

---

# Introducción al desarrollo de software

*Tecnicatura Universitaria en Software Libre*

Emiliano P. López

Mayo de 2015

UNIVERSIDAD NACIONAL DEL LITORAL  
Facultad de Ingeniería y Ciencias Hídricas



*A todos los que están mirando,  
por el amor*



<b>1. Introducción</b>	<b>3</b>
1.1. Motivación	3
1.1.1. ¿Por qué Python?	3
1.2. Instalando Python	3
1.2.1. Windows	3
1.2.2. GNU/Linux	4
1.3. Entornos de programación	4
1.3.1. El intérprete interactivo	4
1.3.2. El intérprete interactivo mejorado	4
1.3.3. Entorno integrado de desarrollo (IDE)	5
1.4. Algoritmos computacionales	5
1.4.1. El primer programa “Adiós mundo!”	6
1.5. Elementos de un programa	6
1.5.1. Números y expresiones	6
1.5.2. Cadenas de caracteres	7
1.5.3. Variables	8
1.5.4. Lectura de datos	9
1.5.5. Escritura de datos	11
1.5.6. Operadores relacionales y lógicos	11
1.5.7. Funciones	11
1.5.8. Módulos	11
1.6. Ejercicios	11
<b>2. Estructuras de control</b>	<b>13</b>
2.1. Condicionales	13
2.1.1. if	13
2.1.2. else	13
2.1.3. Estructuras anidadas	13
2.1.4. elif	13
2.2. Repeticiones	13
2.2.1. while	13
2.3. Ejercicios	13
<b>3. Más estructuras de datos y control</b>	<b>15</b>

3.1.	Listas . . . . .	15
3.2.	for . . . . .	15
3.3.	Manipulando textos con strings . . . . .	15
<b>4.</b>	<b>Funciones, archivos, diccionarios</b>	<b>17</b>
4.1.	Definiendo funciones . . . . .	17
4.1.1.	Variables globales y locales . . . . .	17
4.2.	Números aleatorios . . . . .	17
4.3.	Lectura y escritura de archivos . . . . .	17
4.4.	Diccionarios . . . . .	17
<b>5.</b>	<b>Clases y objetos</b>	<b>19</b>
<b>6.</b>	<b>This is a Title</b>	<b>21</b>
6.1.	Subject Subtitle . . . . .	21
6.2.	Inline Markup . . . . .	21
<b>7.</b>	<b>Indices and tables</b>	<b>23</b>

Contents:





---

# Introducción

---

En el presente capítulo introduciremos los conceptos necesarios para desarrollar los primeros algoritmos computacionales. Además, se explican las herramientas necesarias para llevar a cabo el desarrollo y sus diferentes alternativas.

## 1.1 Motivación

que lindo es programar

### 1.1.1 ¿Por qué Python?

lo lindo que es python y quienes lo usan, su crecimiento, su ámbito de aplicación: web, científico, etc.

## 1.2 Instalando Python

Actualmente existen dos versiones de Python comúnmente utilizadas, la versión 2 y 3, ambas son completamente funcionales y muy utilizadas. En este curso nos basaremos en la versión 3.

### 1.2.1 Windows

Para instalar Python en una máquina con Windows, debemos seguir los siguientes pasos:

- Apuntar el navegador a: <https://www.python.org/downloads/windows/>
- Ir al link de la última versión disponible (por ej: latest python 3 relase)
- En la sección Files, descargar el instalador correspondiente a su arquitectura (64/32 bits), por ej: <https://www.python.org/ftp/python/3.4.3/python-3.4.3.msi>
- Ejecutar el instalador (por ej: python-3.4.3.msi) aceptando las opciones por defecto

### 1.2.2 GNU/Linux

En los sistemas operativos serios, es muy probable que ya contemos con el intérprete instalado, incluso en sus dos versiones. Para instalarlo utilizando los administradores de paquetes debemos ejecutar los siguientes comandos desde una terminal:

Para sistemas basados en Debian (como Ubuntu o sus derivados):

```
apt-get install python3
```

## 1.3 Entornos de programación

### 1.3.1 El intérprete interactivo

Ya con el intérprete de Python podemos comenzar programar. Si ejecutamos en una terminal `python3`, ingresaremos al intérprete en modo interactivo y veremos una salida similar a la siguiente:

```
Python 3.4.2 (default, Oct 8 2014, 10:45:20)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Podemos ejecutar algunas operaciones matemáticas para corroborar su funcionamiento.

```
>>> 2*5
10
>>> 2*5+10
20
>>> -3*19+3.1415
-53.8585
>>>
```

### 1.3.2 El intérprete interactivo mejorado

**IPython**<sup>1</sup> es una interfaz mejorada del intérprete nativo. Se lo puede utilizar en modo consola o a través de una interfaz web. La instalación en GNU/Linux es: `apt-get install ipython3`.

La ejecución de `ipython` desde una terminal nos arroja una pantalla similar a la siguiente:

```
emiliano@pynandi:~ $ ipython3
Python 3.4.2 (default, Oct 8 2014, 10:45:20)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
```

---

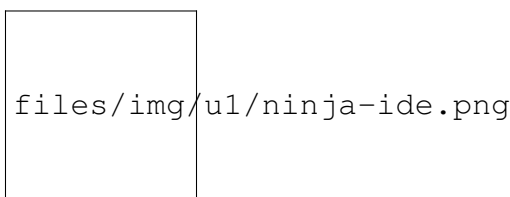
<sup>1</sup><http://ipython.org>

```
%quickref -> Quick reference.  
help      -> Python's own help system.  
object?   -> Details about 'object', use 'object??' for extra details.
```

In [1]:

Otra alternativa muy interesante son los notebooks de ipython, una interfaz que permite programar utilizando el navegador web como entorno. No entraremos en detalle ya que posteriormente analizaremos su funcionamiento. Se debe ejecutar en una terminal `ipython3 notebook` y esto abrirá el navegador por defecto con el entorno cargado.

### 1.3.3 Entorno integrado de desarrollo (IDE)



Un IDE es un entorno que nos facilita las tareas a la hora de programar. Consiste en la integración de un editor de texto, con características de resaltado de sintaxis autocompletado -entre otras-, y el intérprete de Python. Existen cientos de entornos muy buenos, como por ejemplo [Spyder<sup>2</sup>](#), [PyCharm<sup>3</sup>](#) o [Ninja-IDE<sup>4</sup>](#). Para el presente curso, nos basaremos en Ninja-IDE, software libre que ha sido desarrollado por la comunidad de Python Argentina, [PyAr<sup>5</sup>](#).

Una lista bastante completa sobre las IDEs disponibles pueden encontrarse en la [wiki oficial de Python<sup>6</sup>](#)

## 1.4 Algoritmos computacionales

En forma simplificada, un programa o software es un conjunto de instrucciones que la computadora puede ejecutar. Este procedimiento formado por un conjunto de instrucciones es lo que denominamos algoritmo computacional. Una analogía a un algoritmo computacional es una receta de cocina, por ejemplo:

```
Prender el fuego  
Salar la carne  
Controlar cada 5 minutos hasta que haya brasas  
Poner la carne a la parrilla  
Cocinar hasta que esté la carne, controlar cada 5 minutos  
Dar vuelta la carne
```

---

<sup>2</sup><https://github.com/spyder-ide/spyder>

<sup>3</sup><https://www.jetbrains.com/pycharm>

<sup>4</sup><http://ninja-ide.org>

<sup>5</sup><http://python.org.ar>

<sup>6</sup><https://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

Cocinar hasta que esté la carne, controlar cada 5 minutos  
Si falta sal al probar, salar

En esta receta se ven una serie de instrucciones que deben ser seguidas en un determinado orden, en algunos casos contamos con ingredientes, instrucciones, decisiones y acciones que se repiten. No muy distinto a un programa de computación, comencemos con algunos *ingredientes* simples de Python y veamos lo que podemos hacer con ellos.

### 1.4.1 El primer programa “Adiós mundo!”

El acercamiento inicial a un lenguaje de programación suele ser con el archiconocido programa “Hola mundo”. Consiste simplemente en un programa que muestra en pantalla ese mensaje.

Renunciando a cualquier pretensión de originalidad comenzaremos del mismo modo, pero despidiéndonos. Para esto utilizaremos la instrucción `print()` pasando el mensaje de despedida entre comillas, a continuación la instrucción.

```
print("Adios mundo cruel!")
```

Podemos probar la instrucción directamente desde el intérprete, creando con un editor de texto plano un archivo guardado como `chau.py` y luego ejecutándolo desde la terminal haciendo `python3 chau.py`, o bien utilizando un IDE y haciendo todo desde ahí mismo.

Ahora bien, es muchísimo más lo que podemos hacer programando además de saludar cordialmente. Veamos los elementos de un programa que nos permitirán realizar tareas más complejas y entretenidas.

## 1.5 Elementos de un programa

A continuación veremos los ingredientes fundamentales de un lenguaje de programación como Python, para llevar a cabo los ejemplos utilizaremos el intérprete interactivo mejorado `ipython`.

### 1.5.1 Números y expresiones

Frecuentemente requerimos resolver cálculos matemáticos, las operaciones aritméticas básicas son:

- adición: +
- sustracción: -
- multiplicación: \*
- división: /
- módulo: %
- potencia: \*\*
- división entera: //

Las operaciones se pueden agrupar con parentesis y tienen precedencia estándar. Veamos unos ejemplos.

```
In [9]: 1/3
Out[9]: 0.3333333333333333
```

```
In [10]: 1//3
Out[10]: 0
```

```
In [11]: 10%3
Out[11]: 1
```

```
In [12]: 4%2
Out[12]: 0
```

El caso de la potencia, también nos sirve para calcular raices. Veamos una potencia al cubo y luego una raíz cuadrada, equivalente a una potencia a la 1/2.

```
In [13]: 5**3
Out[13]: 125
```

```
In [14]: 2**(1/2)
Out[14]: 1.4142135623730951
```

Los datos numéricos que obtenidos en las operaciones previas se clasifican en reales y enteros, en python se los clasifica como float e int respectivamente, además existe el tipo complex, para números complejos.

Utilizando la función `type()` podemos identificar el tipo de dato. Veamos:

```
In [15]: type(0.333)
Out[15]: float
```

```
In [16]: type(4)
Out[16]: int
```

## 1.5.2 Cadenas de caracteres

Además de números, es posible manipular texto. Las cadenas son secuencias de caracteres encerradas en comillas simples ('...') o dobles ("..."), el tipo de datos es denominado *str*. Sin adentrarnos en detalles, que posteriormente veremos, aquí trataremos lo indispensable para poder desarrollar los primeros programas. Veamos unos ejemplos:

```
>>> 'huevos y pan'           # comillas simples
'huevos y pan'
```

Los operadores algebraicos para la suma y multiplicación tienen efecto sobre las cadenas:

```
>>> 'eco '*4                # La multiplicación repite la cadena
'eco eco eco eco '
```

```
>>>'yo y ' + 'mi otro yo'    # La suma concatena dos o mas cadenas
'yo y mi otro yo'
```

Es posible utilizar cadenas de más de una línea, anteponiendo **triples comillas** simples o dobles al inicio y al final, por ejemplo (fragmento del poema de Fortunato Ramos *Yo jamás fui un niño*):

```
'''
Mi sonrisa es seca y mi rostro es serio,
mis espaldas anchas, mis músculos duros
mis manos partidas por el crudo frío
sólo ocho años tengo, pero no soy un niño.
'''
```

### 1.5.3 Variables

Las variables son contenedores para almacenar información. Por ejemplo, para elevar un número al cubo podemos utilizar 3 variables, para la base (*num1*), para el exponente (*num2*) y para almacenar el *resultado*:

```
num1 = 5                # num1 toma valor 5.
num2 = 3                # num2 toma 3.
resultado = num1**num2  # resultado toma num1 elevado a num2.
print('El resultado es', resultado)
```

El operador igual (=) sirve para asignar lo que está a su derecha, a la variable que se encuentra a su izquierda. Implementemos la siguiente ecuación para dos valores de  $x$ , 0.1 y 0.2.

$$y = (x - 4)^2 - 3$$

```
x1 = 0.1
y1 = (x1-4)**2-3
```

```
x2 = 0.2
y2 = (x2-4)**2-3
```

```
print(x1,y1)
print(x2,y2)
```

Veremos la siguiente salida por pantalla:

```
0.1 12.209999999999999
0.2 11.44
```

Otros ejemplos utilizando variables que contengan **cadenas de caracteres**:

```
cadena1 = 'siento que '
cadena2 = 'nacé en el viento '
```

```
cadena3 = cadena1 + cadena2
```

```
print(cadena3)
```

Los nombres de las variables (identificador o etiqueta) pueden estar formados por letras, dígitos y guiones bajos, teniendo en cuenta ciertas restricciones, no pueden comenzar con un número y ni ser algunas de las siguientes palabras reservadas:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Se debe tener en cuenta que las variables diferencian entre mayúsculas y minúsculas, de modo que juana, JUANA, JuAnA, JUANa son variables diferentes. Esta característica suele denominarse como *case-sensitive*.

### 1.5.4 Lectura de datos

De los ejemplos que vimos, los valores que almacenan las variables fueron ingresadas en el mismo código, difícilmente sea útil contar con los valores cargados en el programa en forma estática. Por esta razón, generalmente se requiere leer información de diferentes fuentes, puede ser desde un archivo o bien interactuando con un usuario.

La lectura de datos desde el teclado se realiza utilizando la sentencia *input()* del siguiente modo:

```
nombre = input("¿Cómo es su nombre, maestro? ")
print("Hola, " + nombre + "!")
```

El comportamiento es:

```
¿Cómo es su nombre, maestro?
Juan de los palotes
Hola, Juan de los palotes!
```

Es importante tener en cuenta que toda lectura por teclado utilizando la función *input()* va a almacenar lo ingresado como una variable de tipo *str*, es decir una cadena de caracteres. Veamos el comportamiento al sumar dos números:

```
num1 = input("Ingrese un número = ")
num2 = input("Ingrese otro número = ")
print("El resultado es =", num1+num2)
```

```
Ingrese un número = 28
Ingrese otro número = 03
El resultado es = 2803
```

Claramente la suma de los valores ingresados no da el resultado observado. El inconveniente se debe a que ambos valores son tomados como cadenas de caracteres y la operación de suma entre cadenas de caracteres produce la concatenación de las mismas. Es necesaria convertir la cadena de caracteres (str) a un valor numérico, ya sea entero o real (int o float).

Para convertir datos de diferentes tipo se utilizan las funciones `int()`, `float()` o `str()`. Modificando el caso anterior:

```
num1 = int(input("Ingrese un número = "))
num2 = int(input("Ingrese otro número = "))
print("El resultado es =", num1+num2)
```

```
Ingrese un número = 28
Ingrese otro número = 03
El resultado es = 31
```

Veamos un ejemplo para operar directamente el valor leído en una ecuación matemática con el siguiente código:

```
x = input("Ingrese x = ")
y = (x-4)**2-3
print(x, y)
```

```
Ingrese x = 3
```

```
-----
TypeError                                Traceback (most recent call last)

<ipython-input-15-3baa5c95d16e> in <module>()
      1 x = input("Ingrese x = ")
----> 2 y = (x-4)**2-3
      3 print(x, y)
```

```
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

A diferencia del ejemplo visto anteriormente, donde la suma de dos cadenas era una operación perfectamente válida, ahora nos encontramos con operaciones entre diferentes tipos pero incompatibles. En este caso, podemos convertir la entrada en un número flotante para operar con normalidad:

```
x = float(input("Ingrese x = "))
y = (x-4)**2-3
print(x, y)
```

```
Ingrese x = 3
3.0 -2.0
```

Es posible combinar distintos tipos de datos haciendo la conversión correspondiente, en el último ejemplo, tanto *x* como *y* son de tipo *float* y es posible concatenarlos a una cadena de caracteres haciendo la conversión correspondiente, utilizando la función `str()`:



```
mensaje = 'y vale ' + str(y) + ' para un valor de x = ' + str(x)
```

### 1.5.5 Escritura de datos

Hemos hecho uso de la función *print()* en su mínima expresión. Iremos viendo diferentes usos a partir de las siguientes variables:

```
# Variables a imprimir
cad = 'Pi es'
pi = 3.1415
mil = 1000
uno = 1
```

#### Como argumentos

La forma más simple es separar los argumentos a ser impresos mediante comas.

```
print(cad, pi, 'aproximadamente')
```

```
Pi es 3.1415 aproximadamente
```

Por defecto, la separación que se obtiene entre cada argumento es un espacio en blanco, sin embargo, se puede cambiar este comportamiento agregando como argumento *\*sep=* ‘\*’ y entre las comillas incluir el separador deseado, por ejemplo:

```
print(cad, pi, 'aproximadamente', sep=';')
print(cad, pi, 'aproximadamente', sep=',')
print(cad, pi, 'aproximadamente', sep=':-)')
```

```
Pi es;3.1415;aproximadamente
Pi es,3.1415,aproximadamente
Pi es:-)3.1415:-)aproximadamente
```

Como vemos, en cada ejecución la impresión se realiza en diferentes renglones, este es el comportamiento por defecto, que puede ser modificando agregando el parámetro *\*end=* “\*”. Reflejemos esto con un ejemplo:

```
print(1, end=" ")
print(2, end=" ")
print(3)
print(4)
```

```
1 2 3
4
```

#### Usando comodines

Los comodines consisten en una marca especial en la cadena a imprimir que es reemplazada por la variable y el formato que se le indique. Existen tres tipos de comodines, para números enteros, reales (flotantes) y para cadenas de caracteres:

- Comodín para reales: %f

- Comodín para enteros: %d
- Comodín para cadenas: %s

Se utiliza del siguiente modo:

```
print('Pi es %f aproximadamente' %pi)
print('El número %d es %s que %d' %(mil, "menor", mil-1))
```

```
Pi es 3.141500 aproximadamente
El número 1000 es menor que 999
```

Es posible formatear los valores, elegir el ancho del campo, la cantidad de decimales, entre muchas otras funciones.

```
print('%.2f %.4f %.3f' %(pi,pi,pi))
print('%4d' %uno)
```

```
3.14 3.1415 3.142
    1
```

Algunas variantes de lo visto se explica en la siguiente lista:

```
%d : un entero
%5d: un entero escrito en un campo de 5 caracteres, alineado a la derecha
%-5d: un entero escrito en un campo de 5 caracteres, alineado a la izquierda
%05d: un entero escrito en un campo de 5 caracteres, completado con ceros desde
%e: flotante escrito en notación científica
%E: como %e, pero E en mayúscula
%11.3e: flotante escrito en notación científica con 3 decimales en un campo de 11
%.3e: flotante escrito en notación científica con 3 decimales en un campo de ancho
%5.1f: flotante con un decimal en un campo de 5 de caracteres
%.3f: flotante con 3 decimales en un campo de mínimo ancho
%s: una cadena
%-20s: una cadena alineada a la izquierda en un campo de 20 caracteres de ancho
```

## 1.5.6 Operadores relacionales y lógicos

## 1.5.7 Funciones

## 1.5.8 Módulos

## 1.6 Ejercicios

---

## Estructuras de control

---

### 2.1 Condicionales

#### 2.1.1 if

#### 2.1.2 else

#### 2.1.3 Estructuras anidadas

#### 2.1.4 elif

### 2.2 Repeticiones

#### 2.2.1 while

### 2.3 Ejercicios

```
for i in range(1,10):  
    if i%2==0:  
        print(i,"es par")  
    else:  
        print(i, "es impar")
```

```
1 es impar  
2 es par  
3 es impar  
4 es par  
5 es impar  
6 es par  
7 es impar  
8 es par  
9 es impar
```



---

## Más estructuras de datos y control

---

### 3.1 Listas

### 3.2 for

### 3.3 Manipulando textos con strings



---

## Funciones, archivos, diccionarios

---

### 4.1 Definiendo funciones

#### 4.1.1 Variables globales y locales

Agrupando el código en módulos

### 4.2 Números aleatorios

### 4.3 Lectura y escritura de archivos

### 4.4 Diccionarios





---

**Clases y objetos**

---



---

# This is a Title

---

That has a paragraph about a main subject and is set when the ‘=’ is at least the same length of the title itself.

## 6.1 Subject Subtitle

Subtitles are set with ‘-’ and are required to have the same length of the subtitle itself, just like titles.

Lists can be unnumbered like:

- Item Foo
- Item Bar

Or automatically numbered:

1. Item 1
2. Item 2

## 6.2 Inline Markup

Words can have *emphasis in italics* or be **bold** and you can define code samples with back quotes, like when you talk about a command: `sudo` gives you super user powers!



---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*