**IronPython  
  
Desarrollo de aplicaciones profesionales  
para Windows con IronPython**

**INTRODUCCIÓN**

[*IronPython*](https://ironpython.net/) es una *implementación de Python2.7 escrita en C#*. Permite crear aplicaciones perfectamente integradas con *Microsoft .NET* utilizando *python* como lenguaje de programación.

**NOTA:** [*IronPython3*](https://github.com/IronLanguages/ironpython3) se encuentra actualemente *fase alfa de desarrollo*.

Como programadores de *python*, para el desarrollo de aplicaciones en *Windows* voy a destacar dos opciones:

* *CPython* con *[PythonNET](https://tecnobillo.com/sections/python-en-windows/pythonnet/pythonnet.html" \t "_blank)*
* [*IronPython*](https://ironpython.net/)

Ambas son buenas opciones y, en general, **el código fuente de una *aplicación ironpython* es código válido para una *aplicación pythonnet***.

Mientras que *pythonnet* es un móduo que integra el *runtime de CPython* con el *runtime NET*, *IronPython* es una *implementación de Python escrita en C#*, que es el lenguaje de programación insignia de *Microsoft .NET*.

*IronPython* está mejor integrado con *NET* que *pythonnet*. Podría decirse que *ironpython* vive en *NET* mientras que *pythonnet* se comunica con *NET*.

Un ejemplo de esta mejor integración es que en *ironpython* no tenemos que crear [*delegados*](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/delegates/#:~:text=Un%20delegado%20es%20un%20tipo,un%20tipo%20de%20valor%20devuelto.&text=Cuando%20se%20crea%20una%20instancia,un%20tipo%20de%20valor%20devuelto.) explícitamente, sino que el intérprete sabe cuando transformar una función en un delegado, cosa que *pythonnet* no puede hacer. Veáse [*crear un cronómetro con pythonnet*](https://tecnobillo.com/sections/python-en-windows/pythonnet/sections/cronometro-pythonnet/cronometro-pythonnet.html).

Un aspecto a tener en cuenta es que en *ironpython* no existe el polémico **GIL** de *cpython*.

**CÓMO UTILIZAR IRONPYTHON**

Hay dos formas de utilizar *IronPython*:

1. Mediante **ipy**, el *intérprete de ironpython*, e **ipyc**, el *compilador de ironpython*.
2. Desde **Visual Studio**, añadiendo *ironpython* desde el gestor de paquetes **NuGet**.

**SIN VISUAL STUDIO: IPY e IPYC**

[*Descarga e instala IronPython*](https://ironpython.net/). Una vez instalado, añade el directorio de instalación al *path del sistema*.

Abre el *cmd* y ejecuta ipy. Verás lo siguiente:

IronPython 2.7.11 (2.7.11.1000)

[.NETFramework,Version=v4.5 on .NET Framework 4.8.4300.0 (64-bit)]

Type "help", "copyright", "credits" or "license" for more information.

>>>

Esto te resultará familiar, ya que es un *REPL* idéntico al de *CPython*. Puedes crear un *script python* como el siguiente:

# mi\_script.py

for i in range(10):

print('Hola {}'.format(i))

Y ejecutarlo mediante ipy mi\_script.py o mediante python mi\_script.py. En el primer caso, el *script* será ejecutado por *IronPython*, y en el segundo caso por *CPython*.

Recuerda que *IronPython* es una implementación de *Python2.7*, por lo que si modificas *mi\_script.py* así:

# mi\_script.py

for i in range(10):

print(f'Hola {i}')

Al ejecutarlo con ipy mi\_script.py se producirá un error, ya que las *f-strings* no existen en *Python2.7*.

Anteriormente he mencionado que *ironpython* es una *implementación de python escrita en C#*, y vamos a comprobarlo desde el *REPL*:

>>> import System

>>> System.String == str

True

>>> System.Int32 == int

True

>>>

Hemos importado a System, que es un **espacio de nombres** de *NET*, y podemos utilizarlo como si fuese un **módulo python**. A continuación hemos comprobado que System.String == str y que System.Int32 == int, es decir, los tipos de datos primitivos de *IronPython* son en realidad tipos de datos de *NET*.

Pero, **¿qué aporta *ironpython* que no aporta *cpython*?**. Para empezar crearemos una pequeña *aplicación WinForms*:

**Ejemplo de Aplicación WinForms con IronPython**

Vamos a ver un ejemplo de *Aplicación con interfaz gráfica Winforms* creada con *IronPython*:

# -\*- coding: utf-8 -\*-

# app.py

import clr # clr (common language runtime) es el módulo mas importante en IronPython

clr.AddReference('System.Windows.Forms') # para usar System.Windows.Forms como un módulo primero hay que usar clr.AddReference

from System.Windows.Forms import \* # importamos System.Windows.Forms como si fuese un módulo python

class App(Form): # Creamos una clase App que hereda de System.Windows.Forms.Form

def \_\_init\_\_(self):

self.Text = 'IronPython App' # Título del formulario/ventana

self.boton = Button() # Creamos un botón que muestra el texto "Saludar"

self.boton.Text = 'Saludar'

self.boton.Click += lambda \*args: MessageBox.Show('Hola desde IronPython! :D') # Bind del evento Click con un manejador

self.boton.Dock = DockStyle.Fill # El botón ocupará todo el espacio disponible del formulario

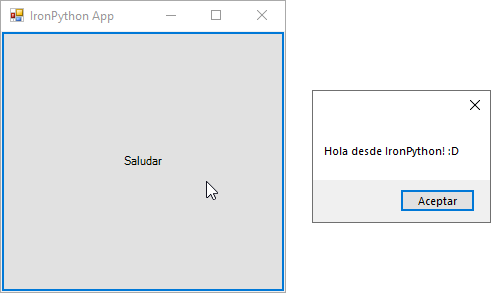
self.Controls.Add(self.boton) # Añadimos el botón como un control del formulario

app = App() # Instanciamos nuestra clase App, que hereda de System.Windows.Forms.Form

Application.EnableVisualStyles() # Activamos los efectos visuales modernos (opcional)

Application.Run(app) # Iniciamos la aplicación

Al ejecutar este código se mostrará un formulario que contiene un botón. El botón ocupa todo el espacio disponible y al hacer click en él se abre un MessageBox con el texto 'Hola desde IronPython! :D'.



En este código es especialemente importante lo siguiente:

* import clr - El *módulo clr* hace referencia al *common language runtime de .NET*, y es el módulo mas importante de *IronPython*, ya que nos permite hacer referencia a los *ensamblados de .NET* y utilizarlos como si fuesen módulos de *python* (se utiliza exactamente igual que en *pythonnet*).
* clr.AddReference('System.Windows.Forms') - Para utilizar *ensamblados de .NET* como módulos primero hay que hacer referencia a ellos.
* from System.Windows.Forms import \* - En el *script* hemos utilizado Form, Button, MessageBox, DockStyle y Application, que pertenecen a System.Windows.Forms. Si desarrollas aplicaciones con *IronPython*, frecuentemente tendrás que leer documentación sobre *C#*, y debes saber que cuando en *C#* se utiliza using en *IronPython* utilizamos from ... import, es decir, from System.Windows.Forms import \* en *C#* sería using System.Windows.Forms.

Ahora que tenemos nuestra primera aplicación, vamos a compilarla para poder distribuirla. En el directorio de instalación de *IronPython* se encuentra *ipyc.exe*, que es el compilador.

Si añadiste el directorio de instalación de *ironpython* al *path del sistema*, en la carpeta donde se encuentra *app.py* puedes ejecutar el siguiente comando para compilar la aplicación:

ipyc /target:winexe /embed /standalone /main:app.py

Se creará un archivo *app.exe* que es tu aplicación. Analicemos los argumentos del comando de compilación:

* /target:winexe - Indica que debe ocultarse el *cmd* al ejecutar la aplicación.
* /embed - Sin este argumento se generaría una *librería dll* de la que depende el ejecutable. Este argumento incluye esta librería como recurso del ejecutable.
* /standalone - Añade los ensamblados de *ironpython* al ejecutable (necesario para poder ejecutarse en equipos que no tienen *ipy* instalado).
* /main:app.py - Indicamos el *script principal* de la aplicación. En este caso es el único *script* de la aplicación.

**NOTA:** Si ejecutas ipyc sin argumentos se imprimirá en consola la *ayuda del compilador*. Por ejemplo, si quieres añadir un icono a tu ejecutable puedes hacerlo con el argumento /win32icon:file.ico.

Ahora que hemos compilado nuestra primera aplicación vamos a enfrentarnos a un nuevo problema. Modifiquemos un poco el código:

# -\*- coding: utf-8 -\*-

import os # <-- Atención!

import clr

clr.AddReference('System.Windows.Forms')

from System.Windows.Forms import \*

class App(Form):

def \_\_init\_\_(self):

self.Text = 'IronPython App'

self.boton = Button()

self.boton.Text = 'Saludar'

self.boton.Click += lambda \*args: MessageBox.Show(os.getcwd()) # <-- Atención!

self.boton.Dock = DockStyle.Fill

self.Controls.Add(self.boton)

app = App()

Application.EnableVisualStyles()

Application.Run(app)

En este caso, hemos importado el *módulo os de la librería estándar de ironpython*. Cuando se haga click en el botón se mostrará el *directorio de trabajo de la aplicación*: MessageBox.Show(os.getcwd())

Si ejecutas ipy app.py y haces click en el botón todo funcionará correctamente, pero si compilas la aplicación con el comando ipyc /target:winexe /embed /standalone /main:app.py, al ejecutar *app.exe* aparecerá un mensaje de error que dice lo siguiente: Error occurred: No module named os.

**Hay dos formas de abordar este problema:**

* No utilizando la *librería estándar* - En su lugar puedes utilizar los recursos que ofrece *NET*. Por ejemplo, para obtener el *directorio de trabajo* utiliza [System.IO.Directory.GetCurrentDirectory()](https://docs.microsoft.com/es-es/dotnet/api/system.io.directory.getcurrentdirectory?view=net-5.0" \t "_blank):

# -\*- coding: utf-8 -\*-

import clr

clr.AddReference('System')

clr.AddReference('System.Windows.Forms')

import System

from System.Windows.Forms import \*

class App(Form):

def \_\_init\_\_(self):

self.Text = 'IronPython App'

self.boton = Button()

self.boton.Text = 'Saludar'

self.boton.Click += lambda \*args: MessageBox.Show(System.IO.Directory.GetCurrentDirectory()) # <-- Atención!

self.boton.Dock = DockStyle.Fill

self.Controls.Add(self.boton)

app = App()

Application.EnableVisualStyles()

Application.Run(app)

* Añadiendo la *librería estándar* - Dentro del directorio donde hemos instalado *ipy* se encuentra el directorio **Lib**. Copialo en el mismo directorio en el que se encuentra *app.py* y podrás utilizar el *módulo os* y cualquier otro de la *librería estándar*. El inconveniente es que **Lib** no está incluido en el ejecutable, y para distribuir tu aplicación tendrías que crear un instalador o comprimir el ejecutable y **Lib** en un archivo *zip*.

Otro inconveniente en este caso es el tamaño final de la aplicación, ya que **Lib ocupa aproximadamente 15mb**. No obstante puedes comprimir **Lib** en un archivo *Lib.zip* (con *winrar* ocupará unos **5.8mb**), pero debes modificar el código de la aplicación así:

# -\*- coding: utf-8 -\*-

import sys, clr # <-- Atención! Módulos que no dependen de Lib

clr.AddReference('System.Windows.Forms')

from System.Windows.Forms import \*

sys.path.append('Lib.zip/Lib') # <-- Atención!

import os # <-- Atención! Solamente puede usarse el módulo os después de añadir el zip a sys.path

class App(Form):

def \_\_init\_\_(self):

self.Text = 'IronPython App'

self.boton = Button()

self.boton.Text = 'Saludar'

self.boton.Click += lambda \*args: MessageBox.Show(os.getcwd()) # <-- Atención!

self.boton.Dock = DockStyle.Fill

self.Controls.Add(self.boton)

app = App()

Application.EnableVisualStyles()

Application.Run(app)

Puedes utilizar import sys, clr antes de añadir Lib.zip/Lib al *path de python*, ya que estos módulos no están definidos en **Lib**.

**Hemos visto como ejecutar *scripts python* con *ipy* y también como compilar y distribuir nuestras aplicaciones.**

No obstante, **las aplicaciones compiladas de esta forma tienen dos importantes desventajas**:

1. Al compilar, no tenemos manera de añadir información sobre el nombre de la aplicación, descripción, versión, etc, al ejecutable.
2. El ejecutable creado con *ipyc.exe* frecuentemente es detectado como malicioso, al igual que sucede con los ejecutables creados con *PyInstaller* para *CPython*. Puedes comprobarlo tú mismo en *[virustutal](https://www.virustotal.com/" \t "_blank)*.

Por estos motivos, **a continuación veremos otra forma de crear aplicaciones con *IronPython***.

**CON VISUAL STUDIO: NuGet**

Instala [*Visual Studio Community*](https://visualstudio.microsoft.com/es/free-developer-offers/) y crea un nuevo proyecto *Aplicación de Windows Forms (C#)* llamado, por ejemplo, *ironapp*.

Una vez creado, a la derecha se muestra el *Explorador de Soluciones*. Donde pone **[C#] ironapp** haz click derecho y a continuación selecciona *Administrar paquetes NuGet...*.

Una vez abierto *NuGet*, en la parte superior haz click en la pestaña *Examinar* y busca *ironpython*. Encontrarás los siguientes paquetes que te interesan:

* *IronPython* - Es el *runtime de ironpython*.
* *IronPython.StdLib* - Es la *librería estándar de ironpython*.

Instala los dos paquetes seleccionándolos y haciendo click en el botón *Instalar* que aparece a la derecha.

Vuelve al *Explorador de Soluciones* y ahora haz click derecho en **[C#] ironapp** y a continuación selecciona *Abrir carpeta en el Explorador de Archivos*. En la carpeta del proyecto crea un archivo *aplicacion.py*. De momento no escribas nada en su interior. En el *Explorador de Soluciones* haz click derecho en **[C#] ironapp**, navega a *Agregar - Elemento existente...* y selecciona tu archivo *aplicacion.py*. Una vez añadido, haz click en él y abajo, donde se muestran sus propiedades, en el campo *Copiar en el directorio de salida* selecciona *Copiar siempre*.

Ahora haz doble click en *Program.cs*, que es el punto de entrada de la aplicación. Tenemos que modificar *Program.cs* para que al iniciar la aplicación se ejecute nuestro archivo *aplicacion.py*:

using System;

using Microsoft.Scripting.Hosting;

namespace ironapp

{

static class Program

{

[STAThread]

static void Main()

{

ScriptEngine engine = IronPython.Hosting.Python.CreateEngine();

ScriptRuntime runtime = engine.Runtime;

ScriptScope scope = runtime.CreateScope();

ScriptSource script = engine.CreateScriptSourceFromFile("aplicacion.py");

var compiled = script.Compile();

compiled.Execute(scope);

runtime.Shutdown();

}

}

}

Si ahora presionas **F5** se compilará y ejecutará la aplicación, pero se cerrará inmediatamente porque todavía no hemos escrito nada en el archivo *aplicacion.py*, y hemos editado *Program.cs* para que lo único que haga sea que *ironpython* ejecute nuestro *script*.

En este punto tienes dos opciones. Fíjate que en tu *proyecto ironapp* hay un formulario vacío que puedes *modificar mediante el editor visual del IDE*. Las particularidades de este formulario vienen definidas en los archivos *Form1.cs*, *Form1.Designer.cs* y *Form1.resx*. Este formulario está codificado en *C#*, no en *python*.

1. Puedes utilizarlo, es decir, añadir controles a tu formulario desde *visual studio* arrastrando y soltando, para luego *añadirle funcionalidad en el archivo aplicacion.py*.
2. Puedes eliminar los archivos *Form1.cs*, *Form1.Designer.cs* y *Form1.resx* para crear una *aplicación ironpython pura*.

**Utilizando *Form1.cs*, *Form1.Designer.cs* y *Form1.resx***

Edita el archivo *aplicacion.py* de la siguiente manera:

# -\*- coding: utf-8 -\*-

import clr

clr.AddReference('System.Windows.Forms')

clr.AddReference('ironapp') # namespace ironapp definido en Program.cs

from System.Windows.Forms import \*

from ironapp import Form1

app = Form1()

Application.EnableVisualStyles()

Application.Run(app)

Presiona **F5** y se compilará y ejecutará la aplicación, mostrando el *formulario Form1*.

Desde *visual studio*, añade un botón a *Form1* (arrastrando y soltando desde el *cuadro de herramientas* a la izquierda del *IDE*). Selecciona el botón y en el *Explorador de propiedades* (en la parte derecha del *IDE*) aparecerán sus propiedades. Modifica las siguientes propiedades del botón:

* (Name) - Es el identificador del botón, es decir, el nombre del botón en el código fuente. Asigna el valor boton para referirte a él como self.boton en el *código ironpython*.
* Text - Es el texto que aparece escrito en el botón. Asigna el valor Saludar.
* Modifiers - **Muy importante**. Para poder *utilizar desde ironpython* controles que hemos añadido al formulario desde *visual studio*, estos controles deben ser *públicos*. Asigna el valor Public.

Desde el *Explorador de propiedades* puedes inspeccionar todas las propiedades de los *controles WinForms*. Por ejemplo, además de las propiedades del botón que ya hemos modificado, puedes buscar su propiedad Dock y hacer que el botón ocupe todo el espacio disponible, tal y como hicimos en la sección anterior desde código, pero ahora de forma mas *visual* desde el *IDE*.

Ahora edita el archivo *aplicacion.py* de la siguiente forma:

# -\*- coding: utf-8 -\*-

import clr

clr.AddReference('System.Windows.Forms')

clr.AddReference('ironapp')

from System.Windows.Forms import \*

from ironapp import Form1

class App(Form1):

def \_\_init\_\_(self):

self.Text = 'IronApp' # Texto/Título del formulario

self.boton.Click += lambda \*args: MessageBox.Show('Hola desde IronPython! :D') # Añadimos funcionalidad al botón con código python

app = App()

Application.EnableVisualStyles()

Application.Run(app)

Presiona **F5** y haz click en el botón. Todo funciona perfectamente!

Hemos creado la *GUI* de la aplicación de forma *visual* desde el *IDE*, y mediante *código python* hemos añadido funcionalidad al botón del formulario: self.boton.Click += lambda \*args: MessageBox.Show('Hola desde IronPython! :D').

Por otra parte, fíjate en la asignación self.Text = 'IronApp'. De esta manera conseguimos modificar el texto/título de la aplicación desde *código python*, pero podríamos haber asignado este *título* desde el *Explorador de propieades de visual studio*.

**No utilizando *Form1.cs*, *Form1.Designer.cs* ni *Form1.resx***

Si no vas a utilizar *Form1.cs*, *Form1.Designer.cs* y *Form1.resx*, que son los archivos que utiliza *visual studio* para administrar el formulario que se crea dentro del *IDE*, elimínalos. Puedes crear una *aplicación ironpython pura*:

# -\*- coding: utf-8 -\*-

import clr

clr.AddReference('System.Windows.Forms')

from System.Windows.Forms import \*

class App(Form):

def \_\_init\_\_(self):

self.Text = 'IronApp'

self.boton = Button()

self.boton.Text = 'Saludar'

self.boton.Dock = DockStyle.Fill

self.boton.Click += lambda \*args: MessageBox.Show('Hola desde IronPython! :D')

self.Controls.Add(self.boton)

app = App()

Application.EnableVisualStyles()

Application.Run(app)

La aplicación está completamente escrita en *IronPython*, a excepción del punto de entrada de la aplicación (*Program.cs*). Compila y ejecuta presionando **F5**.

**RESUMEN**

En este artículo hemos visto cómo crear aplicaciones con *IronPython*, que es una implementación de *python2.7* escrita en *C#*. Te recomiento de utilices el *REPL de IronPython* para investigar y aprender mas sobre esta fantástica implementación.

Para crear aplicaciones personales, o que solamente vas a instalar en ordenadores que tú mismo administras, puedes utilizar *ipy* e *ipyc* (incluso puedes simplemente utilizar *ipy* y nunca compilar aplicaciones para ser distribuidas). En caso de compilar con *ipyc*, para instalar aplicaciones en equipos que no tienen instalado *ironpython*, te recomiendo que si la aplicación es muy pequeña leas un poco de documentación para no tener que utilizar la *librería estándar de ironpython*. Por el contrario, si creas una aplicación de gran tamaño, los *megabytes* adicionales de la *stdlib* serán despreciables y merece la pena utilizarla.

Si quieres distribuir aplicaciones a *desconocidos*, por ejemplo a través de *Internet*, te recomiendo que desarrolles tu aplicación desde *visual studio*, añadiendo *ironpython* desde *NuGet*. La razón es que los programas empaquetados con *ipyc* serán reconocidos como maliciosos por muchos antivirus, de manera que los potenciales usuarios de tu aplicación podrían desconfiar de tí como desarrollador.

Compilar desde *visual studio* presionando **F5** no tiene este problema, y es la mejor opción para crear *aplicaciones profesionales*. En este caso, la aplicación no será *standalone*, pero puedes crear un instalador facilmente con *[InstallForge](https://installforge.net/" \t "_blank)* o con *[InnoSetup](https://jrsoftware.org/isinfo.php" \t "_blank)* (recomiendo *InnoSetup* porque, aunque es un poco mas difícil, los instaladores que produce son *mas limpios* a ojos de los antivirus).

Recomiendo que empieces creando la *interfaz gráfica de tus aplicaciones* desde *visual studio*, aprovechando la vista diseño, ya que el *Explorador de propiedades* es una buena vía para aprender qué propiedades, así como qué eventos soportan, los distintos controles que podemos añadir a un formulario.

No obstante, con el tiempo querrás hacer cosas mas avanzadas, y preferirás que tus aplicaciones sean *puro ironpython*. Personalmente yo suelo aprovechar *Form1.cs*, *Form1.Designer.cs* y *Form1.resx* (o sea el formulario que puedes editar visualmente) para crear la *ventana principal de la aplicación*. Pero para crear ventanas secundarias que se abren en respuesta a eventos prefiero crearlas completamente en *python*.

Espero que te haya gustado este artículo, y que haya sido de utilidad para tí. Intentaré escribir mas sobre *IronPython* próximamente.

Un saludo!

**CONTINÚA LEYENDO:**

* [IronPython - Página principal](https://tecnobillo.com/sections/python-en-windows/ironpython/ironpython.html)
* [Form - Propiedades](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/form-propiedades/form-propiedades.html)
* [Form - Menú](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/form-menu/form-menu.html)
* [MessageBox](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/messagebox/messagebox.html)
* [NotifyIcon](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/notifyicon/notifyicon.html)
* [Diálogos de selección](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/dialogs/dialogs.html)
* [Eventos de Form y Controles](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/eventos/eventos.html)
* [Diseño GUI](https://tecnobillo.com/sections/python-en-windows/ironpython/sections/disenho-gui/disenho-gui.html)

LIKEDISLIKE

**¿Te ha gustado esta página?**

**comparte esta página**

Principio del formulario

** https://www.paypal.com/es_ES/i/scr/pixel.gif**

Final del formulario

**mis redes sociales**

[**tecnobillo**22 de Mayo de 2022 a las 11:48:13](https://tecnobillo.com/apps/contacta-con-tecnobillo/contacta-con-tecnobillo.html)