# Capítulo 2

# Primeros Pasos en Python

## 2.1. Primer programa

El primer programa que escribiremos en Python es, el clásico, *hola mundo!*. Para crear este script en Python requerimos de un editor de texto plano (vi, jed, xemacs, gedit. . . elija su favorito). El archivo creado debe tener la exstensión `.py` y lo llamamos `hola.py`. Contiene las siguientes lineas:

```
# hola.py
# Mi primer programa de Python

print ("Hola Mundo!")
```

Para ejecutar el programa, en la linea de comandos de la terminal

```
gprieto > python hola.py
```

que da como resultado

```
gprieto > python hola.py
Hola Mundo!
```

Note que la ejecución del programa se hace directamente en la linea de comandos y no abriendo Python. Sin embargo, si se tiene abierto Python, se puede ejecuar el programa directamente dentro de Python.

### 2.1.1. Explicación del primer programa

La primera línea del código es un comentario. Cualquier texto después de un símbolo # es un comentario y Python no lo tiene en cuenta. Al terminar la linea termina el comentario. No hay necesidad de terminar el comentario con otro símbolo (aunque en otros lenguajes esto si es necesario). También es posible hacer comentarios dentro de una línea

```
print ("Hola Mundo!")    # imprimir texto
```

lo cual es válido.

La segundo linea es una linea en blanco. Python no tiene en cuenta estas lineas, y se puede poner tantas lineas en blanco como se quiera, ya que serán descartadas por Python. El uso de lineas en blanco permite hacer el código mas fácil de leer, y permite escribir el código en bloques pequeños.

La siguiente linea si es interpretada por el programa. El comando `print` ordena que en la terminal se imprima el texto que está dentro del paréntesis. Para lineas de texto, deben ir entre comillas o comillas dobles.

### 2.1.2. Alternativas para el primer programa

Python permite variaciones al programa anterior que también funcionan

```
# hola3.py
# Mi primer programa de Python

x = "Hola Mundo!"
print (x)
```

y que permiten posteriormente realizar operaciones con variables.

```
# hola4.py
# Mi primer programa de Python

x = "Hola"
y = "Mundo!"
print (x+' '+y)
```

Note que en este último ejemplo, realizamos una operación para *pegar* variables con caracteres. Con el fin de separar las palabras en las variables $x$ y $y$, se pone un espacio. El resultado de estos dos programas es idéntico al primero.

## 2.2. Multiplicar dos números enteros

A continuación, queremos realizar operaciones matemáticas en Python. Abajo, un programa simple para multiplicar dos enteros (2 y 3).

```
# multint.py
# Simple code to multiply two integers
#

a = 2
b = 3
```

```
c = a*b

print ("Product = ", c)
```

El programa usa tres variables, las letras `a`, `b` y `c`. Las variables pueden tener nombres largos, pero no pueden tener espacios. Si quiere unir palabras use _. El primer caracter de una variable debe ser una letra, el resto puede ser una combinación de letras, números y _. No se recomienda usar puntos o símbolos de menos (−). A diferencia de Fortran o C, Python define de manera automática si un número es entero o real, aunque si el usuario así lo prefiere, puede definir la variable como entero:

```
a = int(2)
b = int(3)
```

Cuando esto sucede, Python asume entonces que `c` será un número entero. Aparentemente en Python existía un problema cuando se dividían dos números enteros, pero la respuesta era un real. En la versión que yo tengo (3.4+) ese problema ya no se presenta.

El resultado del programa anterior da como resultado

```
gprieto > python multint.py
Product =  6
```

como es de esperarse. El símbolo $*$ indica multiplicación casi como en la mayoría de lenguajes de programación. Adición es +, resta es − y división es /. En Python, para elevar un número $a$ a la $b$ potencia, se escribe $a**b$, y las dos variables pueden ser reales.

## 2.3. Una tabla trigonemétrica con `for` loops

Cualquier lenguaje de programación debe permitir realizar una serie de operaciones de manera repetida. Esto significa permitir realizar un *loop* para una serie de valores de una variable. En C o matlab, esto se lleva a cabo con un `for` loop, en Fortran se hace con un `do` loop. En Python se usa el `for`. A continuación se muestra un ejemplo de código para realizar una tabla de funciones trigonométricas:

```
# trigtable.py
# Create a simple trigonometric table
#

import math
import numpy as np

degrad = 180.0/3.1415927
```

```
i = np.arange(90)

for ang in i :
   theta = float(ang)
   ctheta = math.cos(theta/degrad)
   stheta = math.sin(theta/degrad)
   ttheta = math.tan(theta/degrad)

   print ("%5.1f, %7.4f, %7.4f, %8.4f"
          % (theta,ctheta,stheta,ttheta))
```

****NOTA IMPORTANTE**** Python tiene una larga lista de librerías (conocidas como `modules`) para realizar operaciones de todo tipo. Para poder utilizarlas en un programa, es necesario cargarlas. Esto se realiza con el comando `import`. Por ejemplo, para utilizar operaciones trigonométricas, se requiere el modulo `math`, que se carga

```
import math
```

y permite utilizar las funciones coseno (`math.cos`), seno (`math.sin`), y tangente (`math.tan`). Adicionalmente el programa carga también el modulo `numpy`, que tiene multiples funciones para realizar operaciones en variables, vectores, etc. Se carga:

```
import numpy as np
```

Para que llamar las diferentes operaciones no sea muy largo, le ponemos un nombre más corto `np`. Esto permite que para generar un vector

```
i = np.arange(90)
```

no sea necesario escribir algo más largo como:

```
i = numpy.arange(90)
```

Este comando genera un arreglo, con valores que van desde 0 hasta 89. **Python empieza los arreglos con 0, no con 1**, lo que significa que el arreglo `i` tiene valores $0, 1, 2, ..., 88, 89$.

Tal como lo hacen C, Fortran o Matlab, el cálculo trigonométrico se hace en radianes (no grados) como argumento en las funciones trigonométricas. Por esto, es necesario convertir los ángulos en grados, a radianes con un operador `degrad`$= 180./\pi$, que realiza la operación con una simple multiplicación.

Después de estos pasos, finalmente podemos empezar el *loop*,

```
for ang in i :
   theta = float(ang)
   ctheta = math.cos(theta/degrad)
   ...
```

donde el loop genera una variable `ang` que toma sucesivamente los valores del arreglo `i`, es decir en cada iteración $0, 1, 2, ..., 88, 89$. A partir de ese valor, definimos nuestro ángulo `theta`, como el número real de `ang` usando la función `float`. Es decir, `theta` asume valores sucesivos de 0,0, 1,0, 2,0, ..., 88,0, 89,0 en cada iteración. Personalmente recomiendo siempre realizar las iteraciones sobre un vector de números enteros (`i`), ya que dependiendo de errores de *rounding* pueden evitar que se llegue exactamente hasta 89,0.

**NOTA: EL `FOR` SE CIERRA CON DOS PUNTOS (`:`). Y DEBE HABER UNA INDENTACIÓN DE LOS COMANDOS QUE ESTAN DENTRO DEL LOOP**. A diferencia de otros lenguajes, Python no tiene un `end` para el loop.

Dentro de cada `for` loop, se calculan los cosenos, senos y tangentes del ángulo `theta`, después de convertir el ángulo a radianes dividiendo por `degrad`. Finalmente, en cada loop, se imprime a la terminal los resultados

```
print ("%5.1f, %7.4f, %7.4f, %8.4f"
          % (theta,ctheta,stheta,ttheta))
```

donde el comando `print` tiene un formato especificado por el usuario. En este caso `%5.1f` ordena que la variable `theta` se imprima como un número real con 5 espacios en total, y 1 dígito a la derecha del punto decimal. Igualmente, el uso de `7.4f` ordena que ctheta tenga 7 espacios y 4 números a la derecha del decimal. Las números están justificados a la derecha. Como Python no tiene en cuenta espacios o saltos de linea, la continuidad del comando dentro del paréntesis es automático. Es decir que el comando en el bloque de código arriba, es un sólo comando.

### 2.3.1. Posibles Variaciones

El código a continuación sugiere dos posibles variaciones que pueden ser útiles.

```
# trigtable2.py
# Create a simple trigonometric table (2)
#

from math import *
import numpy as np

fmt = "%5.1f, %7.4f, %7.4f, %8.4f"

degrad = 180.0/pi

i = np.arange(90)

for ang in i :
```

```
theta = float(ang)
ctheta = cos(theta/degrad)
stheta = sin(theta/degrad)
ttheta = tan(theta/degrad)

print (fmt %(theta,ctheta,stheta,ttheta))
```

donde se importan todas las funciones dentro del modulo `math` de tal forma que no hay necesidad de llamarlos con `math.cos`, sino directamente con `cos`. Adicionalmente, uno puede definir una variable `fmt`, que tiene los caracteres que definen el formato de salida. Esto es útil cuando uno va a utilizar comandos `print` en múltiples casos dentro de un programa. El modulo `math` tiene guardado el valor de $\pi$ en la variable `pi`.

## 2.4. Funciones del modulo `math`

Acá se muestran otras (no todas) funciones que se encuentran en el modulo.

```
ceil(x).      smallest integer greater or equal than x
fabs(x)       Absolute calue of x
floor(x)      largest integer value less than or equal to x
isinf(x).     Check if the float x is positive or negative infinity.
isnan(x)      Check if the float x is a NaN (not a number).
exp(x)        exponential
log10(x)      base 10 log
log(x)        return the natural logarithm of x
sqrt(x)       square root
acos(x)       arccosine
asin(x)       arcsine
atan(x)       arctangent
atan2(y,x)    arctangent of y/x in correct quadrant (***very useful!)
cos(x)        cosine
cosh(x)       hyperbolic cosine
sin(x)        sine
sinh(x)       hyperbolic sine
tan(x)        tangent
tanh(x)       hyperbolic tangent
degrees(x)    Convert angle x from radians to degrees.
radians(x)    Convert angle x from degrees to radians.
```

y constantes dentro del módulo

```
pi  valor de pi=3.14159265...
e   valor de la constante e=2.718281...
```

## 2.5. Sobre formatos

Python permite desplegar en la terminal los números y caracteres en diferentes formatos. Acá algunos ejemplos útiles

```
%5i   =  entero, con 5 espacios, justificado a la derecha

%5.4i =  igual, pero con 0's a la izquierda (88 es 0088)

%8.3f =  real, 8 espacios, con 3 a la derecha del punto decimal

%12.4e = real con exponente, 4 valores a la derecha del decimal
         p.e., b-0.2342E+02 donde "b" is espacio en blanco
         (sirve para escribir valores grandes y pequeños,
         o cuando no se sabe que tamaño tendrán los valores)

%8s  =  lista de caracteres, ocho espacios, justificado a la
        derecha. Si la lista de  caracteres es mayor a 8, se
        imprimen todas.

%.8s =  igual, pero sólo los primeros 8 caracteres se imprimen.
```

## Problemas

2.1. Escriba un programa de Python que imprima una frase célebre.

2.2. Escriba un programa en Python que imprima en la pantalla una tabla con

```
x    sinh(x)    cosh(x)
```

las funciones hiperbólicas del seno y coseno, para valores de X entre 0.0 a 6.0 con incrementos de 0.5. Utilice un formato para el resultado que sea presentable. No convierta `x` a radianes.

2.3. Modifique el programa inicial, para imprimir en la Terminal, dos números enteros y el resultado de las 4 operaciones `+, -, *, /`. Use un formato de impresión adecuado para cada caso.

2.4. Genere una tabla de multiplicar para los números del 1 al 9, es decir $1 \times 1, 1 \times 2, \ldots, 1 \times 10$, la siguiente linea $2 \times 1, 2 \times 2, \ldots, 2 \times 10$ y así hasta el 9.

.