Lenguajes, Tecnologías y Paradigmas de la Programación

Práctica 1

Introducción a C y Python

Contenido

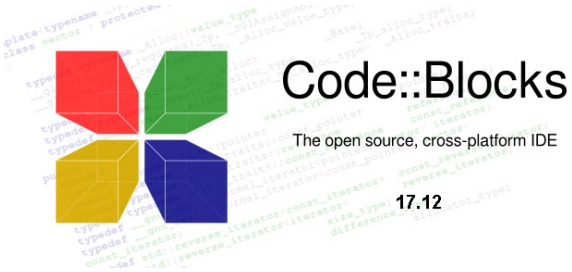
1. Introducción

El objetivo de esta práctica consiste en introducir al alumno los lenguajes de programación C y Python.

Para ello, primero se presentarán las herramientas con las que se trabajará a lo largo de la práctica: GCC + Code::Blocks para C y Python + IDLE para Python. Posteriormente se realizarán una serie de ejercicios para ir conociendo ambos lenguajes, así como sus respectivos entornos de desarrollo y de ejecución. Finalmente, se propondrán dos ejercicios que deberán ser entregados en el tiempo estipulado por el profesor en PoliformaT.

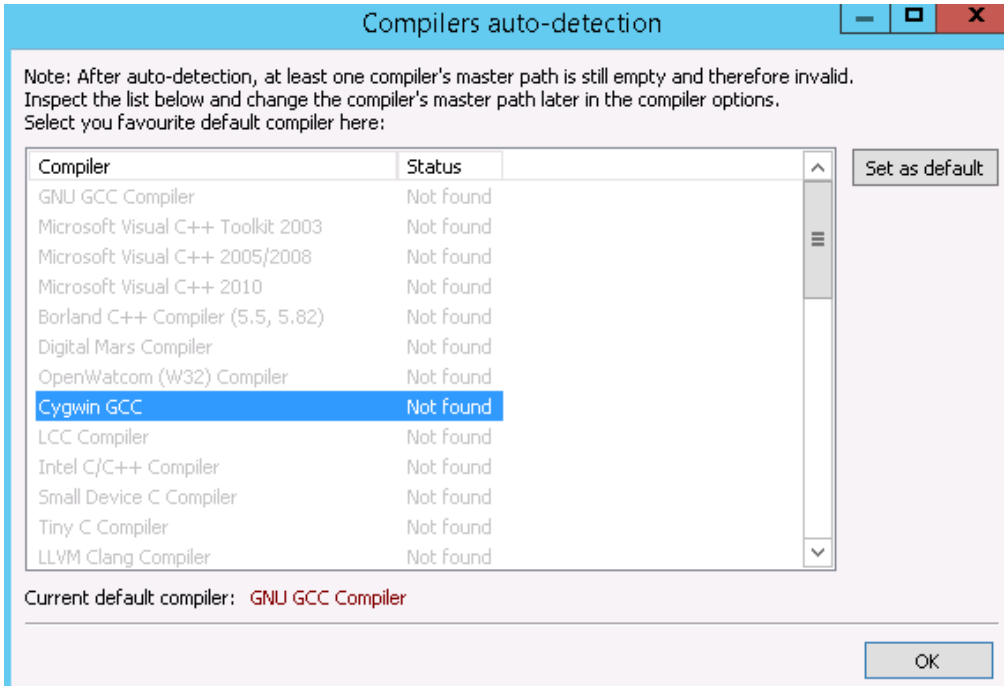
2. Programando en C

Code::Blocks es un entorno de desarrollo libre que permite desarrollar software utilizando los lenguajes de programación C/C++ y Fortran, entre otros. Puede utilizar distintos compiladores. En la práctica se usará Cygwin/GCC, que ya está preinstalado en las máquinas del laboratorio.

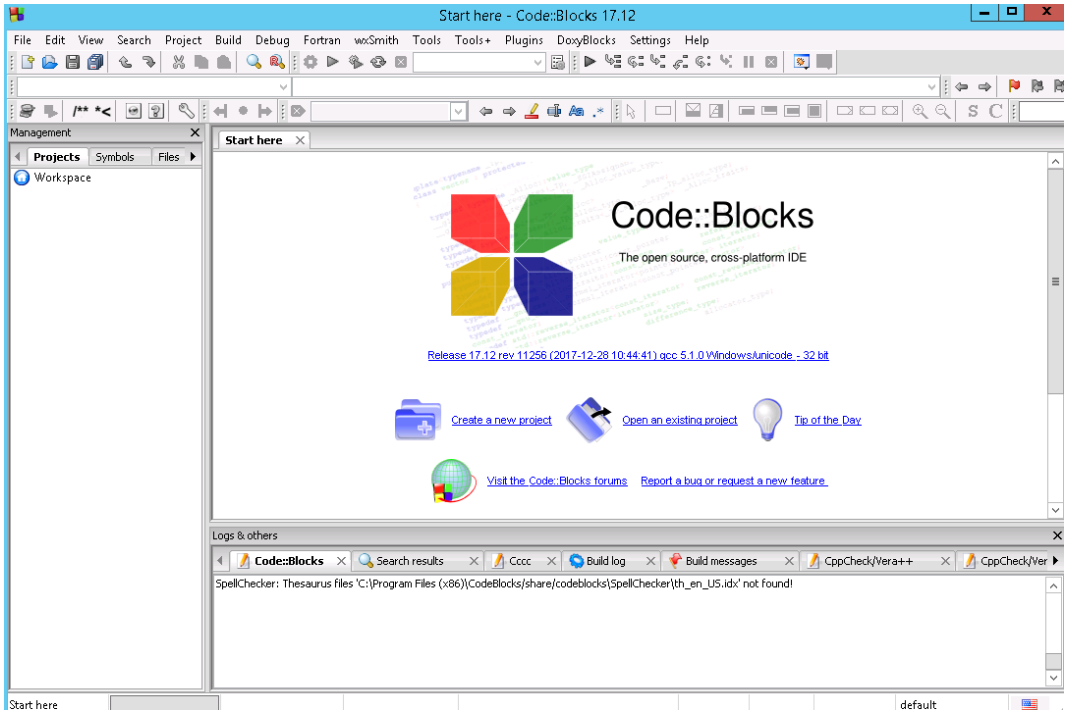


Para descargar el entorno de desarrollo hay que acceder al URL: https://codeblocks.org

El entorno ya está instalado en las máquinas del laboratorio. Cuando se arranque por primera vez, detectará que no hay ningún compilador asociado y mostrará los compiladores soportados por el entorno. Hay que seleccionar Cygwin GCC.

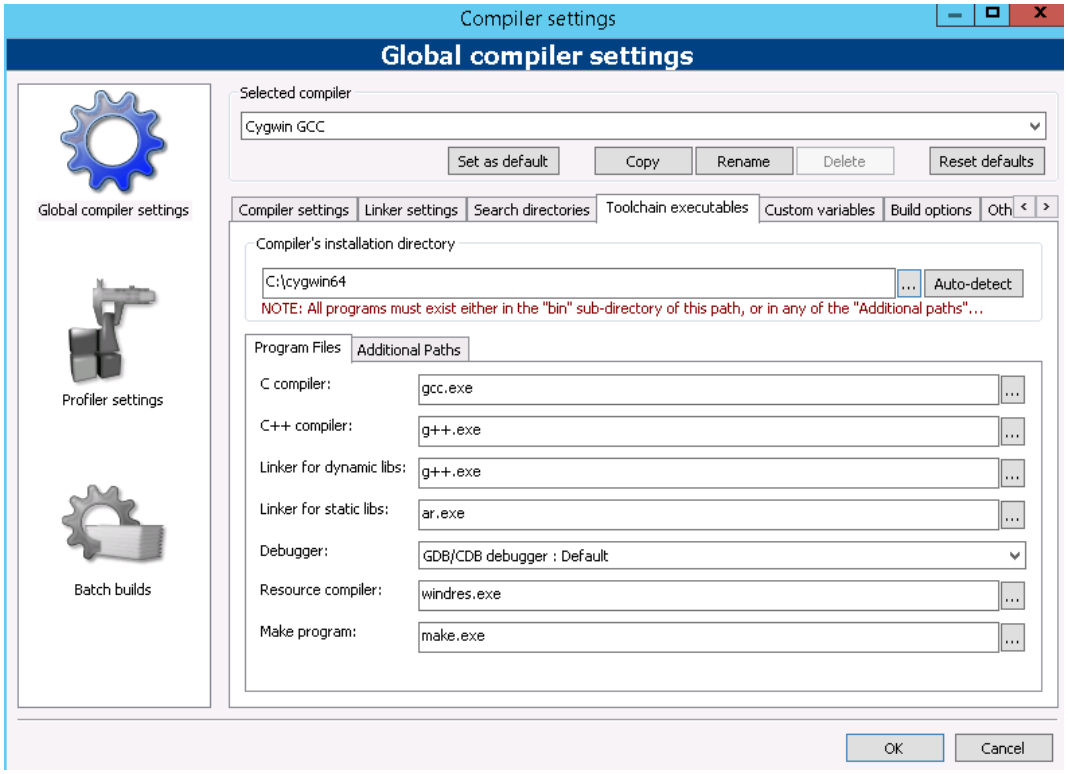


Una vez abierto, se dispondrá de un entorno de desarrollo como el que se muestra en la siguiente figura:



Lo primero que hay que hacer es indicarle al entorno cuál es el compilador por defecto con el que hay que trabajar y el lugar donde se encuentra. Para ello hay que seleccionar *Settings* / *Compiler*... Ahí hay que escoger el compilador Cygwin GCC y pulsar el botón "*Set as default*". Después hay que seleccionar la pestaña "*Toolchain executables*" e indicar la ubicación del directorio donde se encuentra Cygwin, en este caso **c:\cygwin64**.

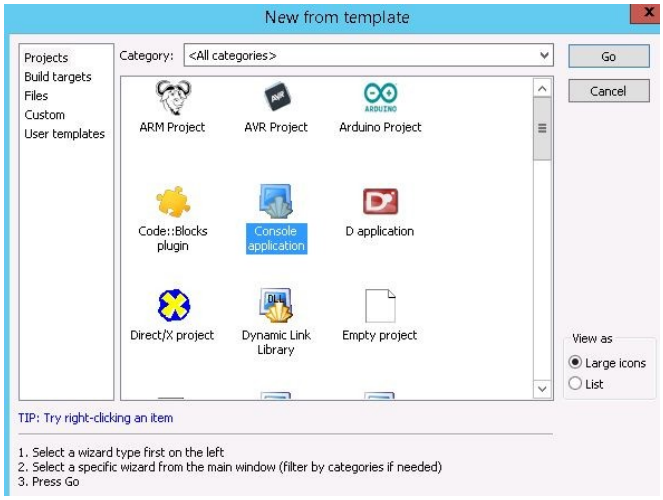
Es posible que sea necesario reiniciar el IDE Para que detecte los cambios correctamente.

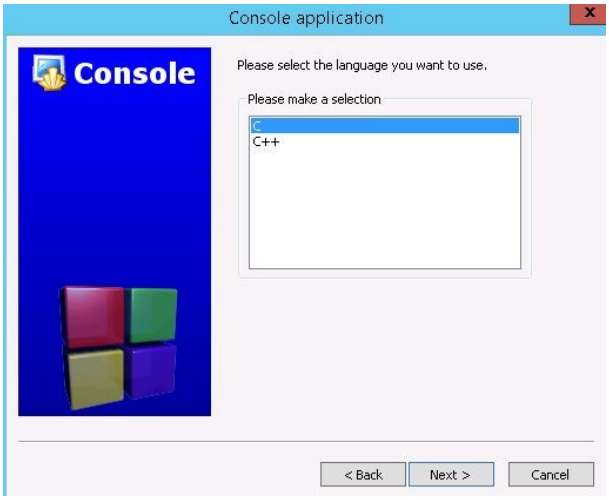


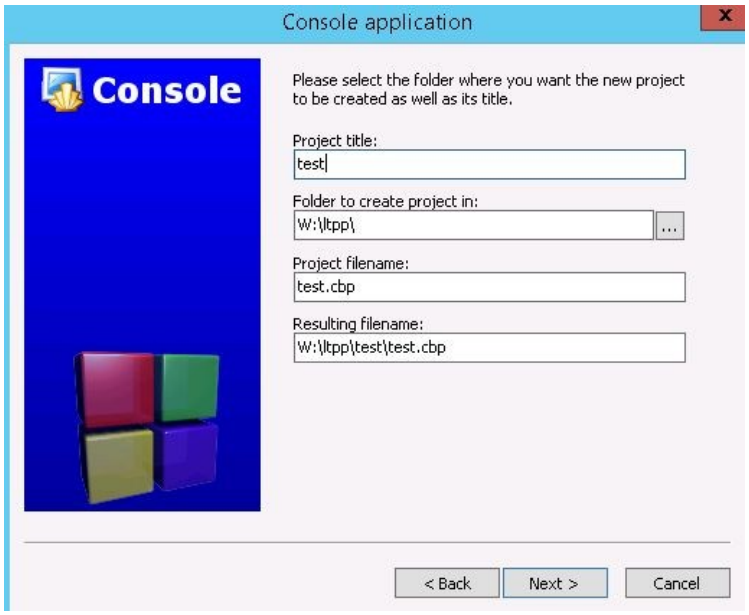
Como puede observarse, Code::Blocks es un entorno de desarrollo orientado a proyectos. Aunque es posible crear, compilar y ejecutar ficheros independientes, generalmente siempre se crea un contexto que permite trabajar con los fuentes, este contexto es un proyecto.

A continuación se creará un proyecto seleccionando *File* / *New* / *Project ...* / *Console application*. En el asistente de creación del proyecto hay que escoger el lenguaje de programación C. Después hay que dar un nombre al proyecto (por ejemplo "test") y una ubicación (por ejemplo W:\ltpp).

Es conveniente usar la unidad W:, tanto si se trabaja con las máquinas de laboratorio como si se hace con Polilabs, para no perder el trabajo realizado.







Otros IDEs que pueden resultar útiles son Eclipse o CLion (Jetbrains). Para compilar y ejecutar C con Eclipse hay que instalar unos componentes (*Help / Install New Software,* buscar en el menú *Programming Language* y seleccionar *C/C++ Developmen Tools*). Con la cuenta de la UPV se dispone de una licencia educativa de CLion.

Code::Blocks automáticamente creará un proyecto y bajo la carpeta de "Sources" habrá un archivo main.c generado automáticamente. Si se abre este archivo habrá una función `main()`, que es el punto de entrada de cualquier programa escrito en C. El contenido de esta función será similar al siguiente:

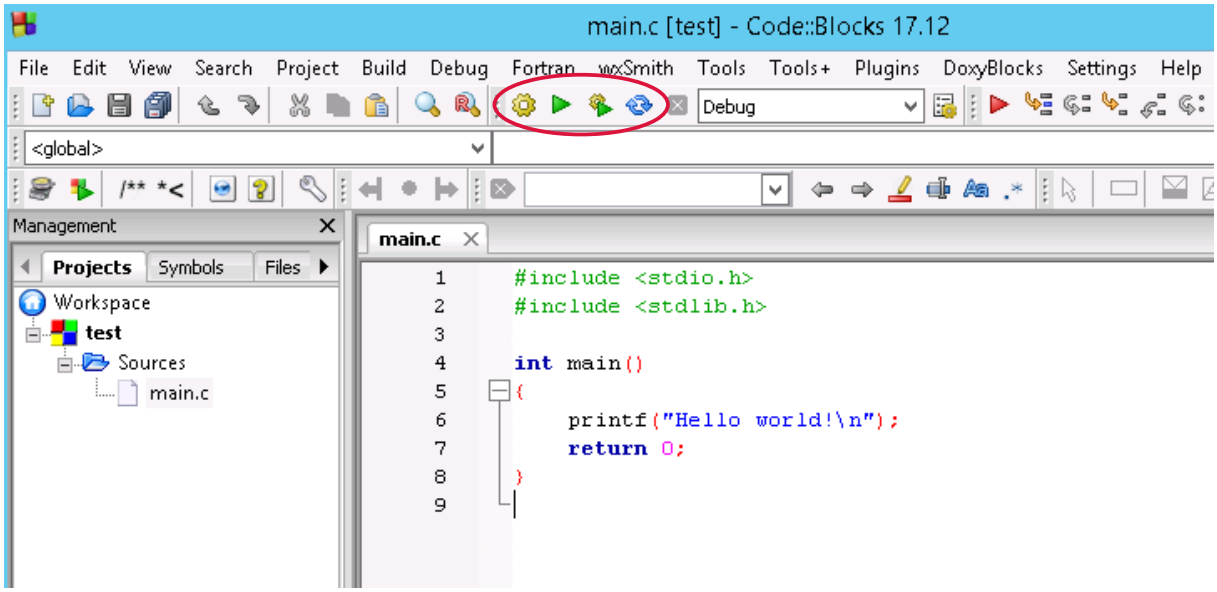
```
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Hello world!\n");
    return 0;
}
```

El programa es muy simple. Las dos primeras líneas incluyen los ficheros de cabecera estándar '*stdio.h*' y '*stdlib.h*'. Un fichero de cabecera generalmente describe funciones y tipos de datos, pero no los implementa. Su objetivo es describir el código que se implementa en otro/s fichero/s con extensión '.c' que generalmente se proporciona en forma de librería. En este caso, '*stdio.h*' define rutinas básicas de entrada/salida en el lenguaje C y '*stdlib.h*' otras utilidades básicas del lenguaje.

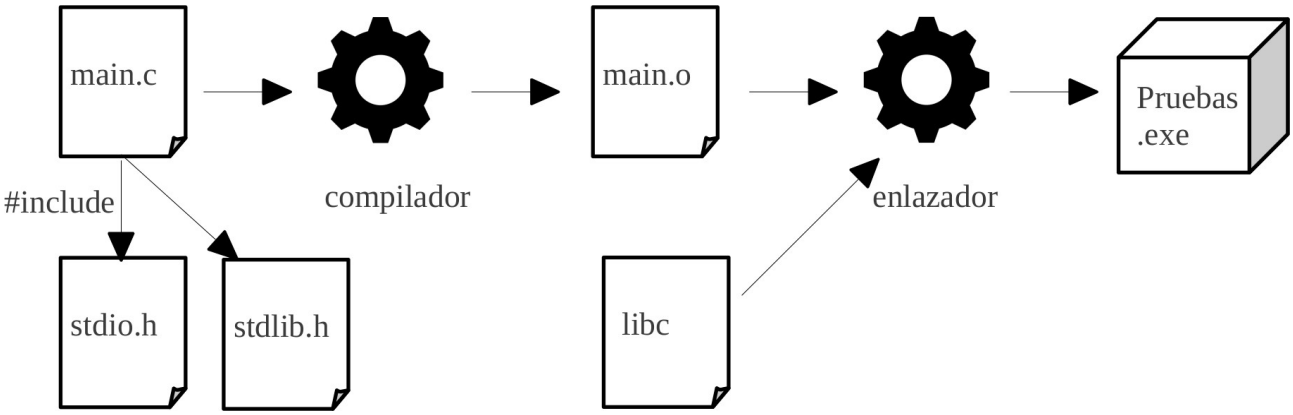
La función `main()` es el punto de entrada de cualquier programa en C. Cuando el programa se ejecuta, automáticamente se pasa el control a `main()`. La función `main()` del ejemplo anterior muestra el mensaje "Hello world!" y termina devolviendo un código 0 (es decir, correcto).

El siguiente paso consiste en compilar el programa y ejecutarlo. En la barra de herramientas de Code::Blocks hay botones para compilar el proyecto (*build*), ejecutarlo (*run*), compilar y ejecutarlo (*build and run*) o recompilar todo el proyecto (*rebuild*), tal como se remarca en la siguiente figura:



Si se presiona el botón de compilación (o se selecciona la opción del menú *Build / Build*) se compilará el fichero '*main.c*'. A continuación se puede navegar en el sistema de ficheros hasta el directorio donde está guardado el proyecto. Se verá que se ha generado un fichero objeto '*.o*' con el mismo nombre que el fichero fuente (*ruta_proyecto\obj\Debug\main.o* en este ejemplo) y un fichero ejecutable '*.exe*' con el mismo nombre que el proyecto (*ruta_proyecto\bin\Debug\test.ext* en este ejemplo).

El fichero '*.o*' es el resultado de compilar el código fuente '*.c*' en código nativo de la máquina. Sin embargo, este fichero objeto '*.o*' no es ejecutable, ya que hace referencia a funciones que no están implementadas en el fichero fuente '*.c*', como por ejemplo `printf()`. Es necesario, por tanto, enlazar dichero fichero objeto '*.o*' con otro fichero con código nativo donde se encuentren las implementaciones de estas funciones y generar finalmente el fichero ejecutable '*.exe*'. En este caso, la implementación de la función `printf()` se encuentra en la librería estándar de C. A continuación se muestra gráficamente el proceso:



A continuación se modificará el programa anterior para trabajar con un *string*. En C un *string* es un vector de caracteres, donde el último carácter es el carácter nulo '\0'. Es posible definir constantes de tipo *string* utilizando literales como "hola". A continuación se presenta una variación del primer ejemplo:

```
char msg[] = "Hola mundo!\n";
printf("%s", msg);
return 0;
```

En este ejemplo se usa una variable `msg` para almacenar un valor que se usará posteriormente para imprimir su valor por pantalla.

La función `printf()` es una función muy sencilla definida en '*stdio.h*'. Recibe como primer parámetro una cadena de formato, donde se pueden intercalar caracteres normales y expresiones de formato. Una expresión de formato comienza con el carácter % y puede ser contemplada como un hueco que será formateado de una determinada manera y que será rellenado con el valor que se suministre en el resto de los parámetros de la función. En el ejemplo anterior se imprime un *string* y por ello se usa el formato %s. Para otros tipos de datos se usan las expresiones %d o %i (enteros), %c (caracteres), %f (coma flotante). En el siguiente ejemplo se imprimen distintos tipos de datos:

```
printf("Me llamo %s, tengo %d años, y peso %f kg", "Pepe", 30, 80.5);
```

Con las expresiones de formato también se puede controlar el número de decimales a imprimir.

Ya se sabe cómo imprimir por la salida estándar. El siguiente paso sería obtener datos de la entrada estándar. Existen diversas funciones para ello, una de ellas es `scanf()`, que también está definida en '*stdio.h*'. Es similar a `printf()`, recibe como primer parámetro una cadena de formato indicando el tipo de los datos que se desea obtener de la entrada estándar, en los siguientes parámetros se indican las variables en las que hay que almacenar los datos. La cadena de formato utiliza las mismas expresiones que `printf()`, es decir, %s (*strings*), %d o %i (enteros), %c (caracteres), %f (coma flotante). En el siguiente ejemplo el usuario introduce una cadena y el programa la imprime por pantalla.

```
char msg[250];
printf("Introduce una cadena: ");
scanf("%s", msg);
printf("%s\n", msg);
return 0;
```

En el código anterior se define un vector de 250 caracteres. Es un *buffer* en el que se pueden escribir cadenas. Posteriormente se pedirá al usuario una cadena de texto, se leerá de la entrada estándar con `scanf()` y finalmente se imprimirá. Se puede introducir este código en un bucle infinito. Para ello se puede usar un bucle *while*. Este bucle acepta un número como condición de tal forma que 0 es interpretado como falso y cualquier otro valor como cierto.

```
char msg[250];
while (1) {
    printf(" > ");
    scanf("%s", msg);
    printf("%s\n", msg);
}
return 0;
```

¿Cómo se podría detener el bucle anterior? Por ejemplo, cuando el usuario introduzca la palabra *exit*. Para ello, habría que comparar carácter a carácter, o bien usar las utilidades definidas en '*string.h*', que permite trabajar con *strings* (comparaciones, búsquedas, concatenaciones, copias,...). Existe una función `strcmp()` que acepta dos *strings* como parámetros y devuelve 0 si son iguales. También será necesario usar una bifurcación con la sentencia *if*, cuya condición se interpreta como en la sentencia *while*.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char msg[250];
    while (1) {
        printf(" > ");
        scanf("%s", msg);
        if (strcmp(msg, "exit") == 0) {
            printf("Bye bye\n");
            break;
        } else {
            printf("%s\n", msg);
        }
    }
    return 0;
}
```

La función `scanf()` lee un dato de la entrada estándar y lo deposita en la variable que se pasa como argumento. Necesita por tanto modificar dicho argumento. Este paso de parámetros se denomina "por referencia" y en C se implementa utilizando punteros (direcciones de memoria). Todo esto se verá más adelante en teoría y prácticas. De momento es suficiente con saber que se se desea recuperar un entero de la entrada estándar hay que indicar como segundo argumento de `scanf()` la dirección de memoria de la variable donde hay que almacenar dicha entrada. Esto en C se hace con el operador &, como se muestra en el siguiente ejemplo:

```
int edad;
printf("Introduce tu edad:");
scanf("%d", &edad);
printf("Tu edad es: %d\n", edad);
edad++; // equivalente a: edad = edad + 1;
printf("Dentro de poco tu edad sera: %d", edad);
```

Otra función que es de mucha utilidad es `strcpy()`, que permite copiar *strings*, pues en C no se puede asignar una cadena a otra "así como así":

```
char str1[6] = “hello”;
// char str2[6] = str1; // Error
char str2[6];
strcpy(str2, str1);
```

En el siguiente ejercicio se usará una función muy útil para generar números aleatorios. Se trata de la función `rand()`, definida en '*math.h*'. Esta función devuelve un número aleatorio entre 0 y `RAND_MAX`, que es una constante que varía entre implementaciones. Si se quiere obtener un número aleatorio comprendido en un rango [0, MAX[(intervalo abierto por la derecha) se puede hacer de forma muy sencilla mediante el operador módulo (%), que devuelve el resto de una división entera.

Sin embargo, si se usa `rand()` únicamente, siempre generará los mismos números aleatorios, ya que parte de la misma semilla y usa el mismo algoritmo generador. Hay que cambiar la semilla en cada ejecución del programa usando la función `srand()`. Se puede usar como semilla la hora actual, así en cada ejecución del programa se obtendrá una secuencia de números diferente. Por ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int max;
    srand(time(NULL));
    printf("Introduce MAX: ");
    scanf("%d", &max);
    int num = rand() % max;
    printf("El numero es: %d", num);
}
```

3. El entregable de C

A continuación se propone desarrollar un pequeño juego en C que el alumno debería ser capaz de crear con las habilidades adquiridas a lo largo de esta práctica.

Se trata de un juego basado en el popular juego de la morra. Es un juego en el que dos jugadores eligen un número del 1 al 5 y después, al mismo tiempo, deben hacer una predicción indicando la suma total reunida entre ambos. Gana el jugador que acierte el número total.

En esta práctica se adaptará el juego:

Al comenzar la partida, el ordenador elegirá un número entre 1 y 5. Después hará una predicción, teniendo en cuenta el número escogido y el que estima que escogerá el usuario, y se lo comunicará a éste. Utilizará para ello el generador de números aleatorios estudiado anteriormente. Una vez hecha la predicción se solicitará al usuario el número que ha escogido y su predicción. Entonces se calculará quién ha ganado o si no ha ganado nadie. El jugador podrá optar por continuar o finalizar el juego. Al finalizar, el ordenador calculará las estadísticas del juego y las mostrará.

A continuación se propone una posible traza del juego. La entrada del usuario está subrayada y en negrita:

```
> Hola campeón. Vamos a jugar.
> Voy a elegir un número entre 1-5 ... hummm ... ya lo tengo.
> Mi predicción es 9.
> Introduce tu elección (1-5): 3
> Introduce tu predicción (2-10): 8
> --- Yo he sacado 5 dedos y tú has sacado 3
> --- Yo he predicho 9 y tú 8
> Has ganado, :-(
> ¿Quieres volver a jugar? (s/n) s

> Voy a elegir un número entre 1-5... hummm... ya lo tengo.
> Mi predicción es 7.
> Introduce tu elección (1-5): 2
> Introduce tu predicción (2-10): 9
> --- Yo he sacado 4 dedos y tú has sacado 2
> --- Yo he predicho 7 y tú 9
> Empate... nadie ha ganado...
> ¿Quieres volver a jugar? (s/n) n

> --------------------------------
>          Estadísticas
>          IA: 0 partidas
>          Tú: 1 partida
>
>          TÚ GANAS
> --------------------------------
```

4. Programando en Python

Para instalar la plataforma Python es necesario acceder al www.python.org y seguir las instrucciones proporcionadas, aunque en las máquinas del laboratorio ya está instalado. Si trabajamos en sistemas Linux o macOS, lo habitual es que Python venga instalado por defecto.

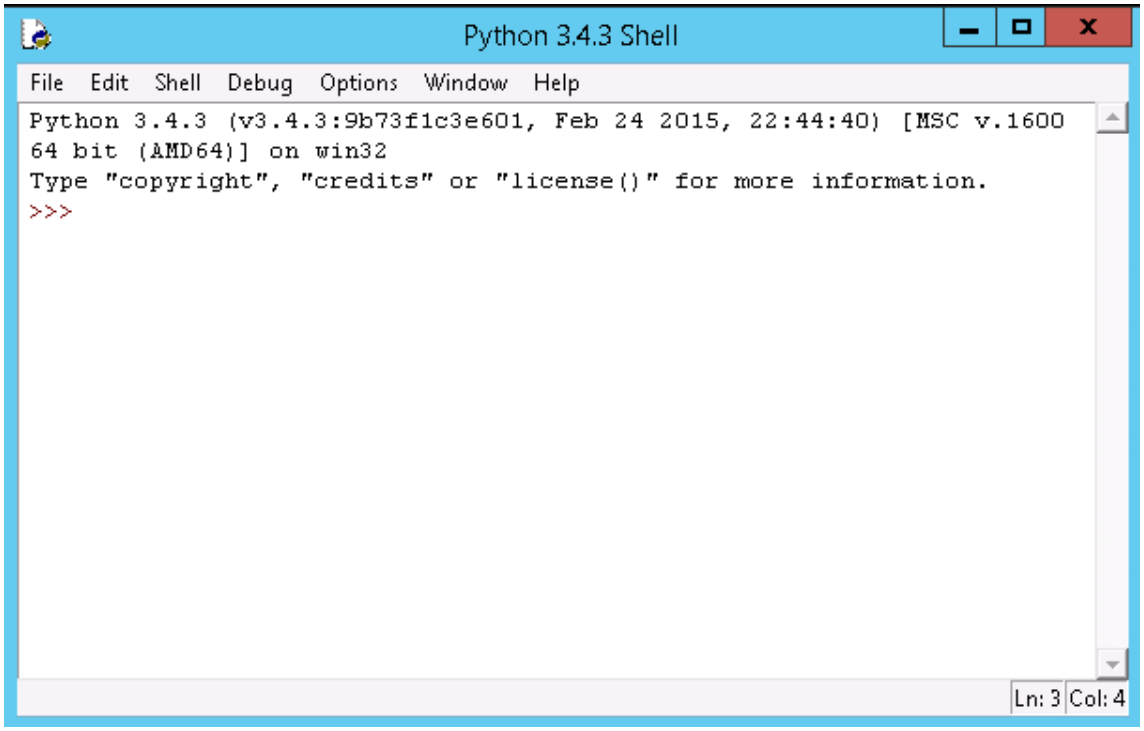
En esta práctica se trabajará con la versión 3.x de Python, que incorpora algunas novedades importantes con respecto a la versión 2.7.

Hay distintas posibilidades a la hora de interactuar con Python:

1. Ejecutar el intérprete interactivo (línea de comandos) de Python: en este caso se abrirá un terminal y se podrá interactuar de manera dinámica con Python escribiendo sentencias y ejecutándolas.
2. Desarrollar el código de los módulos de Python en cualquier editor de textos y posteriormente ejecutarlo usando el intérprete de Phthon en línea de comandos.
3. Usar un entorno de desarrollo ligero denominado IDLE y proporcionado por la instalación básica de Python.
4. Usar el entorno de desarrollo Thonny diseñado para principiantes. Thonny Es el entorno por defecto en Raspberry Pi: https://thonny.org

De momento se escogerá la opción 3, de forma que se ejecutará IDLE para abrir el entorno Python.

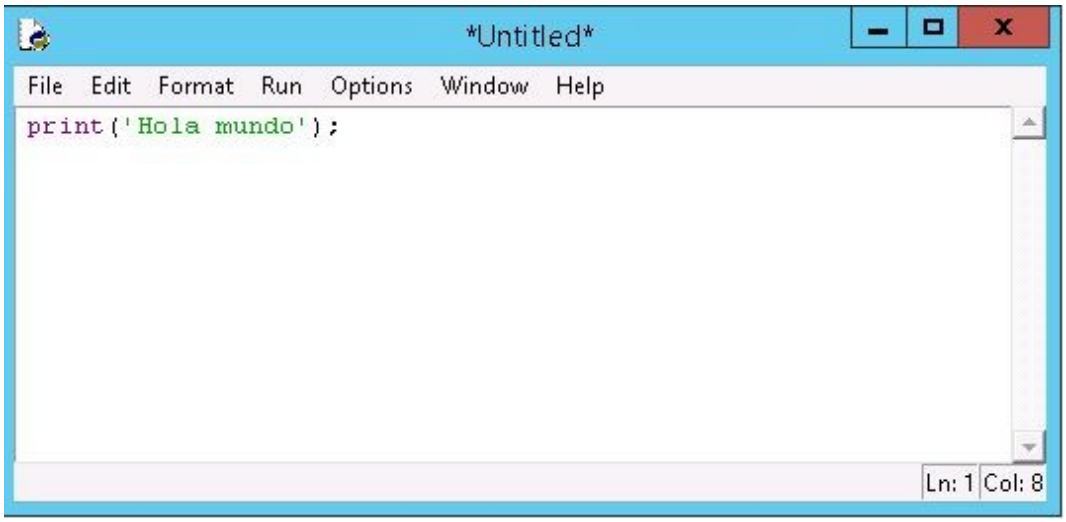
Si se selecciona IDLE, automáticamente se abrirá una ventana con el Shell de Python, que no es más que una versión del intérprete de Python, donde podemos evaluar cualquier comando de manera dinámica.



A continuación se va a ejecutar un comando muy simple:

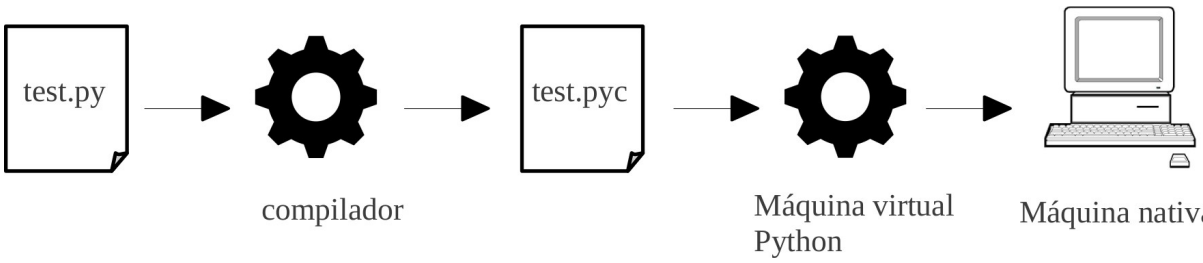
```
>>> print('Hola mundo!')
```

El resultado se mostrará inmediatamente. Desde esta ventana se pueden crear, editar y ejecutar ficheros fuentes '.py', que en Python se denominan módulos. A continuación se va a crear un módulo. Para ello, en el Shell hay que escoger la opción del menú *File* / *New File...* para abrir un sencillo editor de texto que colorea y sugiere sintaxis. Ahí se va a escribir la sentencia anterior:



Para ejecutar el código anterior hay que acceder a la opción del menú *Run* / *Run Module* (F5) guardando el fichero durante el proceso como 'test.py'. El módulo se ejecutará en el Shell y se podrá observar su salida.

Cuando se ejecuta un programa Python primero se compila a un lenguaje de bajo nivel no nativo (lenguaje intermedio), y después es interpretado por la máquina virtual de Python. A continuación se ilustra este proceso.



El archivo compilado '.pyc' no se genera para el módulo principal del programa Python, es decir, para el módulo en el que comienza la ejecución del programa, pero el proceso de traducción siempre sucede en memoria.

Una vez conocidas las bases del entorno de desarrollo se puede empezar a trabajar con Python.

El intérprete interactivo de Python puede resultar muy útil, ya que se evalúan comandos en tiempo real. A continuación se proponen algunas operaciones aritméticas:

```
>>> 2 + 2   # esto es un comentario
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
```

No es necesario declarar las variables antes de usarlas:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Los *strings* en Python se pueden definir con comillas simples o dobles:

```
>>> 'spam eggs' # single quotes
'spam eggs'
>>> "doesn't" # ...or use double quotes instead
"doesn't"
```

Estas son algunas características interesantes sobre *strings*:

- Se puede obtener la longitud de un *string* con la función `len()`
- Se puede acceder a los caracteres de un *string* indexándolo con un entero `[i]` (basado en 0).
- Se puede acceder a secciones del *string* usando un rango de índices `[i0:i1]`
- Se pueden concatenar *strings* usando el operador `+`
- Se pueden usar los operadores de relacionales: `==`, `!=`, `<`, `<=`, `>`, `>=`

Por ejemplo:

```
>>> word = 'Python'
>>> len(word)
6
>>> word[0]
'P'
>>> word[2:5]
'tho'
>>> 'hola ' + 'mundo'
'hola mundo'
>>> 'hola' == 'hola'
True
```

La función predefinida `print()` permite imprimir cualquier cosa por pantalla. Su sintaxis es: `print(expr1, expr2, ...)`

```
>>> s = 'First line.\nSecond line.'
>>> print(s, [2,3,4], "adios")
First line.
Second line. [2,3,4] adios
```

Python posee unos tipos de datos complejos muy útiles, como listas, tuplas y diccionarios. Las listas son colecciones ordenadas de valores que se definen con el literal [1, 2, 3 ...]. Soportan la operación `len()`, indexación, secciones, concatenación y operadores del mismo modo que los *strings*.

```
lst = [1,2,3,4]
>>> len(lst)
4
>>> lst[0]
1
>>> lst[1:3]
[2,3]
>>> lst + [5,6,7]
[1,2,3,4,5,6,7]
>>> lst == [1,2,3,4]
True
```

Como ya se ha visto, la salida por consola es muy simple usando la función `print()`, aunque en el intérprete cualquier expresión será evaluada y mostrada por consola. La entrada puede obtenerse usando la función `input()`, que devuelve un *string* tal y como se muestra a continuación:

```
>>> s = input()
hello
>>> print(s)
hello
>>> s = input('Enter message: ')
Enter message: hello
>>> print(s)
hello
```

Python también proporciona funciones de conversión de datos: `int()`, `float()`, `list()`, `str()`, ...

```
>>> int('110')
110
>>> str(200)
'200'
```

La bifurcación por excelencia en Python es la sentencia `if`. Acepta una condición (sin paréntesis) que puede ser cualquier valor aunque al final se transformará en `True` o `False` (`0`, `[]`, `{}`, `None` son `False`; cualquier otro valor es `True`). El comienzo de bloque se marca con ':' y todas las instrucciones con una indentación diferente pertenecerán a otro bloque, por tanto, no existe símbolo de fin de bloque.

```
s = input('Enter word: ')
if s == 'hello':
    print('hello :-)')
else:
    print(s)
```

Al principio puede resultar extraña esta manera de marcar el comienzo y el final de un bloque de instrucciones complejas, pero obliga a crear un código legible. En Python la indentación no es una cuestión de estilo, sino de sintaxis.

Lo mismo sucede con el bucle `while` tal y como se muestra a continuación:

```
while(True):
   s = input('Enter word: ')
   if s == 'hello':
       print('hello :-)')
   elif s == 'exit':
       print('bye bye :-(')
       break
   else:
       print(s)
```

Para más detalles sobre la sintaxis del lenguaje de programación Python es posible consultar el tutorial oficial, o bien la especificación de la sintaxis con BNF en las siguientes URLs, respectivamente:

- https://docs.python.org/3/tutorial/index.html
- https://docs.python.org/3/reference/index.html

Lo último que se aprenderá en esta práctica es la importación de módulos de terceros y la utilización de sus funciones. Python posee una potente librería estándar donde se puede encontrar casi cualquier cosa que se necesite. Es posible investigar sobre la misma en el siguiente URL:

- https://docs.python.org/3/library/index.html

En el siguiente ejemplo se importará el módulo `random`, que permite generar números aleatorios. Para importar un módulo basta con utilizar la sentencia `import` y hacer referencia a su nombre. Al importar un módulo Python crea un objeto con el mismo nombre que el módulo que contiene todas las características (funciones, variables, etc.) exportadas por aquél. Si se mira la documentación, el módulo `random` posee una función `randint()` que genera un número aleatorio comprendido en un rango. En el siguiente ejemplo se hace uso de la misma con el objetivo de generar una lista con 5 números aleatorios en el rango [1, 10].

```
import random
lst = []
i = 0
while i < 5:
    lst = lst + [random.randint(1,10)]
    i = i + 1
print(lst)
```

5. El entregable de Python

A continuación se propone desarrollar un pequeño juego en Python que el alumno debería ser capaz de crear con las habilidades adquiridas a lo largo de esta práctica.

Se trata de una versión reducida del MasterMind. Al comenzar la partida, el programa generará aleatoriamente una lista de 5 enteros aleatorios en el rango [1, 10]. El jugador deberá adivinarla. Para ello, efectuará distintos intentos listando en cada jugada una nueva lista con 5 enteros. El programa indicará los aciertos devolviendo, por ejemplo, una lista con unos y ceros, donde 1 indicará un acierto y 0 un fallo.

A continuación se propone una posible traza del juego. La entrada del usuario está subrayada y en negrita:

```
> Hola campeón. Vamos a jugar.
> Déjame que piense ... hummm ... ya he elegido la lista de números
> Adivínala: [1,2,3,4,5]
> Mi respuesta es: [0,1,0,0,0]
> Adivínala: [2,2,4,7,3]
> Mi respuesta es: [0,1,0,1,0]
> Adivínala: [3,2,6,7,5]
> Mi respuesta es: [1,1,0,1,0]
> Adivínala: [3,2,8,7,9]
> Mi respuesta es: [1,1,1,1,0]
> Adivínala: [3,2,8,7,8]
> Muy bien!!! Ganaste!!! Has usado 5 intentos.
```

Además, si el usuario introduce como entrada la palabra "respuestas", hay que mostrar cuáles son los números elegidos, a modo de depuración, y no contará como intento:

```
> Adivínala: respuestas
> Mi respuesta es: --- La solución es [4,7,5,10,2]
> Adivínala:
...
```

Como **restricción** adicional, no se permite usar la función `split()` para analizar el *string* que introduce el usuario.

Se plantea como extensión que el usuario pueda escoger un número máximo de jugadas. Si se utilizan todas ellas, entonces ganará la IA.