

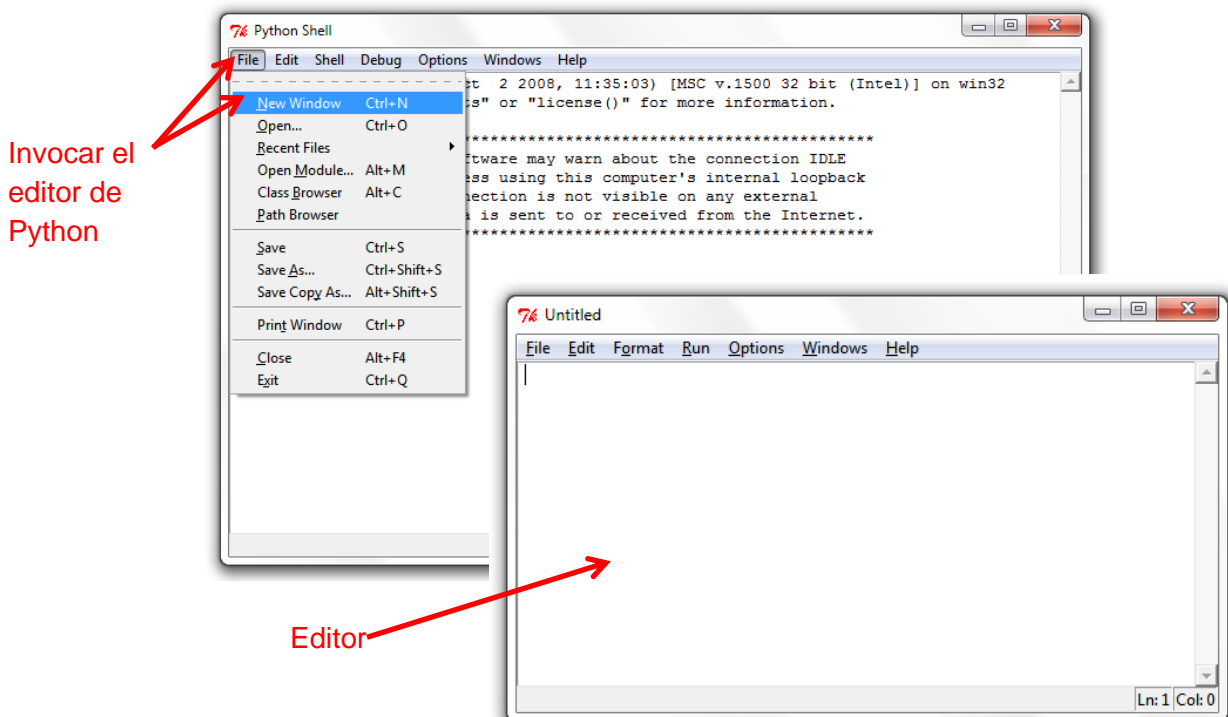
Estructura de programas en Python

Hasta ahora hemos trabajado con la versión interactiva del intérprete de Python ingresando sentencias para ejecutar funciones, evaluar el comportamiento de los operadores aritméticos, operar con números enteros y no enteros entre otras actividades. Pero esto tiene una gran desventaja: al cerrar el ambiente interactivo, **perdemos las sentencias** que hemos estado utilizando.

Ahora aprenderemos a crear conjuntos ordenados de sentencias Python que permiten obtener un resultado objetivo y que podemos aplicar una y otra vez a diferentes conjuntos de datos. A estos conjuntos le llamaremos **programas**.

Para poder utilizar un programa varias veces, éste tiene que estar guardado en memoria permanente (como un disco duro o un *pendrive*). Los programas en Python se almacenan en archivos con extensión **".py"**.

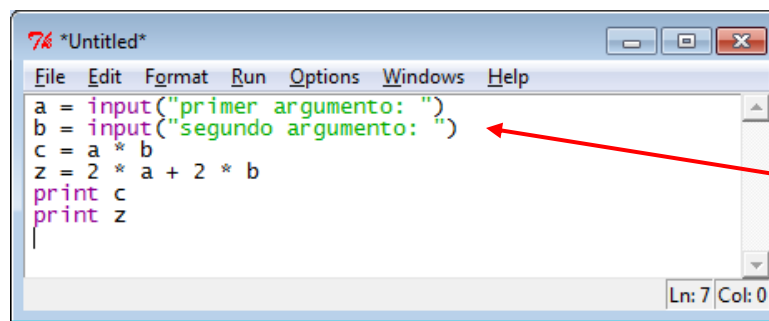
Para escribir un programa necesitaremos de un **editor**. El ambiente IDLE de Python que hemos estado utilizando provee uno haciendo *click* en la opción **"File → New Window"**, que abre una segunda ventana con el editor de programas. Ahí podemos escribir las sentencias que necesitamos, en el orden que deben ejecutarse, y guardarlo con algún nombre.



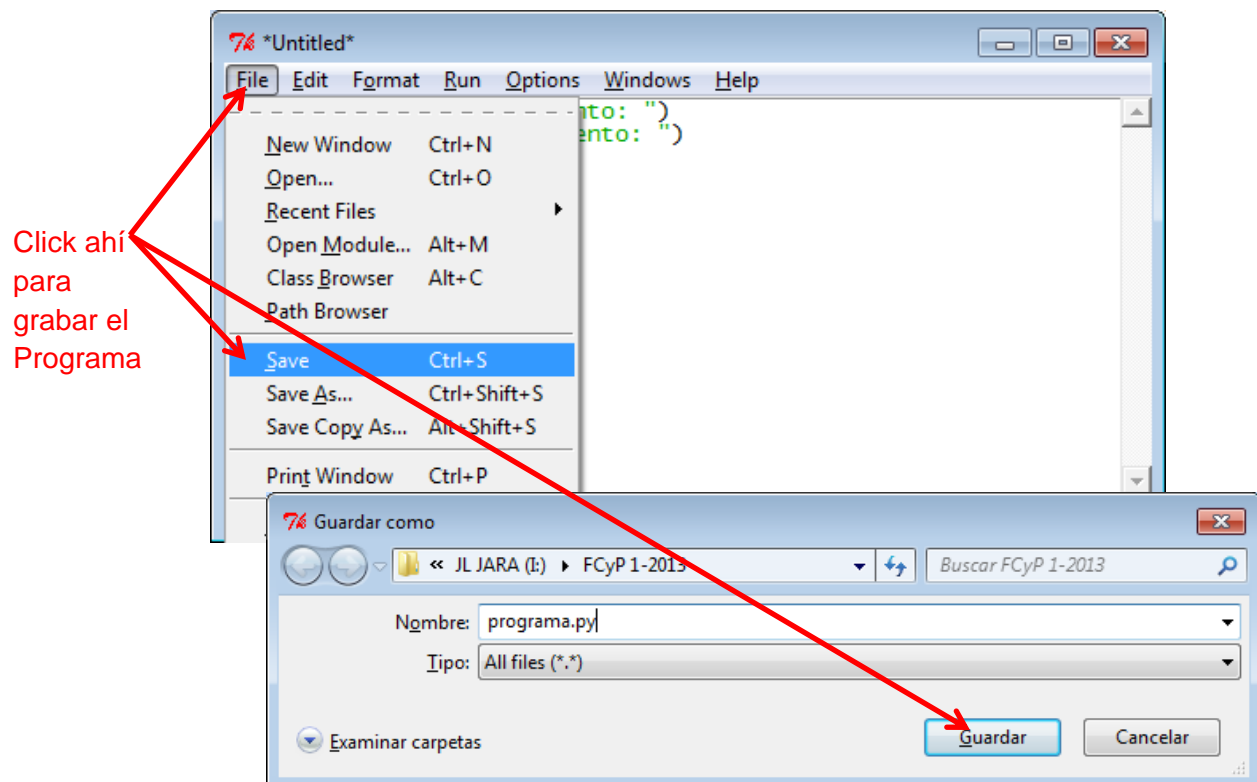
Probemos ahora ingresar un programa en el editor.

Programa.py

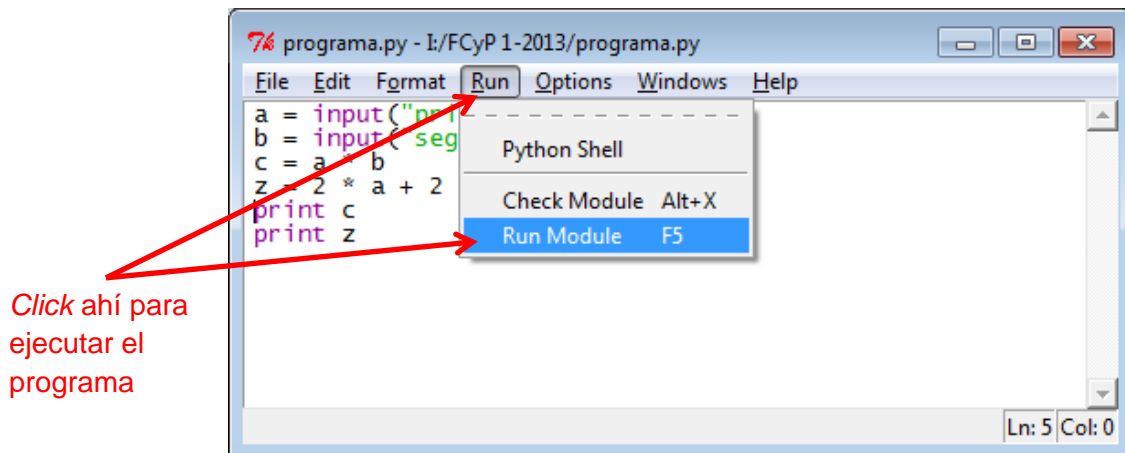
```
a = input("primer argumento: ")
b = input("segundo argumento: ")
c = a * b
z = 2 * a + 2 * b
print c
print z
```



Ahora grabamos el programa haciendo *click* en la opción "**File** → **Save as**" y otorgando un nombre al programa (recordando agregar ".py" al final).

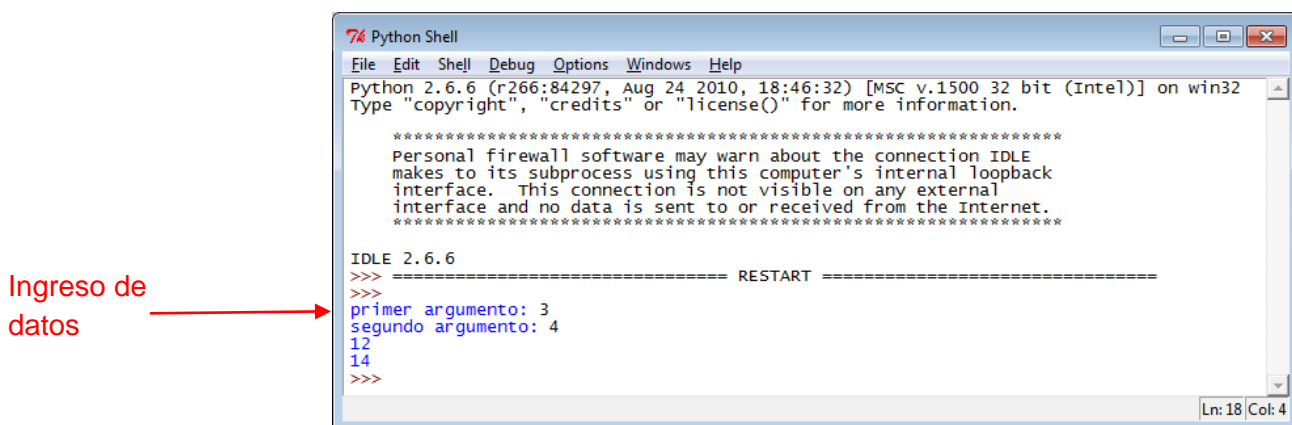


¡Ya hemos hecho nuestro primer programa! Ahora necesitamos “ejecutarlo” o hacerlo “correr”. Estos términos informáticos se usan para decir que solicitaremos al intérprete de Python que siga cada una de las sentencias del programa. Para ejecutar el programa debemos hacer *click* en la opción “Run → Run Module” (o presionar la tecla F5).



Al ejecutar el programa, el ambiente interactivo de Python interpreta, una a una, cada sentencia en el orden en que aparecen. En el programa ejemplo, las primeras dos sentencias corresponden a la función nativa `input()`, que muestra un **mensaje en pantalla** (su argumento) y devuelve un **valor ingresado por teclado**. Es decir, esta función se utiliza para solicitar **entrada de datos** al usuario.

Ingresemos los valores 3 y 4. Podemos ver que Python nos entrega dos valores resultados: 12 y 14. Esto se debe a la sentencia `print()` que aparece como última sentencia del programa. Para obtener estos valores, el intérprete de Python tuvo que evaluar previamente las expresiones que dieron valor a las variables que se muestran.



¡Hemos ejecutado nuestro primer programa con éxito!

Claro que los programadores no hacen programas por nada, sino que tienen un **propósito**.

Pregunta 1

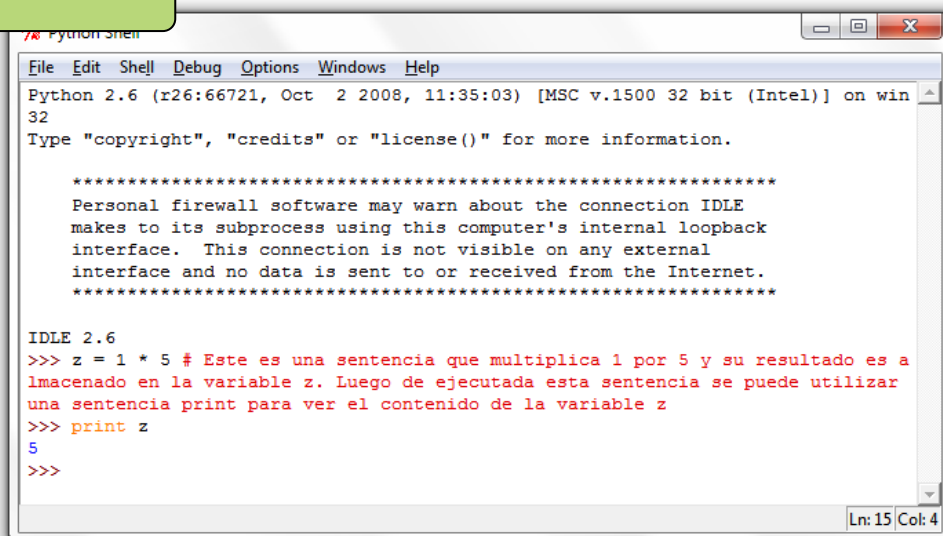
Con esta información, ya podemos resolver la primera pregunta del apunte.

Podemos darnos cuenta que no es fácil descubrir el propósito de este programa. Esto es porque quien escribió este programa no era buen programador y no consideró **buenas prácticas de programación** básicas que ayudan a mantener una **buena legibilidad** del programa, lo que harían el trabajo de **revisarlo y entenderlo** más fácil.

Para aumentar la legibilidad de un programa podemos intercalar comentarios que expliquen lo que hace el programa y cada una de sus partes. Los comentarios son para los **humanos**, no para Python, quien los ignora al momento de la ejecución.

Podemos incorporar comentarios escribiendo el símbolo **#** (*hash*, *sharp* o gato), todo lo que esté hacia la derecha de este símbolo será considerado un comentario hasta el final de la línea.

Ejemplo 1



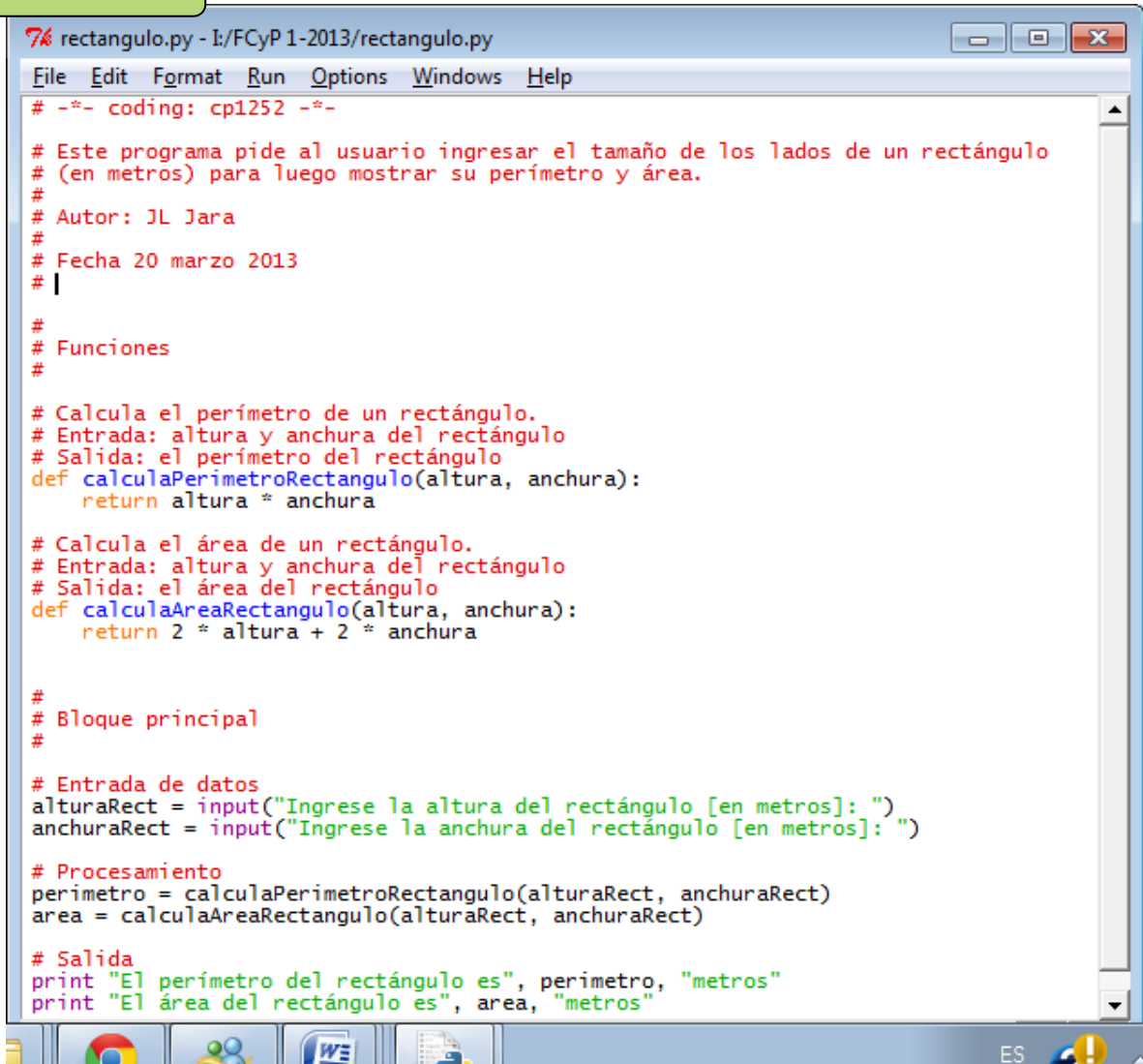
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.6 (r26:66721, Oct 2 2008, 11:35:03) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 2.6
>>> z = 1 * 5 # Este es una sentencia que multiplica 1 por 5 y su resultado es a
lmacenado en la variable z. Luego de ejecutada esta sentencia se puede utilizar
una sentencia print para ver el contenido de la variable z
>>> print z
5
>>>
```

Un poco más arriba nos preguntamos qué hacía el programa ejemplo. Tal vez incorporando algunos comentarios podamos mejorar su legibilidad.

Ejemplo 2



```
rectangulo.py - I:/FCyP 1-2013/rectangulo.py
File Edit Format Run Options Windows Help
# -*- coding: cp1252 -*-

# Este programa pide al usuario ingresar el tamaño de los lados de un rectángulo
# (en metros) para luego mostrar su perímetro y área.
#
# Autor: JL Jara
#
# Fecha 20 marzo 2013
# |

#
# Funciones
#

# Calcula el perímetro de un rectángulo.
# Entrada: altura y anchura del rectángulo
# Salida: el perímetro del rectángulo
def calculaPerimetroRectangulo(altura, anchura):
    return altura * anchura

# Calcula el área de un rectángulo.
# Entrada: altura y anchura del rectángulo
# Salida: el área del rectángulo
def calculaAreaRectangulo(altura, anchura):
    return 2 * altura + 2 * anchura

#
# Bloque principal
#

# Entrada de datos
alturaRect = input("Ingrese la altura del rectángulo [en metros]: ")
anchuraRect = input("Ingrese la anchura del rectángulo [en metros]: ")

# Procesamiento
perimetro = calculaPerimetroRectangulo(alturaRect, anchuraRect)
area = calculaAreaRectangulo(alturaRect, anchuraRect)

# Salida
print "El perímetro del rectángulo es", perimetro, "metros"
print "El área del rectángulo es", area, "metros"
```

Si observamos el programa con comentarios podemos notar que ha mejorado mucho su legibilidad: con una **simple lectura** podemos saber **el objetivo** para el cual fue creado.

Otros aspectos que ayudan a la legibilidad:

- Los nombres que se usen en el programa debe ser **indicativos**. Una función denominada `z()` es misteriosa, en cambio una función que se llama `calculaVolumenEsfera()` es indicativo del objetivo de la función. Lo mismo sucede con el nombre de las variables y constantes.
- Los mensajes dirigidos al usuario deben ser informativos y claros.

- En muchos aspectos, el intérprete de Python no requiere un programa muy estructurado, sin embargo, un programa que diferencia claramente **secciones de código distintas** tiene mejor legibilidad. Tratemos siempre de mantener el siguiente esquema de organización:

Encabezado	<ul style="list-style-type: none"> Descripción general Autor(es) Fecha/versión
Definición de constantes	<ul style="list-style-type: none"> Constantes
Definición de funciones propias	<ul style="list-style-type: none"> Funciones
Bloque principal	<ul style="list-style-type: none"> Entrada de datos
	<ul style="list-style-type: none"> Procesamiento
	<ul style="list-style-type: none"> Entrega de los resultados (salida)

Necesitamos hacer tres observaciones del código del programa `rectangulo.py`. La primera línea del programa contiene el texto `"# -*- coding: cp1252 -*-"`, que no es un comentario para aumentar la legibilidad del programa, sino que una indicación al intérprete de Python de el programa usa un **juego de caracteres con tildes**. Esto es necesario para que nuestros mensajes se desplieguen correctamente en pantalla y funciona en el sistema operativo Windows™. En otros sistemas operativos esto generalmente se consigue escribiendo el texto `"# -*- coding: utf-8 -*-"` en la primera línea del programa e indicando al editor que se esté usando que guarde los archivos `".py"` con este juego de caracteres.

La segunda observación es que cuando la sentencia `print` se usa con varios argumentos separados con comas, estas comas son reemplazadas por un espacio en la salida a pantalla. Debemos considerar esto para no obtener espacios innecesarios en nuestros mensajes al usuario.

Pregunta 2

Ahora es posible resolver la pregunta número 2 del apunte. Recuerden trabajar como equipo.

Finalmente, debemos saber que la función nativa `input()` no sólo acepta valores constantes como entrada, sino que **cualquier expresión válida**. El siguiente ejemplo

muestra la ejecución del programa `rectangulo.py` cuando ingresamos las expresiones Python `12 / 4.0` y `2 ** 3` respectivamente.

Ejemplo 3

```
>>> ===== RESTART =====  
>>>  
Ingrese la altura del rectángulo [en metros]: 12 / 4.0  
Ingrese la anchura del rectángulo [en metros]: 2 ** 3  
El perímetro del rectángulo es 24.0 metros  
El área del rectángulo es 22.0 metros  
>>>
```

Podemos ver que Python primero evalúa las expresiones ingresadas para cada valor antes de realizar la asignación.

Pregunta 3

Trabaja con tu grupo en la última pregunta de la actividad de hoy