

## Apéndice A. Uso de un entorno integrado de desarrollo de programas



**Michael González Harbour**  
**José Javier Gutiérrez García**  
**José Carlos Palencia Gutiérrez**  
**José Ignacio Espeso Martínez**  
**Adolfo Garandal Martín**

Departamento de Ingeniería  
Informática y Electrónica

Este material se publica con licencia:  
[Creative Commons BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/)

# Apéndice

---

## *Apéndice A. Uso de un entorno integrado de desarrollo de programas*

- Entorno de desarrollo de programas ***spyder***
- Gestión de proyectos
- Analizar, cargar y ejecutar el programa
- La depuración
- Generación de documentos

# A.1 Entorno de desarrollo spyder

C:/Users/Administrador/Documents/docencia/prog-python/ejemplos/cap2 - Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda

Explora... Editor - C:\Users\Administrador\Documents\docencia\prog-python\ejemplos\cap... Explorador de variables

cap2

temp.py plano\_inclinado.py muelle.py

28 #Constantes:  
29 # CTE\_ELASTICA: Constante elástica del muelle  
30 # G: Gravedad terrestre, en m/s^2  
31 CTE\_ELASTICA: float = 0.52  
32 G: float = 9.8  
33  
34 def fuerza\_muelle(alt: float)-> float:  
35 """  
36 Calcula la fuerza que ejerce el muelle  
37 dada la altura de su extremo  
38  
39 Args:  
40 alt: la altura del extremo del muelle  
41 Returns:  
42 la fuerza que ejerce el muelle, en N  
43 """  
44  
45 # retornamos la fuerza calculada por medio de la ley de Hooke  
46 return -CTE\_ELASTICA\*alt  
47  
48 def avanza\_tiempo(alt: float, vel: float, mas: float, tiempo: float)-> (float, float):  
49 """  
50 Calcula la nueva altura y velocidad de la masa  
51 transcurrido un tiempo especificado  
52 """  
53

Nombre Tipo Tamaño Valor

**Ventana multiusos**

Explorador de varia... Explorador de archi... Ay... Análisis estático del cód...

Terminal de IPython

Terminal 1/A

Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more information.  
IPython 7.2.0 -- An enhanced Interactive Python.  
In [1]:

**Terminal Python**

Terminal de IPython Historial de comandos

Permisos: RW Fin de línea: CRLF Codificación: UTF-8 Línea: 9 Columna: 1 Memoria: 41 %

# A.2. Gestión de proyectos

---

Para programar utilizando *spyder* es conveniente crearse un “*proyecto*” por programa o práctica

El proyecto es un directorio en el disco donde se guarda toda la información que se va generando relacionada con el programa:

- código fuente
- módulos compilados
- documentación
- imágenes
- etc.

# Creación de un nuevo proyecto

---

Elegir **Proyectos=>Nuevo Proyecto**, y luego situarse en el directorio deseado y darle al proyecto un nombre. Ej:

`practica1`

Añadir un nuevo módulo al proyecto. Para ello, con botón derecho sobre el nombre del proyecto seleccionar **Crear Nuevo -> Módulo**

- luego darle un nombre

Para editar el módulo, hacer “doble click” sobre él en el explorador del proyecto

# Ejercicio 1: editar este programa

---

```
# -*- coding: utf-8 -*-  
"""
```

Pone un mensaje de bienvenida en pantalla

```
@author: <tu nombre>  
@date   : feb/2019  
"""
```

```
def main():  
    """
```

Hola Mundo

Este programa pone un mensaje en pantalla  
"""

```
print("Hola, ¿qué tal?")
```

# A.3 Analizar, cargar y ejecutar

---

## Analizar

- el editor va mostrando errores y avisos al escribir
- hay un analizador estático de las reglas de estilo: Pulsar F8
  - es muy importante acordarse de usarlo

Para cargar el programa en el intérprete: Pulsar F5 o el botón 

- las instrucciones sueltas se ejecutan al cargar
- pero las funciones o clases no

Para ejecutar un `main()`, teclear desde el intérprete:

```
main()
```

Para ejecutar una función, teclear desde el intérprete:

```
funcion(parámetros)
```

# Ejercicio 2: simulación de un muelle

---

- Meter en el proyecto el ejemplo de clase `muelle.py` que se suministra
- Cargarlo y ejecutarlo



# A.4. La depuración

---

Es el proceso de prueba del programa para localizar errores

La depuración se puede realizar:

- manualmente: insertando instrucciones de salida (`print`) que muestren el flujo de control del programa y el valor de las variables de interés
- mediante un depurador de alto nivel

El depurador de alto nivel permite:

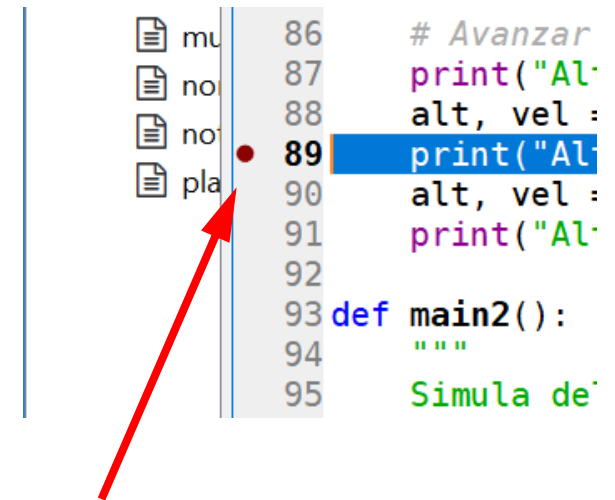
- parar el programa en los puntos deseados: *puntos de ruptura* o *breakpoints*
- ejecutar paso a paso
- visualizar el contenido de las variables
- visualizar secuencia de llamadas a funciones y sus argumentos

# Depurador de Spyder

Es un depurador para programas Python con interfaz gráfica

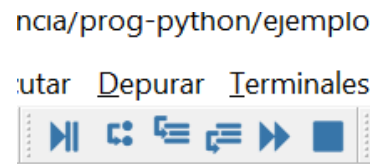
Permite introducir puntos de ruptura (*breakpoints*)

- “doble click” en la columna a la izquierda del número de línea (aparece un ●)
- se quita también con “doble click”



```
86 # Avanzar
87 print("Al
88 alt, vel :
89 print("Al
90 alt, vel :
91 print("Al
92
93 def main2():
94     """
95     Simula de
```

En el menú superior se dispone de los siguientes botones de depuración



# Depuración: Comienzo

---

Para comenzar la depuración el programa debe tener instrucciones que ejecuten el `main()`







- si no el depurador no se ejecutará

Para ello, en la ventana del editor poner una invocación al `main()` al final del módulo. Por ejemplo:

```
def main():  
    """  
    Hola Mundo  
    """  
    print("Hola, ¿qué tal?")  
  
main() # invocamos el main()
```

Añadir para depurar.  
Luego quitar,  
o proteger como se  
indica en el capítulo 4

# Botones de depuración

	Comenzar la depuración. Se detiene al principio del programa
	Ejecutar línea <sup>a</sup>
	Entrar en la función de la línea actual <sup>b</sup>
	Ejecutar hasta que la función actual termine <sup>c</sup>
	Continuar la ejecución hasta el próximo punto de ruptura <sup>d</sup>
	Finalizar la depuración <sup>e</sup>

a. Equivale al comando `n` o `next` en la consola del depurador (`ipdb>`); para repetir el comando, presionar `'enter'`

b. Equivale al comando `s` o `step`

c. Equivale al comando `r` o `return`

d. Equivale al comando `c` o `continue`

e. Equivale al comando `q`, `quit` o `exit`

# Visualizar información

---

Variables: en la ventana multiusos, pestaña "*Explorador de variables*"

Con el programa detenido en una función se pueden teclear comandos en el terminal, tras el símbolo "*ipdb>*":

- Secuencia de llamadas a funciones:
  - *w o where*: muestra la secuencia, lo más antiguo arriba
- Argumentos y variables de funciones
  - *args*: muestra los argumentos
  - *up*: cambia a la llamada anterior (más antigua)
  - *down*: cambia a la llamada posterior (más nueva)

# Ejercicio 3: depuración

---

Depurar el programa `muelle.py`

- ejecutar paso a paso
- añadir un punto de ruptura en el bucle
- continuar la ejecución hasta el punto de ruptura varias veces
  - observar cómo varían los valores de las variables y cómo crecen las listas
- quitar el punto de ruptura y continuar la ejecución hasta el final

# A.5. Generación de documentos

---

Para usar un módulo con sus funciones y clases, lo único que se necesita conocer de él es la interfaz pública:

- clases: sus nombres, atributos y métodos
- funciones y métodos: sus cabeceras y descripción de lo que hacen

Se puede extraer esta información de manera automática, por medio de herramientas de documentación

Para ello ejecutaremos `pydoc` desde una shell del sistema operativo:

```
pydoc -w nombre_modulo
```

Observar que no se pone la extensión ".py"

El Windows, usar como shell "Anaconda Prompt"

El resultado es un fichero `.html`. Se puede ver en un navegador web