

Arquitectura y Lenguajes de Programación

El computador y su arquitectura básica

Podríamos partir preguntándonos qué es un computador, también llamado ordenador o computadora, pero probablemente nos ahogaríamos en un sinfín de definiciones. Las más simples tienen que ver con la idea de **computar**, que tiene su origen en la palabra del latín **computare**, que significa "contar" o "calcular". Las más complejas tienen relación con el aparato que conocemos hoy: una **máquina de cálculo electrónica digital programable**. Diremos entonces que **un computador** es un artefacto capaz de almacenar, computar y manipular **datos**, tanto de manera lógica como matemática, que además puede interactuar con **un usuario**.

Todos hemos escuchado del **hardware** del computador, que no es otra cosa que los **componentes físicos** del computador, es decir, aquellos que podemos tocar. En otras palabras, el hardware corresponde a la máquina en sí misma. En esta categoría podemos encontrar muchos elementos.

La estructura de los computadores ha ido evolucionando a través del tiempo, pero prácticamente todos los computadores comerciales hoy en día, responden a lo que se conoce como **arquitectura Eckert-Mauchly**, también conocida como la **arquitectura de Von Neumann**, propuesta en 1945. Esta consiste esencialmente de cinco componentes (Figura 1.):

1. Una **unidad aritmético-lógica (ALU)**, donde se llevan a cabo las operaciones aritméticas y lógicas.
2. Una **unidad de control**, que se encarga de coordinar los otros elementos para realizar una determinada operación.
3. Una **memoria**, que almacena datos y las instrucciones a ejecutar (programa).
4. **Dispositivos de entrada/salida**, para alimentar la memoria con datos e instrucciones y para entregar los resultados del cómputo almacenados en memoria.
5. **Buses**, que proporcionan un medio de transporte de los datos e instrucciones entre los distintos elementos (cableado).

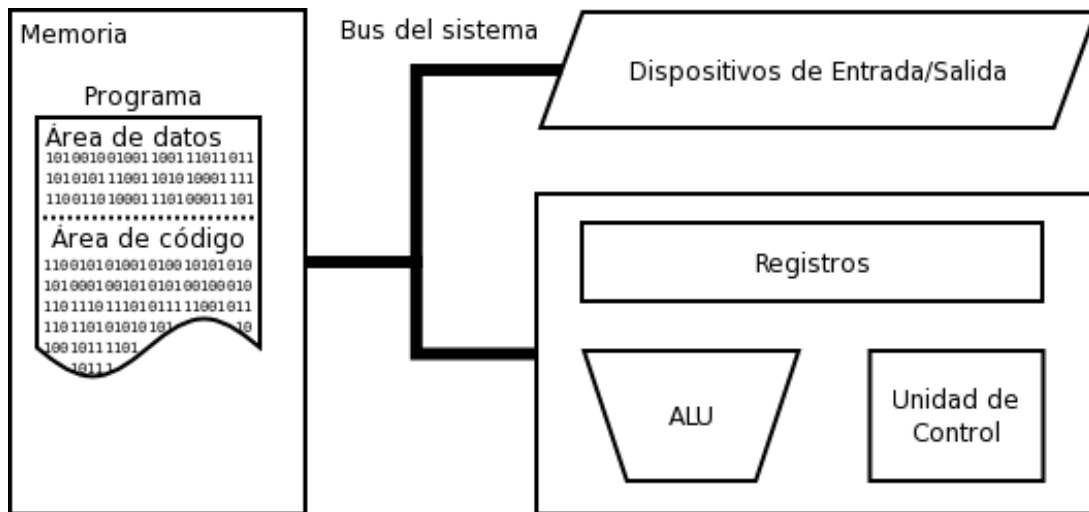


Figura 1. Modelo básico de la arquitectura de Eckert-Mauchly

Los primeros dos elementos más unos dispositivos de almacenamiento más rápidos y más pequeños que la memoria principal, llamados **registros**, constituyen la **unidad central de procesamiento** (UCP o **CPU** por su nombre en inglés: Central Processing Unit), que hoy en día normalmente reside íntegramente en el **procesador**.

Preguntas 1 y 2

En este punto, en forma grupal trabajen en las primeras dos preguntas de la actividad

Sistema binario

Los computadores son máquinas que funcionan con electricidad. En consecuencia, necesitan de un sistema para manejar datos que pueda resumirse en “hay voltaje” o “no hay voltaje”. A esta unidad mínima de electrónica digital se le conoce como **bit**. A través de convenciones y estándares de codificación, los bits permiten representar números, textos, imágenes, videos, etc. Para manejar números enteros se usa el **sistema numérico binario**, en que una secuencia de n bits corresponde a un número natural (entero ≥ 0) de acuerdo a la siguiente fórmula:

$$decimal(b_{n-1}b_{n-2} \dots b_2b_1b_0) = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

En las arquitecturas actuales **n** normalmente es múltiplo de 8, como 32 ó 64 bits. A una secuencia de ocho bits se le llama **byte**, por lo que las arquitecturas actuales usan **palabras** de 4 y 8 bytes.

Ejemplo 1

En un computador con palabras de un byte,

$$(00000011)_2 = 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (3)_{10}$$

$$(00001011)_2 = 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (11)_{10}$$

$$(10010010)_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (146)_{10}$$

Esto último se lee “el número binario 10010010 es igual al número decimal 146”.

Para realizar una conversión de natural a binario, existen dos métodos, el primero, es similar al que se muestra en el ejemplo 1, consiste en **descomponer** un **número decimal** en potencias de 2, sin embargo, éste método requiere **memorizar** o **calcular manualmente** los valores de dichas potencias desde 2^0 hasta por lo menos 2^8 . Con ésta información se descompone el número para obtener los 0's y 1's que componen el número binario.

Ejemplo 2

En un computador con palabras de un byte,

$$(18)_{10} = 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (00010010)_2$$

$$(254)_{10} = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (11111110)_2$$

$$(131)_{10} = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (10000011)_2$$

Como podemos ver, éste método corresponde a una transformación en muchos casos “al ojo”, que puede llevarnos a error si no estamos familiarizados a descomponer en potencias de 2, un método más formal consiste en **dividir repetidamente por 2**, desde el número original hasta llegar a 0 o un resto 1, y obtener el número a partir de los **restos** de cada una de éstas divisiones, éste método **nos asegura** que estamos representando correctamente el número deseado.

Ejemplo 3

Supongamos que tenemos el número $(138)_{10}$ para convertirlo en binario sería:

$$\begin{array}{rcl} 138 \div 2 & = & 69 \text{ con resto } 0 \\ 69 \div 2 & = & 34 \text{ con resto } 1 \\ 34 \div 2 & = & 17 \text{ con resto } 0 \\ 17 \div 2 & = & 8 \text{ con resto } 1 \\ 8 \div 2 & = & 4 \text{ con resto } 0 \\ 4 \div 2 & = & 2 \text{ con resto } 0 \\ 2 \div 2 & = & 1 \text{ con resto } 0 \\ 1 \div 2 & = & 0 \\ & & \text{con resto } 1 \end{array}$$

Tomamos cada uno de los restos obtenidos y los ordenamos del último al primero lo que nos da 10001010, lo cual corresponde a la representación binaria del 138 como se demuestra a continuación:

$$(10001010)_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (38)_{10}$$

Pregunta 3

En este punto, ya podemos comenzar a trabajar en la pregunta 3 de la actividad.

Como vimos anteriormente, la posición en la que había un 0 o un 1, en una secuencia de bits, nos indicaba la potencia de 2 a la que teníamos que elevar para obtener el número decimal original, a esto se le denomina **codificación posicional en bytes**, y posee muchas ventajas, en especial una aritmética muy simple.

Ejemplo 4

$$\begin{array}{r} 00000100 \\ + 00000111 \\ \hline 00001011 \end{array}$$

Podemos observar en este ejemplo, que la suma en el sistema binario moderno es idéntica a la decimal, que conocemos desde la escuela: se suma cada par de dígitos en una misma posición. La diferencia radica en cuándo ocurre una reserva: cuando ambos dígitos son uno, el resultado es $1 + 1 = 10$, es decir se escribe 0 y se acarrea 1 en la reserva.

Sin embargo, esta notación tan simple presenta dificultades para trabajar con los números enteros negativos. En estos casos, el estándar actual es codificarlos en **complemento a dos**. El procedimiento es muy simple:

1. primero, obtenemos la representación binaria del **valor absoluto** del número (la versión positiva)
2. luego esta representación **se complementa**, es decir los ceros se cambian por unos y los unos por ceros
3. a este resultado, **le sumamos** la representación binaria del valor uno para obtener la representación final

Ejemplo 5

La representación del valor -7 sería:

$$(7)_{10} \rightarrow (00000111)_2$$

$$\text{Complementando } (00000111)_2 \rightarrow (11111000)_2$$

$$\text{Sumando } (00000001)_2 \rightarrow (11111001)_2$$

$$\text{Luego } (-7)_{10} = (11111001)_2$$

Una de las ventajas por las cuales se utiliza el estándar complemento a dos es que **mantiene la lógica aritmética**, lo que además permite implementar la operación de resta fácilmente (i.e. sin hardware adicional). Por ejemplo, la operación $4 - 7$ en lógica binaria de 8 bits sería:

Ejemplo 6

$$4 - 7 = + \begin{array}{r} 00000100 \\ 11111001 \\ \hline 11111101 \end{array} = -3$$

Pregunta 4

Desarrollemos ahora la cuarta pregunta de la actividad.

Software

No podemos hablar de software sin definir antes la noción de **programa**. Así, podemos decir que un programa es una **secuencia de instrucciones que un computador puede interpretar y ejecutar**. Así, un programa es algo que necesitamos para que los computadores funcionen, pero es algo **intangible**. Solo podemos ver el resultado de su funcionamiento.

Un concepto similar, a veces usado como sinónimo, es el término **software**. Sin embargo éste agrupa algo más que instrucciones ejecutables por un computador y se refiere más bien a los programas, procesos, reglas y documentación asociada para la operación de un **sistema de información**.

Clasificaciones de Software

Una de las clasificaciones que se le da comúnmente al software, es según su utilización. De esta forma se pueden distinguir tres grandes grupos: los sistemas de aplicación, los sistemas operativos y los sistemas utilitarios.

Los sistemas de aplicación son aquellos que interactúan con el usuario, permitiéndole realizar distintas funciones en el computador o equipo tecnológico que esté utilizando. Entre ellos se pueden mencionar los sistemas de ofimática (v.g. LibreOffice y MS Office), para el manejo de imágenes (v.g. InkScape y Adobe PhotoShop), para aplicaciones científicas (e.g. Octave y Matlab), que permiten al usuario realizar diferentes tareas, según su propósito.

Los sistemas operativos son aquellos que permiten que los sistemas de aplicación puedan ser ejecutados. En los computadores actuales normalmente se puede encontrar a Unix, Solaris, Linux, Windows, Irix, FreeBSD, etc..

Los sistemas utilitarios son programas adicionales que ayudan a la interconexión interna entre programas de aplicación o dispositivos externos, y un sistema operativo. Algunos ejemplos de ellos son: drivers, codecs de video, bibliotecas dinámicas (DLLs), entre otros.

Lenguajes de programación

Como hemos visto, los computadores operan con unidades digitales binarias. Es decir, todo es almacenado y manejado en forma de ceros y unos. Desde luego, esto resulta muy difícil de comprender y manejar para las personas, por lo que existen **lenguajes** que buscan acercarse un poco más a la forma en que nosotros nos comunicamos.

En la enseñanza básica y media, todos aprendemos que los lenguajes, como el español o el inglés, tienen una **estructura** y un **conjunto de normas** para poder expresarnos adecuadamente. Los **lenguajes de programación** también son, como su nombre lo indica, lenguajes, aunque su finalidad no es facilitar la comunicación entre personas, sino permitir que los seres humanos puedan interactuar con máquinas, y tienen diversas **reglas** que permiten escribir correctamente **sentencias** y **bloques**, que corresponden respectivamente a las ideas de oraciones y párrafos del español. Los siguientes párrafos describen los tipos de reglas que encontramos normalmente en los lenguajes de programación.

Reglas léxicas que establecen cómo debe ser escrita cada *palabra* del programa. Por ejemplo, en muchos lenguajes “los nombres deben comenzar con una letra, aunque puedan contener también dígitos”.

Reglas sintácticas que determinan cómo debe ser la estructura de un programa. Vale decir, cómo se deben ordenar las palabras en una sentencia o las sentencias en un bloque. Por ejemplo, si en matemáticas decimos $c = a + b$ estamos indicando claramente que el valor c corresponde a la suma de los valores de a y de b , mientras que si escribimos $c\ b = a +$ ¡no se entiende!

Reglas semánticas que comprueban que cada palabra presente en una sentencia tenga sentido de acuerdo a su contexto. Por ejemplo, la sentencia $c = b + a$ es sintácticamente correcta. Pero... ¿qué pasa si c e b son números y a es una palabra? ¿Será posible hacer esa suma?

Algunos lenguajes de programación son más cercanos a nosotros que otros, por lo que podemos hablar de lenguajes de bajo y alto nivel.

Pregunta 5

En este punto, ya podemos resolver la quinta pregunta de la actividad.

Lenguajes de bajo nivel

Los lenguajes de programación de bajo nivel son muy **cercanos al lenguaje propio de la máquina** (i.e. secuencias de unos y ceros), pero escritos de una manera más clara para nosotros. Por ejemplo, en vez de escribir un valor o una instrucción en su formato binario, podemos simplemente escribir el número en formato decimal o el nombre de la instrucción. Eso sí, estos lenguajes se llaman de bajo nivel porque **no nos proporcionan capacidad de abstracción**. Es decir, tenemos que explicar al computador lo que debe hacer con muy pocas herramientas. La gran ventaja, sin embargo, es que es muy fácil traducir el programa al lenguaje de la máquina y es sencillo lograr un uso eficiente de los recursos. El ejemplo 7 muestra un programa escrito en un lenguaje de bajo nivel **para el**

procesador MOS Technology 6502 (usado en computadores clásicos como ATARI, Apple II, y Commodore 64), que escribe la palabra **HOLA** en pantalla¹. Podemos ver que el programa no es más que una secuencia de instrucciones, sin una estructura aparente que facilite su comprensión.

Ejemplo 7

```
1      = $6000
2      inicio
3          lda #'H
4          jsr $f2b0
5          lda #'O
6          jsr $f2b0
7          lda #'L
8          jsr $f2b0
9          lda #'A
10         jsr $f2b0
11     loop
12         jmp loop
13         *=$2e0
14         .word inicio
```

Lenguajes de alto nivel

De la explicación anterior podemos deducir que escribir programas en lenguajes de bajo nivel puede ser muy complejo. Por esta razón, existen también otros lenguajes de programación que son más **cercanos a la forma de comunicación humana**. Estos lenguajes **nos proporcionan diversos mecanismos de abstracción** que nos permiten crear programas de manera más clara para nosotros. El ejemplo 8 muestra un fragmento de un programa escrito en el lenguaje de programación de alto nivel Pascal que, al igual que el ejemplo 7, muestra la palabra **HOLA** en pantalla². Se puede observar que la terminología es bastante más comprensible al ser más cercana al idioma inglés, y también que los elementos están organizados de un modo jerárquico.

Ejemplo 8

```
1      program helloworld;
2      begin
3          writeln(HOLA)
4      end.
```

¹ www.retrogames.cl/cap4.html

² rosettacode.org/wiki/Hello_world#Pascal

Por supuesto, estos lenguajes también tienen sus **desventajas**: necesitan ser traducidos al lenguaje de máquina y a veces no son tan eficientes como los de bajo nivel en el uso de recursos.

Lenguaje de programación Python

Vamos a usar el lenguaje de programación **Python** para aterrizar los conceptos que se estudian en este curso, principalmente porque es un lenguaje con una gramática bastante simple y clara, lo que lo hace ideal para iniciarse en "*las artes de la programación*". Python es un lenguaje **interpretado** de propósito general que sustenta diferentes paradigmas de programación, pero en el curso se enfoca principalmente en el **paradigma procedural-imperativo**, es decir en programas que describen **procedimientos detallados** para un computador de **cómo hacer** una tarea. Esto contrasta con otros paradigmas que enfrentan la programación describiendo **qué hacer** o basados en la interacción de **entidades abstractas** que describen datos de la realidad.

Como Python es interpretado, un **usuario** puede interactuar directamente con su **intérprete** y utilizarlo, en el ejemplo 9 vemos como pedirle al intérprete de Python que despliegue el mismo mensaje que los ejemplos 7 y 8.

Ejemplo 9

```
Python 2.6 (r26:66721, Oct 2 2008, 11:06:43) [MSC v.1500 64 bit (AMD64)]
on win 32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "HOLA"
HOLA
>>>
```

Compiladores e intérpretes

Hemos hablado de lenguajes de alto nivel y de bajo nivel, que deben ser traducidos al lenguaje de la máquina para que puedan ser ejecutados. Esto quiere decir que necesitamos algún mecanismo que nos permita llevar nuestro programa al **lenguaje binario del computador**, ocupando únicamente las instrucciones disponibles.

El primer mecanismo a considerar es el **compilador**. Esta herramienta lee un programa escrito en un lenguaje fuente (o sea, el lenguaje que ocupó el programador) y revisa que no contenga errores léxicos, sintácticos o semánticos. Si hay errores, notifica al programador. En caso contrario, crea un nuevo programa equivalente escrito en **lenguaje objeto**, generalmente el lenguaje de la máquina (en Windows, este programa equivalente corresponde al archivo ejecutable). Una de las grandes ventajas de este esquema es que logra que la ejecución del programa sea más rápida al no tener que hacer la revisión y la traducción cada vez que queramos usarlo. Sin embargo, tiene la desventaja de que el programa compilado **no sirve en cualquier máquina**.

El segundo mecanismo es el **intérprete**. En este caso no se crea un programa equivalente que pueda ser reutilizado, sino que se van tomando **una a una las sentencias escritas en lenguaje fuente**, en forma secuencial, para revisarlas y traducirlas al lenguaje de máquina. Una vez traducida una sentencia, esta es ejecutada y la traducción se descarta. Se puede concluir que, con este enfoque, la ejecución de un programa es más lenta porque debemos esperar la traducción cada vez que queramos usar el programa. Sin embargo, basta con que una máquina tenga el intérprete del lenguaje empleado para poder ejecutarlo, sin necesidad de hacer modificaciones.

Instalando el intérprete de Python

Ahora, llegó la hora de instalar el intérprete de Python en sus equipos, el profesor pasará distribuyendo el archivo ejecutable por los grupos, que contiene el interprete de Python para la versión **2.6** del lenguaje, además podemos descargarlo directamente desde la web desde el sitio (<http://www.python.org/download/releases/2.6/>) es importante que descarguemos el ejecutable que va de acuerdo a la máquina en que lo instalaremos, ya sea **Linux, Mac, Windows de 32(x86) o 64(Amd64)** bits.

Importante

Es importante que descargues la versión de Python con la que trabajaremos en el curso, que será **Python 2.6**, ten en cuenta que los ejemplos, la documentación y los libros que usaremos estarán orientados a funcionar en esa versión.

Por lo que si instalas una versión diferente y los ejemplos no funcionan como los apuntes indican, es de tu **exclusiva responsabilidad**, y puede significar incluso que tus programas no funcionen y se te califique con menor nota.

Se eligió esta versión de Python debido a que es estable, cercana al lenguaje natural y existen diversos recursos en la web que pueden apoyar las prácticas fuera del aula, a diferencia de las versiones más recientes.

Una vez que tengas el archivo, comenzaremos a instalarlo como cualquier otro programa, es decir, haciendo **doble clic** sobre el ejecutable, lo que nos llevará a la pantalla que se muestra en la figura 2.

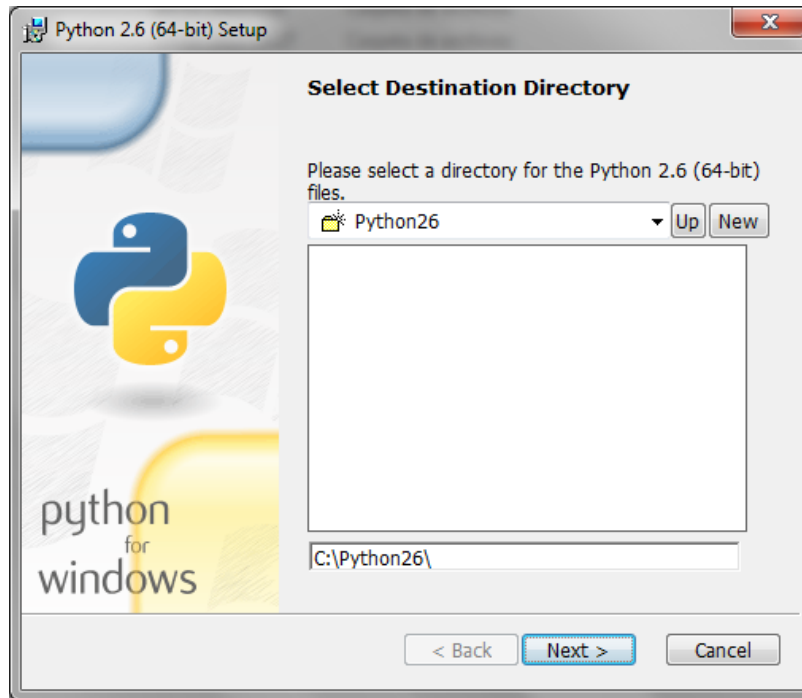


Figura 2. Pantalla de Bienvenida del Intérprete de Python

Seleccionaremos el directorio en el que queremos que se instale Python y luego presionamos **Next >**, a continuación veremos la pantalla de personalización de la instalación de Python, en la cual podremos indicar que módulos de Python queremos que se instalen, se recomienda, debido a su peso, **instalar todo directamente** para evitar tener que volver a cargar módulos de nuevo, esto se realiza seleccionando en Python, la opción **Entire feature will be installed on local hard drive**, como se muestra en la figura 3., seleccionamos esta opción y volvemos a pulsar **Next >**.

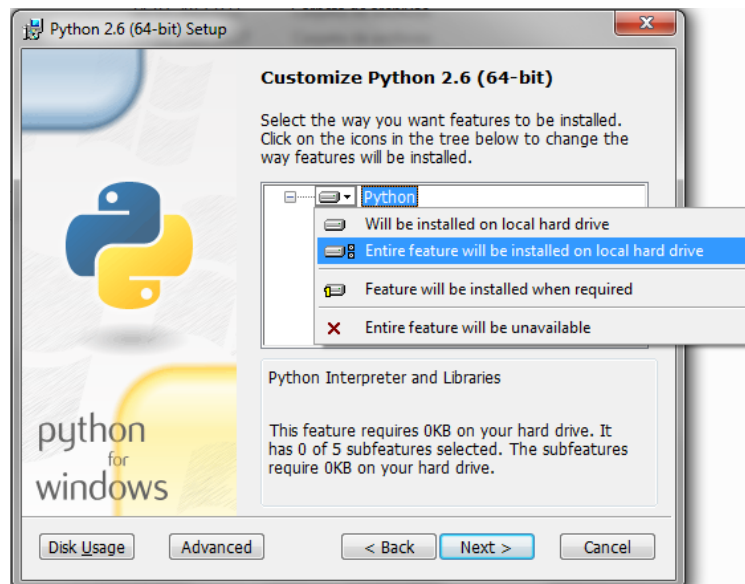


Figura 3. Pantalla de personalización de la instalación

Esperamos unos minutos, en los que se nos mostrará el **progreso de la instalación**, como se muestra en la figura 4.

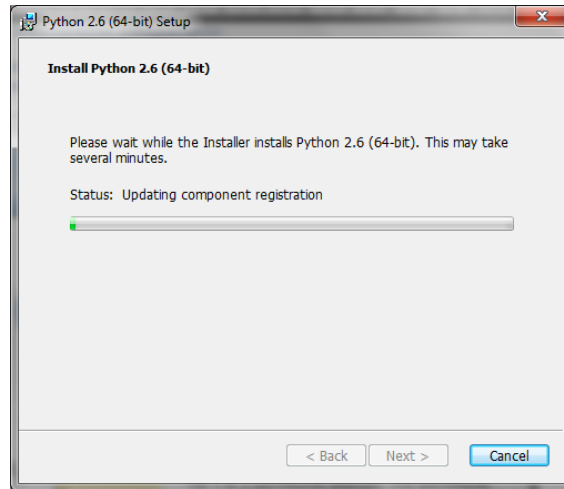


Figura 4. Progreso de la instalación

Finalmente, si la instalación ha sido exitosa, deberíamos ver la pantalla que se muestra en la figura 5, con esto pulsamos **Finish** y terminamos exitosamente la instalación del intérprete de Python.



Figura 5. Finalización exitosa de la instalación

Tarea

Si tienes un equipo para practicar en tu hogar, instala el intérprete de **Python 2.6** bajándolo desde internet y pruébalo como se indica a continuación.

Una vez realizada la instalación, probaremos si esta ha sido exitosa, para ello iremos a **Inicio → Todos los programas** y buscaremos entre el listado de programas la carpeta **Python 2.6** y abriremos ahí el programa llamado **Python (command line)**, deberíamos ver una pantalla negra con letras blancas con las primeras líneas del **ejemplo 6**, y deberíamos ver una línea horizontal parpadeante inmediatamente después del signo **>>>**, como se muestra en la figura 6.

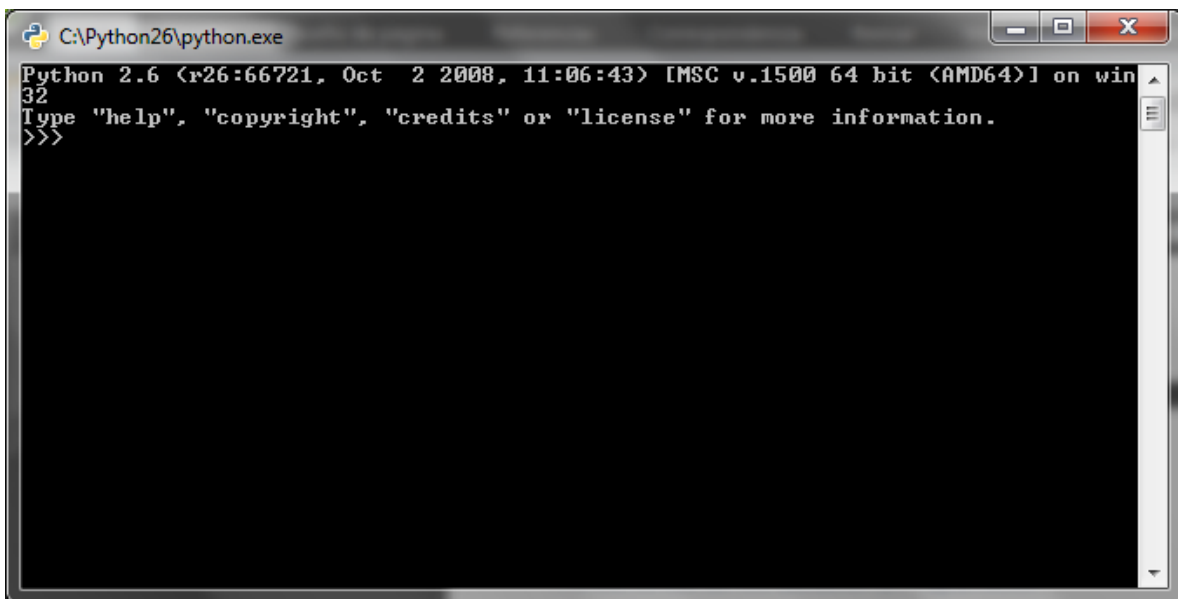


Figura 6. Intérprete de Python en consola

En ella escribiremos `print "HOLA"`, y presionaremos **enter**, ahora la pantalla debería tener el mismo texto presentado en el ejemplo 6, es decir, el computador debería haber respondido **HOLA**, como se le solicitó con el comando, si lo hizo, lo hemos logrado, se ha instalado **el intérprete de Python correctamente** y has escrito tu **primera línea de código** en Python, conocida en el mundo de la informática como **HOLA MUNDO**. ¡Felicitaciones!

Forma alternativa de ejecutar el intérprete

Es posible ejecutar el intérprete de Python directamente desde la **consola de Windows**, esto se realiza abriendo la consola con el comando **cmd** en la barra de búsqueda de programas de **Windows Vista y 7** o en la opción **ejecutar** en el menú de inicio de **Windows XP** y versiones anteriores.

Al abrir la ventana de comandos, nos muestra la consola de Windows, y debemos navegar en ella con los comandos `cd ..` y `cd <Nombre del Directorio>` hasta encontrar la carpeta de Python, llamada comúnmente **Python26**, una vez allí, tipeamos **Python**, presionamos la tecla **enter** y la consola de Windows funcionará como el intérprete de Python.