UNL VIRTUAL

Tecnicatura Universitaria **en Software Libre**

Introducción al Desarrollo de Software

Unidad Nº 1
Introducción

Autor
Emiliano López

Colaborador
Maximiliano Boscovich

FICH UNL

Tecnicatura Universitaria en Software Libre

Introducción al Desarrollo de Software

Docente: Emiliano López
Tutor: Maximiliano Boscovich

Contenidos

Introducción al Desarrollo de Software

Este documento fue generado el 2015-10-03 18:11

1 Unidad 1: 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

Gran parte de las tecnologías utilizadas en la actualidad tienen algo en común, y es que por lo general basan su funcionamiento en algún tipo de programa. Que una computadora tenga la flexibilidad de ser utilizada para jugar, predecir el comportamiento climático o gestionar un sanatorio depende exclusivamente de los programas que ejecuta.

Saber programar nos permite intervenir sobre parte de esta realidad desde una postura activa, comprender su funcionamiento y con esto nos abre un gran abanico de posibilidades, limitadas únicamente por nuestra imaginación. Pensemos por un momento en todas las aplicaciones que usamos a diario en el teléfono celular, en la PC, en la tablet, etc. Saber que si necesitamos algo en concreto seremos capaces de crearlo nosotros mismos es pura libertad.

Lo más importante es que todos podemos programar, simplemente tenemos que aprender un conjunto de reglas básicas, saber como aplicarlas y tener muchas ganas de crear cosas nuevas. Además, programar es muy divertido, al contrario de lo que mucha gente podría pensar en un principio. Es como un gran rompecabezas en el que debemos encajar ciertas piezas de una forma específica para conseguir el resultado deseado.

Aprender a programar implica conocer por un lado cierta lógica y por el otro una determinada sintaxis. En este curso haremos énfasis en adquirir el pensamiento lógico utilizando Python como lenguaje de programación por lo que aprenderemos su sintaxis.

Del mismo modo que un enófilo se convierte en un experto en vinos probándolos, a programar se aprende programando y leyendo código. Aquí es donde la sintaxis se vuelve relevante y puede facilitarnos la vida o definitivamente complicarla.

Esta es la principal razón por la que hemos decidido utilizar como primer lenguaje de programación Python, además que existen una gran cantidad de programas desarrollados en este lenguaje, desde herramientas para servidores, hasta programas para usuarios finales, pasando por aplicaciones empresariales, herramientas de desarrollo, plataformas web, juegos de todo tipo, y muchísimas aplicaciones software libre, por lo que se puede obtener y estudiar el código con el que están hechas.

1.1.1 ¿Por qué Python?

Python es un lenguaje de programación multipropósito, poderoso y fácil de aprender. Es del tipo interpretado, lo que significa que los programas no necesitan ser compilados, en su lugar, simplemente requieren que el equipo donde van a ser ejecutados cuente con un interprete instalado.

Python es un lenguaje de programación de tipado dinámico y multiplataforma, cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Es también multiparadigma, ya que soporta orientación a objetos, programación imperativa y programación funcional.

Es sencillo de aprender, y muchos programadores Python reconocen un sustancial aumento en su productividad además de sentir que el lenguaje mismo los incentiva al desarrollo de código de mayor calidad. Está disponible en múltiples plataformas, desde una PC hasta teléfonos celulares, y muchos sitios de Internet utilizan Python como soporte de sus servicios.

Es un lenguaje que cuenta con estructuras de datos eficientes y de alto nivel. Su elegante sintaxis y su tipado dinámico hacen de éste un lenguaje ideal para el desarrollo rápido de aplicaciones en diversas áreas como:

- Aplicaciones WEB
- Aplicaciones científicas

- Gráficas
- Multimedia
- Juegos
- Etc.

Otra de las grandes virtudes de python, es que su interprete puede ejecutarse en la mayoría de los sistemas operativos utilizados en la actualidad (GNU/Linux, Microsoft Windows, Mac OSX, etc.).

Dada su versatilidad y simplicidad, Python es utilizado por compañías como Google, Youtube, Netflix, Yahoo, NSA, NASA, Canonical, IBM, entre otras tantas.

1.2 Instalando Python

Actualmente existen dos versiones de Python comúnmente utilizadas, la versión 2 y 3, ambas son completamente funcionales. 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 release)
- 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 la mayoría de las distribuciones GNU/Linux, es muy probable que ya contemos con el intérprete instalado, incluso en sus dos versiones. En caso de no ser así, para instalarlo utilizando los administradores de paquetes debemos ejecutar los siguientes comandos desde una terminal:

Para sistemas basados en Debian (o sus derivados):

```
sudo apt-get install python3
```

Para sistemas que utilizan yum como sistema de paquetes (Fedora, CentOS, RedHat)

```
sudo yum install *python*
```

1.3 Entornos de programación

1.3.1 El intérprete interactivo

Ya con el intérprete de Python instalado, podemos comenzar a 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.
>>>
```

Con esto, el interprete de python esta listo para empezar a interpretar las instrucciones (las cuales llamaremos sentencias) que forman parte de nuestro programa, por lo que podemos decir que ya estamos listos para empezar a programar. Pero vayamos de lo más sencillo a lo más complejo, y lo mejor para comenzar es realizando ciertos cálculos matemáticos sencillos, y corroborando su resultado. Por ejemplo, escribamos lo siguiente:

```
>>> 2*5
10
>>>
```

Como vemos, si ingresamos 2*5, le estamos diciendo al interprete que debe realizar la multiplicación entre 2 y 5. El interprete analiza la instrucción ingresada (2*5), y contesta con el resultado (10 en este caso).

Hagamos otros cálculos para entrar en calor

```
>>> 2*5+10
20
>>> -3*19+3.1415
-53.8585
>>> 2/10.0
0.2
>>>
```

1.3.2 IPython, el intérprete interactivo mejorado

IPython es una interfaz mejorada del intérprete nativo. Se lo puede utilizar en modo consola o a través de una interfaz web. La instalación en sistemas basados en Debian GNU/Linux es similar a la de python: `apt-get install ipython3`.

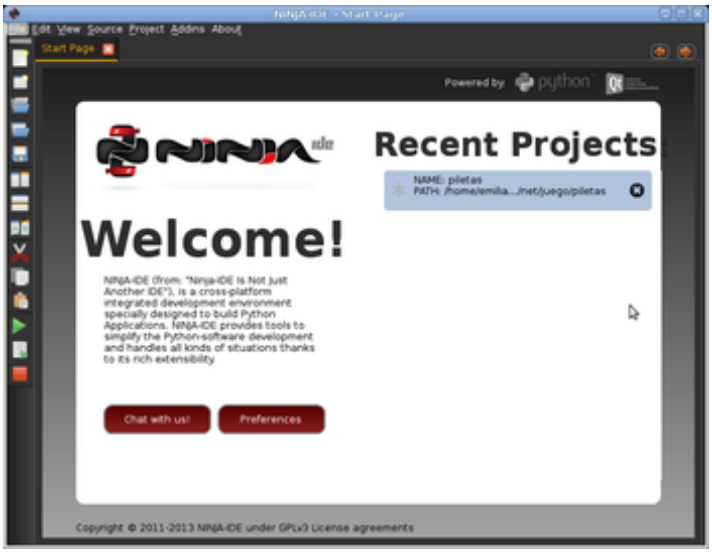
La ejecución de ipython desde una terminal nos arroja una pantalla similar a la siguiente:

```
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.
%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. Si bien no entraremos en detalles sobre su uso, para lanzar la aplicación se debe ejecutar desde una consola el comando `ipython3 notebook`, 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, auto-completado -entre otras-, y el intérprete de Python. Existen cientos de entornos muy buenos, como por ejemplo Spyder, PyCharm o Ninja-IDE. Para el presente curso, nos basaremos en Ninja-IDE, software libre que ha sido desarrollado por la comunidad de Python Argentina, PyAr.



Una lista bastante completa sobre las IDEs disponibles pueden encontrarse en la wiki oficial de Python

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
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

El acercamiento inicial a un lenguaje de programación suele ser con el popular programa "Hola mundo", que consiste 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 entre los paréntesis el mensaje de despedida entre comillas.

```
print("Adiós 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 Modos de ejecutar tus programas

El intérprete interactivo de Python es una gran ayuda para realizar pruebas y experimentar en tiempo real sobre el lenguaje. Sin embargo, cuando cerramos el intérprete perdemos lo escrito, por lo que no es una solución para escribir programas mas largos y con mayores complejidades. Por otro lado, tampoco resulta poco práctico abrir el IDE para correr un script Python. Entonces, para un programa guardado con el nombre hola_mundo.py, lo podemos ejecutar de las siguientes maneras:

1.5.1 Desde la línea de comandos

Abriendo una terminal, e invocando al intérprete python y luego la ruta y nombre del archivo:

```
$python3 hola_mundo.py
```

1.5.2 Como un script

Es posible ejecutarlo sin invocar al intérprete desde la línea de comandos, para esto, se debe incluir al principio del programa la siguiente línea:

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

Con esa línea, estaremos especificando en el mismo programa la ruta del intérprete que debe ejecutarlo. Antes de poder ejecutarlo, debemos otorgarle permisos de ejecución con el comando del sistema operativo chmod:

```
$chmod +x hola_mundo.py
```

Una vez realizado lo anterior, es posible ejecutarlo desde la terminal, como cualquier ejecutable del sistema operativo, llamándolo con el nombre del programa antecediendo "./" (punto barra, sin comillas):

```
$./hola_mundo.py
Adiós mundo cruel
```

1.6 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.

1.6.1 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 paréntesis 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 raíces. 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 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
```

1.6.2 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* (string). 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.
'''
```

1.6.3 Comentarios en el código

En los ejemplos previos y siguientes, veremos dentro del código comentarios explicativos que no serán ejecutados por el intérprete. Su uso solamente está destinado a quien lea el código, como texto explicativo para orientar sobre lo que se realiza.

Los comentarios pueden ser de una única o múltiples líneas. Para el primer caso se utiliza el símbolo numeral. Lo que continúa a la derecha de su uso no es ejecutado.

Los comentarios de múltiples líneas se deben escribir entre triples comillas, ya sean simples o dobles.

1.6.4 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 = 'nací 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*.

1.6.5 Lectura de datos

De los ejemplos que vimos, los valores que almacenan las variables fueron ingresados 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-3-3baa5c95d16e> 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)
```

1.6.6 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
```

1.6.6.1 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
```

1.6.6.2 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 utilizan 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)
```

La sintaxis general del uso de comodines es:

```
%[opciones][ancho][.precisión]tipo
```

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 la izquierda (ej. 00041)
- %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 caracteres
- %.3e: flotante escrito en notación científica con 3 decimales en un campo de ancho mínimo
- %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

Con lo visto hasta aquí tenemos suficientes alternativas para mostrar en pantalla información de diferentes tipos. Existen una alternativa para imprimir en pantalla utilizando el método `format`, el lector interesado puede indagar más al respecto en http://docs.python.org.ar/tutorial/3/inputoutput.html , en el capítulo Entrada y Salida del tutorial de Python oficial http://docs.python.org.ar/tutorial/pdfs/TutorialPython3.pdf ó también en http://www.python-course.eu/python3_formatted_output.php

1.6.7 Funciones

Las funciones son programas o subprogramas que realizan una determinada acción y que pueden ser invocados desde otro programa. En los capítulos posteriores trabajaremos en mayor profundidad, en esta sección solamente presentaremos algunas de las muchas que nos provee Python en su biblioteca estándar.

El uso de funciones nativas en Python es directo, veamos algunas:

```
frase = 'simple es mejor que complejo'
num_letras = len(frase)
print(num_letras)
```

```
28
```

El ejemplo previo hicimos uso de dos funciones, por un lado la función `print()`, presentada ya desde el primer programa y una nueva función, `len()`, que recibe como dato de entrada una cadena de caracteres y calcula la cantidad de caracteres de la misma y lo retorna de manera tal que lo podemos asignar a una variable (num_letras).

1.6.8 Módulos

Python posee cientos de funciones que se organizan o agrupan en módulos. Veamos un ejemplo para calcular la raíz cuadrada, el seno y coseno de un número haciendo uso de las funciones `sqrt()`, `sin()` y `cos()`, todas ubicadas bajo el módulo `math`.

```
import math

nro = 2
raiz = math.sqrt(nro)
print("La raíz de %d es %.4f" %(nro,raiz))
print("El seno de %d es %.4f" %(nro,math.sin(nro)))
print("El coseno de %d es %.4f" %(nro,math.cos(nro)))
```

```
La raíz de 2 es 1.4142
El seno de 2 es 0.9093
El coseno de 2 es -0.4161
```

Del ejemplo previo, hemos visto como indicarle a Python que importe -o haga uso de- un módulo en particular y de algunas de sus funciones incluidas.

En capítulos posteriores veremos en profundidad distintos modos de importar módulos e invocar sus funciones.