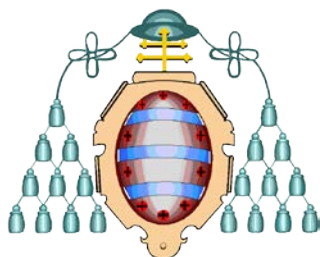


FUNDAMENTOS DE INFORMÁTICA

INTRODUCCIÓN A LA PROGRAMACIÓN

**Introducción al entorno
de desarrollo
(PyCharm Edu)**



*Departamento de Informática
Universidad de Oviedo*



ÍNDICE

1	DESARROLLO DE LA PRÁCTICA	3
2	HERRAMIENTAS NECESARIAS PARA PROGRAMAR EN PYTHON	3
2.1	IDE PYCHARM EDU	4
2.1.1	Instalación de PyCharm Edu.....	4
3	TRABAJANDO CON EL IDE PYCHARM EDU	6
3.1	PRIMERA EJECUCIÓN	6
3.2	LOS PROYECTOS DE PYCHARM.....	7
3.2.1	Creación de un proyecto	7
3.2.2	Creación de archivos Python.....	10
3.3	INTERFAZ DE USUARIO DE PYCHARM	12
3.4	USO DE LA CONSOLA DE PYTHON.....	13
3.5	MÁS FUNCIONES DE LA “SUPERCALCULADORA”	14
3.6	EL EDITOR	15
3.7	EJECUCIÓN DE PROGRAMAS	17
4	ERRORES	18
4.1	ERRORES DE SINTAXIS	18
4.2	ERRORES EN TIEMPO DE EJECUCIÓN.....	19
4.3	ERRORES SEMÁNTICOS.....	20
4.4	DEPURACIÓN DE PROGRAMAS	21
	EJERCICIOS	23



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 *modo 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”.

Si bien cualquier editor de “texto plano” sirve para escribir programas en Python, sin embargo, resulta 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 sangrado automático, 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 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:

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.



- Editor
- Intérprete
- IDE (con depurador)

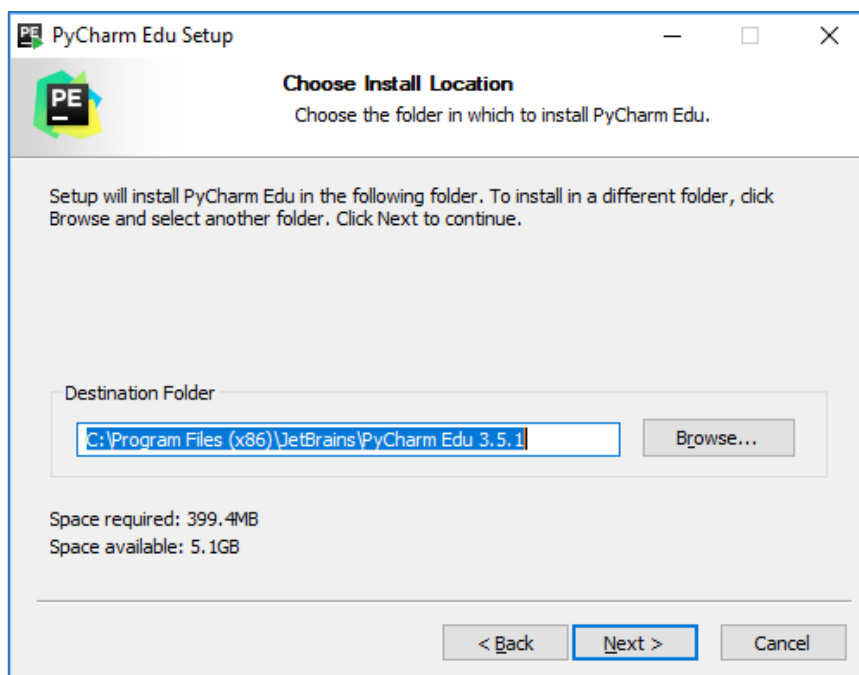
2.1 IDE PyCharm Edu

Se ha buscado un IDE que estuviera disponible para cualquier plataforma (Windows, Linux o Mac). Así, tras analizar varios IDEs disponibles para las tres plataformas, se ha optado por utilizar el IDE **PyCharm Edu** (*Educational Edition*). En el momento de hacer estos apuntes la versión disponible para es la 3.5.1. Puede descargarse de: <https://www.jetbrains.com/pycharm-edu/download/>. Esta edición es una versión especialmente diseñada para introducirse en la programación con Python. Existen otras ediciones de PyCharm con características más avanzadas que podrían ser útiles para aquellos alumnos que profundicen en la programación con Python.

PyCharm es un IDE que permite programar con cualquier versión de Python. En la actualidad coexisten dos versiones de este lenguaje: Python 2 y Python 3. Son muy similares, pero contienen algunas diferencias que hacen que los programas de una versión no funcionen directamente en la otra. En esta asignatura se va a explicar **Python 3**, por tanto, debes instalar también una versión de Python 3 en tu ordenador. PyCharm Edu instala automáticamente la versión de Python que indiquemos. Por si alguno quisiera instalar otra versión diferente de Python, todas las versiones están disponibles en <https://www.python.org/downloads/>.

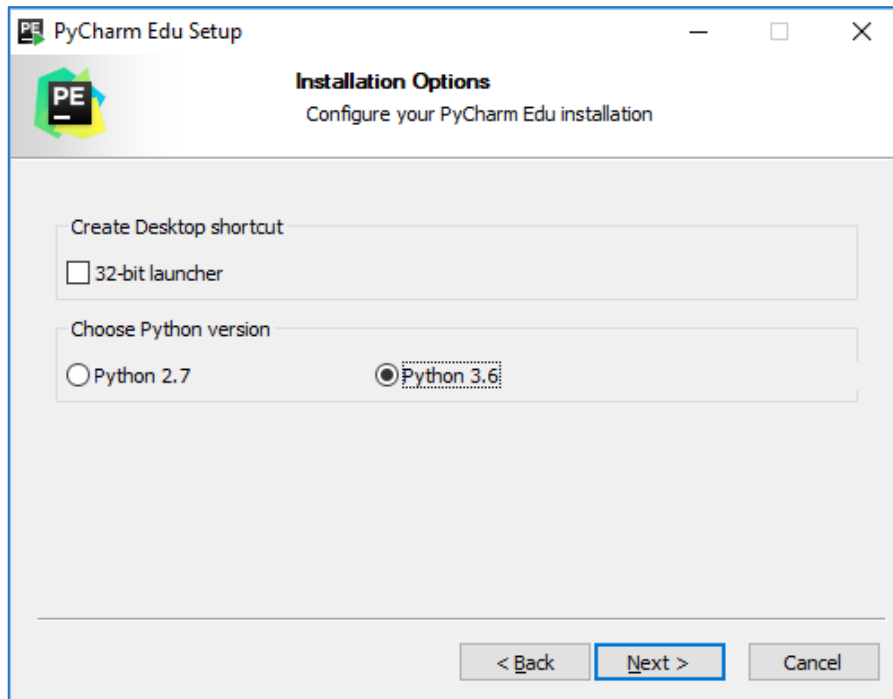
2.1.1 Instalación de PyCharm Edu

Los pasos que se detallan en este punto corresponde a una instalación en Windows 10. Una vez descargado el archivo de instalación (**pycharm-edu-3.5.1.exe**), lo ejecutamos y comienza la instalación el IDE en nuestro ordenador. Lo primero que nos pide es la ruta al directorio donde se guardarán los archivos de la aplicación:

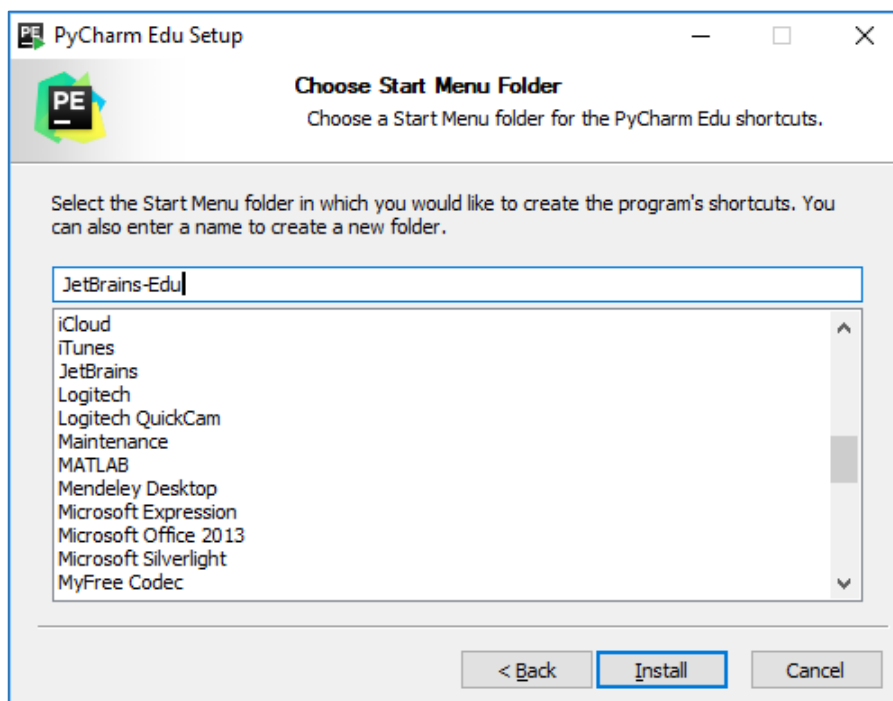




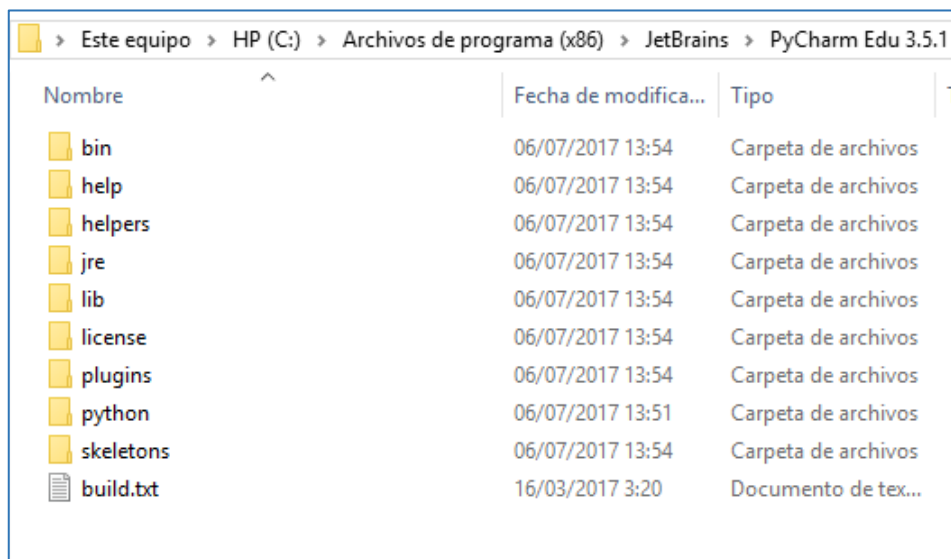
A continuación, nos pregunta si queremos crear un acceso directo en el escritorio y nos pide la versión de Python que queremos utilizar. Escogemos Python 3.6:



Y por último nos solicita el nombre de la carpeta del Menú Inicio:

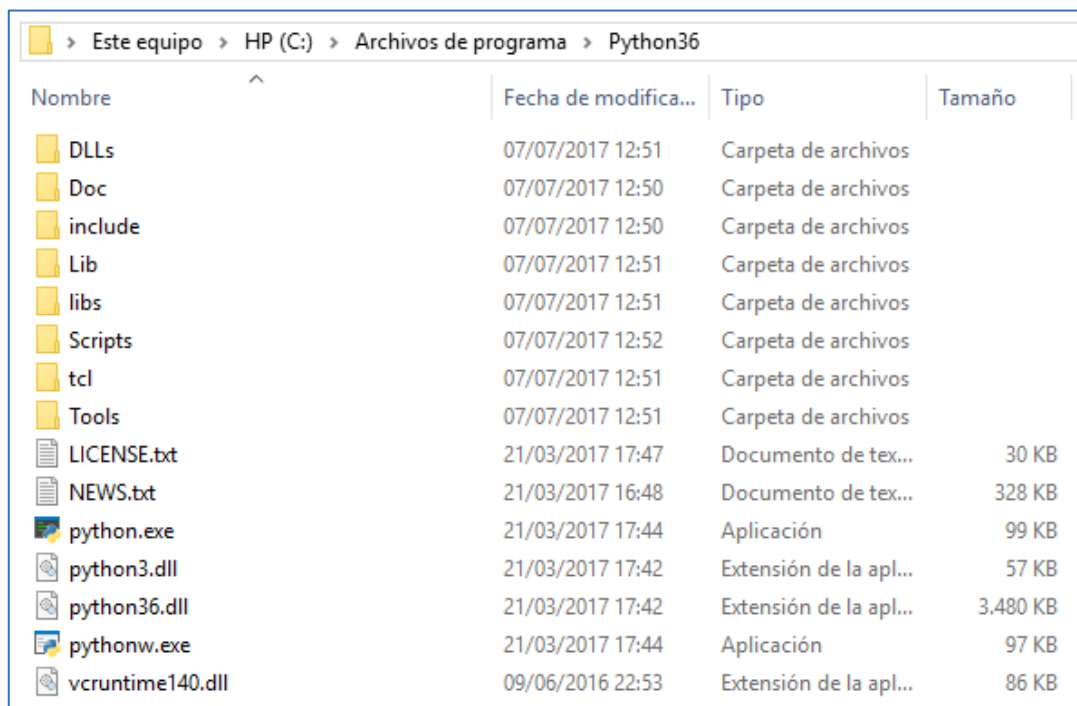


Una vez completada la instalación el directorio de la aplicación tendrá un aspecto similar al que se muestra en la siguiente imagen:



Nombre	Fecha de modifica...	Tipo
bin	06/07/2017 13:54	Carpeta de archivos
help	06/07/2017 13:54	Carpeta de archivos
helpers	06/07/2017 13:54	Carpeta de archivos
jre	06/07/2017 13:54	Carpeta de archivos
lib	06/07/2017 13:54	Carpeta de archivos
license	06/07/2017 13:54	Carpeta de archivos
plugins	06/07/2017 13:54	Carpeta de archivos
python	06/07/2017 13:51	Carpeta de archivos
skeletons	06/07/2017 13:54	Carpeta de archivos
build.txt	16/03/2017 3:20	Documento de tex...

Además, también se habrá instalado Python 3 en esta otra carpeta:

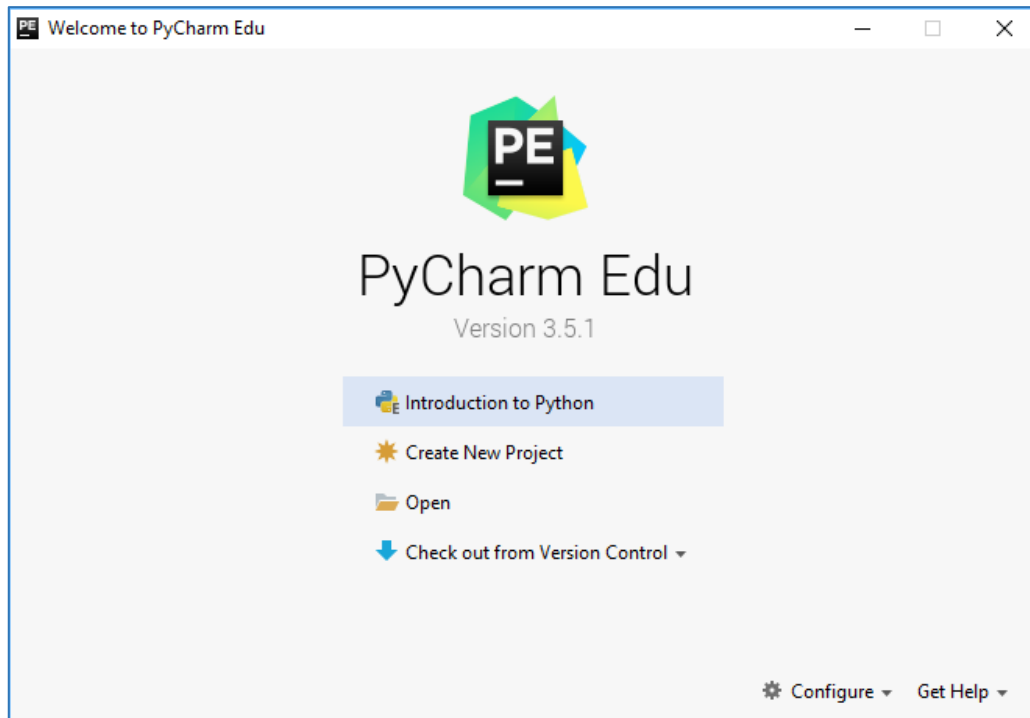


Nombre	Fecha de modifica...	Tipo	Tamaño
DLLs	07/07/2017 12:51	Carpeta de archivos	
Doc	07/07/2017 12:50	Carpeta de archivos	
include	07/07/2017 12:50	Carpeta de archivos	
Lib	07/07/2017 12:51	Carpeta de archivos	
libs	07/07/2017 12:51	Carpeta de archivos	
Scripts	07/07/2017 12:52	Carpeta de archivos	
tcl	07/07/2017 12:51	Carpeta de archivos	
Tools	07/07/2017 12:51	Carpeta de archivos	
LICENSE.txt	21/03/2017 17:47	Documento de tex...	30 KB
NEWS.txt	21/03/2017 16:48	Documento de tex...	328 KB
python.exe	21/03/2017 17:44	Aplicación	99 KB
python3.dll	21/03/2017 17:42	Extensión de la apl...	57 KB
python36.dll	21/03/2017 17:42	Extensión de la apl...	3.480 KB
pythonw.exe	21/03/2017 17:44	Aplicación	97 KB
vcruntime140.dll	09/06/2016 22:53	Extensión de la apl...	86 KB

3 Trabajando con el IDE PyCharm Edu

3.1 Primera ejecución

La primera vez que se ejecuta PyCharm Edu, bien a través del enlace directo en el escritorio de Windows o a través del Menú Inicio, aparece la siguiente “Ventana de bienvenida”:



PyCharm trabaja en base a proyectos (ver apartado siguiente). La edición de Educación viene con un proyecto llamado “*Introduction to Python*” que podemos abrir escogiendo la primera opción de la ventana anterior. La segunda opción permite “crear un proyecto nuevo” y la tercera “abrir un proyecto” guardado en disco.

3.2 Los proyectos de PyCharm

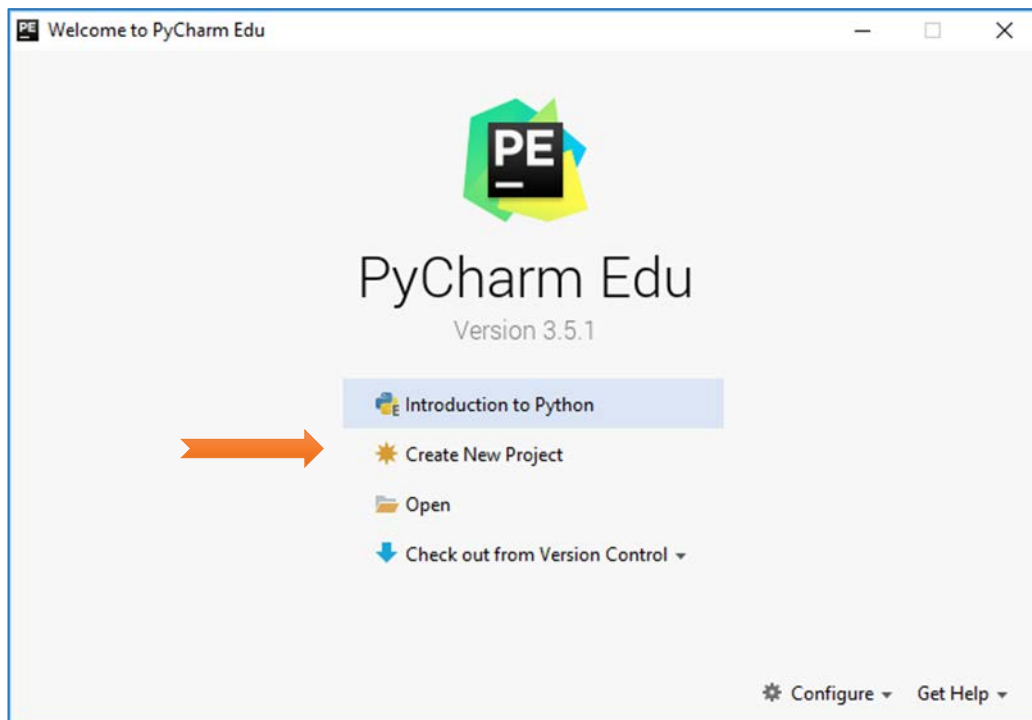
Como ya hemos dicho, el entorno de programación PyCharm trabaja en base a proyectos. Todo programa en PyCharm requiere la **creación de un proyecto**. Los proyectos permiten organizar mejor el código que creamos, especialmente cuando este empieza a crecer en tamaño y en cantidad de archivos.

La manera ideal de trabajar durante el curso será creando un proyecto para cada sesión de prácticas. Los proyectos constarán de un archivo de código Python por cada programa que se desarrolle.

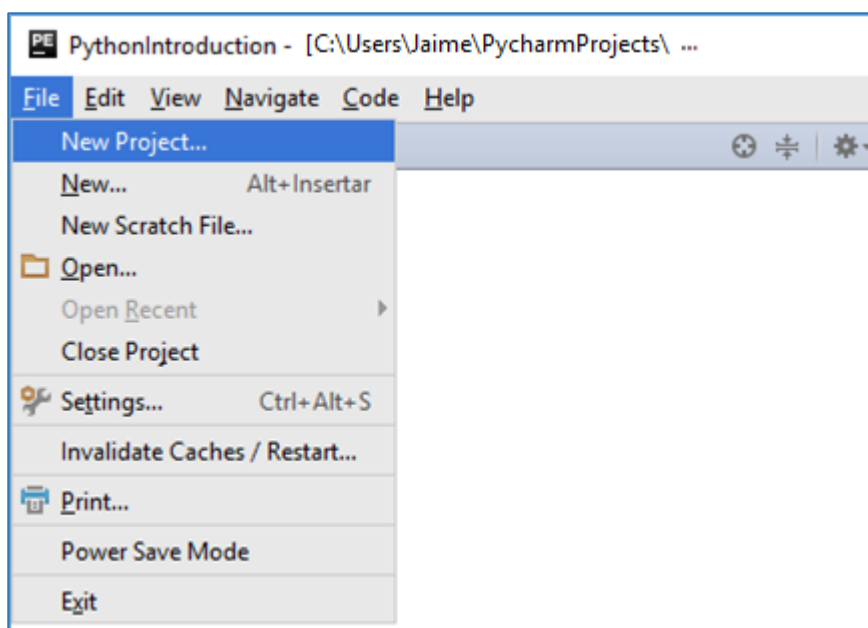
Se puede acceder a un video de introducción a PyCharm en Youtube a través del enlace siguiente: <https://youtu.be/ztXR9tP1KVc>

3.2.1 Creación de un proyecto

Podemos crear un proyecto nuevo desde la “*Pantalla de bienvenida*” o desde el *menú* de la aplicación. En la “*Pantalla de bienvenida*” escogeremos la opción “*Create New Project*”:

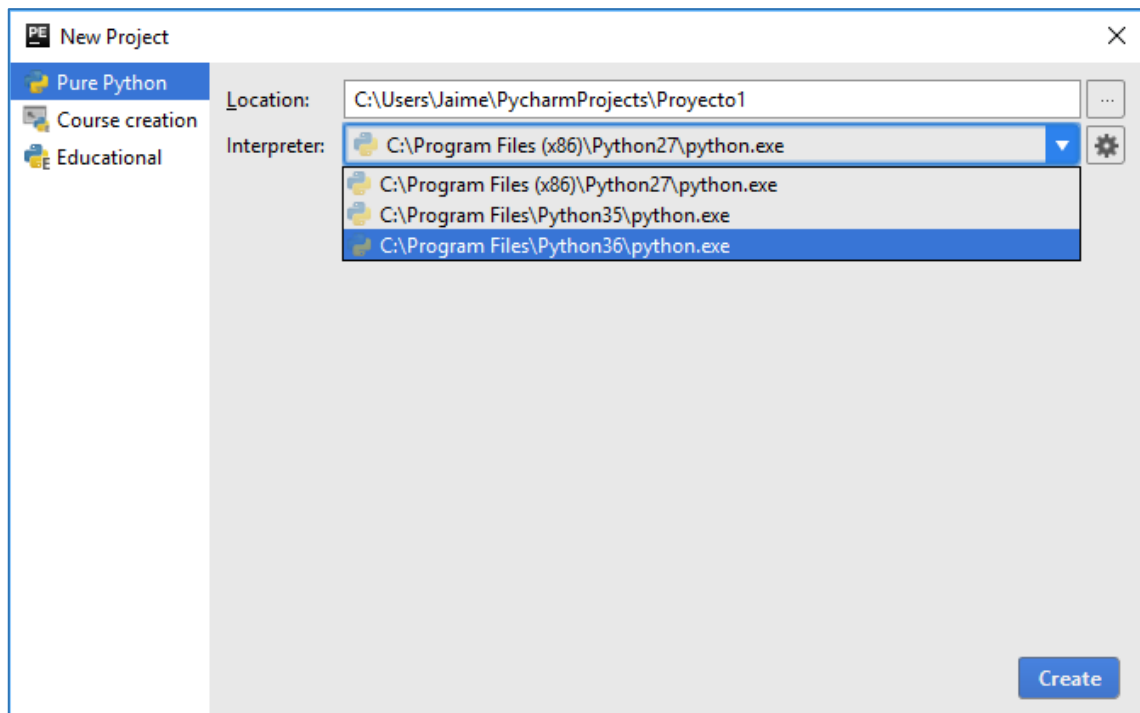


En el *menú* de la aplicación seleccionaremos “*File + New Project*”:

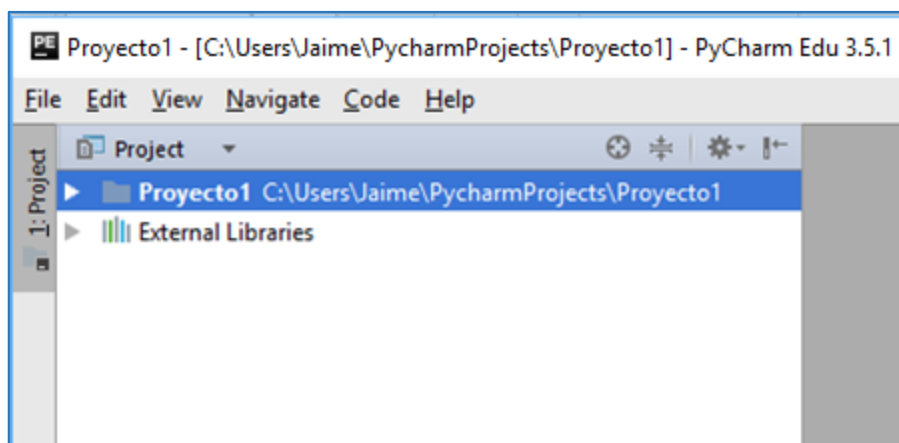
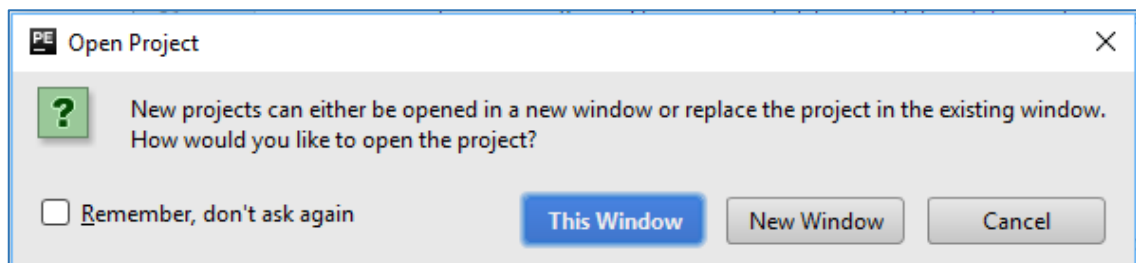


En ambos casos, después de escoger la opción de “Crear Proyecto Nuevo” sale una ventana en la que tenemos que elegir:

- I. El tipo del proyecto. Seleccionar **Pure Python**.
- II. El nombre del proyecto. Escribir un nombre (**Proyecto1** en el ejemplo)
- III. La versión del intérprete Python. Seleccionar **Python 3.6**. (PyCharm permite trabajar con distintas versiones de Python instaladas en el mismo ordenador).



Cuando creamos un proyecto nuevo, este se puede abrir en una ventana nueva del *PyCharm* o añadirse a la ventana que ya tenemos abierta. En el primer caso, se abre otra instancia de *PyCharm* y se trabaja con los dos proyectos independientemente. En el segundo caso, se abre el proyecto nuevo en la instancia actual de *PyCharm* y se cierra el proyecto con el que se estaba trabajando (la versión *PyCharm Community* permite tener varios proyectos abierto a la vez en la misma instancia).

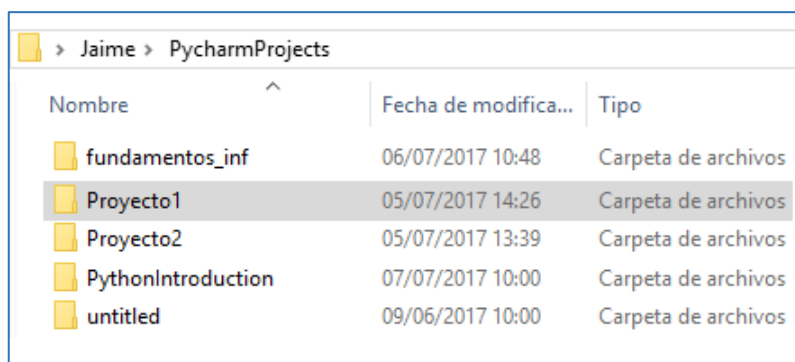


Cada proyecto va asociado a un directorio en el espacio de almacenamiento. En este directorio se guardan todos los archivos que forman parte del proyecto. Los directorios de los proyectos se crean por defecto en `C:\Users\usuario\PycharmProjects` (siendo *usuario* el *uoxxxxxx* de cada alumno). El proyecto y el directorio reciben inicialmente el mismo nombre. El nombre del proyecto y del directorio se podrían cambiar posteriormente si fuese necesario (se verá más adelante).

En las ventanas de ejemplo de este tutorial el nombre de usuario es **Jaime**. Por este motivo, el ejemplo de la imagen crearía un directorio llamado:

`C:\Users\Jaime\PycharmProjects\Proyecto1`.

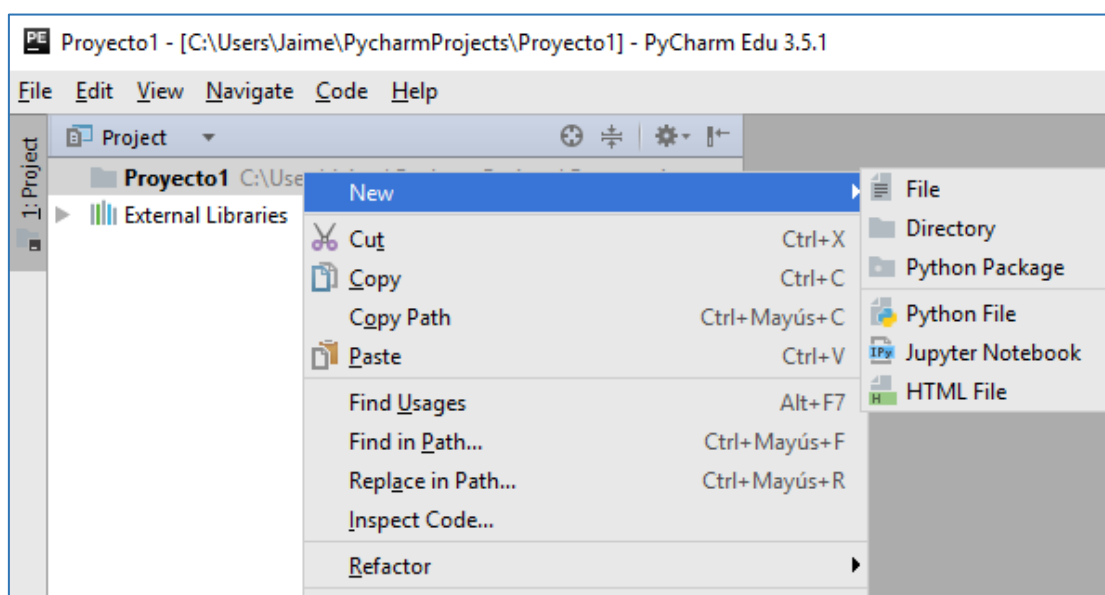
La ventana siguiente muestra el directorio de proyectos de PyCharm para el usuario **Jaime**:



Jaime > PycharmProjects		
Nombre	Fecha de modifica...	Tipo
fundamentos_inf	06/07/2017 10:48	Carpeta de archivos
Proyecto1	05/07/2017 14:26	Carpeta de archivos
Proyecto2	05/07/2017 13:39	Carpeta de archivos
PythonIntroduction	07/07/2017 10:00	Carpeta de archivos
untitled	09/06/2017 10:00	Carpeta de archivos

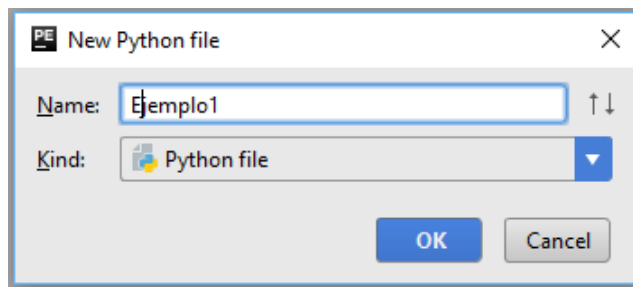
3.2.2 Creación de archivos Python

Un proyecto puede contener uno o más archivos con código Python. Para añadir un archivo a un proyecto colocaremos el cursor del ratón sobre el nombre del proyecto y haremos *click* en el botón derecho. Aparece un menú contextual con una serie de opciones que se pueden realizar sobre un proyecto. Escogeremos la opción “*New+Python File*”.

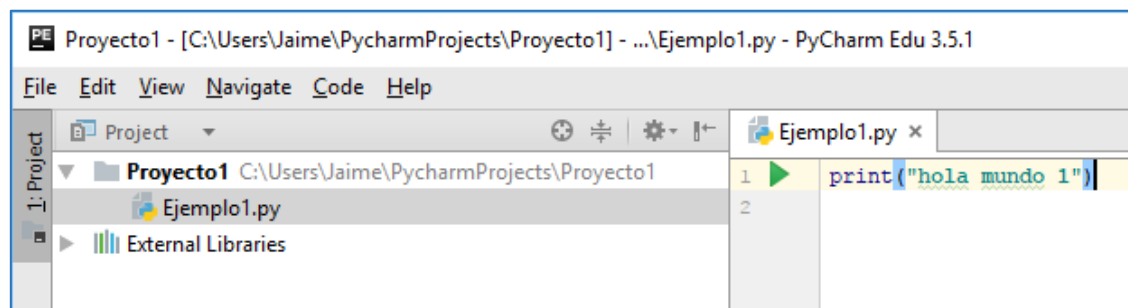




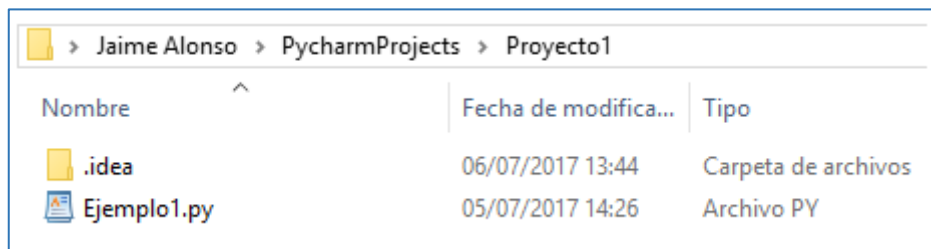
Escogemos un nombre y botón “OK”.



El archivo se añade al proyecto y ya se puede empezar a introducir código Python. En el ejemplo siguiente añadimos una sentencia `print` al archivo “*Ejemplo1.py*” que permite sacar por pantalla un mensaje.

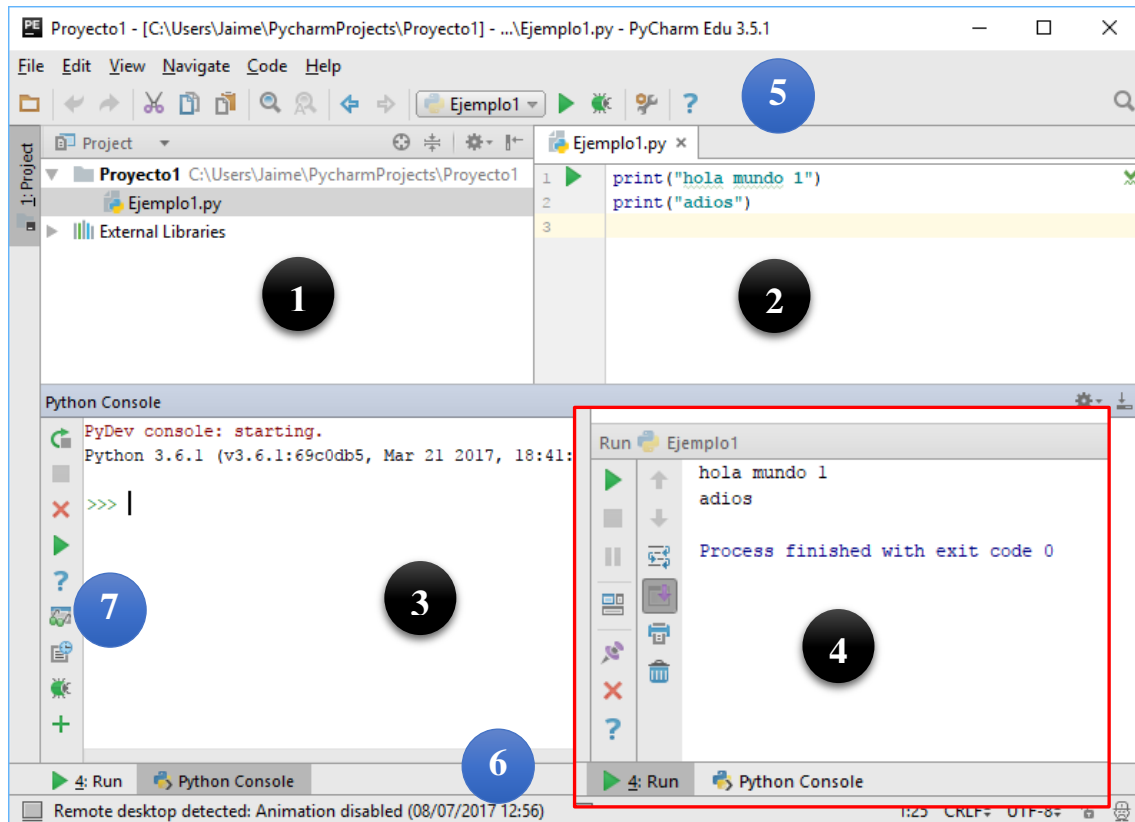


Los archivos se guardan en el directorio creado para el proyecto al que pertenecen.



3.3 Interfaz de usuario de PyCharm

La apariencia típica de PyCharm con un proyecto abierto debería ser similar a la que se muestra en la figura siguiente:



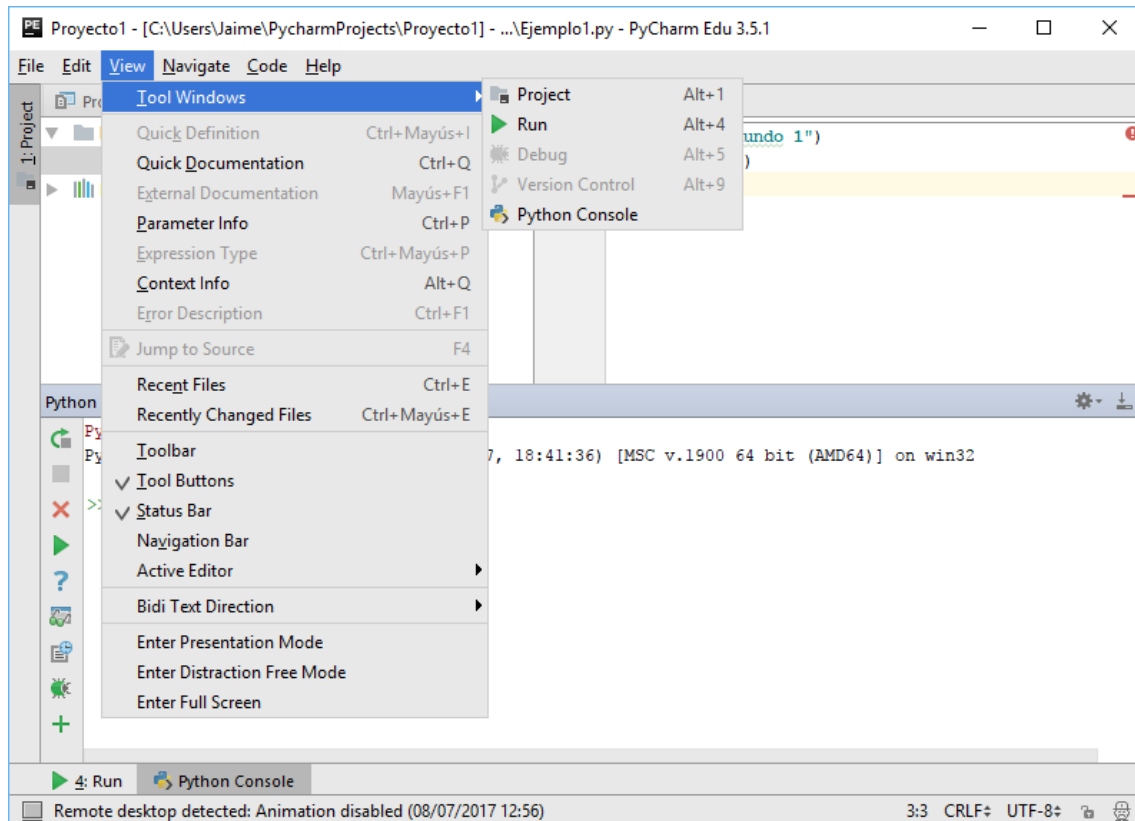
A continuación se describen los elementos más importantes de la interfaz de usuario:

1. Ventana de proyectos. Contiene información sobre el proyecto con el que se está trabajando, sus directorios y archivos.
2. Ventana de edición. Sirve para escribir programas en Python que puedan ser guardados en disco. Tiene coloreado sintáctico y ayuda automática al escribir código.
3. Consola de Python. 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.
4. Ventana de ejecución. Contiene el resultado de la ejecución de los programas Python.

Otros elementos de la interfaz son:

5. Barra de herramientas (*Toolbar*). Barra de botones con herramientas de uso frecuente.
6. Barra de estado (*Status Bar*). Indica el estado del IDE y permite llevar a cabo algunas tareas de mantenimiento del entorno.
7. Menús de Consola y de la Ventana de ejecución.

La configuración de la visualización se realiza desde la opción “View” del menú principal de la aplicación:



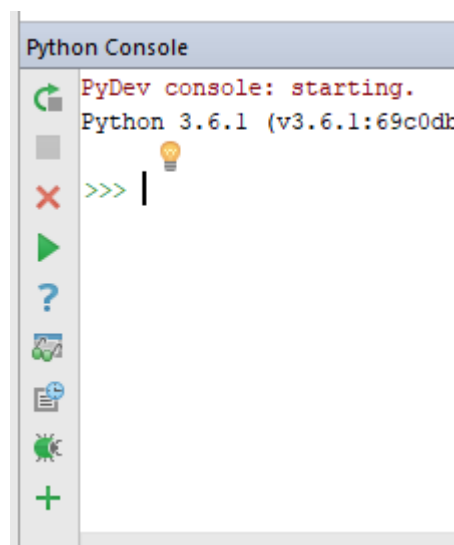
3.4 Uso de la Consola de Python

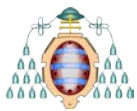
Puedes comenzar a usar ya mismo el intérprete de Python a través de esta consola, incluso sin saber nada de programación, como si fuera una potente calculadora. Prueba a introducir cualquier expresión numérica y verás cómo te responde con el resultado.

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.

Hay una barra de botones lateral con este significado:

- Reiniciar consola
- Detener proceso
- Cerrar consola
- Ejecutar sentencia
- Ayuda
- Ver variables
- Ver historial
- Vincular depurador
- Abrir otra consola





3.5 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.

- 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 y proporciona funciones tanto para obtener el código ASCII de una letra, como 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, asume 10.

- `int('1B', 16)` sale 27 (¿podrías explicar por qué?)
- `int('27')` sale 27 (conversión de cadena de caracteres a entero).

Puede convertir un número dado a hexadecimal o binario. El número puede ser la evaluación de una expresión. El resultado es una cadena de caracteres, delante de la cual añade los caracteres "0x" para indicar que está en hexadecimal y "0b" para indicar que está en binario.

- `bin(2**8)` sale `'0b100000000'`
- `hex(2**8)` sale `'0x100'`

También puedes operar con expresiones lógicas realizando comparaciones entre números (expresiones numéricas) o entre cadenas de caracteres (o expresiones de cadenas). En Python los valores lógicos son `True` para representar verdadero y `False` para representar falso. Los operadores booleanos son los siguientes:

- El operador `and` es la conjunción (\wedge): `2 < 3 and 'bc' < 'adf'` sale `False`
- El operador `or` es la disyunción (\vee): `2 < 3 or 'bc' < 'adf'` sale `True`
- El operador `not` es la negación (\neg): `not 2 < 3` sale `False`

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.

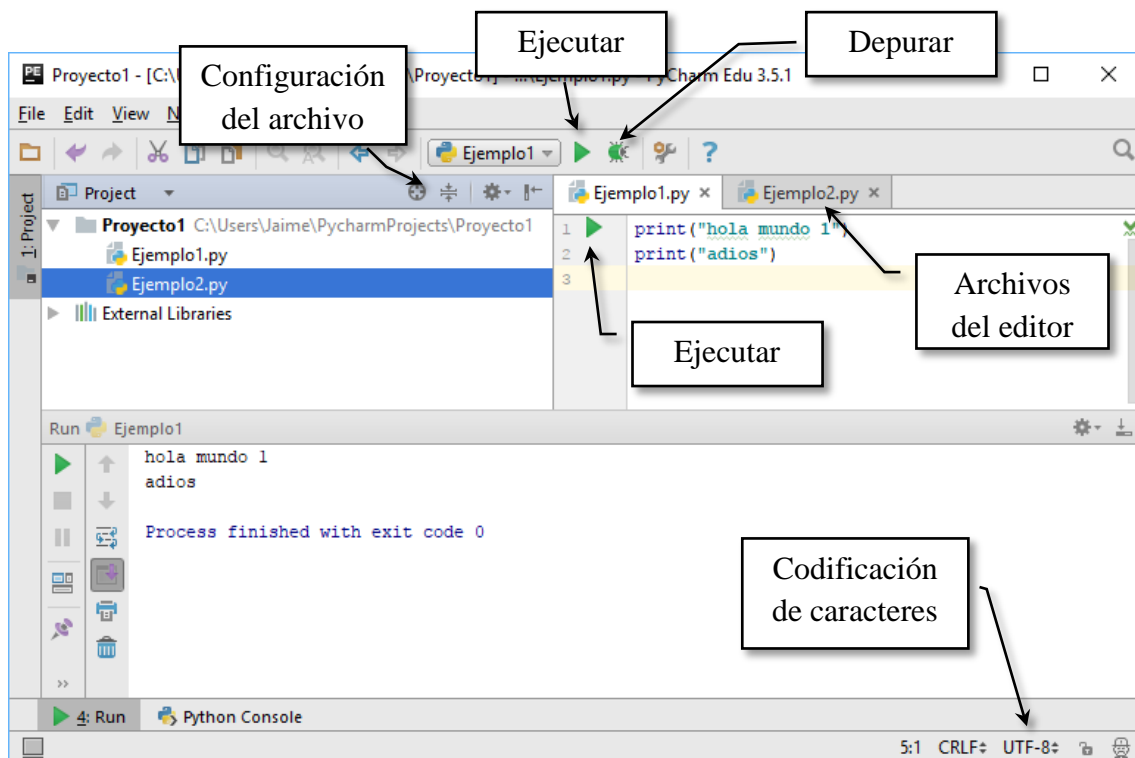


- `round(real, n)`: redondea el número real dado a n decimales. Se puede omitir n en cuyo caso redondea al entero más próximo (aunque el resultado seguirá siendo float).
- `type(dato)`: te dice de qué tipo es ese dato (o el resultado de la expresión)
- `help(x)`: te da ayuda sobre x , que puede ser una función, un tipo de dato, una variable, un módulo que hayas importado, etc.

REALIZA AHORA EL EJERCICIO 1

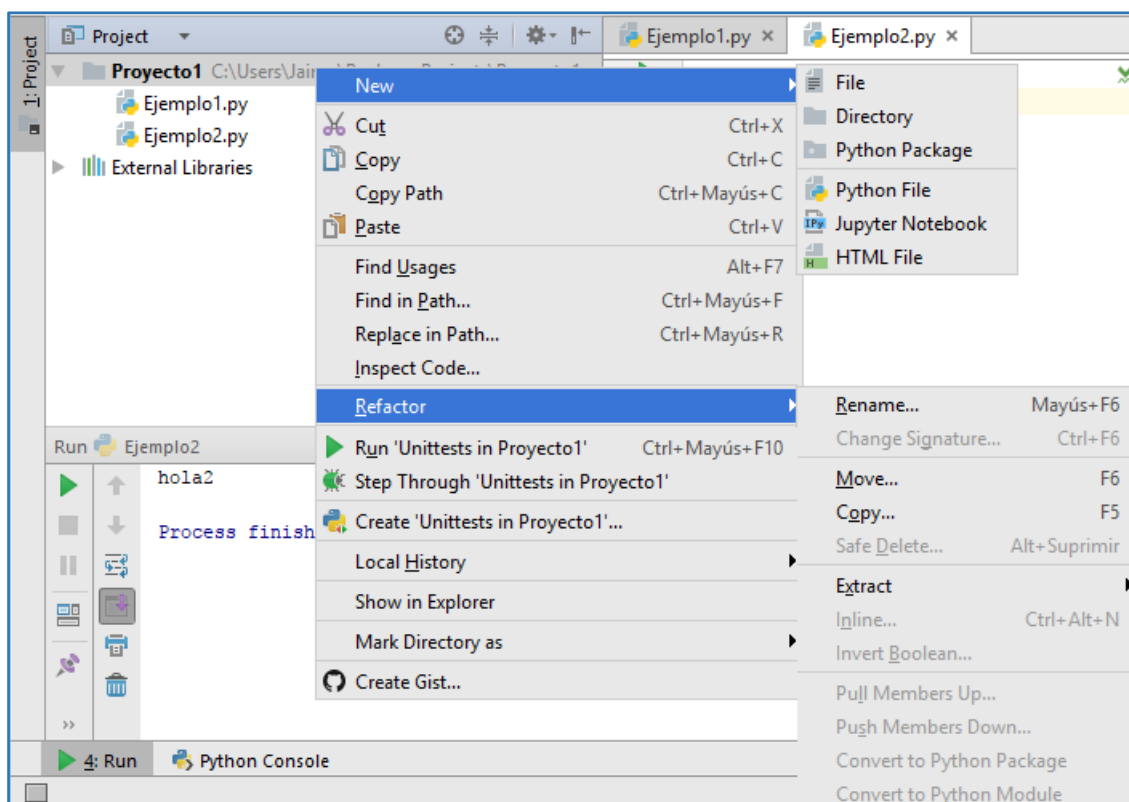
3.6 El editor

El panel del editor te permite escribir código Python asociado a un archivo y ejecutarlo. Es la forma habitual de desarrollar programas en Python. El intérprete interactivo es sólo adecuado para pequeñas pruebas con fragmentos muy cortos de código. Se recuerda que todos los archivos Python deben ir vinculados a un proyecto. Python tiene autoguardado automático de archivos por lo que no aparece un botón u opción de menú para guardar. La codificación de caracteres es UTF-8 por defecto.

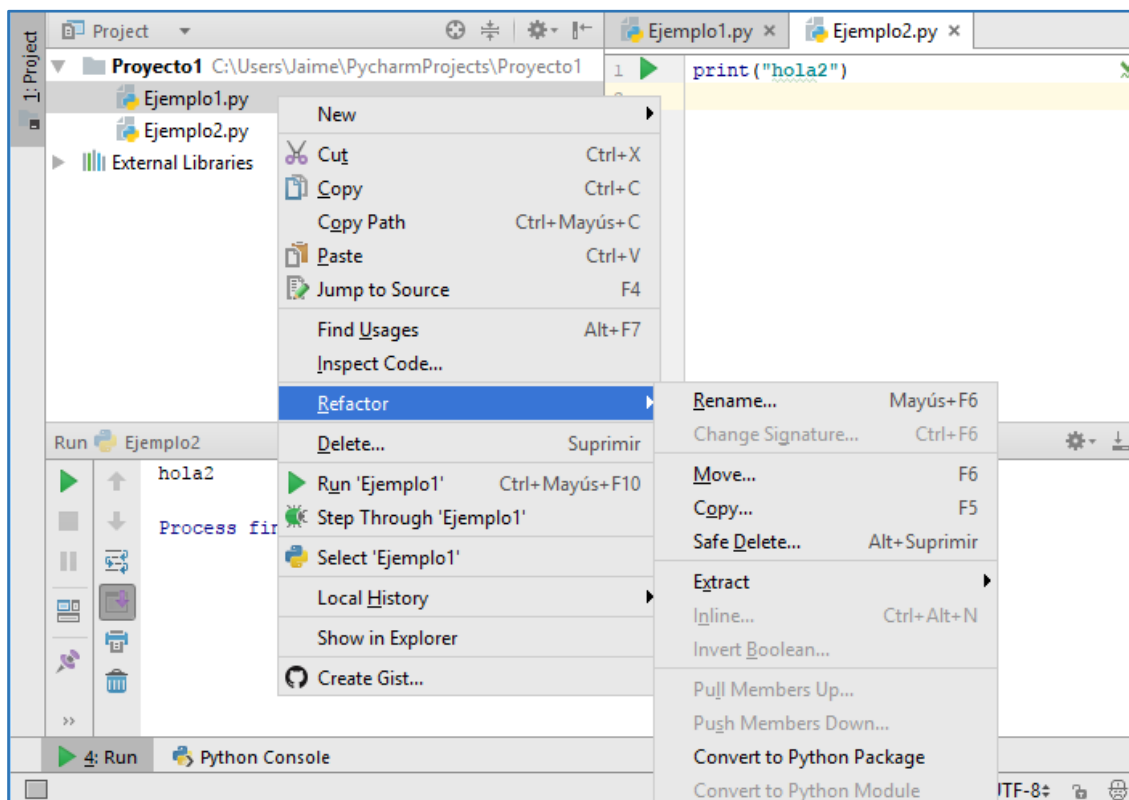


Cada archivo de Python tiene una configuración independiente que podemos modificar si fuese necesario. Durante el curso, las opciones por defecto son suficientes y no va a hacer falta cambiar nada.

La forma más rápida de efectuar acciones sobre los proyectos y los archivos es mediante los menús contextuales. Un menú de contexto aparece cuando se hace *clic* con el botón derecho del ratón sobre un determinado elemento. Por un lado, tenemos el siguiente menú de contexto para un proyecto:

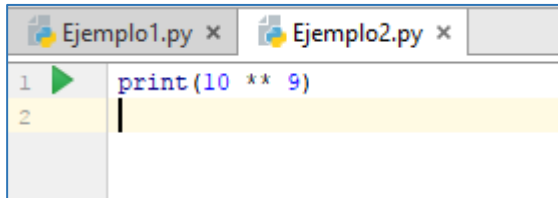


Las opciones más destacables son “New”, que permite crear archivos nuevos de Python en el proyecto y “Refactor”, que permite cambiar el nombre del proyecto. Por otro lado, para un archivo se dispone del siguiente menú contextual:



Mediante este menú de archivos podemos, entre otras cosas, ejecutar un archivo, verlo en una ventana del explorador de archivos, moverlo, borrarlo o renombrarlo.

Para mostrar un mensaje por la pantalla Python dispone de la sentencia `print`. Aunque esto se verá en detalle en la sesión sobre Entrada/Salida por consola, de momento baste decir que la palabra `print` seguida de una expresión entre paréntesis mostrará en pantalla el resultado de evaluar esa expresión (en la *Ventaja de Ejecución*). Prueba lo siguiente:

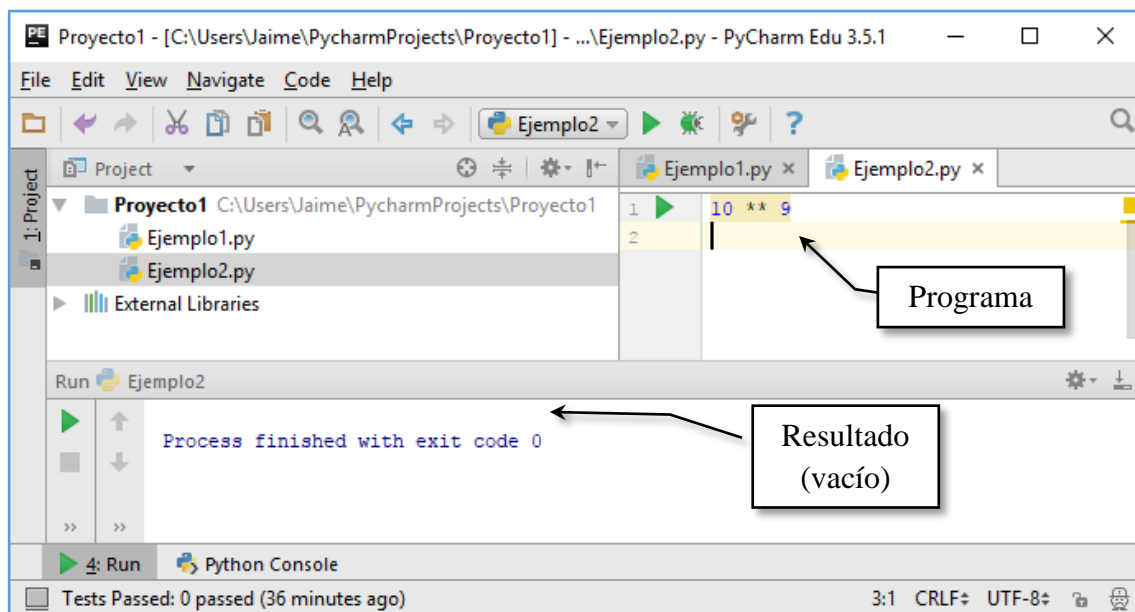


```
1 print(10 ** 9)
2
```

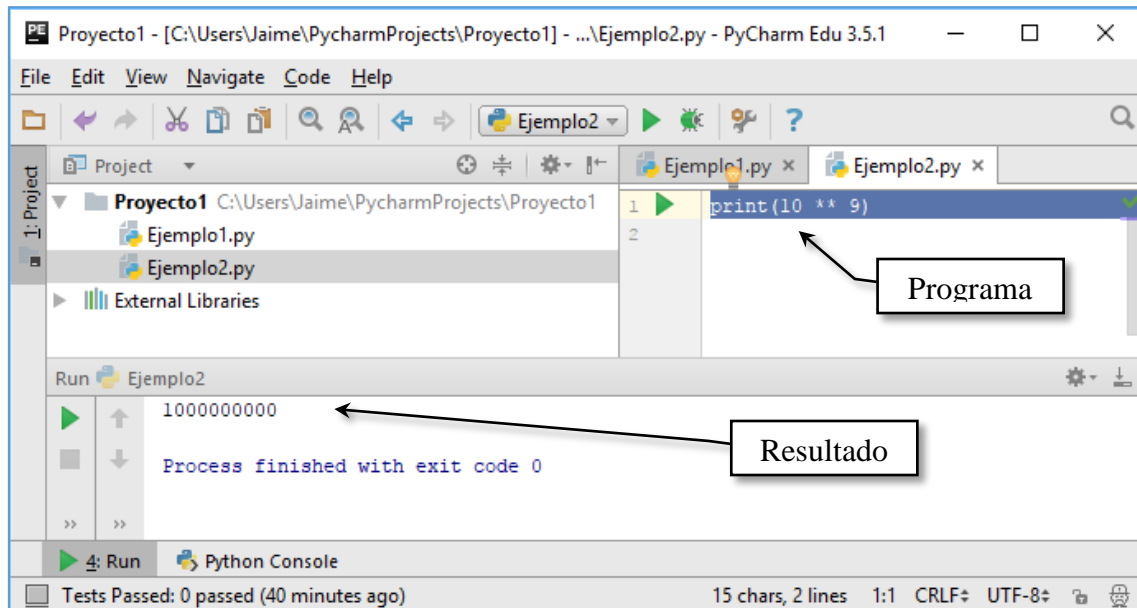
REALIZA AHORA EL EJERCICIO 2

3.7 Ejecución de programas

En los apartados anteriores hemos visto que hay varias formas de ejecutar archivos que contienen programas Python: desde el editor, desde la barra de herramientas o desde el menú contextual de un archivo. En *PyCharm*, el resultado de la ejecución de un archivo se muestra en la *Ventaja de ejecución*. Hay programas cuya ejecución no proporciona información al usuario y otros que sí. Veamos dos ejemplos:



Este primer programa no contiene ninguna salida por pantalla que proporcione información al usuario.



Este segundo programa sí contiene salida por pantalla del resultado ya que incorpora una sentencia `print`.

4 Errores

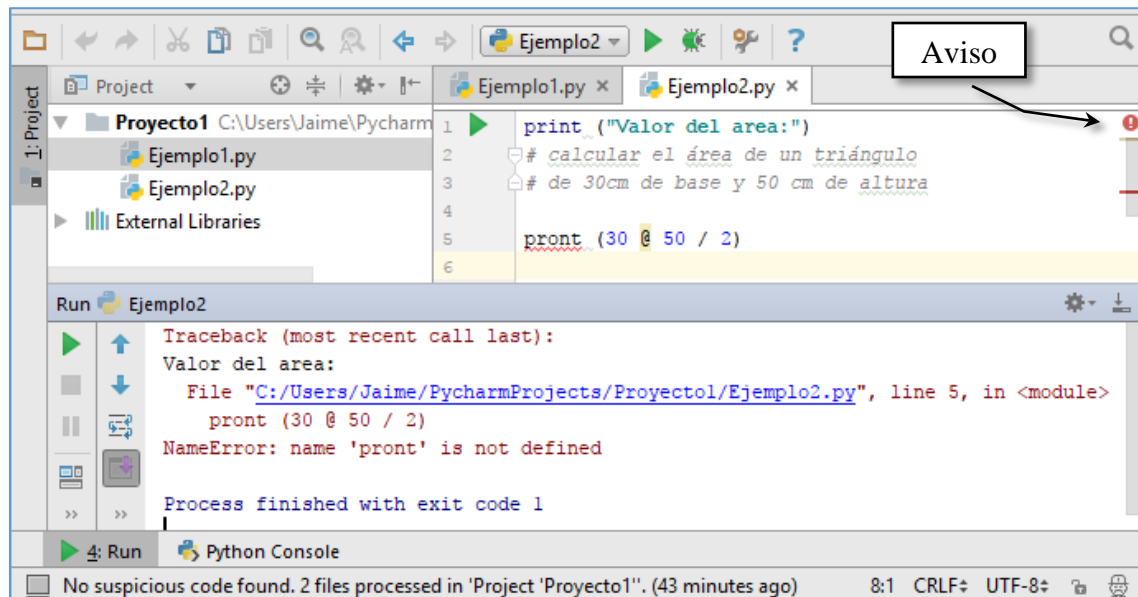
Los programadores no siempre hacen los programas bien a la primera, pueden cometer errores. Los errores en programación se denominan “*bugs*”, y encontrar esos errores se llama “*debugging*” o “*depuración*”. Podemos catalogar los errores en tres tipos, por orden creciente de su dificultad de detección.

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.

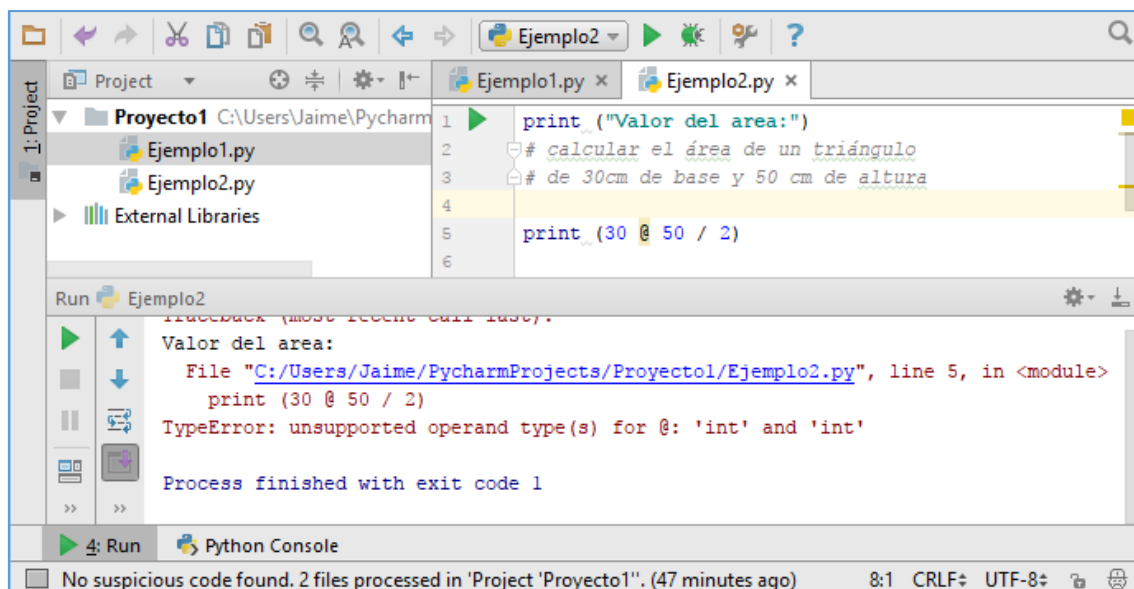
Los errores de sintaxis son fáciles de encontrar porque, antes siquiera de ejecutar nuestro código, Python le echa un vistazo general en busca de este tipo de errores y, si encuentra alguno, nos lo señala sin llegar a ejecutar ni una línea del código.

Por ejemplo, introduce el siguiente código en el editor, en el que se han cometido dos errores de sintaxis: escribir `pront` en lugar de `print` y poner `@` en lugar de `*` (y resulta que `@` no es una operación válida en Python):



Al escribir el código en el editor del IDE se detectan los errores sintácticos, nos avisa y nos los señala: `pront` subrayado y `@` en color amarillo. Otros IDE's no permiten ejecutar el programa si se detecta un error sintáctico. PyCharm avisa pero permite la ejecución, lo que provocaría en error del tipo que se va a explicar en el apartado siguiente.

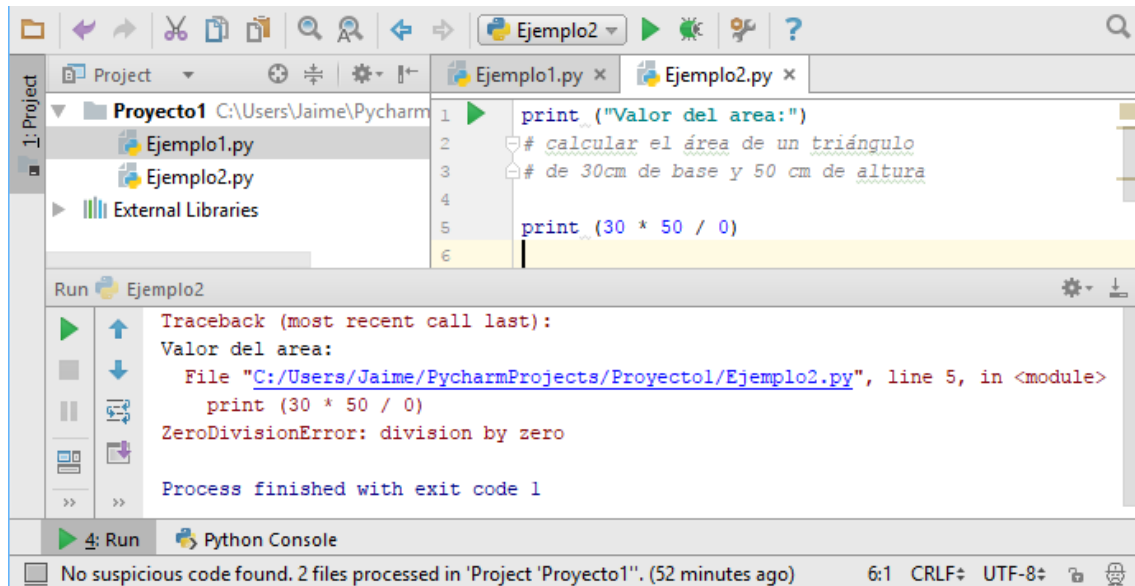
El mismo ejemplo si sólo corregimos el primer error:



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.

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. Por ejemplo:



```
1 print('Valor del area:')
2 # calcular el área de un triángulo
3 # de 30cm de base y 50 cm de altura
4
5 print(30 * 50 / 0)
6
```

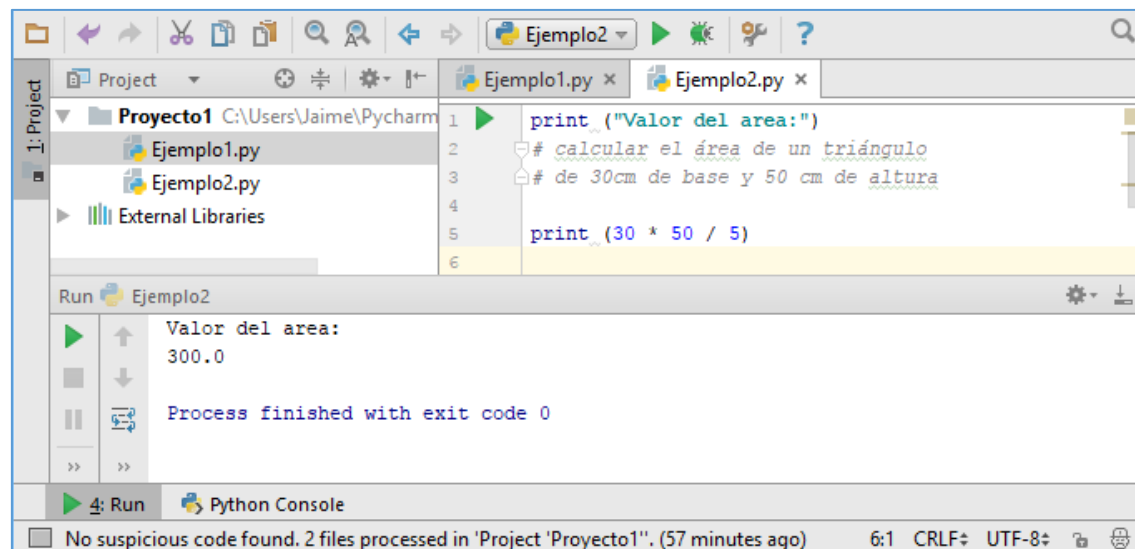
Run Ejemplo2

Traceback (most recent call last):
Valor del area:
File "C:/Users/Jaime/PycharmProjects/Proyecto1/Ejemplo2.py", line 5, in <module>
print(30 * 50 / 0)
ZeroDivisionError: division by zero
Process finished with exit code 1

En este caso, la expresión $30 * 50 / 0$ es sintácticamente correcta, pero no se puede dividir por 0. Por tanto, al llegar a esa expresión se produce un error en tiempo de ejecución. Comprueba que la ejecución del código en este caso proporciona el mensaje correspondiente a la primera línea del código.

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:



```
1 print('Valor del area:')
2 # calcular el área de un triángulo
3 # de 30cm de base y 50 cm de altura
4
5 print(30 * 50 / 5)
6
```

Run Ejemplo2

Valor del area:
300.0
Process finished with exit code 0

En lugar de dividir por 2 hemos dividido por 5 (debería salir 750.0 y sale 300.0). 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.

4.4 Depuración de programas

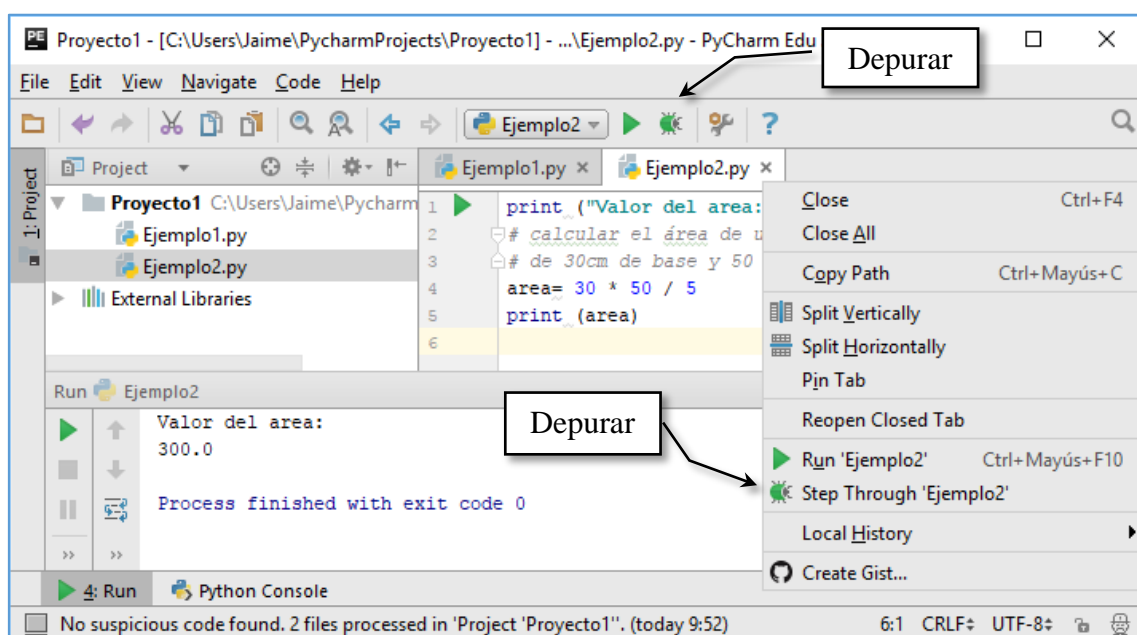
En este ejemplo anterior lo teníamos fácil porque era un ejemplo muy sencillo, pero en general se plantea una interesante cuestión. Si ya sabíamos que resultado debía dar ¿para qué hicimos un programa que lo calcula? Y si no lo sabíamos ¿cómo podremos detectar que ha salido mal?

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.

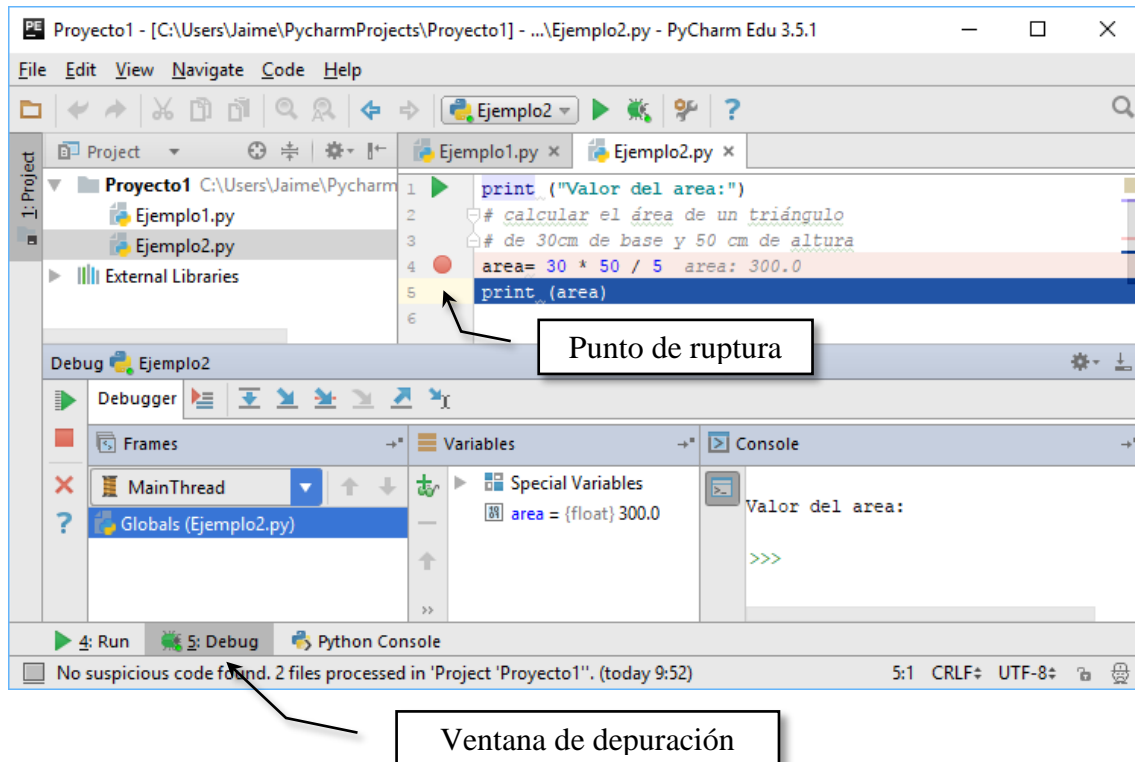
Si damos con un caso en el que la respuesta no es correcta, habrá que encontrar en qué punto del programa hemos cometido el error. El método general es usar un *depurador* que permite ejecutar una a una las instrucciones del programa y ver qué sale en cada una de ellas. De nuevo se supone que conocemos 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.

En definitiva, el *depurador* te permite ejecutar el programa “en cámara lenta”, instrucción por instrucción, al ritmo que uno quiera para poder observar en detalle el flujo que toma el programa y el estado de las variables.

Vamos a ver por qué no va bien el programa del apartado anterior que calcula el área de un triángulo. Hemos modificado ligeramente el código para que contenga una variable que almacene el valor que se calcula para el área. Sabemos que el resultado debería ser 750.0 pero sale 300.0.



Es habitual establecer puntos de ruptura (*breakpoints*) o puntos de parada en los programas que queremos depurar. Indican las líneas del programa donde se va a detener la ejecución del mismo. En PyCharm se establecen haciendo *click* con el ratón en la banda lateral izquierda del editor, justo a la altura de la línea donde queremos que se detenga la ejecución. Visualmente se distinguen porque aparece un punto rojo.

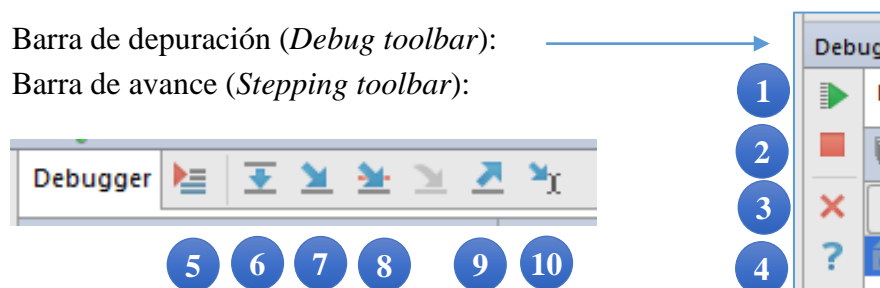


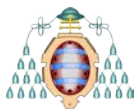
Cuando se ejecuta un programa en modo depuración la “traza” del mismo se muestra en una *ventana de depuración* (*Debug* en la imagen). Está formada por tres paneles:

- *Frames*: muestra el “ámbito” al que pertenece el código que se está ejecutando. En este curso tendremos dos ámbitos: global y local (funciones).
- *Variables*: permite examinar los valores almacenados en las variables del programa.
- *Console*: muestra información del sistema, mensajes de error, y la entrada y salida de la consola del programa.

y una serie de botones agrupados principalmente en dos barras:

- Barra de depuración (*Debug toolbar*):
- Barra de avance (*Stepping toolbar*):





Breve descripción de los botones más relevantes para nosotros:

1. Continuar ejecución (*Resume, F9*)
2. Parar la ejecución (*Stop, Ctrl+F2*)
3. Cerrar depurador (*Close, Ctrl+F4*)
4. Ayuda (*Help, F1*)
5. Mostrar punto de ejecución (*Show Execution Point, Alt+F10*)
6. Avanzar a la sgte. línea sin entrar en funciones (*Step Over, F8*)
7. Avanzar a la sgte. línea pudiendo entrar en funciones (*Step Into, F7*)
8. Avanzar a la sgte. línea pudiendo entrar en funciones pero sólo si son nuestras (*Step Into My Code, Alt+Mayús+F7*)
9. Avanzar a la sgte. línea saliendo de la función actual (*Step Out, Mayús+F8*)
10. Ejecutar hasta la posición del cursor (*Run to Cursor, Alt+F9*)

El depurador de Pycharm ofrece muchas opciones interesantes para ayudar a encontrar los errores de los programas. Recomendamos profundizar en su estudio para que estas opciones se puedan explotar al máximo.

REALIZA AHORA EL EJERCICIO 3

Ejercicios

► EJERCICIO 1. 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 `ord` 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. Usa la función apropiada para encontrar qué número decimal es el que se codifica en binario como 110011001100
5. 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
6. 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
7. Observa la diferencia entre el resultado de `int(2.8)` y `round(2.8)`
8. Comprueba que `type(int(3.14))` y `type(round(3.14))` producen el resultado esperado.
9. ¿Qué resultado produce `type("Hola")`?
¿Y cuál producirá `type(hex(100))`? Compruébalo.



► EJERCICIO 2. USANDO EL IDE

1. Abre un nuevo archivo y escribe con el editor un programa que muestre en pantalla la tabla ASCII correspondiente a las 5 primeras letras mayúsculas del alfabeto. La tabla debe mostrar el carácter y su código en decimal, binario y hexadecimal, tal y como se muestra a continuación:

Car.	Dec	Binario	Hex
A	65	0b1000001	0x41
B	66	0b1000010	0x42
C	67	0b1000011	0x43
D	68	0b1000100	0x44
E	69	0b1000101	0x45

Para hacer la tabla necesitas saber que un `print` puede escribir varios resultados en una misma línea de la pantalla. Para ello debes separar las expresiones correspondientes con el carácter `' , '`.

Así, por ejemplo, la ejecución del código que se muestra en el cuadro adjunto muestra en pantalla:

Car.	Dec
A	65
B	66

```
# -*- coding: utf-8 -*-  
  
# Tabla ASCII para las cinco  
# primeras letras mayúsculas  
  
print 'Car. Dec'  
print '-----'  
print ' A ', ord('A')  
print ' B ', ord('B')
```

2. Una vez realizado el código y ejecutado, guarda el archivo con el nombre `primer_ejemplo.py`. Cuando estés seguro de haberlo guardado, cierra el archivo y vuelve a cargarlo.

► EJERCICIO 3. ERRORES

1. En el campus virtual tienes un fichero llamado `con_errores.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 proporcione la salida esperada:

```
Buenos días  
50  
18.8496  
True  
True
```




► EJERCICIO 4. 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 45°F, 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.
6. ¿Cómo arreglar lo anterior?
7. Prepara una variable `n` con el valor 5. Calcula la raíz `n`-ésima de 2. Debe salir 1.1486983549970351