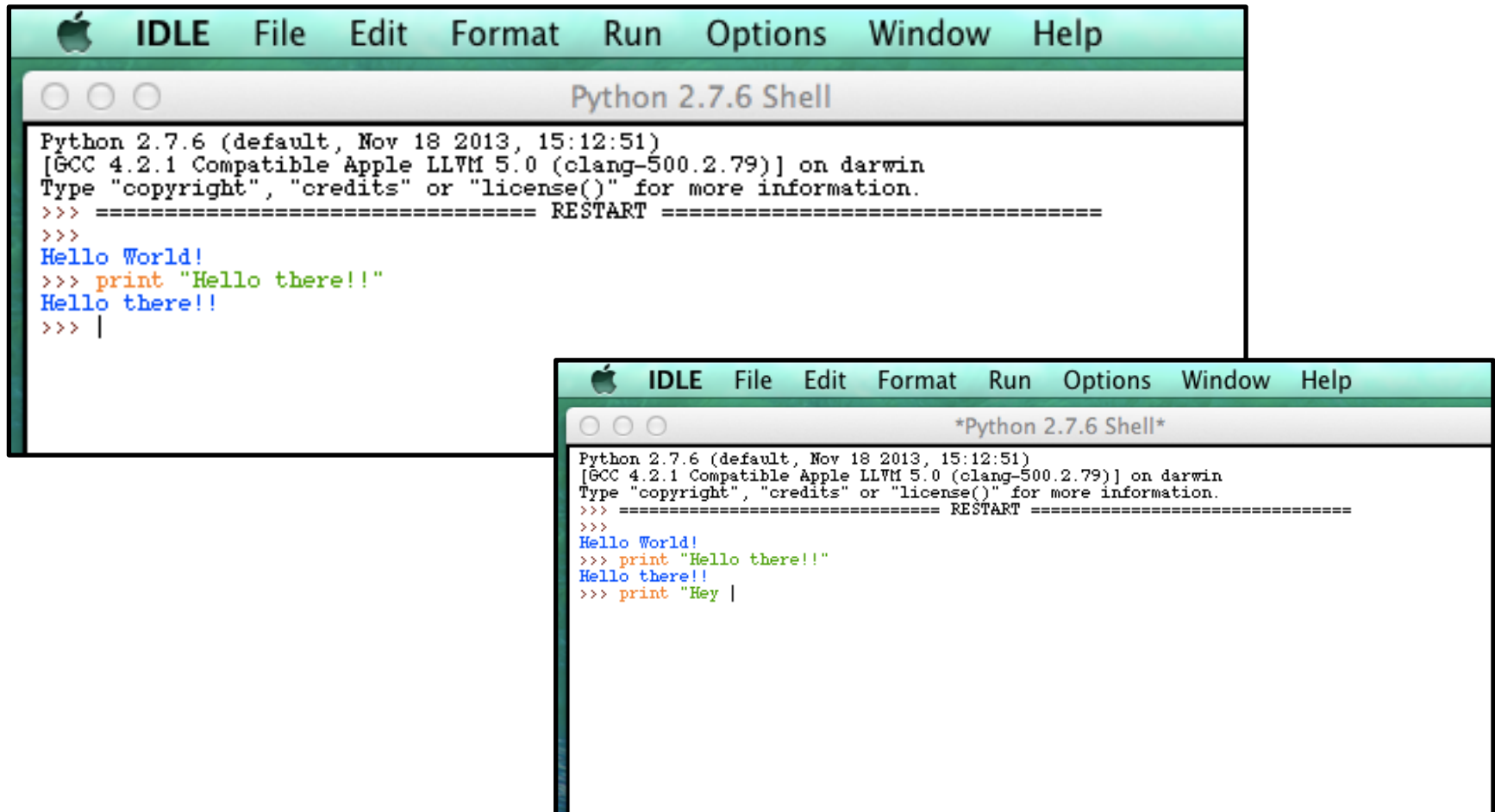
I might be asked to save my program if I have not already

I should then **save** my program

Once that is done (or if I had already saved my program), the code will be run in the shell

Only “print” commands will be outputted (not the result of expressions, unlike when I type in the Python shell)

IDLE: a Python Environment



print statement

- Play around with it
- Play around with expression

```
>>> print "What could I ask of you?"
What could I ask of you?
>>> print 3/4
0
>>> print 3./4.
0.75
>>> print 1/0

Traceback (most recent call last):
  File "<pysbell#4>", line 1, in <module>
    print 1/0
ZeroDivisionError: integer division or modulo by zero
>>> print 4+3+9*3
34
>>> print (4+4+9)*3
51
>>> |
```

Some Useful Links

- To get Python go to: www.python.org/getit/
- Use Python online: www.codeskulptor.org
(this site has a lot of neat demos, such as a [bad] game of Tetris programmed in Python)
- See how Python program work **step by step**:
www.pythontutor.com/visualize.html