

Herramientas Computacionales para Matemática Aplicada

Curso 2020



Introducción a Phyton


Características de Python:

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.² Se trata de un lenguaje de programación **multiparadigma**, ya que soporta **orientación a objetos**, **programación imperativa** y, en menor medida, **programación funcional**. Es un lenguaje interpretado, dinámico y multiplataforma.

Es administrado por la **Python Software Foundation**. Posee una licencia de **código abierto**, denominada **Python Software Foundation License**,³ que es compatible con la **Licencia pública general de GNU** a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Biblioteca estándar [\[editar \]](#)

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas. Esto viene de la filosofía "pilas incluidas" ("*batteries included*") en referencia a los módulos de Python. Los módulos de la biblioteca estándar pueden mejorarse por módulos personalizados escritos tanto en C como en Python. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar, combinada con la habilidad de usar lenguajes de bajo nivel como C y C++, los cuales son capaces de interactuar con otras bibliotecas, Python es un lenguaje que combina su clara sintaxis con el inmenso poder de lenguajes menos elegantes.

Python	
	
Desarrollador(es)	
Python Software Foundation	
Sitio web oficial@	
Información general	
Extensiones comunes	.py , .pyc , .pyd , .pyo , .pyw , .pyz
Paradigma	Multiparadigma: orientado a objetos, imperativo, funcional, reflexivo
Apareció en	1991
Diseñado por	Guido van Rossum
Última versión estable	3.8.2 ¹ (24 de febrero de 2020 (1 mes y 2 días))
Sistema de tipos	Fuertemente tipado, dinámico
Implementaciones	CPython, IronPython, Jython, Python for S60, PyPy, ActivePython, Unladen Swallow
Dialectos	Stackless Python, RPython
Influído por	ABC, ALGOL 68, C, Haskell, Icon, Lisp, Modula-3, Perl, Smalltalk, Java
Ha influido a	Boo, Cobra, D, Falcon, Genie, Groovy, Ruby, JavaScript, Cython, Go Latino
Sistema operativo	Multiplataforma
Licencia	Python Software Foundation License

Características de Python:

Filosofía [\[editar \]](#)

Los usuarios de Python se refieren a menudo a la **filosofía de Python** que es bastante análoga a la filosofía de **Unix**. El código que siga los principios de Python se dice que es "pythonico". Estos principios fueron descritos por el desarrollador de Python **Tim Peters** en **El Zen de Python**

- Bello es mejor que feo.
- Explicito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una —y preferiblemente solo una— manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.²²
- Ahora es mejor que nunca.
- Aunque *nunca* es a menudo mejor que *ya mismo*.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (*namespaces*) son una gran idea ¡Hagamos más de esas cosas!

Tim Peters, **El Zen de Python**

CPython es la implementación oficial y más ampliamente utilizada del lenguaje de programación **Python**. Está escrita en **C**. Además de CPython, hay otras implementaciones con calidad para producción: **Jython**, escrita en **Java**; **IronPython**, escrita para el **Common Language Runtime** y **PyPy**, escrita en un subconjunto del propio lenguaje Python.

Aritmética con Python:

```
fede@fedeLaptopLenovo: ~  
File Edit View Search Terminal Help  
fede@fedeLaptopLenovo:~$ python3  
Python 3.6.9 (default, Nov 7 2019, 10:44:02)  
[GCC 8.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 4.3+3.5  
7.8  
>>> 3.2*2.1  
6.7200000000000001  
>>> round(_, 3)  
6.72  
>>> 4-2-2.1  
-0.10000000000000009  
>>> round(_, 3)  
-0.1  
>>> #con ** denotamos potenciación  
... 2**3  
8  
>>> (2.2+3.5)**3.2  
262.3007534884654  
>>> round(_, 3)  
262.301  
>>> 4/3  
1.3333333333333333  
>>> #con // denotamos la división entera  
... 25 // 3  
8  
>>> 25 % 3  
1  
>>> 
```

Python como potente calculadora científica:

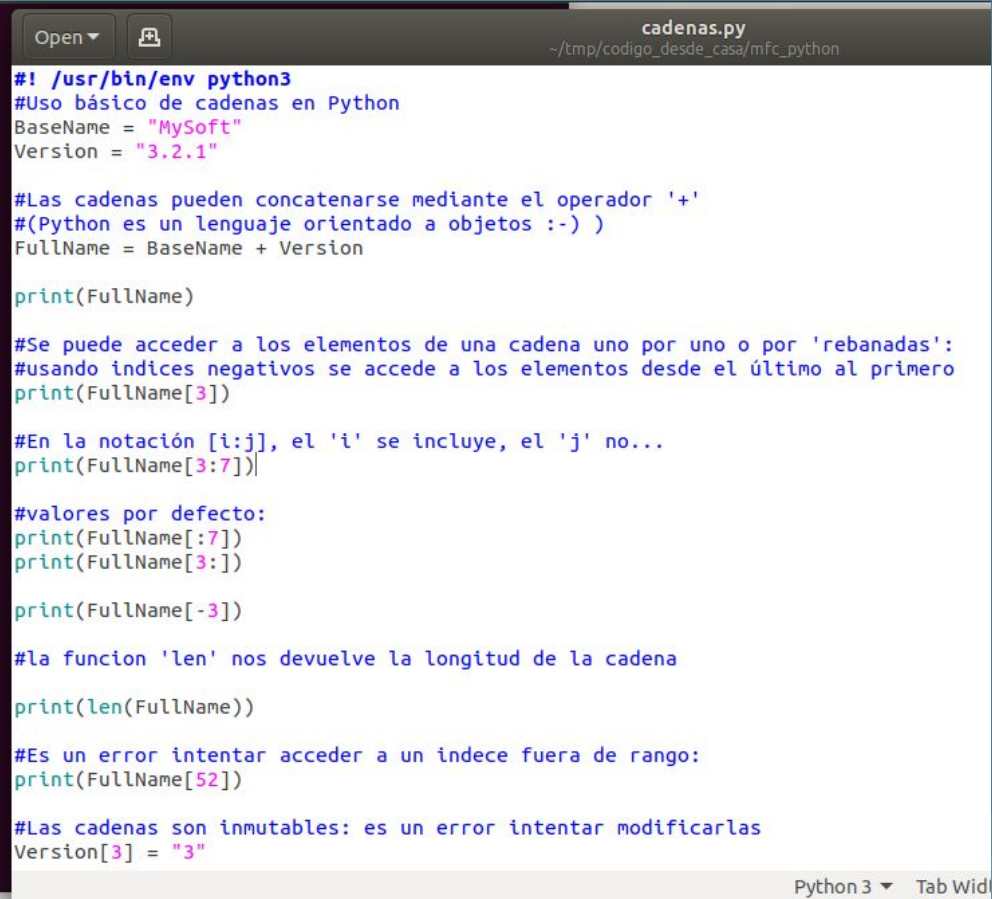
```
fedede@fededeLaptopLenovo: ~  
File Edit View Search Terminal Help  
fedede@fededeLaptopLenovo:~$ python3  
Python 3.6.9 (default, Nov 7 2019, 10:44:02)  
[GCC 8.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import math  
>>> round(math.pi,5)  
3.14159  
>>> round(math.pi,15)  
3.141592653589793  
>>> math.cos(0)  
1.0  
>>> math.cos(math.pi/2)  
6.123233995736766e-17  
>>> round(_, 6)  
0.0  
>>> math.exp(1)  
2.718281828459045  
>>> math.asin(1)  
1.5707963267948966  
>>> z1 = 3 + 2j  
>>> z2 = -2 + j  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'j' is not defined  
>>> z2 = -2 + 1j  
>>> z3 = 3 + 5j  
>>> z1 + 2*z2 + 3*z3  
(8+19j)  
>>> 
```



Cadenas en Python:

```
fede@fedelaptopLenovo: ~  
File Edit View Search Terminal Help  
>>> #Uso básico de cadenas en Python  
... BaseName = "MySoft"  
>>> Version = "3.2.1"  
>>> #Las cadenas pueden concatenarse mediante el operador '+'  
... #(Python es un lenguaje orientado a objetos :-))  
... FullName = BaseName + Version  
>>> print(FullName)  
MySoft3.2.1  
>>> #Se puede acceder a los elementos de una cadena uno por uno o por 'rebanadas':  
... #usando índices negativos se accede a los elementos desde el último al primero  
... print(FullName[3])  
o  
>>> #En la notación [i:j], el 'i' se incluye, el 'j' no...  
... print(FullName[3:7])  
oft3  
>>> #valores por defecto:  
... print(FullName[:7])  
MySoft3  
>>> print(FullName[3:])  
oft3.2.1  
>>> print(FullName[-3])  
2  
>>> #la función 'len' nos devuelve la longitud de la cadena  
... print(len(FullName))  
11  
>>> #Es un error intentar acceder a un índice fuera de rango:  
... print(FullName[52])  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
IndexError: string index out of range  
>>> #Las cadenas son inmutables: es un error intentar modificarlas  
... Version[3] = "3"  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
TypeError: 'str' object does not support item assignment  
>>>  
>>> █
```

Python en modo no-interactivo : generando scripts

```
fed@fedelaptoplenovo:~/tmp$ ls -lrt cadenas.py
-rw-r--r-- 1 fed fed 834 mar 29 12:34 cadenas.py
fed@fedelaptoplenovo:~/tmp$ python3 cadenas.py
MySoft3.2.1
0
oft3
MySoft3
oft3.2.1
2
11
Traceback (most recent call last):
  File "cadenas.py", line 30, in <module>
    print(FullName[52])
IndexError: string index out of range
fed@fedelaptoplenovo:~/tmp$ chmod +x cadenas.py
fed@fedelaptoplenovo:~/tmp$ ls -lrt cadenas.py
-rwxr-xr-x 1 fed fed 834 mar 29 12:34 cadenas.py
fed@fedelaptoplenovo:~/tmp$ ./cadenas.py
MySoft3.2.1
0
oft3
MySoft3
oft3.2.1
2
11
Traceback (most recent call last):
  File "./cadenas.py", line 30, in <module>
    print(FullName[52])
IndexError: string index out of range
fed@fedelaptoplenovo:~/tmp$
```



```
Open  cadenas.py
~/tmp/codigo_desde_casa/mfc_python

#!/usr/bin/env python3
#Uso básico de cadenas en Python
BaseName = "MySoft"
Version = "3.2.1"

#Las cadenas pueden concatenarse mediante el operador '+'
#(Python es un lenguaje orientado a objetos :-))
FullName = BaseName + Version

print(FullName)

#Se puede acceder a los elementos de una cadena uno por uno o por 'rebanadas':
#usando índices negativos se accede a los elementos desde el último al primero
print(FullName[3])

#En la notación [i:j], el 'i' se incluye, el 'j' no...
print(FullName[3:7])

#valores por defecto:
print(FullName[:7])
print(FullName[3:])

print(FullName[-3])

#la función 'len' nos devuelve la longitud de la cadena

print(len(FullName))

#Es un error intentar acceder a un índice fuera de rango:
print(FullName[52])

#Las cadenas son inmutables: es un error intentar modificarlas
Version[3] = "3"
```

Python 3 ▾ Tab Width

Listas en Python:

```
fedefedeLaptopLenovo: ~  
File Edit View Search Terminal Help  
fedefedeLaptopLenovo:~$ python3  
Python 3.6.9 (default, Nov 7 2019, 10:44:02)  
[GCC 8.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> #Uso básico de listas en Python  
... Lista1 = [1,3,5,'a','euler']  
>>> print(Lista1)  
[1, 3, 5, 'a', 'euler']  
>>>  
>>> print(Lista1[3])  
a  
>>>  
>>> print(Lista1[3:])  
['a', 'euler']  
>>> #Las listas son 'mutables': se pueden reasignar sus elementos  
... #y también es posible agregar y quitar elementos  
... Lista1[3] = 'b'  
>>> print(Lista1)  
[1, 3, 5, 'b', 'euler']  
>>> Lista1.append('gauss')  
>>> del Lista1[1]  
>>> print(Lista1)  
[1, 5, 'b', 'euler', 'gauss']  
>>> #las listas se pueden concatenar  
... Lista2 = Lista1 + ['cauchy', 'lagrange']  
>>>  
>>> print(Lista2)  
[1, 5, 'b', 'euler', 'gauss', 'cauchy', 'lagrange']  
>>> □
```


Ciclos for:

```
File Edit View Search Terminal Help
>>> paises = ['Argentina', 'Uruguay', 'Italia', 'Rusia', 'Francia']
>>>
>>> for x in paises:
...     print(x, 'tiene', len(x), 'letras', end=' | ')
...
Argentina tiene 9 letras | Uruguay tiene 7 letras | Italia tiene 6 letras | Rusia tiene 5 letra
s | Francia tiene 7 letras | >>>
>>> #inicializando una lista mediante un ciclo 'for'
... N = 10
>>> Nat_n = []
>>>
>>> for x in range(N):
...     Nat_n.append(x)
...
>>> #es necesaria la linea en blanco de arriba, para indicar la terminación
... #del ciclo 'for'
... print(Nat_n)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> #importamos el módulo math, para tener acceso a operaciones matemáticas complejas
... import math
>>>
>>> Sqrt_Nat_n = []
>>>
>>> for x in range(N):
...     Sqrt_Nat_n.append(round(math.sqrt(x),5))
...
>>> print(Sqrt_Nat_n)
[0.0, 1.0, 1.41421, 1.73205, 2.0, 2.23607, 2.44949, 2.64575, 2.82843, 3.0]
>>>
```

Ciclos while:

File Edit View Search Terminal Help

fedede@fededeLaptopLenovo:~/tmp\$./ciclos_while.py

La suma de los primeros 20 números naturales es: 210

Este resultado también se puede obtener con la fórmula $N(N+1)/2$: 210.0

La suma de los primeros 20 números naturales elevados al cuadrado es: 2870

Este resultado también se puede obtener con la fórmula $N(N+1)(2N+1)/6$: 2870.0

```
#!/usr/bin/env python3
```

```
#ejemplificamos el uso del while calculando la suma de los primeros N  
#números naturales
```

```
N = 20
```

```
index = 1
```

```
suma = 0
```

```
while index <= N:
```

```
    suma = suma + index
```

```
    index = index + 1
```

```
print(70*'-')
```

```
print('La suma de los primeros', N, 'números naturales es:', suma)
```

```
print('\nEste resultado también se puede obtener con la fórmula  $N(N+1)/2$ :', N*(N+1)/2)
```

```
print(70*'-')
```

```
#python también soporta asignación múltiple, como en el siguiente ejemplo:
```

```
index, suma_cuad = 1, 0
```

```
while index <= N:
```

```
    suma_cuad, index = suma_cuad + index*index, index + 1
```

```
#Las expresiones a la derecha en la asignación múltiple de arriba
```

```
#son evaluadas antes de que suceda cualquier asignación. Las
```

```
#expresiones a la derecha son evaluadas de izquierda a derecha
```

```
print('\n\nLa suma de los primeros', N, 'números naturales elevados al cuadrado es:',  
suma_cuad)
```

```
print('\nEste resultado también se puede obtener con la fórmula  $N(N+1)(2N+1)/6$ :',  
N*(N+1)*(2*N+1)/6)
```

```
print(70*'-')
```

Estructuras condicionales if-else:

```
_ ./if_else.py
Ingrese un entero, por favor: 12
El entero es positivo
El entero es par
```

```
----- otro ejemplo: -----
```

```
chau
euler
gauss
_ █
```

```
#!/usr/bin/env python3
```

```
#ingresando datos desde el teclado
```

```
x = int(input("Ingrese un entero, por favor:  "))
```

```
if x < 0:
```

```
    print("El entero es negativo")
```

```
elif x == 0:
```

```
    print("El entero es cero")
```

```
elif x > 0:
```

```
    print("El entero es positivo")
```

```
else:
```

```
    print("Este caso no debería haber ocurrido, se cancela la ejecución...")
```

```
    exit(1)
```

```
if (x % 2) == 0:
```

```
    print("El entero es par")
```

```
else:
```

```
    print("El entero es impar")
```

```
print(40*'-')
```

```
print(13*'-', 'otro ejemplo:', 12*'-')
```

```
print(40*'-')
```

```
List = ["hola", "chau", "euler", "gauss", "juan", "pepe"]
```

```
for j in List:
```

```
#imprime solo los elementos (lexicográficamente) mayores que 'h',
```

```
#es decir, los que comienzan con una letra anterior a la 'h' en el
```

```
#alfabeto
```

```
    if (j >= 'h') == 0:
```

```
        print(j)
```

Definición de funciones:

```
fedelaptoplenovo:$ ./ej_funcion.py
La cadena hola NO es capicua ...
La cadena SOS es capicua ...
La cadena chau NO es capicua ...
La cadena formosa NO es capicua ...
La cadena neuquen es capicua ...
fedelaptoplenovo:$
```

```
#!/usr/bin/env python3
```

```
#Definimos la función DetectarCapicua
```

```
def DetectarCapicua(cadena = 'prueba'):
    """Esta función solo imprime información respecto de si la
    cadena que recibió como argumento es o no capicua"""
    aux = 0
    for i in range(len(cadena)):
        if cadena[i] != cadena[-(i+1)]:
            aux = 1
            break

    if aux == 0:
        print ("La cadena", cadena,"es capicua ...")
    else:
        print ("La cadena", cadena,"NO es capicua ...")
```

```
DetectarCapicua('hola')
```

```
DetectarCapicua('SOS')
```

```
DetectarCapicua('chau')
```

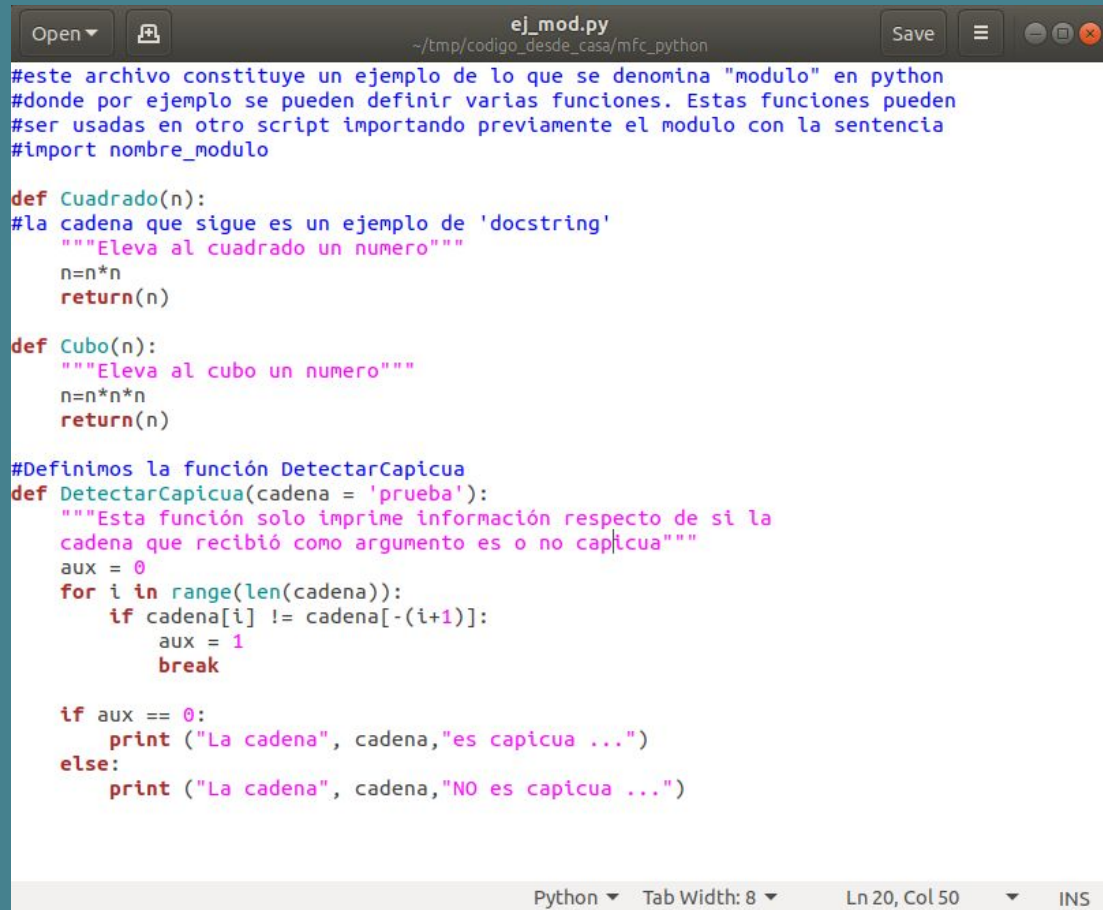
```
provincia1 = "formosa"
```

```
provincia2 = "neuquen"
```

```
DetectarCapicua(provincia1)
```

```
DetectarCapicua(provincia2)
```

Generando nuestros propios módulos:



```
Open  ej_mod.py  Save  ~/tmp/codigo_desde_casa/mfc_python

#este archivo constituye un ejemplo de lo que se denomina "modulo" en python
#donde por ejemplo se pueden definir varias funciones. Estas funciones pueden
#ser usadas en otro script importando previamente el modulo con la sentencia
#import nombre_modulo

def Cuadrado(n):
    """Eleva al cuadrado un numero"""
    n=n*n
    return(n)

def Cubo(n):
    """Eleva al cubo un numero"""
    n=n*n*n
    return(n)

#Definimos la función DetectarCapicua
def DetectarCapicua(cadena = 'prueba'):
    """Esta función solo imprime información respecto de si la
    cadena que recibió como argumento es o no capicua"""
    aux = 0
    for i in range(len(cadena)):
        if cadena[i] != cadena[-(i+1)]:
            aux = 1
            break

    if aux == 0:
        print ("La cadena", cadena,"es capicua ...")
    else:
        print ("La cadena", cadena,"NO es capicua ...")

Python  Tab Width: 8  Ln 20, Col 50  INS
```


Testeando nuestro módulo:

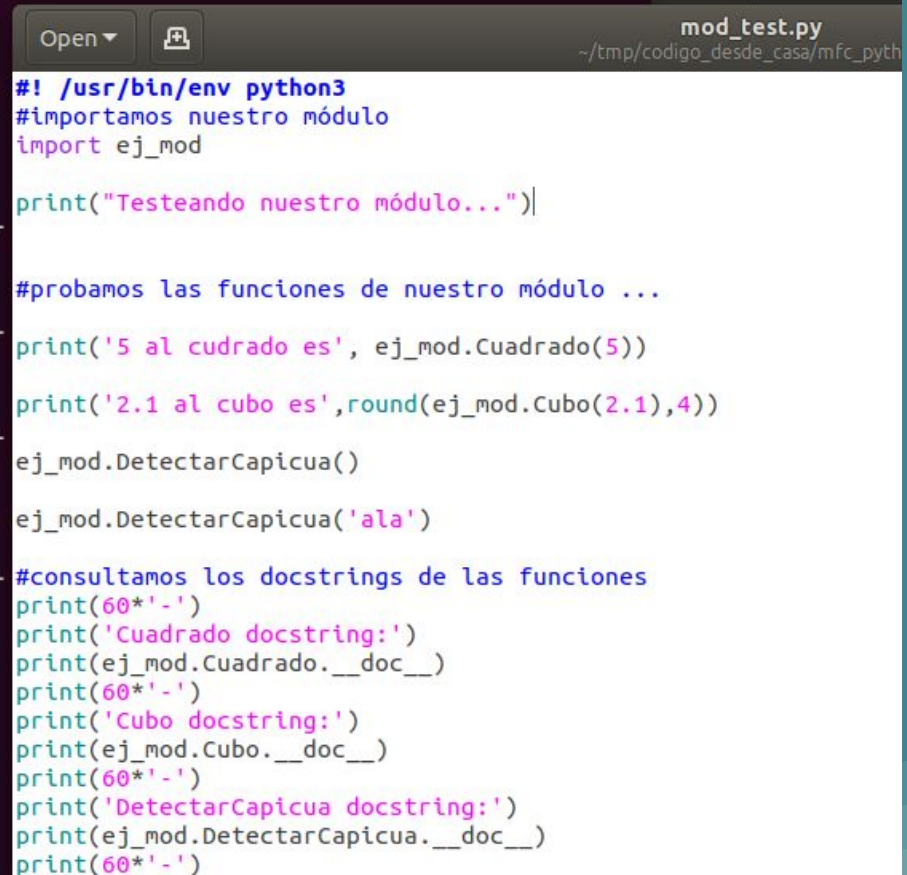
```
fedelaptoplenovo:~$ ./mod_test.py
Testeando nuestro módulo...
5 al cuadrado es 25
2.1 al cubo es 9.261
La cadena prueba NO es capicua ...
La cadena ala es capicua ...
```

```
-----
Cuadrado docstring:
Eleva al cuadrado un numero
```

```
-----
Cubo docstring:
Eleva al cubo un numero
```

```
-----
DetectarCapicua docstring:
Esta función solo imprime información respecto de si la
cadena que recibió como argumento es o no capicua
```

```
-----
fedelaptoplenovo:~$
```



```
mod_test.py
~/tmp/codigo_desde_casa/mfc_pyth

#!/usr/bin/env python3
#importamos nuestro módulo
import ej_mod

print("Testeando nuestro módulo...")

#probamos las funciones de nuestro módulo ...

print('5 al cuadrado es', ej_mod.Cuadrado(5))

print('2.1 al cubo es', round(ej_mod.Cubo(2.1),4))

ej_mod.DetectarCapicua()

ej_mod.DetectarCapicua('ala')

#consultamos los docstrings de las funciones
print(60*'-')
print('Cuadrado docstring:')
print(ej_mod.Cuadrado.__doc__)
print(60*'-')
print('Cubo docstring:')
print(ej_mod.Cubo.__doc__)
print(60*'-')
print('DetectarCapicua docstring:')
print(ej_mod.DetectarCapicua.__doc__)
print(60*'-')
```

Recomendaciones de estilo:

- Usar sangrías de 4 espacios, no tabs.

4 espacios son un buen compromiso entre una sangría pequeña (permite mayor nivel de sangrado) y una sangría grande (más fácil de leer). Los tabs introducen confusión y es mejor dejarlos de lado.

- Recortar las líneas para que no superen los 79 caracteres.

Esto ayuda a los usuarios con pantallas pequeñas y hace posible tener varios archivos de código abiertos, uno al lado del otro, en pantallas grandes.

- Usar líneas en blanco para separar funciones y clases, y bloques grandes de código dentro de funciones.

- Cuando sea posible, poner comentarios en una sola línea.

- Usar docstrings.

- Usar espacios alrededor de operadores y luego de las comas, pero no directamente dentro de paréntesis:

```
a = f(1, 2) + g(3, 4).
```

- Nombrar las clases y funciones consistentemente; la convención es usar `NotacionCamello` para clases y `minusculas_con_guiones_bajos` para funciones y métodos. Siempre usá `self` como el nombre para el primer argumento en los métodos (mirá [Un primer vistazo a las clases](#) para más información sobre clases y métodos).

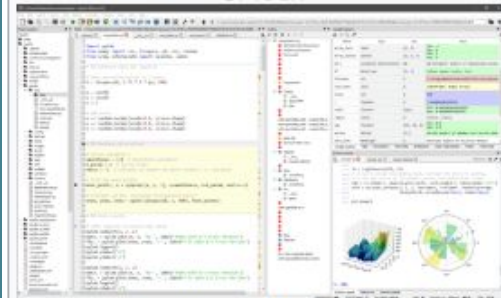
- No uses codificaciones estrafalarias si esperás usar el código en entornos internacionales. El default de Python, UTF-8, o incluso ASCII plano funcionan bien en la mayoría de los casos.

Usando Python desde un IDE: Spyder

Features [\[edit \]](#)

Features include:^[12]

- An editor with [syntax highlighting](#), [introspection](#), [code completion](#)
- Support for multiple [IPython consoles](#)
- The ability to explore and edit [variables](#) from a GUI
- A Help pane able to retrieve and render rich text [documentation](#) on functions, classes and methods automatically or on-demand
- A [debugger](#) linked to IPdb, for step-by-step execution
- [Static code analysis](#), powered by [Pylint](#)
- A run-time [Profiler](#), to benchmark code
- Project support, allowing work on multiple development efforts simultaneously
- A built-in [file explorer](#), for interacting with the filesystem and managing projects
- A "Find in Files" feature, allowing full [regular expression](#) search over a specified scope
- An online help browser, allowing users to search and view Python and package documentation inside the IDE
- A [history log](#), recording every user command entered in each console
- An internal console, allowing for introspection and control over Spyder's own operation



Screenshot of Spyder on Windows

Original author(s)	Pierre Raybaut
Developer(s)	Spyder project contributors
Initial release	18 October 2009; 10 years ago ^{[1][2]}
Stable release	4.1.1 / 19 March 2020; 0 days ago
Repository	github.com/spyder-ide/spyder
Written in	Python
Operating system	Cross-platform
Platform	Qt, Windows, macOS, Linux
Type	Integrated development environment
License	MIT
Website	www.spyder-ide.org

Trabajando con conjuntos (ejemplo usando Spyder):

The image shows the Spyder Python IDE interface. The main editor window displays a Python script named `conjuntos.py` with the following code:

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sun Mar 29 19:14:49 2020
5
6@author: fede
7"""
8#Uso básico del tipo de datos 'set' en Python
9canasta = {"papa", "manzana", "zanahoria", "pera", "naranja", "acelga"}
10
11print("canasta=", canasta)
12
13#verificacion de pertenencia
14var_bool = 'mandarina' in canasta
15print('mandarina pertenece a canasta?', var_bool)
16
17var_bool = 'pera' in canasta
18print('pera pertenece a canasta?', var_bool)
19
20
21frutas = {'manzana', 'mandarina', 'kiwi', 'pera', 'naranja', 'damasco'}
22print("frutas=", frutas)
23
24#resta de conjuntos
25verduras = canasta - frutas
26
27print("verduras=", verduras)
28
29#Otra manera de definir conjuntos
30A = set('enhorabuena')
31#en un conjunto, no cuentan los elementos repetidos
32print(A)
33
34B = set('callejero')
35print(B)
```

The IPython console on the right shows the output of the script:

```
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('/home/fede/tmp/codigo_desde_casa/mfc_python/conjuntos.py',
wdir='/home/fede/tmp/codigo_desde_casa/mfc_python')
canasta= {'acelga', 'manzana', 'naranja', 'pera', 'zanahoria', 'papa'}
mandarina pertenece a canasta? False
pera pertenece a canasta? True
frutas= {'kiwi', 'mandarina', 'manzana', 'naranja', 'damasco', 'pera'}
verduras= {'acelga', 'zanahoria', 'papa'}
{'n', 'h', 'o', 'u', 'a', 'r', 'e', 'b'}
{'l', 'o', 'j', 'a', 'c', 'r', 'e'}
{'l', 'n', 'h', 'o', 'j', 'u', 'a', 'c', 'r', 'e', 'b'}
{'o', 'e', 'r', 'a'}
{'h', 'c', 'b', 'l', 'n', 'j', 'u'}

In [2]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 1, Column: 1, Memory: 94 %.

Más tipos de datos: diccionarios

```
fedelLaptopLenovo$ ./diccionarios.py
```

```
{'juan': 2021, 'pepe': 2152, 'carla': 2127, 'ana': 2054, 'ale': 2143}
```

```
{'pepe': 2152, 'carla': 2127, 'ana': 2054, 'ale': 2143, 'eva': 2213}
```

```
El interno de Carla es: 2127
```

```
lunes rendir parcial
```

```
martes estudiar álgebra
```

```
miércoles ir a la biblioteca
```

```
jueves clase de análisis
```

```
viernes hacer prácticas de HCMA
```

```
sábado merecido descanso!
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
fedelLaptopLenovo$
```

```
Open
```



```
diccionarios.py
```

```
~/tmp/codigo_desde_casa/mFc_python
```

```
Save
```



```
#!/usr/bin/env python3
```

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

```
#Uso básico del tipo de datos 'dictionary' en Python
```

```
#Un diccionario es un conjunto de pares clave-valor
```

```
agenda = {'juan':2021, 'pepe':2152, 'carla':2127, 'ana':2054, 'ale':2143}
```

```
print(agenda)
```

```
#agregando entradas:
```

```
agenda['eva'] = 2213
```

```
#borrando entradas:
```

```
del agenda['juan']
```

```
print(agenda)
```

```
list(agenda.keys())
```

```
#accediendo al valor asociado a cierta clave:
```

```
print('El interno de Carla es:', agenda['carla'])
```

```
#otra manera de definir diccionarios
```

```
tareas = {'lunes':'rendir parcial', 'martes':'estudiar álgebra', 'miércoles':'ir a la biblioteca', 'jueves':'clase de análisis', 'viernes':'hacer prácticas de HCMA', 'sábado':'merecido descanso!'}
```

```
for k, v in tareas.items():
```

```
    print(k, v)
```


Pasando argumentos al script desde línea de comandos:

```
Open  arg_cmd_line.py  Save  ~/tmp/codigo_desde_casa/mfc_python

#ejemplificamos el uso del while calculando la suma de los primeros N
#numeros naturales
#Definimos la función suma_primeros_n_naturales(n)
def suma_primeros_n_naturales(n):
    """Esta función recibe como argumento un número natural n y
    retorna la suma de los primeros n naturales"""
    index = 1
    suma = 0
    while index <= n:
        suma = suma + index
        index = index + 1

    return(suma)

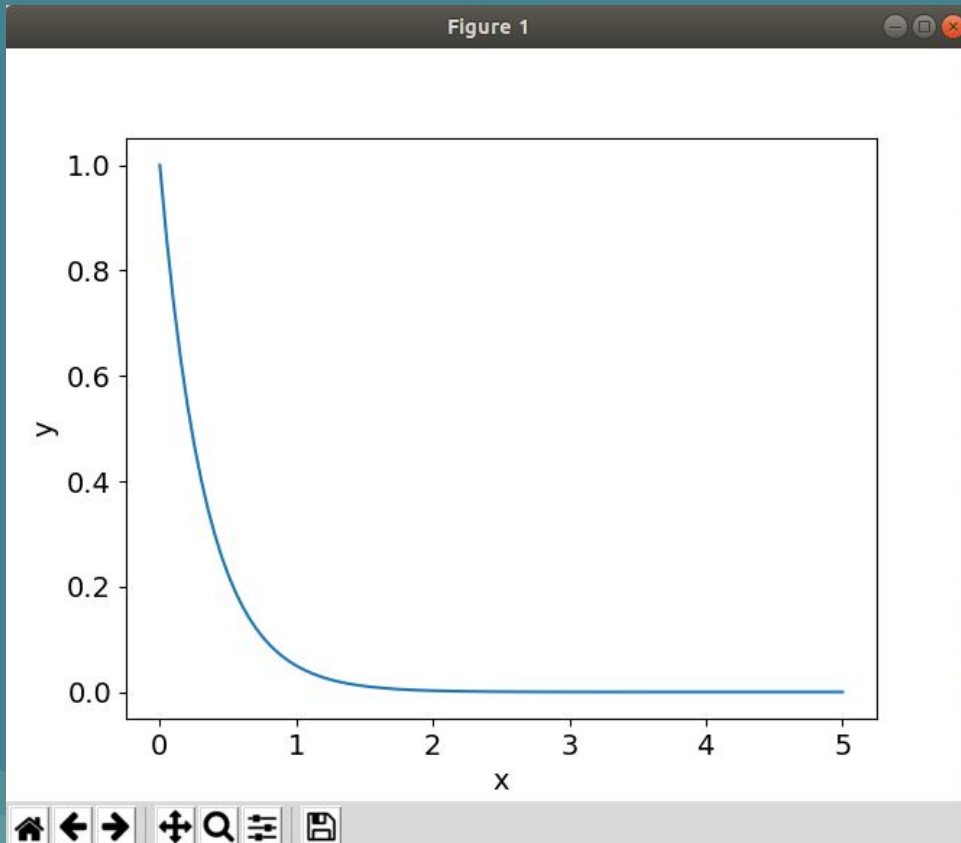
#importamos sys para acceder los argumentos pasados al script
import sys
if len(sys.argv) == 2:
    print(70*'-')
    print('La suma de los primeros', int(sys.argv[1]), 'números naturales es:',
    suma_primeros_n_naturales(int(sys.argv[1])))
    print('\nEste resultado también se puede obtener con la fórmula N(N+1)/2:',
    int(sys.argv[1])*(int(sys.argv[1])+1)/2)
    print(70*'-')
else:
    print('Numero incorrecto de argumentos. Se cancela la ejecución...')
    exit(1)

Python 3  Tab Width: 8  Ln 16, Col 62  INS

fedelaptoplenovo:~$ python3 arg_cmd_line.py 16
-----
La suma de los primeros 16 números naturales es: 136

Este resultado también se puede obtener con la fórmula N(N+1)/2: 136.0
-----
```

Usando Scipy para integrar numéricamente ecs. diferenciales



```
#!/usr/bin/env python3
#ejemplo de uso de scipy

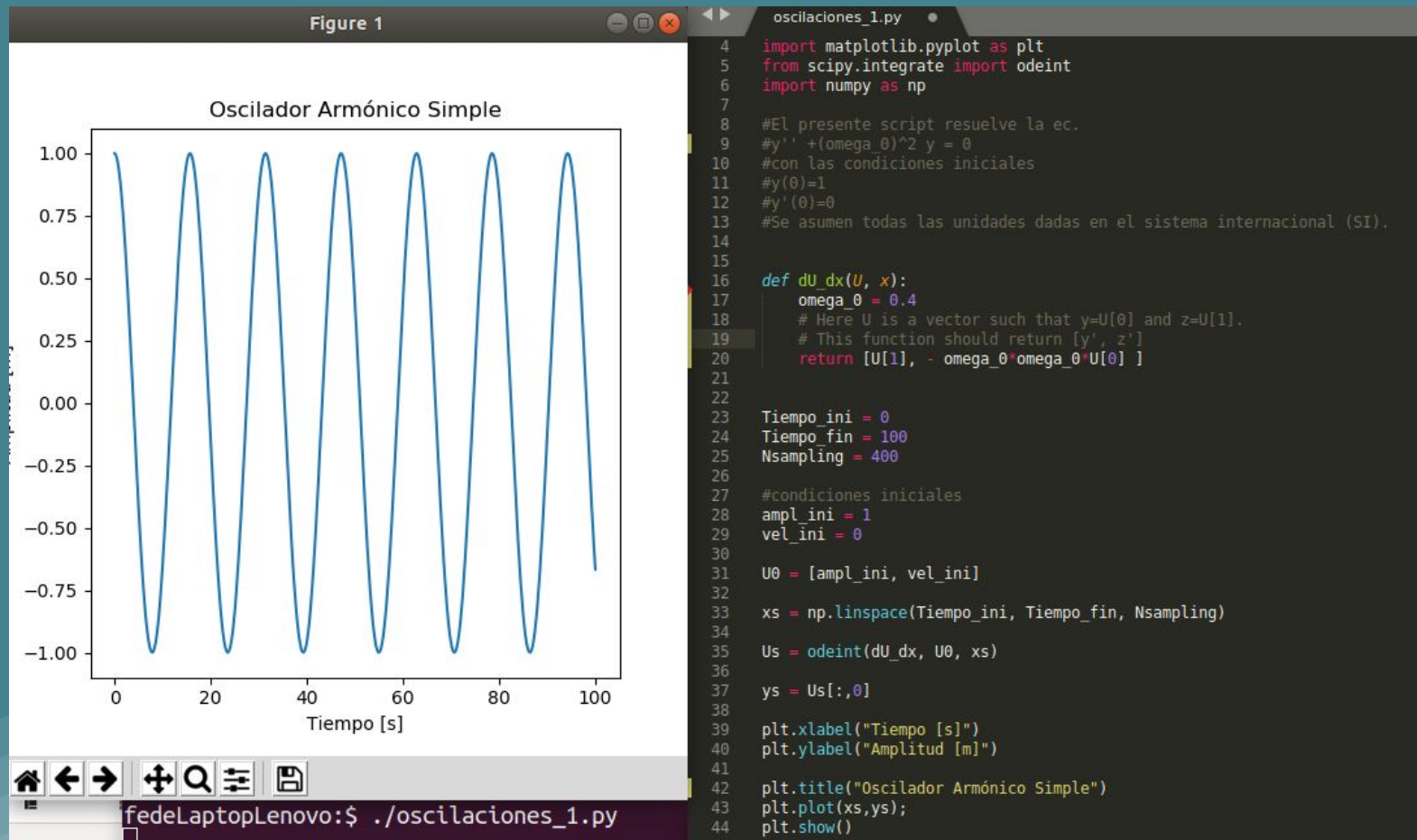
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

# Define a function which calculates the derivative
def dy_dx(y, x):
    return -3*y

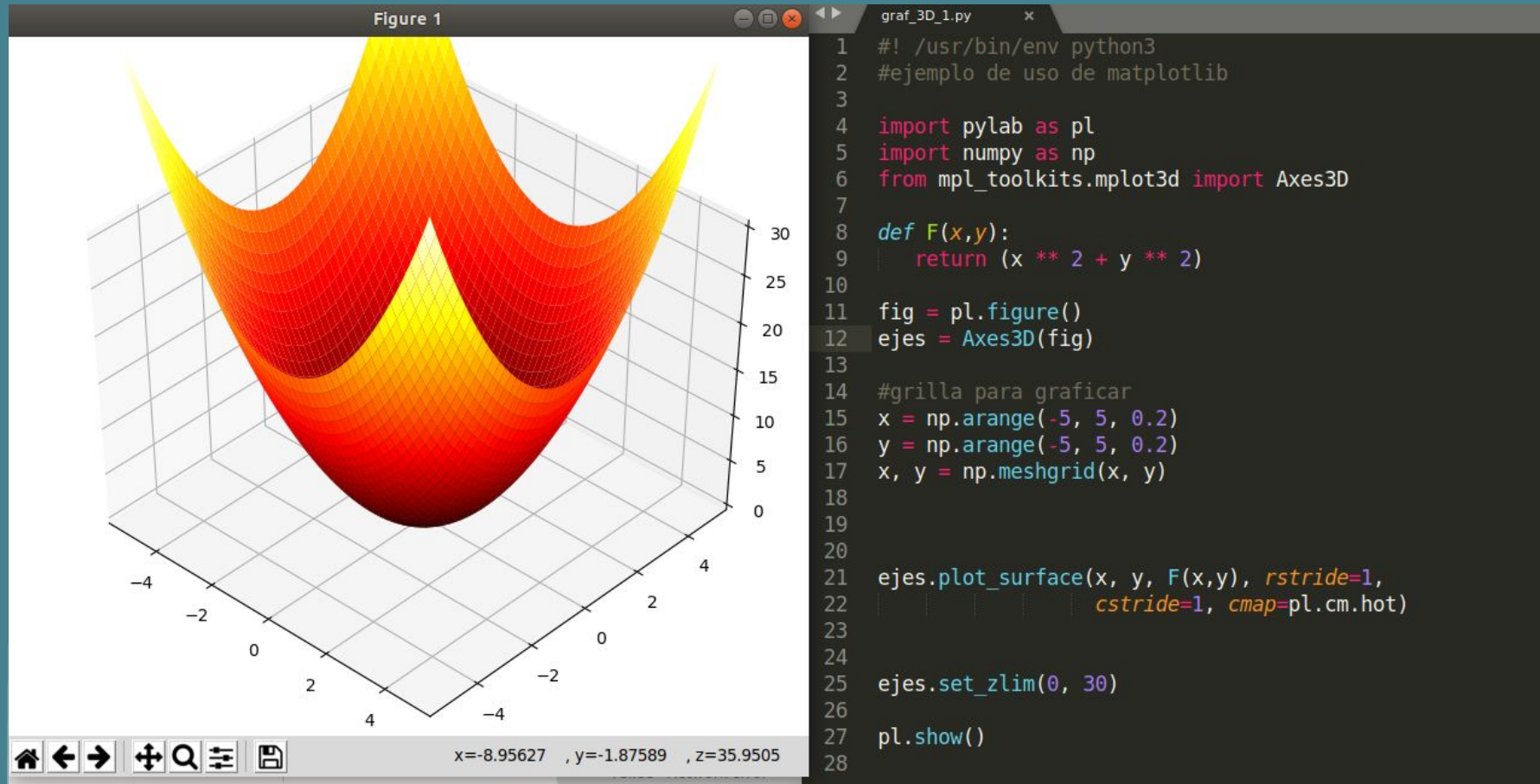
xs = np.linspace(0,5,100)
y0 = 1.0 # the initial condition
ys = odeint(dy_dx, y0, xs)
ys = np.array(ys).flatten()

# Plot the numerical solution
plt.rcParams.update({'font.size': 14}) # increase the font size
plt.xlabel("x")
plt.ylabel("y")
plt.plot(xs, ys)
plt.show()
```

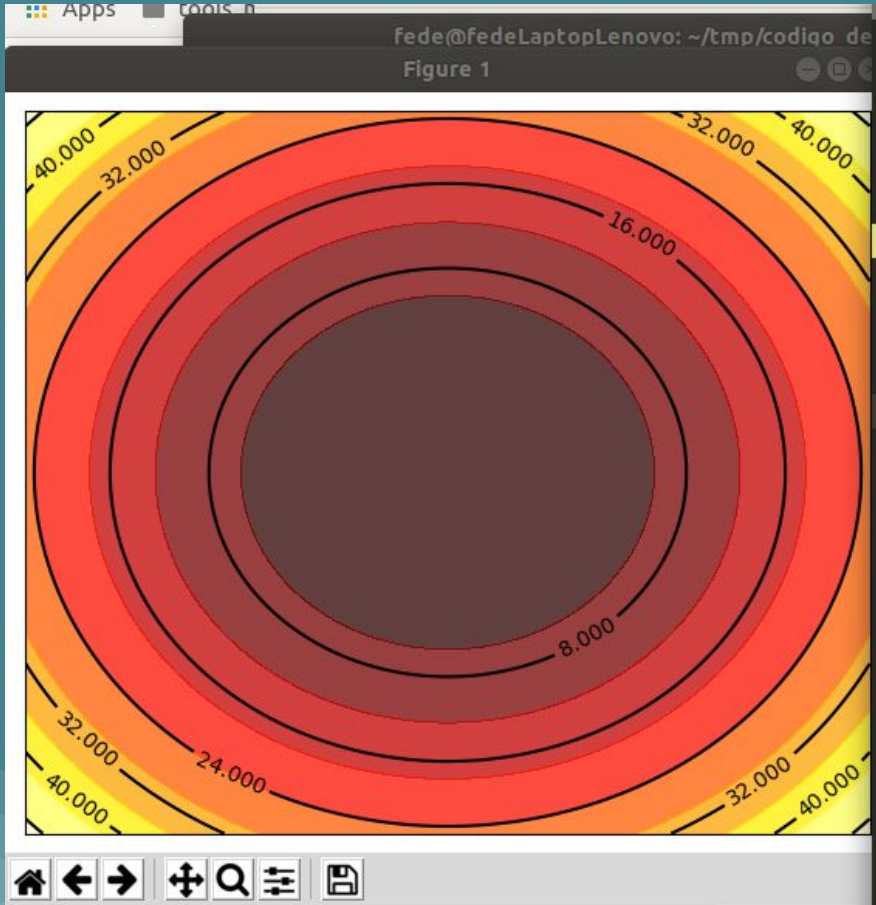
Ej. resolver la ED del oscilador armónico simple con Scipy:



Graficando funciones con numpy + matplotlib:



Curvas de nivel con matplotlib:



```

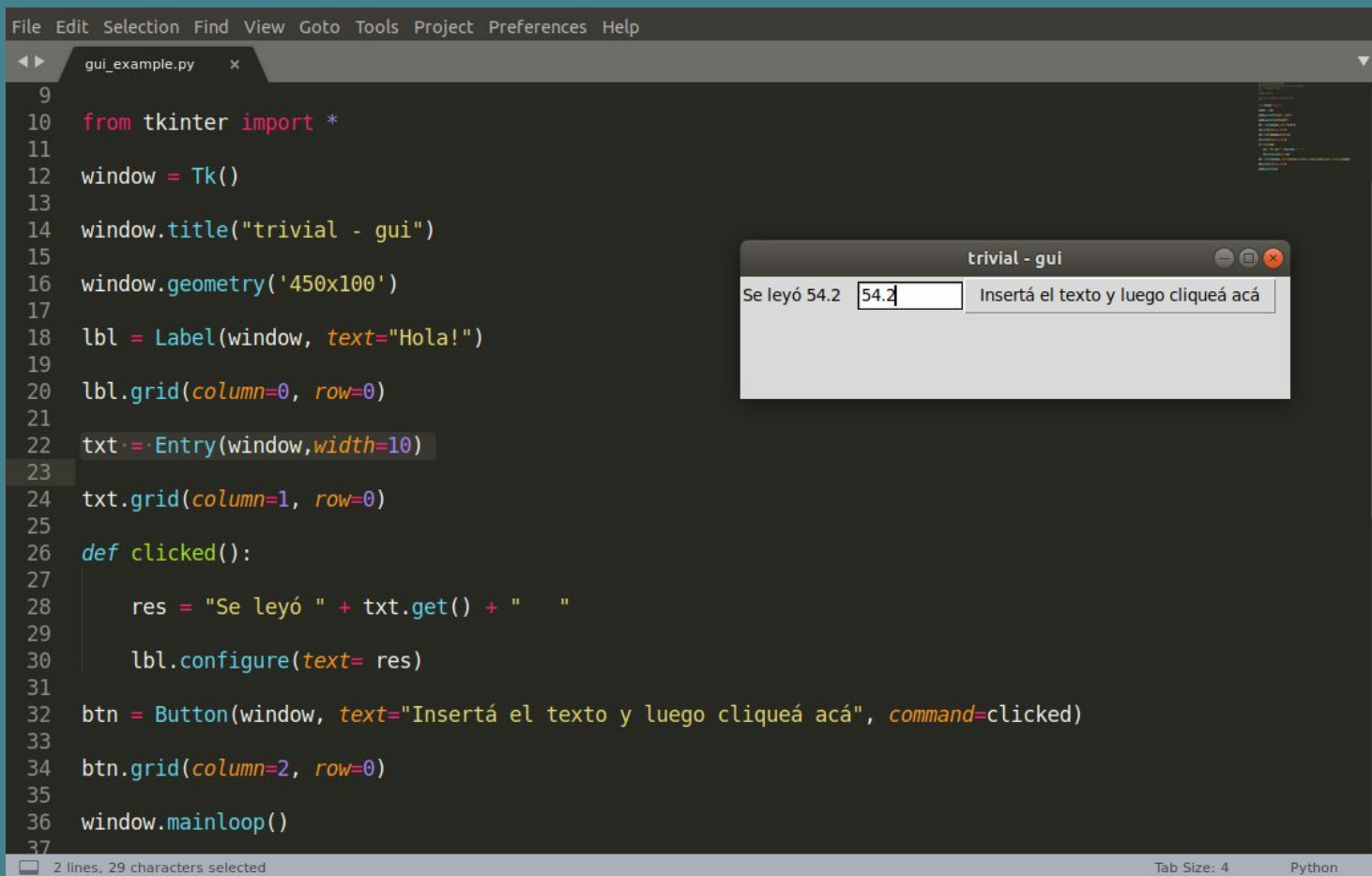
1  #! /usr/bin/env python3
2  #ejemplo de uso de numpy
3  import numpy as np
4  import pylab as pl
5
6  def F(x,y):
7      return (x ** 2 + y ** 2)
8
9  n = 256
10 x = np.linspace(-5, 5, n)
11 y = np.linspace(-5, 5, n)
12 X,Y = np.meshgrid(x, y)
13
14 pl.axes([0.025, 0.025, 0.95, 0.95])
15
16 pl.contourf(X, Y, F(X, Y), 8, alpha=.75,
17 cmap=pl.cm.hot)
18
19 NumCurvas = 6
20 C = pl.contour(X, Y, F(X, Y), NumCurvas, colors='black',
21 linewidth=.5)
22 pl.clabel(C, inline=1, fontsize=10)
23
24 pl.xticks(())
25 pl.yticks(())
26 pl.show()

```


Ajuste de curvas por cuadrados mínimos



Interfaces gráficas con tkinter



The image shows a Python IDE window with a file named `gui_example.py` open. The code defines a simple GUI application using the `tkinter` library. The application window is titled "trivial - gui" and has a geometry of `450x100`. It contains a label with the text "Hola!", an entry field, and a button. The button's command is a function named `clicked`, which updates the label's text to "Se leyó " followed by the text entered in the entry field. The IDE status bar at the bottom indicates "2 lines, 29 characters selected".

```
9
10 from tkinter import *
11
12 window = Tk()
13
14 window.title("trivial - gui")
15
16 window.geometry('450x100')
17
18 lbl = Label(window, text="Hola!")
19
20 lbl.grid(column=0, row=0)
21
22 txt = Entry(window, width=10)
23
24 txt.grid(column=1, row=0)
25
26 def clicked():
27     res = "Se leyó " + txt.get() + " "
28     lbl.configure(text= res)
29
30 btn = Button(window, text="Insertá el texto y luego cliqueá acá", command=clicked)
31
32 btn.grid(column=2, row=0)
33
34
35
36 window.mainloop()
37
```

The running GUI window, titled "trivial - gui", displays the text "Se leyó 54.2" next to the label. The entry field contains the text "54.2". The button text is "Insertá el texto y luego cliqueá acá".

2 lines, 29 characters selected

Tab Size: 4 Python

Para tener en cuenta: jupyter

El **Proyecto Jupyter** es una [organización sin ánimo de lucro](#) creada para "desarrollar [software de código abierto](#), estándares abiertos y servicios para [computación interactiva](#) en docenas de lenguajes de programación". Creado a partir de [IPython](#) en 2014 por [Fernando Pérez](#), el proyecto Jupyter soporta entornos de ejecución en varias docenas de lenguajes de programación. El nombre del proyecto Jupyter es una referencia a los tres lenguajes de programación principales soportados por Jupyter, que son [Julia](#), [Python](#) y [R](#), y también un homenaje a los cuadernos de [Galileo](#) que registran el descubrimiento de los [satélites de Júpiter](#). El proyecto Jupyter ha desarrollado y respaldado los productos de computación interactiva Jupyter Notebook, JupyterHub y JupyterLab, la versión de próxima generación de Jupyter Notebook.

Índice [\[ocultar\]](#)

- 1 [Historia](#)
- 2 [Filosofía](#)
- 3 [Productos](#)
 - 3.1 [Jupyter Notebook](#)
 - 3.2 [Jupyter kernels](#)
 - 3.3 [JupyterHub](#)
 - 3.4 [JupyterLab](#)
- 4 [Uso en la industria](#)
- 5 [Cobertura mediática](#)
- 6 [Premios](#)
- 7 [Véase también](#)
- 8 [Referencias](#)
- 9 [Enlaces externos](#)

Proyecto Jupyter



Tipo [organización sin ánimo de lucro](#)

Objetivos Apoyar la ciencia de datos interactiva y la informática científica en todos los lenguajes de programación.¹

Fundación 2015

Miembro de [Joint Roadmap for Open Science Tools](#)

Sitio web jupyter.org [↗](#) y jupyterlab.readthedocs.io/en/stable [↗](#)

[\[editar datos en Wikidata\]](#)

Ejemplo de jupyter notebook

```
[12] print(interseccion)
```

```
{'r', 'n', 'o', 'a', 'e'}
```

Vamos a crear un conjunto de datos x-y para graficar...

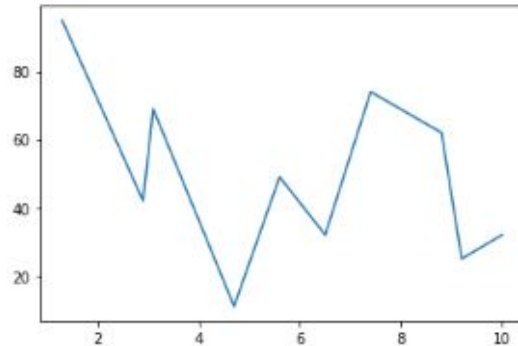
```
x = [1.3, 2.9, 3.1, 4.7, 5.6, 6.5, 7.4, 8.8, 9.2, 10]
```

```
[14] y = [95, 42, 69, 11, 49, 32, 74, 62, 25, 32]
```

```
[16] import matplotlib.pyplot as plt
```

```
[17] plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x7fedd38834a8>]
```



Referencias:

- <http://www.python.org.ar/>
- <http://docs.python.org.ar/tutorial/>
- <https://numpy.org/doc/1.18/>
- <https://docs.scipy.org/doc/scipy/reference/>
- <https://matplotlib.org/users/index.html>
- <https://docs.python.org/3/library/tkinter.html>
- <https://jupyter.org/>
- Para probar Python en la nube: https://www.onlinegdb.com/online_bash_shell
- Para probar Python en la nube: <https://techiedelight.com/compiler/>