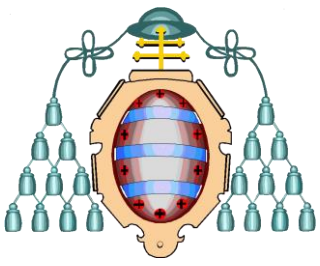
FUNDAMENTOS DE INFORMÁTICA

INTRODUCCIÓN A LA PROGRAMACIÓN

Introducción al entorno de desarrollo



Departamento de Informática
Universidad de Oviedo

ÍNDICE

1 Desarrollo de la práctica.

Si sigues esta práctica en la sesión de laboratorio, no necesitas ir leyendo los apartados siguientes, el profesor los irá explicando en la pantalla. Puedes ir directamente a las últimas páginas donde están los enunciados de los ejercicios que harás cuando el profesor lo indique.

El resto del texto es para quienes no puedan acudir a la sesión y lo quieran hacer por su cuenta.

2 Herramientas necesarias para programar en Python

Un programa en Python no es más que un archivo de texto, cuyo contenido son órdenes escritas en el lenguaje Python.

Para escribir este archivo se puede utilizar cualquier **editor de texto** que pueda guardar su contenido en "texto plano" (esto es, sin tipos de letra)[1]. Algunos editores válidos serían el bloc de notas de Windows, Notepad++ para Windows, textedit de Mac, gedit de Linux, etc.

Para ejecutar el programa se requiere el **intérprete Python**, que es a su vez otro programa que lee el archivo que hemos escrito y va ejecutando una línea cada vez. El intérprete también se puede usar en *nodo interactivo*, en el cual las líneas no se toman de un fichero, sino de lo que el usuario va escribiendo en ese momento, y tras cada línea evaluada muestra el resultado de su evaluación. Una buena noticia para los usuarios de Mac y de Linux es que sus operativos ya vienen con un intérprete de Python integrado. Si abres una terminal y escribes "python" entrarás en él. Los usuarios de Windows en cambio, necesitarán instalar uno.

El editor y el intérprete son las dos herramientas indispensables. Sólo con ellas ya se podría hacer la mayor parte de las prácticas de la asignatura. Sin embargo, si vas a usar Python seriamente en el futuro para resolver problemas prácticos reales (quizás relacionados con otras asignaturas), seguramente te resultarán muy útiles los **módulos** extras, que añaden muchas funciones a las que Python ya trae "de fábrica". Uno de estos módulos (NumPy) se utiliza frecuentemente en diversas materias relacionadas con la informática pues permite usar Python para el manejo de matrices y otras estructuras de datos similares.

Si bien cualquier editor de "texto plano" sirve para escribir programas en Python, resulta sin embargo recomendable disponer de un editor que proporcione funcionalidades específicas para este lenguaje. Por ejemplo, que coloree el listado según la sintaxis del mismo, que trate correctamente los espacios al inicio de la línea, que proporcione indentación (sangrado) automáticos, etc. Si además el editor es capaz de llamar al intérprete para ejecutar el programa, y proporciona capacidades para inspeccionar el programa mientras se ejecuta (depuración), estaríamos ya ante lo que se

[1] Debido a que Python utiliza los espacios al inicio de las líneas de una forma especial que ya veremos, algunos editores que cambian estos espacios por "tabuladores" podrían causar problemas.

denomina un Entorno de Desarrollo Integrado (IDE), que es una herramienta indispensable cuando el proyecto de programación adquiere ya una cierta envergadura.

Resumiendo, los componentes que necesitaremos serán:

- Editor
- Intérprete
- Módulos extra (esto es opcional)
- IDE con depurador integrado

2.1 Python portable

Si bien es posible descargarse Python[2] e instalarlo en la máquina como cualquier otro programa, esto requerirá de permisos de administrador.

También es posible instalar una versión de Python llamada "Python portable" que no requiere de tales privilegios, ya que no se instala "realmente". Simplemente se copia en una carpeta (que puede estar en una unidad externa, como un lápiz USB) y está especialmente diseñado para ir a buscar a la carpeta de instalación todo lo que necesite para funcionar. Además, esta solución incorpora no sólo el intérprete, sino también un gran número de módulos extra que pueden serte útiles en el futuro, y más de un IDE para elegir, incluyendo sus depuradores.

La gran ventaja de Python portable es que puedes llevarlo contigo a todas partes, en un lápiz USB, y dondequiera que tengas un ordenador en el que puedas enchufar tu lápiz, tendrás un Python listo para funcionar. La desventaja es que es más complejo instalar en él módulos extra nuevos que vayan saliendo.

Python portable puede descargarse de https://portablepython.com/wiki/Download/ , pero encontrarás que hay allí dos versiones para elegir. Ambas versiones, aunque algo antiguas, son estables y funcionan a la perfección. En nuestro caso emplearemos Python 3 ya que la versión 2 es a extinguir.

Pueden encontrarse versiones más modernas de Python Portable en el enlace https://sourceforge.net/projects/portable-python/files/ donde de nuevo elegiremos la versión 3 de Python y dentro de su carpeta elegiremos a su vez la versión de 32 bits o la de 64 bits dependiendo de que nuestro sistema operativo sea de 32 o de 64 bits. La de 32 bits va a funcionar correctamente en todos y es suficiente para esta asignatura, aunque casi todos los equipos modernos llevan ya un sistema operativo de 64 bits.

Si se va a usar Python portable en un USB es recomendable que éste sea USB 3.0, dado que con memorias USB 2.0 resulta un tanto lento al inicio.

Para una instalación permanente en nuestro equipo personal es mejor emplear las versiones estándar no portables tanto del intérprete Python como del entorno integrado de desarrollo PyScripter. El intérprete lo descargaremos de https://www.python.org/

[2] Desde http://www.python.org/getit/

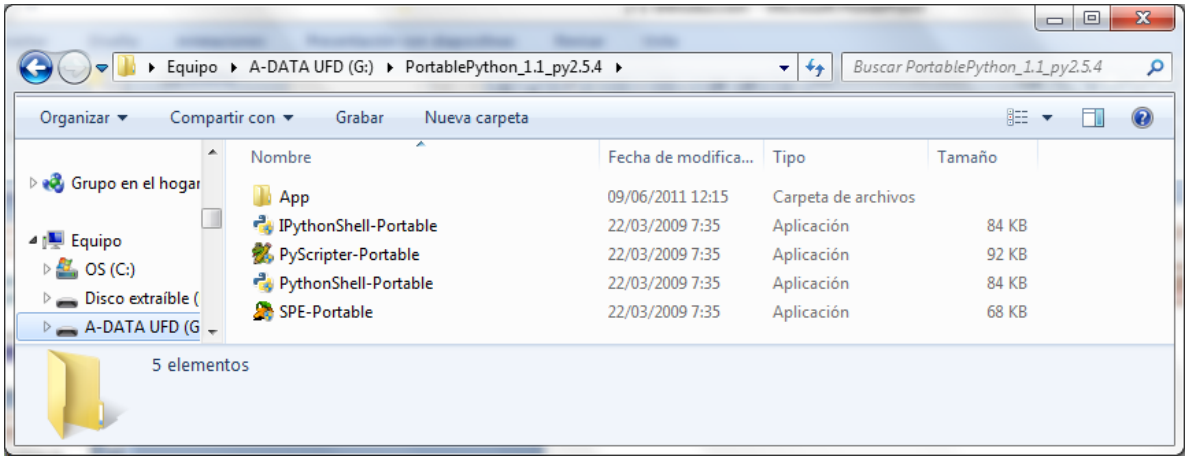
(vale cualquier versión que empiece por 3) y el entorno de desarrollo lo descargaremos de https://sourceforge.net/projects/pyscripter/

Muy importante: debemos bajar e instalar o bien la versión de 32 bits de Python y PyScripter o bien la de 64 bits de ambos si nuestro sistema operativo lo permite, no podemos hacer mezclas y bajar la de 64 bits de Python y la de 32 del PyScripter, no funcionará.

REALIZA AHORA EL EJERCICIO 1

3 Ejecutando el IDE pyscripter

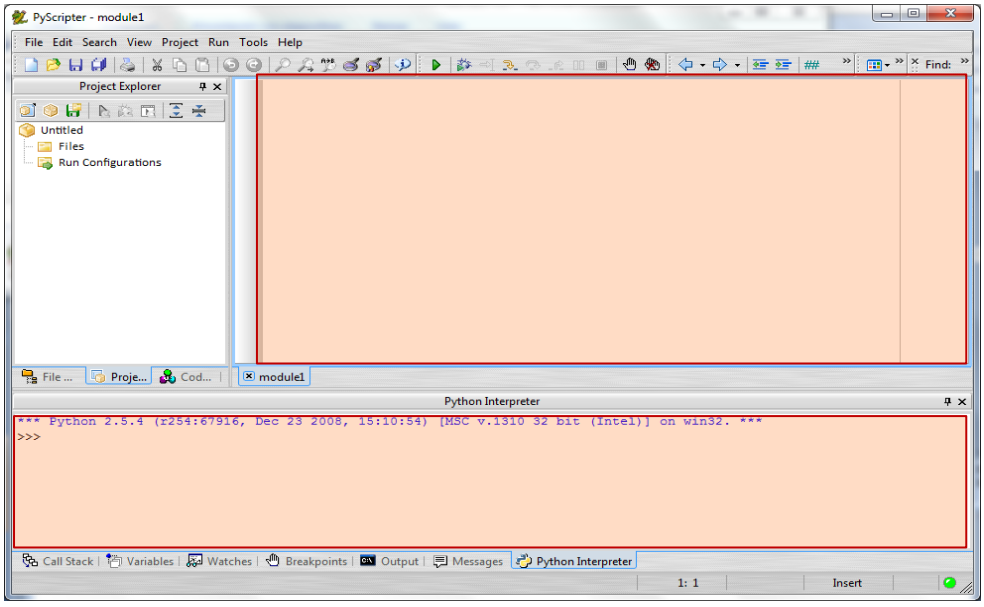
En la carpeta que acaba de crearse durante la instalación podrás ver algo como esto:



La carpeta App contiene todo lo que Python necesita para funcionar (el intérprete, las bibliotecas, los módulos extra, herramientas adicionales, documentación, etc.) No necesitarás entrar en ella.

Los restantes ficheros que ves son *interfaces* para interactuar con Python. En esta asignatura usaremos sólo el llamado PyScripter[3].

Haciendo doble-clic en PyScripter, se abre este IDE que presenta el siguiente aspecto:



Este interfaz tiene tres partes claramente diferenciadas:

[3] Un inconveniente de PyScripter es que sólo existe en Windows. Si en tu casa tienes otro operativo, deberás investigar otros IDEs apropiados para tu operativo. O bien, no usar ningún IDE, sino un editor de texto que te guste para escribir los programas, y el intérprete python desde la terminal para ejecutarlos.

- El navegador (a la izquierda del editor). Sirve para buscar ficheros, ya sea en el proyecto en que estás trabajando, o en el sistema de ficheros. No usaremos esta parte del IDE.
- El editor. Sirve para escribir programas en Python que puedan ser guardados en disco. Veremos que tiene *coloreado sintáctico* y ayuda automática al escribir código.
- El intérprete interactivo. En esta parte puedes escribir órdenes simples que son ejecutadas inmediatamente por el intérprete, quien muestra a continuación el resultado. Es perfecto para hacer pequeñas pruebas que no necesites guardar.

3.1 Uso del intérprete interactivo

Puedes comenzar a usar ya mismo este intérprete, incluso sin saber nada de programación, como si fuera una potente calculadora. Prueba a introducir cualquier expresión numérica y verás cómo te responde con el resultado.

Si quieres calcular potencias, necesitas saber que el operador es ** en lugar de ^ como en Excel. Prueba 2**100 y observa cómo da el resultado con todos sus dígitos.

REALIZA AHORA EL EJERCICIO 2

3.2 Más funciones de la “supercalculadora”

Puede operar con cadenas de caracteres, las cuales se escriben poniéndolas entre comillas. Puedes usar "comillas dobles" o 'comillas simples', como prefieras, para Python son lo mismo. Pero no las mezcles, si se abre con las de un tipo se debe cerrar con las del mismo.

- El operador + concatena: 'a' + 'b' sale 'ab'
- El operador * repite la cadena: 'a' * 3 sale 'aaa'
- La función len() te dice cuántas letras tiene la cadena: len('hola') sale 4
- Puedes comparar cadenas para ver si son iguales (==) o cuál es mayor alfabéticamente.

Python codifica sus cadenas en Unicode (que coincide en parte con ASCII). Proporciona por tanto funciones para obtener el código Unicode de una letra, o para obtener qué letra es un código dado:

- `chr(65)` sale 'A'
- `ord('A')` sale 65

También tiene funciones para interpretar una cadena de caracteres, entendida como si fuera un número escrito en una cierta base, y devolvernos qué número es. Si no especificamos la base, supone 10.

- `int('1B', 16)` sale 27 (¿podrías explicar por qué?)

- `int('27')` sale 27 (puede parecer algo inútil, pero más adelante comprenderemos su utilidad).

Puede convertir un número dado a hexadecimal. El número puede ser el resultado de una expresión. El resultado es una cadena de caracteres, delante de la cual siempre añade los caracteres '0x' para indicar que está en hexadecimal.

- `hex(2**8)` sale '0x100'

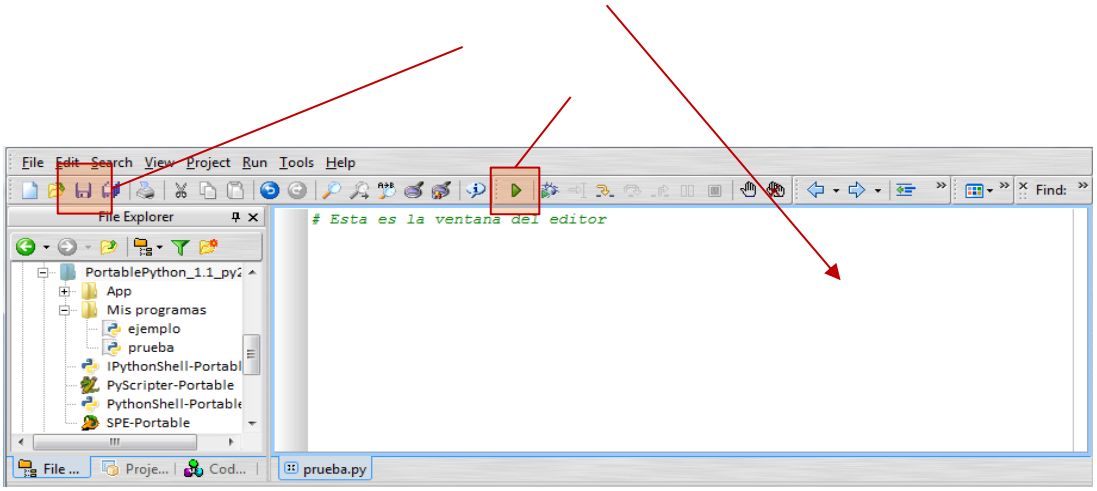
Otras funciones que encontrarás útiles:

- `max(), min()`: encuentran el mayor (o menor) de una serie de números
- `float()`: convierte un entero dado en el mismo número, pero representado en coma flotante. Esto afecta, como se verá, a la división.
- `round(real, n)`: redondea el número real dado a n decimales. Se puede omitir n en cuyo caso redondea al entero más próximo (aunque el resultado seguirá siendo float).
- `type(dato)`: te dice de qué tipo es ese dato (o el resultado de la expresión)
- `help(x)`: te da ayuda sobre x, que puede ser una función, un tipo de dato, una variable, un módulo que hayas importado, etc.

REALIZA AHORA EL EJERCICIO 3

3.3 El editor

El panel del editor te permite escribir código Python, ejecutarlo, y guardarlo en el disco para poder cargarlo de nuevo en otra sesión. Es la forma habitual de desarrollar programas en Python. El intérprete interactivo es sólo adecuado para pequeñas pruebas con fragmentos muy cortos de código.



La línea que comienza por # es un *comentario*. Todo lo que aparece tras un signo # es ignorado por el intérprete. Prueba a escribir el código anterior y ejecútalo (pulsando en el triángulo verde). No ocurre nada, porque ese código no contiene líneas ejecutables.

Prueba a introducir en el editor el código siguiente:

```
# Calculamos cuántos bytes son 1 GiB y 1 GB
2**30
10**9
```

Si lo ejecutas, tampoco ocurre aparentemente nada. En realidad, Python ha calculado esas expresiones, pero a diferencia del intérprete interactivo, cuando está ejecutando código de un fichero, no va mostrando automáticamente el resultado de cada expresión que evalúa, a menos que se le pida. Ya que en este caso no se le pide, los resultados no se ven, aunque realmente han sido calculados.

El resultado de estas expresiones puede ser guardado en *variables.* El concepto se explicará en la siguiente sesión. De momento valga decir que son una zona de memoria a la que ponemos un nombre.

Intenta ahora el siguiente código que usa variables para almacenar los resultados:

```
# Calculamos cuántos bytes son 1 GiB y 1 GB
GiB = 2**30
GB = 10**9
```

Al ejecutarlo, de nuevo no verás ningún resultado, pues este programa aún sigue sin pedirle a Python que muestre nada. Sin embargo, las variables GiB y GB han quedado definidas, y desde la ventana del intérprete podrás ver su valor, sin más que escribir sus nombres. Inténtalo.

Por último, veamos cómo pedirle a Python que muestre un resultado en pantalla. Aunque esto se verá en detalle en la sesión sobre Entrada/Salida por consola, de momento baste decir que la función print(expresión), mostrará en pantalla el resultado de evaluar esa expresión (PyScripter lo mostrará en el panel del intérprete). Prueba lo siguiente:

```
# Calculamos y mostramos cuántos bytes son 1 GiB y 1 GB
print(2**30)
print(10**9)
```

REALIZA AHORA EL EJERCICIO 4

4 Errores

Errar es humano, y la programación la hacen personas. Los errores en programación se denominan "gazapos ("bugs" en inglés), y encontrar esos errores se llama "depuración" ("debugging" en inglés). Podemos catalogar los errores en tres tipos, por orden creciente de su dificultad de detección.

4.1 Errores de sintaxis

Todos los lenguajes de programación tienen unas reglas de sintaxis. Por ejemplo, un número entero válido es una secuencia de los dígitos 0 a 9, como 88212. Si apareciera en medio una letra, como en 882I2 se estaría violando esta regla. Se trataría de un error de sintaxis.

Los errores de sintaxis son fáciles de encontrar porque, antes siquiera de ejecutar nuestro código, Python le echa un vistazo general en busca de este tipo de errores y, si encuentra alguno, nos lo señala sin llegar a ejecutar ni una línea del código.

Por ejemplo, introduce el siguiente código en el editor, en el que se ha cometido el error de sintaxis de poner `@` en lugar de `*` (y resulta que `@` no es una operación válida en Python):

```
# Calcular el área de un triángulo
# de 30cm de base y 50cm de altura
prueba = 0
print(30 @ 50 / 2)
```

Al intentar ejecutar este código, Python detectará el error y nos lo señalará. En la ventana del intérprete puedes intentar ver el valor de la variable `prueba`, y descubrirás que esa variable no existe. Esta es la demostración de que, aunque el error de sintaxis estaba en la última línea del programa, no llegó a ejecutar ninguna de las anteriores (pues de lo contrario la variable `prueba` existiría con el valor 0).

4.2 Errores en tiempo de ejecución.

Aún si nuestro programa tiene toda su sintaxis correcta, puede contener líneas que no sea posible ejecutar. Por ejemplo, una línea que intente realizar una división por cero. O una línea donde se intente usar una variable que no ha recibido ningún valor.

En este caso Python no detectará el problema hasta que no llegue a ejecutar esa línea. En ese momento detiene su ejecución y nos señala el problema. Por tanto, cuando el error aparece, se habrán ejecutado todas las anteriores hasta ella.

Ejemplo:

```
# Calcular el área de un triángulo
# de 30cm de base y 50cm de altura
prueba = 0
print(30 * 50 / dos)
```

En este caso, la expresión `30 * 50 / dos` es sintácticamente correcta, ya que `dos` podría ser el nombre de una variable. Pero en este caso, tal variable no está definida (no se ha ejecutado antes una línea que haya puesto un valor en `dos`), por tanto, al llegar a esa expresión se produce un error en tiempo de ejecución.

Comprueba en el intérprete que la variable `prueba` sí está definida, lo que demuestra que la línea anterior se había ejecutado.

Modifica el programa para añadirle una línea que ponga: `dos = 2`, y comprueba cómo entonces desaparece el error en tiempo de ejecución (naturalmente tal línea debes añadirla antes, no después, de la que usa `dos` en la expresión).

4.3 Errores semánticos

El tercer tipo de error, y el más difícil de detectar, es el error semántico, que consiste en que hemos escrito un programa perfecto, salvo que no hace lo que queríamos. Por ejemplo, si en lugar de la fórmula para el área del triángulo metemos otra fórmula (bien construida, pero que no calcula el área de un triángulo). Por ejemplo:

```
# Calcular el área de un triángulo
# de 30cm de base y 50cm de altura
prueba = 0
print(30 * 50 / 5)
```

En lugar de dividir por 2 hemos dividido por 5. Naturalmente Python no puede saber lo que pretendíamos, por lo que esto no es un error para él. Sólo detectaremos el problema si sabemos qué resultado debería dar, y observamos que no sale eso.

En este caso lo teníamos fácil, pero en general esto plantea una interesante cuestión. Si ya sabíamos que resultado debía dar ¿para qué hicimos un programa que lo calcula? Y si no lo sabíamos ¿cómo podremos detectar que ha salido mal?

La respuesta a esta aparente paradoja es que, si bien generalmente no conocemos el resultado final (¡para eso escribimos programas!) para cualquier caso posible, sí que se suele conocer qué resultado debería producir para algunos casos particulares (por ejemplo, para un triángulo cuya base o altura es cero, o para uno en que ambos valores son 1). Si probamos varios casos "conocidos" y en todos obtenemos la respuesta correcta, podemos suponer que será correcta también en los restantes casos. Naturalmente puede que no sea así.

Si damos con un caso en el que la respuesta no es correcta, habrá que encontrar en qué punto del programa hemos cometido el error. El método general es usar un *depurador* que permite ejecutar una a una las instrucciones del programa y ver qué sale en cada una de ellas. De nuevo se supone que conocemos lo que debería salir en cada línea, y cuando encontramos una en la que no se obtiene lo esperado, habremos localizado el error. O uno de ellos.

Se entiende que la depuración es la parte más compleja de la programación. De hecho, muchas personas consideran que programar *es depurar*. O sea, corregir un programa hasta que produzca el resultado correcto.

REALIZA AHORA EL EJERCICIO 5

REALIZA DESPUÉS EL EJERCICIO OPCIONAL 6

Ejercicios

►EJERCICIO 1. DESCARGA E INSTALA PYTHON

No tiene mayor dificultad. Descarga de la web el Python portable en su versión 3 y ejecuta el archivo que has descargado.

Te pedirá una carpeta en la cual instalarlo. Si has traído un lápiz USB (deberías haber traído uno), elige la carpeta raíz de esa unidad. Si no lo has traído, puedes instalarlo en cualquier carpeta de tu perfil (el Escritorio, Mis Documentos, etc.)

Verás que en la carpeta que hayas elegido, te crea otra`.` Esta carpeta puedes copiarla a cualquier otro lugar (incluso a otra unidad) y seguirá funcionando. Si no has traído hoy el lápiz USB, tráelo otro día y copia en él esta carpeta.

►EJERCICIO 2. CÁLCULOS

Calcula cuál es la diferencia (en bytes) entre 1 KiB y 1 KB (recuerda que las unidades binarias, son las que llevan una "i" en su abreviatura, y son potencias de 2, mientras que las del Sistema Internacional no llevan la "i" y son potencias de 10).

Realiza el mismo cálculo con EXCEL ¿sale lo mismo?

Completa la tabla siguiente (recordatorio: cada unidad es el resultado de multiplicar la anterior por 1 Ki o 1 K respectivamente).

Diferencia	**Resultado Python**	**Resultado EXCEL**
1 KiB – 1 KB		
1 MiB – 1 MB		
1 GiB – 1 GB		
1 TiB – 1 TB		

►EJERCICIO 3. USANDO LA CALCULADORA DEL INTÉRPRETE

1. Encuentra cuál el valor del mayor de estos números: 2^{15}, 3^{12} o 5^{10}.
2. Usando las funciones `chr` y `ord`, encuentra cuál es la letra que está 10 posiciones más adelante de la A
3. Usando las funciones `chr` y `hex` (y tu cabeza) encuentra cuál es el código binario de la letra 'a' y de la letra 'A' y verifica que se diferencian tan solo en un bit
4. Usando las funciones vistas en la práctica, encuentra cuántos dígitos tendría el número de tu NIF si lo expresaras en hexadecimal.
5. Usa la función apropiada para encontrar qué número decimal es el que se codifica en binario como 110011001100
6. La función `int()` aplicada sobre un número real, elimina su parte decimal y lo deja en entero. Comprueba, por ejemplo, que `int(3.14)` sale 3

7. La función `round()` en cambio, mantiene el resultado como real, incluso si la parte decimal es cero. Comprueba que `round(3.14)` sale 3.0
8. Observa la diferencia entre el resultado de `int(2.8)` y `round(2.8)`
9. Comprueba que `type(int(3.14))` y `type(round(3.14))` producen el resultado esperado.
10. ¿Qué resultado produce `type("Hola")`?
    ¿Y cuál producirá `type(hex(100))`? Compruébalo.

► EJERCICIO 4. USANDO EL EDITOR

1. Intenta combinar los dos ejemplos anteriores, haciendo que desde el editor se almacene el resultado en dos variables, y mostrando dichas variables desde el programa.
2. Comprueba desde la ventana del intérprete interactivo que puedes trabajar con las variables `GiB` y `GB` como si fueran cualquier otro dato. Por ejemplo, prueba `hex(GiB)` o bien la expresión booleana `GiB > GB`

► EJERCICIO 5. ERRORES

1. En el campus virtual tienes un fichero llamado `problema.py`. Lee el enunciado (que aparece en el comentario) y el código que intenta solucionar el problema. ¿Cuántos errores tiene? ¿De qué tipo es cada uno? Corrígelos hasta que funcione como se espera (el resultado debería ser 65.45 horas)

► EJERCICIO 6. PROBLEMAS ADICIONALES OPCIONALES

Usando sólo las funciones explicadas en esta práctica (sin usar otros operadores ni sentencias de control):

1. Escribe una expresión que calcule el resto de dividir 500 entre 7. Comprueba con el intérprete que sale 3
2. Escribe en el editor un programa que haga salir en pantalla una línea formada por 80 asteriscos.
3. Busca en internet la fórmula para pasar grados Fahrenheit a Celsius y calcula cuántos grados Celsius son 45ºF, redondeado a tres decimales. La respuesta correcta es 7.222
4. Calcula la raíz cuadrada de 2 con cinco decimales.
5. ¿Qué crees que debería salir al poner `type(1/2)`? Comprueba qué sale. Comprueba también el resultado de la operación. ¿Y con `type(1//2)`?
6. ¿Cuál es la diferencia entre el operador `/` y el `//`?
7. Prepara una variable `n` con el valor 5. Calcular la raíz n-sima de 2. Debe salir `1.1486983549970351`