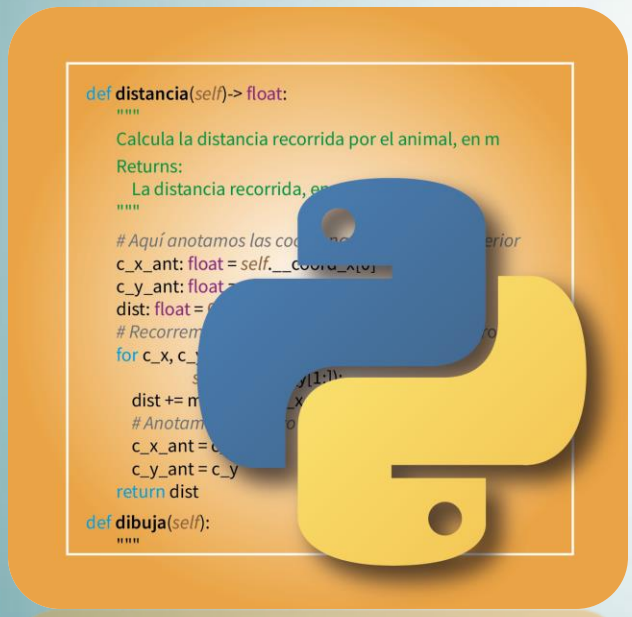
Programación

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

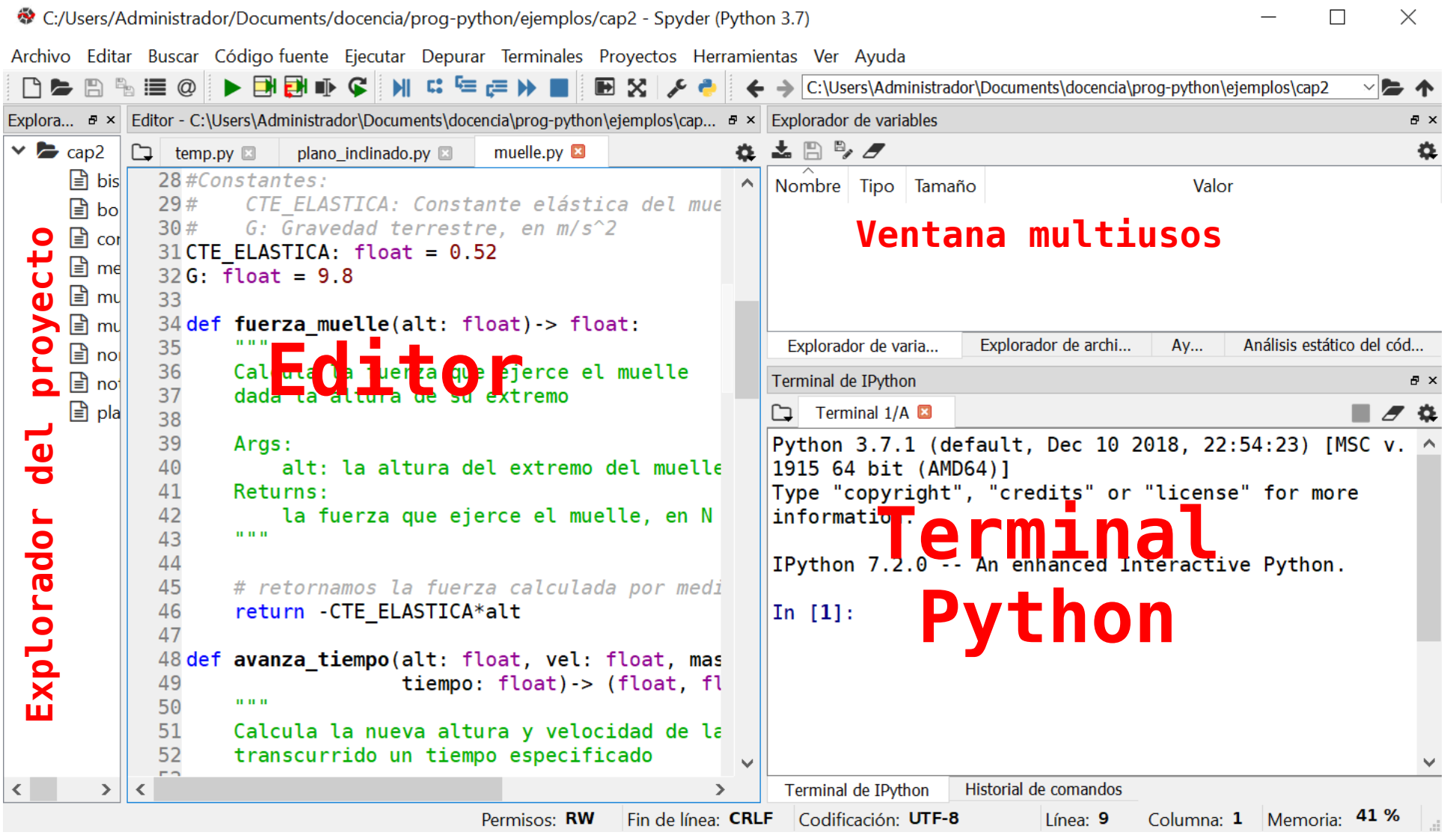
Departamento de Ingeniería Informática y Electrónica

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



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:

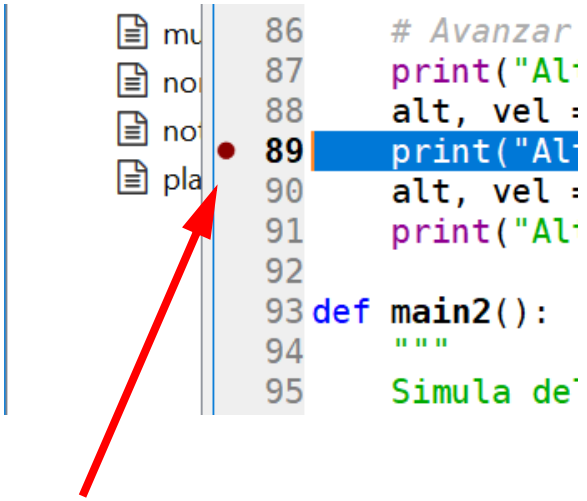
- parar el programa en los puntos deseados: *puntos de ruptura* o *breakpoints*
- ejecutar paso a paso
- visualizar el contenido de las variables
- visualizar secuencia de llamadas a funciones y sus argumentos

Depurador de Spyder

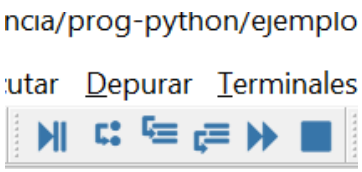
Es un depurador para programas Python con interfaz gráfica

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"



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[a]
	Entrar en la función de la línea actual[b]
	Ejecutar hasta que la función actual termine[c]
	Continuar la ejecución hasta el próximo punto de ruptura[d]
	Finalizar la depuración[e]

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