

2nd December 2020 Iron Python. Uso de código Python en VB.NET

Python es un lenguaje muy versátil y popular, con una sintaxis limpia y una curva de aprendizaje corta por su sencillez. Por eso a veces queremos usarlo en diferentes ámbitos. En este caso, para poder ejecutar código Python directamente desde .NET, Microsoft creó [IronPython](https://es.wikipedia.org/wiki/IronPython) [https://es.wikipedia.org/wiki/IronPython] en el año 2006. Básicamente es una implementación de Python escrita en C#. Esto permite que un programa creado en Python pueda ser interpretado por IronPython, y podemos usar ese código en una aplicación .NET y por consiguiente, puede ser compilado a Bytecode, con la ventaja de que será un código que se ejecute más rápido que su equivalente en el interprete Python.

Y es que admitámoslo, una de las desventajas de Python es que no se ejecuta de la manera más eficiente. Una alternativa a IronPython para optimizar el desempeño del código Python es [PyPy](https://es.wikipedia.org/wiki/PyPy) [https://es.wikipedia.org/wiki/PyPy], una tecnología que permite la compilación de código Python Just In Time a código de máquina. [PyPy](https://www.arsys.es/blog/pypy/) [https://www.arsys.es/blog/pypy/] surge en 2007 así que ya tiene algo de tiempo entre nosotros. Pero como dijimos, usa una compilación Just In Time, no genera compilados para distribuirlos entre sistemas. Así que debes tener instalado PyPy en la máquina donde lo uses, como si fuera un interprete Python. Esto es ya que por el dinamismo inherente de Python, no hay forma de emitir el código JITted resultante como un binario independiente y reutilizarlo. Cada programa tiene que ser compilado para cada ejecución. Si deseas compilar Python en un código más rápido que pueda ejecutarse como una aplicación independiente, debes usar Cython para transformar Python a código C. Otra desventaja de PyPy es que no tiene soporte completo para todas las librerías de Python, que generalmente son creadas en C.

Ahora veamos un ejemplo en VB.NET de como usar IronPython. Para ello comenzaremos creando dos programas Python, que solo contendrán una clase. Estas clases las usaremos directamente en .NET. primero creo el programa calc.py:

```
class Calculator:
    def suma(self, argA, argB):
        return argA+argB
    def resta(self, argA, argB):
        return argA-argB
    def multiplica(self, argA, argB):
        return argA*argB
    def division(self, argA, argB):
        return argA/argB
```

Como vemos esta primer clase no ofrece nada especial, es una sencilla calculadora que realiza las 4 operaciones aritméticas fundamentales. Hay que acotar que IronPython no ofrece soporte completo para todas las librerías de Python, así que por ahora no ejecutaré nada externo ni complejo.

Ahora veamos el segundo archivo, llamado triangulo.py:

```
class triangulo:
    def __init__(self, lado1, lado2, lado3):
        self.lado1 = lado1
        self.lado2 = lado2
        self.lado3 = lado3
    def perimetro(self):
        return self.lado1 + self.lado2 +self.lado3
    def area(self,altura):
        return self.lado1*altura/2
    def tipoTriangulo(self):
        if self.lado1 != self.lado2 and self.lado1 != self.lado3 and self.lado2 != self.lado3:
            return "escaleno"
        if self.lado3 == self.lado1 and self.lado3 == self.lado2:
```

```

        return "equilatero"
    if self.lado1== self.lado2 or self.lado2==self.lado3:
        return "isosceles"

```

Una vez más es una clase sencilla, agregue un constructor para ver como pasarle los datos desde VB.NET, y tengo 3 métodos, uno que me retorna perímetro del triángulo, otro su área y uno que me dice que tipo de triángulo es.

Para probar IronPython, usaré una aplicación de consola. Una vez creada es importante agregar las siguientes librerías, que están en el directorio de instalación de IronPython (olvide decirlo, debes instalar IronPython en tu máquina antes):

-IronPython.dll

-IronPython.Modules.dll

-Microsoft.Dynamic.dll

-Microsoft.Scripting.dll

Una vez hecho esto, podemos poner el siguiente código, pero antes hagamos los siguientes Imports:

Imports IronPython.Hosting

Imports IronPython.Runtime.Types

Imports Microsoft.Scripting.Hosting

```

Sub Main()
    Dim engine = Python.CreateEngine()

    Dim source As ScriptSource = engine.CreateScriptSourceFromFile("C:\Python\calc.py")
    Dim scope As ScriptScope = engine.CreateScope()
    source.Execute(scope)

    'Creo la clase
    Dim Calculator As Object = scope.GetVariable("Calculator")
    'instancio un objeto calculadora
    Dim calc = Calculator()
    Dim a As Double = 7.5
    Dim b As Double = 2.5
    Dim Res As Double

    Res = calc.suma(a, b)
    Console.WriteLine("{0} + {1} = {2}", a, b, Res)
    Res = calc.resta(a, b)
    Console.WriteLine("{0} - {1} = {2}", a, b, Res)
    Res = calc.multiplica(a, b)
    Console.WriteLine("{0} * {1} = {2}", a, b, Res)
    Res = calc.division(a, b)
    Console.WriteLine("{0} / {1} = {2}", a, b, Res)

```

[https://1.bp.blogspot.com/-iZ0-myyvztrc/X8feu5TIGFI/AAAAAAAAA7sw/KTDW_hMhc-01qfyc2YImOZ4QxlpZQ0JsgCLcBGAsYHQ/s685/calculaPython.png]

Primero creamos un engine con `Python.CreateEngine`. Luego definimos el script que será nuestra fuente, usando `CreateScriptSourceFromFile` pasándole el path de nuestro archivo Python.

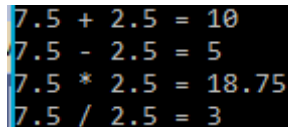
Necesitamos un scope, que creamos usando `CreateScope`, luego en nuestro script damos `Execute` pasándole el scope.

Con eso ya podemos llamar los métodos o clases del script Python, por ejemplo defino la clase `Calculator`, usando `scope.GetVariable` y pasándole el nombre de la clase en Python. Esta se define como `Objetc`, dinámicamente va a recibir desde Python sus métodos y propiedades.

Es importante el siguiente paso, ya tenemos una clase Python para usar en nuestro programa .NET, pero aun no hemos instanciado ningún objeto de la misma. Así que instanciamos `calc` como una `Calculator`.

Una vez hecho esto podemos usar el objeto de la clase. Por ejemplo, aquí ejecutamos todos sus métodos igual que si fuera un objeto de VB.NET. Por que de hecho lo es, podemos usar todo el código Python como si fuera nativo. De esa manera se puede reutilizar código escrito en Python.

El resultado sería:



```
7.5 + 2.5 = 10
7.5 - 2.5 = 5
7.5 * 2.5 = 18.75
7.5 / 2.5 = 3
```

[<https://1.bp.blogspot.com/-20G1jEFyZ7M/X8fiZpCiazI/AAAAAAAAA7s8/wvcYyfJcGzUE2CaG9sJ7W-WWfJ9Pa87-wCLcBGAsYHQ/s159/calculaPythonRes.png>]

Como vemos el resultado fue el esperado. Y la salida la tuvimos en VB.NET.

Veamos un ejemplo un poco más complejo:

```

source = engine.CreateScriptSourceFromFile("C:\Python\triangulo.py")
scope = engine.CreateScope()
source.Execute(scope)
'Creo la clase
Dim Figura As Object = scope.GetVariable("triangulo")
'instancio un objeto triangulo
Console.WriteLine("Ingresa lado1:")
Dim lado1 As Integer = Console.ReadLine()
Console.WriteLine("Ingresa lado2:")
Dim lado2 As Integer = Console.ReadLine()
Console.WriteLine("Ingresa lado3:")
Dim lado3 As Integer = Console.ReadLine()
Dim triangulo = Figura(lado1, lado2, lado3)
Console.WriteLine("Ingresa la altura:")
Dim altura As Integer = Console.ReadLine()

Dim perimetro = triangulo.perimetro
Console.WriteLine("El perimetro del triangulo es: {0}", perimetro)
Dim area = triangulo.area(altura)
Console.WriteLine("El área del triangulo es: {0}", area)

Dim tipo = triangulo.tipoTriangulo
Console.WriteLine("Es un triángulo: {0}", tipo)
Console.WriteLine("Presiona cualquier tecla para finalizar...")
Console.ReadLine()

```

[[https://1.bp.blogspot.com/-](https://1.bp.blogspot.com/-euOvrKWJfDs/X8fi1esT4EI/AAAAAAAAA7tE/ied6xXZOWUMOlrvLRxJwwugemN9tYuf4QCLcBGAsYHQ/s594/trianguloPython.png)

[euOvrKWJfDs/X8fi1esT4EI/AAAAAAAAA7tE/ied6xXZOWUMOlrvLRxJwwugemN9tYuf4QCLcBGAsYHQ/s594/trianguloPython.png](https://1.bp.blogspot.com/-euOvrKWJfDs/X8fi1esT4EI/AAAAAAAAA7tE/ied6xXZOWUMOlrvLRxJwwugemN9tYuf4QCLcBGAsYHQ/s594/trianguloPython.png)]

Es muy similar, pero ahora estoy leyendo datos por consola en VB.NET. Los utilizo para instanciar el objeto triangulo, ya que la clase de Python tiene un constructor. En este caso se identifica por el `__init__`. A ese constructor le envío los 3 datos que leí desde VB.NET y son asignados a propiedades de la clase en Python. Recuerda para usar propiedades de clase en Python, se usa la palabra reservada `self` que indica la instancia actual (equivale al `Me` de VB.NET).

Así ya puedo ejecutar el método `perimetro` que trabaja con esas propiedades. Para el método `area`, le envío además la altura del triángulo, y usando un lado como base, calcula el área. Finalmente invoco `tipoTriangulo`. Todos los datos devueltos puedo usarlos en mi programa, en este caso los imprimo en pantalla.

El resultado sería:

```
Ingresa lado1:
5
Ingresa lado2:
7
Ingresa lado3:
8
Ingresa la altura:
5
El perimetro del triangulo es: 20
El área del triangulo es: 12
Es un triángulo: escaleno
Presiona cualquier tecla para finalizar...
```

[[https://1.bp.blogspot.com/-](https://1.bp.blogspot.com/-DxHDv1CIVpA/X8fkmcczAkl/AAAAAAAAA7tQ/cF--La3Bj1oVcvxmAjU-PPSWcVcGT-U_gCLcBGAsYHQ/s354/PythonVB.png)

[DxHDv1CIVpA/X8fkmcczAkl/AAAAAAAAA7tQ/cF--La3Bj1oVcvxmAjU-PPSWcVcGT-U_gCLcBGAsYHQ/s354/PythonVB.png](https://1.bp.blogspot.com/-DxHDv1CIVpA/X8fkmcczAkl/AAAAAAAAA7tQ/cF--La3Bj1oVcvxmAjU-PPSWcVcGT-U_gCLcBGAsYHQ/s354/PythonVB.png)]

Como podemos ver los cálculos son correctos. Así de fácil podemos integrar Python con .NET gracias a IronPython.

Publicado hace 2nd December 2020 por Unknown

Etiquetas: [__init__](#), [_self](#), [Cython](#), [IronPython](#), [POO](#), [PyPy](#), [Python](#), [VB.NET](#)



Agregar un comentario



Escribir comentario