chap02

July 7, 2021

1 Capítulo 2

1.1 Primeros pasos con Python

1.2 Mi primer programa de Python

El primer programa que escribiremos en Python es, el clásico, `hola mundo!`.

En este ejercicio, si está usando Jupyter Notebooks, simplemente se puede correr el código en una de las celdas (cells) abajo y simplemente ejecutarlo. Si, Ud quiere crear un script en Python, requiere de un editor de texto plano (vi, jed, xemacs, gedit. . . elija su favorito). El archivo creado debe tener la exstensión `.py` y lo podemos llamar `hola.py`.

Contiene las siguientes lineas:

[10]:
```
print("Hola Mundo!")
```

```
Hola Mundo!
```

Una segunda versión de este programa puede ser más informativo

```
Escriba el mismo programa, pero ahora
ponga más información en comentarios,
para que el usuario que lo vea sepa
qué es lo que hace el programa.
```

[11]:
```
# hola.py
# Mi primer programa de Python

print ("Hola Mundo!")
```

```
Hola Mundo!
```

1.2.1 Observaciones

El código `hola.py` tiene varias características - El resultado es idéntico a la versión más sencilla. - Las dos primeras líneas comienzan con el símbolo # - Hay una linea que se encuentra vacía.

1.2.2 Explicación

Las primeras dos lineas 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.

La tercera 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 y separados.

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.

Hágalo Ud. mismo Cree una versión de `hola.py` con comentarios, explicando que hace el comando `print`.

[1]:
```
"""
hola2.py
esto puede ser un comentario, no es ejecutado por Python
"""

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

```
Hola Mundo!
```

1.3 Alternativas para mi primer programa

Python (y otros languajes de programación) pueden asignarle a una variable (`x`, `y` o `var`) una serie de caracteres.

[1]:
```
# hola3.py
# Otras alternativas para el programa

# Asignar variable x
x = "Hola Mundo!"

# Imprimir
print(x)
```

```
Hola Mundo!
```

Hágalo Ud. mismo Asigna el contenido del texto que quiere imprimir a dos o más variables. En Python se puede concatenar (*pegar*) las variables con una suma (`+`).

[3]:
```
# hola4.py
# Otras alternativas para el programa
"""
Asignar tres variables (x,y,z), pegarlas para
el mismo resultado.
"""
```

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

W1 = x+" "+y+z
print(W1)
```

```
Hola Mundo!
Hola Mundo!
```

1.3.1 Observaciones

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.

1.4 Multiplicación de dos números enteros

A continuación, queremos realizar operaciones matemáticas en Python, que es algo común en la investigación científica.

Hágalo Ud. mismo Multiplicar dos números enteros (`2` y `3`).

[5]:
```
# multint.py
# Código para multiplicar dos números enteros

a = 2
b = 3

c = a*b

print(a,"x",b," = ",c)
```

```
2 x 3  =  6
```

1.4.1 Explicación

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 (`-`), ya que esto significa resta para Python. El resultado del programa anterior da como resultado `6`, como es de esperarse.

1.4.2 Observaciones

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 y la respuesta era un real, no un entero. En versiones recientes de Python (3.4 o mayor) ese problema no se presenta.

[11]:
```
a = int(a)
b = int(b)
a/b
```

[11]:
```
0.6666666666666666
```

[12]:
```
b/a
```

[12]:
```
1.5
```

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.

Hágalo Ud. mismo Eleve un número (`a=2.01`) a la `b=3.2` potencia.

$$c = a^b$$

[41]:
```
# real_elev.py
"""
Código simple para elevar un número a la b
potencia. Imprima el resultado
"""
a = 2.01
b = 3.2
c = a**b

print(a, "elevado a la ",b," = ", c)
```

```
2.01 elevado a la  3.2  =  9.337430530829119
```

1.5 Una tabla trigonométrica 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 `for`.

Ejemplo. Genere la tabla del 7 (7x0, 7x1, 7x2, ...)

[14]:
```
# mult_table.py
# Genere la tabla de multiplicar del 7.

x = 7    # la tabla del 7

for y in range(10):            # loop 0, 1, 2, ..., 9
    z = x*y
    print(x,'x',y,' = ',z)
print('Acabó el loop')
```

```
7 x 0  =  0
7 x 1  =  7
7 x 2  =  14
7 x 3  =  21
7 x 4  =  28
7 x 5  =  35
7 x 6  =  42
7 x 7  =  49
7 x 8  =  56
7 x 9  =  63
Acabó el loop
```

1.5.1 Explicación

El `for` loop es un comando que ejecuta una serie de comandos (que deben estar indentados). El número de veces o la forma como los ejecuta depende código

`for y in range(10):`

have que la variable `y` tome valores en un rango de `0` hasta `9`. Es decir repite la operación 10 veces, empezando en cero, hasta llegar al nueve. Porqué Python no incluye el 10? El comando `range` genera 10 números, pero Python (como C, C##) empiezan el contéo con el cero, es decir (0, 1, 2, ..., 9), para un total de 10 números.

Como se puede ver, en el primer *loop*, `y=0`, en el segundo `y=1` y así sucesivamente.

Una segunda opción de código

[52]:
```
# mult_table2.py
"""
Genere la tabla de multiplicar del 7.
Ahora use numpy
"""
import numpy as np

x = 7          # la tabla del 7
i = np.arange(10)

for y in i:        # loop 0, 1, 2, ..., 9
    z = x*y
```

```
    print(x,'x',y,' = ',z)
```

```
7 x 0  =  0
7 x 1  =  7
7 x 2  =  14
7 x 3  =  21
7 x 4  =  28
7 x 5  =  35
7 x 6  =  42
7 x 7  =  49
7 x 8  =  56
7 x 9  =  63
```

1.5.2 Explicación

Como se puede observar, el resultado de esta segunda versión del código es la misma a la anterior, pero tiene más cosas.

****NOTA IMPORTANTE****

Python tiene una larga lista de librerías (conocidas como *modules*, o *packages*) para realizar operaciones de todo tipo. Para poder utilizarlas en un programa, es necesario cargarlas. Esto se realiza con el comando `import`. En este caso, importamos el paquete NumPy.

```
import numpy as np
```

NumPy tiene multiples funciones para realizar operaciones en variables, vectores, etc. Para usar una función dentro de NumPy, si debe llamar el paquete y la función. Para que esta operación no sea muy larga, le ponemos un nombre más corto `np`. Esto permite usar la función `arange` dentro de NumPy, y generar un vector

```
i = np.arange(10)
```

Este comando genera un arreglo (o un vector), con valores que van desde 0 hasta 9. Es algo muy similar a lo que hace `range(10)`, pero lo pone en un arreglo `i`

[53]:
```
i
```

[53]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Ahora el `for` loop es:

```
for y in i:
```

que lo que hace es que `y` tome el primer valor del arreglo, después el segundo y así sucesivamente.

1.5.3 Observaciones

La indentación es muy importante

- No mezcle espacios y “tabs”
- Python busca por un número de espacios exacto en la indentación
- Aunque no se vea, un tab no es equivalente a ocho espacios para Python.

- Notebooks generalmente hace la indentación por Ud, pero no se confie.

Abrir y cerrar loops

- La linea con el comando `for` debe terminar con dos puntos (`:`).
- Todo lo que se quiere dentro del loop debe estar intentado.
- Para terminar el loop, simplemente retire la indentación.
- A diferencia de otros lenguajes, Python no tiene un `end` para el loop.

Hágalo Ud. mismo Genere una tabla trigonométrica, para ángulos de `ang=0,1, ..., 89` grados y resultados para

`coseno(ang) seno(ang) tangente(ang)`

****NOTA IMPORTANTE**** 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

`arad= ang*pi/180.`

Y cómo sabe Python el valor de π?

El módulo `math` el valor de π en la variable `math.pi`. Además tiene funciones para calcular coseno (math.cos), seno (math.sin), y tangente (math.tan) y mucho más.

[15]:
```
# trigtable.py
# Genera una tabla trigonométrica simple.
# Imprima los resultados para cos, sin, tan.
#
import math
import numpy as np

i = np.arange(90)

"""
Haga un loop, y calcule los tres valores
No olvida transformar el ángulo
"""
for ang in i:
    theta  = ang/180.*math.pi
    ctheta = math.cos(theta)
    stheta = math.sin(theta)
    ttheta = math.tan(theta)

    print(ang, ctheta, stheta, ttheta)
```

```
0 1.0 0.0 0.0
1 0.9998476951563913 0.01745240643728351 0.017455064928217585
2 0.9993908270190958 0.03489949670250097 0.03492076949174773
3 0.9986295347545738 0.05233595624294383 0.0524077792830412
4 0.9975640502598242 0.0697564737441253 0.06992681194351041
```

5 0.9961946980917455 0.08715574274765817 0.087488663525924
6 0.9945218953682733 0.10452846326765346 0.10510423526567646
7 0.992546151641322 0.12186934340514748 0.1227845609029046
8 0.9902680687415704 0.13917310096006544 0.14054083470239143
9 0.9876883405951378 0.15643446504023087 0.15838444032453627
10 0.984807753012208 0.17364817766693033 0.17632698070846498
11 0.981627183447664 0.1908089953765448 0.19438030913771848
12 0.9781476007338057 0.20791169081775931 0.2125565616700221
13 0.9743700647852352 0.22495105434386498 0.2308681911255631
14 0.9702957262759965 0.24192189559966773 0.2493280028431807
15 0.9659258262890683 0.25881904510252074 0.2679491924311227
16 0.9612616959383189 0.27563735581699916 0.2867453857588079
17 0.9563047559630355 0.2923717047227367 0.30573068145866034
18 0.9510565162951535 0.3090169943749474 0.3249196962329063
19 0.9455185755993168 0.3255681544571567 0.34432761328966527
20 0.9396926207859084 0.3420201433256687 0.36397023426620234
21 0.9335804264972017 0.35836794954530027 0.3838640350354158
22 0.9271838545667874 0.374606593415912 0.4040262258351568
23 0.9205048534524404 0.3907311284892737 0.4244748162096047
24 0.9135454576426009 0.40673664307580015 0.4452286853085361
25 0.9063077870366499 0.42261826174069944 0.46630765815499864
26 0.898794046299167 0.4383711467890774 0.4877325885658614
27 0.8910065241883679 0.45399049973954675 0.5095254494944288
28 0.882947592858927 0.4694715627858908 0.5317094316614788
29 0.8746197071393957 0.48480962024633706 0.554309051452769
30 0.8660254037844387 0.49999999999999994 0.5773502691896256
31 0.8571673007021123 0.5150380749100542 0.6008606190275603
32 0.848048096156426 0.5299192642332049 0.6248693519093275
33 0.838670567945424 0.544639035015027 0.6494075931975104
34 0.8290375725550417 0.5591929034707468 0.6745085168424265
35 0.8191520442889918 0.573576436351046 0.7002075382097097
36 0.8090169943749475 0.5877852522924731 0.7265425280053609
37 0.7986355100472928 0.6018150231520483 0.7535540501027942
38 0.7880107536067219 0.6156614753256583 0.7812856265067175
39 0.7771459614569708 0.6293203910498375 0.8097840331950074
40 0.7660444431189781 0.6427876096865393 0.8390996311772798
41 0.754709580222772 0.6560590289905073 0.8692867378162267
42 0.7431448254773942 0.6691306063588582 0.9004040442978399
43 0.7313537016191705 0.6819983600624985 0.9325150861376617
44 0.7193398003386512 0.6946583704589973 0.9656887748070739
45 0.7071067811865476 0.7071067811865475 0.9999999999999999
46 0.6946583704589974 0.7193398003386511 1.0355303137905694
47 0.6819983600624985 0.7313537016191705 1.0723687100246826
48 0.6691306063588583 0.7431448254773941 1.1106125148291925
49 0.6560590289905075 0.7547095802227719 1.150368407221009
50 0.6427876096865394 0.766044443118978 1.1917535925942098
51 0.6293203910498376 0.7771459614569708 1.2348971565350508
52 0.6156614753256584 0.7880107536067219 1.2799416321930783

```
53 0.6018150231520484 0.7986355100472928 1.3270448216204098
54 0.5877852522924732 0.8090169943749473 1.3763819204711731
55 0.5735764363510462 0.8191520442889917 1.4281480067421142
56 0.5591929034707469 0.8290375725550416 1.4825609685127399
57 0.5446390350150272 0.8386705679454239 1.5398649638145825
58 0.5299192642332049 0.848048096156426 1.6003345290410504
59 0.5150380749100544 0.8571673007021121 1.664279482350517
60 0.5000000000000001 0.8660254037844386 1.7320508075688767
61 0.48480962024633717 0.8746197071393957 1.8040477552714234
62 0.46947156278589086 0.8829475928589269 1.8807264653463316
63 0.4539904997395468 0.8910065241883678 1.9626105055051504
64 0.4383711467890774 0.898794046299167 2.0503038415792965
65 0.42261826174069944 0.9063077870366499 2.1445069205095586
66 0.4067366430758004 0.9135454576426009 2.246036773904215
67 0.3907311284892737 0.9205048534524404 2.355852365823753
68 0.3746065934159122 0.9271838545667873 2.4750868534162946
69 0.35836794954530016 0.9335804264972017 2.6050890646938023
70 0.3420201433256688 0.9396926207859083 2.747477419454621
71 0.32556815445715676 0.9455185755993167 2.9042108776758218
72 0.30901699437494745 0.9510565162951535 3.077683537175253
73 0.29237170472273677 0.9563047559630354 3.2708526184841404
74 0.27563735581699916 0.9612616959383189 3.487414443840909
75 0.25881904510252074 0.9659258262890683 3.7320508075688776
76 0.24192189559966767 0.9702957262759965 4.0107809335358455
77 0.22495105434386514 0.9743700647852352 4.331475874284153
78 0.20791169081775923 0.9781476007338057 4.704630109478457
79 0.19080899537654492 0.981627183447664 5.144554015970307
80 0.17364817766693041 0.984807753012208 5.6712818196177075
81 0.15643446504023092 0.9876883405951378 6.313751514675041
82 0.13917310096006547 0.9902680687415704 7.115369722384208
83 0.12186934340514748 0.992546151641322 8.144346427974593
84 0.10452846326765344 0.9945218953682733 9.514364454222587
85 0.08715574274765835 0.9961946980917455 11.43005230276132
86 0.06975647374412523 0.9975640502598242 14.30066625671194
87 0.052335956242943966 0.9986295347545738 19.08113668772816
88 0.03489949670250108 0.9993908270190958 28.636253282915515
89 0.0174524064372836 0.9998476951563913 57.289961630759144
```

Aunque el resultado es correcto, no es la forma más amable de presentar una tabla.

[2]:
```python
# trigtable2.py
# Genera una tabla trigonométrica simple.
# Imprima los resultados para cos, sin, tan.
#    Ahora, imprima los resultados con mejor presentación
#

import math
import numpy as np
```

```
i = np.arange(90)

for ang in i :
    theta = float(ang/180.*math.pi)
    ctheta = math.cos(theta)
    stheta = math.sin(theta)
    ttheta = math.tan(theta)

    """Desplegar resultados más amables"""
    print ("%5.1f, %7.4f, %7.4f, %8.4f"
           % (ang,ctheta,stheta,ttheta))
```

```
  0.0,  1.0000,  0.0000,   0.0000
  1.0,  0.9998,  0.0175,   0.0175
  2.0,  0.9994,  0.0349,   0.0349
  3.0,  0.9986,  0.0523,   0.0524
  4.0,  0.9976,  0.0698,   0.0699
  5.0,  0.9962,  0.0872,   0.0875
  6.0,  0.9945,  0.1045,   0.1051
  7.0,  0.9925,  0.1219,   0.1228
  8.0,  0.9903,  0.1392,   0.1405
  9.0,  0.9877,  0.1564,   0.1584
 10.0,  0.9848,  0.1736,   0.1763
 11.0,  0.9816,  0.1908,   0.1944
 12.0,  0.9781,  0.2079,   0.2126
 13.0,  0.9744,  0.2250,   0.2309
 14.0,  0.9703,  0.2419,   0.2493
 15.0,  0.9659,  0.2588,   0.2679
 16.0,  0.9613,  0.2756,   0.2867
 17.0,  0.9563,  0.2924,   0.3057
 18.0,  0.9511,  0.3090,   0.3249
 19.0,  0.9455,  0.3256,   0.3443
 20.0,  0.9397,  0.3420,   0.3640
 21.0,  0.9336,  0.3584,   0.3839
 22.0,  0.9272,  0.3746,   0.4040
 23.0,  0.9205,  0.3907,   0.4245
 24.0,  0.9135,  0.4067,   0.4452
 25.0,  0.9063,  0.4226,   0.4663
 26.0,  0.8988,  0.4384,   0.4877
 27.0,  0.8910,  0.4540,   0.5095
 28.0,  0.8829,  0.4695,   0.5317
 29.0,  0.8746,  0.4848,   0.5543
 30.0,  0.8660,  0.5000,   0.5774
 31.0,  0.8572,  0.5150,   0.6009
 32.0,  0.8480,  0.5299,   0.6249
 33.0,  0.8387,  0.5446,   0.6494
```

```
34.0,  0.8290,  0.5592,   0.6745
35.0,  0.8192,  0.5736,   0.7002
36.0,  0.8090,  0.5878,   0.7265
37.0,  0.7986,  0.6018,   0.7536
38.0,  0.7880,  0.6157,   0.7813
39.0,  0.7771,  0.6293,   0.8098
40.0,  0.7660,  0.6428,   0.8391
41.0,  0.7547,  0.6561,   0.8693
42.0,  0.7431,  0.6691,   0.9004
43.0,  0.7314,  0.6820,   0.9325
44.0,  0.7193,  0.6947,   0.9657
45.0,  0.7071,  0.7071,   1.0000
46.0,  0.6947,  0.7193,   1.0355
47.0,  0.6820,  0.7314,   1.0724
48.0,  0.6691,  0.7431,   1.1106
49.0,  0.6561,  0.7547,   1.1504
50.0,  0.6428,  0.7660,   1.1918
51.0,  0.6293,  0.7771,   1.2349
52.0,  0.6157,  0.7880,   1.2799
53.0,  0.6018,  0.7986,   1.3270
54.0,  0.5878,  0.8090,   1.3764
55.0,  0.5736,  0.8192,   1.4281
56.0,  0.5592,  0.8290,   1.4826
57.0,  0.5446,  0.8387,   1.5399
58.0,  0.5299,  0.8480,   1.6003
59.0,  0.5150,  0.8572,   1.6643
60.0,  0.5000,  0.8660,   1.7321
61.0,  0.4848,  0.8746,   1.8040
62.0,  0.4695,  0.8829,   1.8807
63.0,  0.4540,  0.8910,   1.9626
64.0,  0.4384,  0.8988,   2.0503
65.0,  0.4226,  0.9063,   2.1445
66.0,  0.4067,  0.9135,   2.2460
67.0,  0.3907,  0.9205,   2.3559
68.0,  0.3746,  0.9272,   2.4751
69.0,  0.3584,  0.9336,   2.6051
70.0,  0.3420,  0.9397,   2.7475
71.0,  0.3256,  0.9455,   2.9042
72.0,  0.3090,  0.9511,   3.0777
73.0,  0.2924,  0.9563,   3.2709
74.0,  0.2756,  0.9613,   3.4874
75.0,  0.2588,  0.9659,   3.7321
76.0,  0.2419,  0.9703,   4.0108
77.0,  0.2250,  0.9744,   4.3315
78.0,  0.2079,  0.9781,   4.7046
79.0,  0.1908,  0.9816,   5.1446
80.0,  0.1736,  0.9848,   5.6713
81.0,  0.1564,  0.9877,   6.3138
```

```
 82.0,  0.1392,  0.9903,   7.1154
 83.0,  0.1219,  0.9925,   8.1443
 84.0,  0.1045,  0.9945,   9.5144
 85.0,  0.0872,  0.9962,  11.4301
 86.0,  0.0698,  0.9976,  14.3007
 87.0,  0.0523,  0.9986,  19.0811
 88.0,  0.0349,  0.9994,  28.6363
 89.0,  0.0175,  0.9998,  57.2900
```

1.5.4 Explicación

Dentro de cada `for` loop, se calculan los cosenos, senos y tangentes del ángulo theta, después de convertir el ángulo `ang` a radianes. Finalmente, en cada loop, se imprime a la terminal los resultados. **Note:** el comando `print` está indentado, hace parte de cada loop.

```
    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, a pesar de usar dos lineas.

1.5.5 Versiones alternativas

[62]:
```
# trigtable3.py
"""
Genera una tabla trigonométrica simple.
Ahora, imprima los resultados con mejor presentación
Defina el formato desde el comienzo
"""

from math import *
import numpy as np

"""Defina el formato acá"""
fmt = "%5.1f, %7.4f, %7.4f, %8.4f"

i = np.arange(90)

for ang in i :
    theta  = float(ang/180.*pi)
    ctheta = cos(theta)
    stheta = sin(theta)
    ttheta = tan(theta)
```

```
    """Imprima al resultado"""
    print(fmt %(ang, ctheta, stheta, ttheta))
```

```
 0.0,  1.0000,  0.0000,   0.0000
 1.0,  0.9998,  0.0175,   0.0175
 2.0,  0.9994,  0.0349,   0.0349
 3.0,  0.9986,  0.0523,   0.0524
 4.0,  0.9976,  0.0698,   0.0699
 5.0,  0.9962,  0.0872,   0.0875
 6.0,  0.9945,  0.1045,   0.1051
 7.0,  0.9925,  0.1219,   0.1228
 8.0,  0.9903,  0.1392,   0.1405
 9.0,  0.9877,  0.1564,   0.1584
10.0,  0.9848,  0.1736,   0.1763
11.0,  0.9816,  0.1908,   0.1944
12.0,  0.9781,  0.2079,   0.2126
13.0,  0.9744,  0.2250,   0.2309
14.0,  0.9703,  0.2419,   0.2493
15.0,  0.9659,  0.2588,   0.2679
16.0,  0.9613,  0.2756,   0.2867
17.0,  0.9563,  0.2924,   0.3057
18.0,  0.9511,  0.3090,   0.3249
19.0,  0.9455,  0.3256,   0.3443
20.0,  0.9397,  0.3420,   0.3640
21.0,  0.9336,  0.3584,   0.3839
22.0,  0.9272,  0.3746,   0.4040
23.0,  0.9205,  0.3907,   0.4245
24.0,  0.9135,  0.4067,   0.4452
25.0,  0.9063,  0.4226,   0.4663
26.0,  0.8988,  0.4384,   0.4877
27.0,  0.8910,  0.4540,   0.5095
28.0,  0.8829,  0.4695,   0.5317
29.0,  0.8746,  0.4848,   0.5543
30.0,  0.8660,  0.5000,   0.5774
31.0,  0.8572,  0.5150,   0.6009
32.0,  0.8480,  0.5299,   0.6249
33.0,  0.8387,  0.5446,   0.6494
34.0,  0.8290,  0.5592,   0.6745
35.0,  0.8192,  0.5736,   0.7002
36.0,  0.8090,  0.5878,   0.7265
37.0,  0.7986,  0.6018,   0.7536
38.0,  0.7880,  0.6157,   0.7813
39.0,  0.7771,  0.6293,   0.8098
40.0,  0.7660,  0.6428,   0.8391
41.0,  0.7547,  0.6561,   0.8693
42.0,  0.7431,  0.6691,   0.9004
43.0,  0.7314,  0.6820,   0.9325
```

```
44.0,  0.7193,  0.6947,   0.9657
45.0,  0.7071,  0.7071,   1.0000
46.0,  0.6947,  0.7193,   1.0355
47.0,  0.6820,  0.7314,   1.0724
48.0,  0.6691,  0.7431,   1.1106
49.0,  0.6561,  0.7547,   1.1504
50.0,  0.6428,  0.7660,   1.1918
51.0,  0.6293,  0.7771,   1.2349
52.0,  0.6157,  0.7880,   1.2799
53.0,  0.6018,  0.7986,   1.3270
54.0,  0.5878,  0.8090,   1.3764
55.0,  0.5736,  0.8192,   1.4281
56.0,  0.5592,  0.8290,   1.4826
57.0,  0.5446,  0.8387,   1.5399
58.0,  0.5299,  0.8480,   1.6003
59.0,  0.5150,  0.8572,   1.6643
60.0,  0.5000,  0.8660,   1.7321
61.0,  0.4848,  0.8746,   1.8040
62.0,  0.4695,  0.8829,   1.8807
63.0,  0.4540,  0.8910,   1.9626
64.0,  0.4384,  0.8988,   2.0503
65.0,  0.4226,  0.9063,   2.1445
66.0,  0.4067,  0.9135,   2.2460
67.0,  0.3907,  0.9205,   2.3559
68.0,  0.3746,  0.9272,   2.4751
69.0,  0.3584,  0.9336,   2.6051
70.0,  0.3420,  0.9397,   2.7475
71.0,  0.3256,  0.9455,   2.9042
72.0,  0.3090,  0.9511,   3.0777
73.0,  0.2924,  0.9563,   3.2709
74.0,  0.2756,  0.9613,   3.4874
75.0,  0.2588,  0.9659,   3.7321
76.0,  0.2419,  0.9703,   4.0108
77.0,  0.2250,  0.9744,   4.3315
78.0,  0.2079,  0.9781,   4.7046
79.0,  0.1908,  0.9816,   5.1446
80.0,  0.1736,  0.9848,   5.6713
81.0,  0.1564,  0.9877,   6.3138
82.0,  0.1392,  0.9903,   7.1154
83.0,  0.1219,  0.9925,   8.1443
84.0,  0.1045,  0.9945,   9.5144
85.0,  0.0872,  0.9962,  11.4301
86.0,  0.0698,  0.9976,  14.3007
87.0,  0.0523,  0.9986,  19.0811
88.0,  0.0349,  0.9994,  28.6363
89.0,  0.0175,  0.9998,  57.2900
```

1.5.6 Explicación

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`. **Nota:** Aunque más corto de escribir, la persona que lee el código no sabe si `cos` viene del modulo `math` o de otra parte. En cuanto a *estilo* de programación, no es recomendable.

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 ocasiones dentro de un programa. El modulo `math` tiene guardado el valor de π en la variable `pi`.

[ ]: