06 - Moduls i Turtle

October 13, 2015

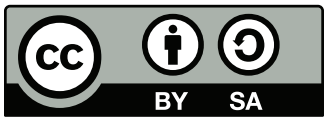

Figure 1: BY-SA

Authors : *Sonia Estradé*
José M. Gómez
Ricardo Graciani
Frank Güell
Manuel López
Xavier Luri
Josep Sabater

1 Mòduls. La llibreria Turtle

1.1 Importació de mòduls

L'intèrpret de Python només proporciona funcionalitats bàsiques del llenguatge, per exemple els tipus numèrics *int*, *float*, *double*, *complex* i funcions com *print()*, *help()*, *type()*.

La majoria de la funcionalitat (i potència) de Python està disponible en **mòduls** (també anomenats llibreries). Python proporciona una gran quantitat de mòduls estàndard, distribuits amb l'intèrpret, i n'hi ha molts més produits i distribuits per grups diversos.

Podem veure un exemple bàsic amb el mòdul estàndard *math*, que proporciona diverses funcions matemàtiques:

```
In [1]: # Importem el mòdul math
        import math
```

Un cop importat el mòdul de la forma anterior, per a usar les funcions que inclou cal mantenir el nom de la llibreria al cridar-les: *math.funcio()*. L'entorn *Jupyter* inclou una ajuda *on*line_ que us facilita saber les funcions disponibles: escriviu `math.`, pulseu la tecla <TAB> i us apareixarà un menú amb les funcions disponibles.

```
In [2]: math.
```

```
          File "<ipython-input-2-186ff497df9b>", line 1
        math.
            ^
    SyntaxError: invalid syntax
```

En l'exemple següent podeu veure com usem les funcions trigonomètriques de la llibreria *math*.

```
In [3]: angle= math.pi
        print(angle)
        print(math.sin(angle))
        print(math.cos(angle))
        print(math.log(angle))
        print(math.sqrt(angle))
```

```
3.141592653589793
1.2246467991473532e-16
-1.0
1.1447298858494002
1.7724538509055159
```

Noteu que els arguments d'aquestes funcions han de ser expressats en radians. La mateixa llibreria ofereix eines de conversió entre radians i graus.

```
In [4]: #Converting radians
        deg = math.degrees(math.pi / 2.)

        #Converting degrees
        rad = math.radians(deg)

        print ('Value in degrees {} and in radians {}.'.format(deg, rad) )
```

```
Value in degrees 90.0 and in radians 1.5707963267948966.
```

Podeu obtenir informació més detallada usant la funcióo *help()*:

```
In [5]: help(math)
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    http://docs.python.org/3.4/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.
```

```
acosh(...)
    acosh(x)

    Return the inverse hyperbolic cosine of x.

asin(...)
    asin(x)

    Return the arc sine (measured in radians) of x.

asinh(...)
    asinh(x)

    Return the inverse hyperbolic sine of x.

atan(...)
    atan(x)

    Return the arc tangent (measured in radians) of x.

atan2(...)
    atan2(y, x)

    Return the arc tangent (measured in radians) of y/x.
    Unlike atan(y/x), the signs of both x and y are considered.

atanh(...)
    atanh(x)

    Return the inverse hyperbolic tangent of x.

ceil(...)
    ceil(x)

    Return the ceiling of x as an int.
    This is the smallest integral value >= x.

copysign(...)
    copysign(x, y)

    Return a float with the magnitude (absolute value) of x but the sign
    of y. On platforms that support signed zeros, copysign(1.0, -0.0)
    returns -1.0.

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).

cosh(...)
    cosh(x)

    Return the hyperbolic cosine of x.
```

```
degrees(...)
    degrees(x)

    Convert angle x from radians to degrees.

erf(...)
    erf(x)

    Error function at x.

erfc(...)
    erfc(x)

    Complementary error function at x.

exp(...)
    exp(x)

    Return e raised to the power of x.

expm1(...)
    expm1(x)

    Return exp(x)-1.
    This function avoids the loss of precision involved in the direct evaluation of exp(x)-1 for sma

fabs(...)
    fabs(x)

    Return the absolute value of the float x.

factorial(...)
    factorial(x) -> Integral

    Find x!. Raise a ValueError if x is negative or non-integral.

floor(...)
    floor(x)

    Return the floor of x as an int.
    This is the largest integral value <= x.

fmod(...)
    fmod(x, y)

    Return fmod(x, y), according to platform C.  x % y may differ.

frexp(...)
    frexp(x)

    Return the mantissa and exponent of x, as pair (m, e).
    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.
```

```
fsum(...)
    fsum(iterable)

    Return an accurate floating point sum of values in the iterable.
    Assumes IEEE-754 floating point arithmetic.

gamma(...)
    gamma(x)

    Gamma function at x.

hypot(...)
    hypot(x, y)

    Return the Euclidean distance, sqrt(x*x + y*y).

isfinite(...)
    isfinite(x) -> bool

    Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(...)
    isinf(x) -> bool

    Return True if x is a positive or negative infinity, and False otherwise.

isnan(...)
    isnan(x) -> bool

    Return True if x is a NaN (not a number), and False otherwise.

ldexp(...)
    ldexp(x, i)

    Return x * (2**i).

lgamma(...)
    lgamma(x)

    Natural logarithm of absolute value of Gamma function at x.

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.

log10(...)
    log10(x)

    Return the base 10 logarithm of x.

log1p(...)
    log1p(x)
```

```
    Return the natural logarithm of 1+x (base e).
    The result is computed in a way which is accurate for x near zero.

log2(...)
    log2(x)

    Return the base 2 logarithm of x.

modf(...)
    modf(x)

    Return the fractional and integer parts of x.  Both results carry the sign
    of x and are floats.

pow(...)
    pow(x, y)

    Return x**y (x to the power of y).

radians(...)
    radians(x)

    Convert angle x from degrees to radians.

sin(...)
    sin(x)

    Return the sine of x (measured in radians).

sinh(...)
    sinh(x)

    Return the hyperbolic sine of x.

sqrt(...)
    sqrt(x)

    Return the square root of x.

tan(...)
    tan(x)

    Return the tangent of x (measured in radians).

tanh(...)
    tanh(x)

    Return the hyperbolic tangent of x.

trunc(...)
    trunc(x:Real) -> Integral

    Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.
```

```
DATA
    e = 2.718281828459045
    pi = 3.141592653589793

FILE
    /Users/ricardo/anaconda3/lib/python3.4/lib-dynload/math.so
```

NOTA IMPORTANT: en alguns entorns, com les versions anteriors de Notebook, us podeu estalviar la importació dels mòduls estàndard. Però això és una característica particular d'aquests entorns, però en general (i en especial pels mòduls no estàndard) heu de seguir les pautes indicades.

```
In [6]: angle = pi
        print(angle)
        print(sin(angle))


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-6-6dcf12e8b41d> in <module>()
    ----> 1 angle = pi
          2 print(angle)
          3 print(sin(angle))


        NameError: name 'pi' is not defined
```

1.2 Importació de funcions específiques

Python també ofereix la possibilitat de no importar tot el mòdul sinó només funcions específiques. En aquest cas la funció es pot usar sense el nom del mòdul; això és útil si en un programa donat hem d'usar una funció amb molta freqüència.

```
In [7]: from math import sin  # import sin method from math module
        from math import pi   # import pi data member from math module

        sin( pi / 2. )

Out[7]: 1.0
```

1.3 Importació global de totes les funcions

Python permet importar individualment totes les funcions d'un mòdul amb una sola comanda (*wild import*). La diferència amb l'import genèric és que això permet usar totes i cadascuna de les funcions sense el prefix de la llibreria.

ATENCIÓ: aquesta opció no es recomana per que pot deixar definides al nostre codi moltes funcions i variables incontrolades

```
In [8]: from math import *
        print(sin(angle))
        print(cos(angle))
        print(log(angle))
        print(sqrt(angle))
```

```
1.2246467991473532e-16
-1.0
1.1447298858494002
1.7724538509055159
```

1.4 Importació amb un nom alternatiu

Python permet la importació de mòduls especificant un nom alternatiu. Això permet per exemple usar noms més curts per a cridar les funcions del mòdul.

```
In [9]: import math as m

        print(m.sin(m.pi))
```

```
1.2246467991473532e-16
```

1.5 Llistat del contingut d'un mòdul

La funció *dir()* permet llistar el contingut d'un mòdul (les variables i funcions que es defineixen dins d'ell).

```
In [10]: # Es pot llistar el contingut d'un mòdul amb la funció "dir"
         print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh',
```

1.6 Introducció a la llibreria turtle

La llibreria turtle (tortuga en anglès) té el seu origen en un projecte de la Universitat de Cambrdige, el llenguatge de programació LOGO (http://ca.wikipedia.org/wiki/Llenguatge_de_programaci%C3%B3_Logo). Aquest llenguatge estava pensat per a ensenyar programació de forma fàcil als nens i originalment estava vinculat al control d'un petit robot en forma de tortuga que traçava dibuixos sobre un paper (d'aquí el nom de *turtle*). Posteriorment el llenguatge es va adaptar al traçat de gràfics en la pantalla d'un ordinador, mantenint la idea d'una "tortuga virtual" que és comandada per les instruccions del llenguatge.

La llibreria *turtle* de Python implementa funcions equivalents a les de LOGO i la usarem per a la iniciació a la programació.

1.6.1 Un programa bàsic usant *turtle*

Un programa mínim que usi la llibreria *turtle* ha de tenir tres parts

1. Importar la llibreria
2. Inicialitzar la finestra gràfica de treball
3. Instruccions de traçat de gràfics

El següent exemple implementa aquestes tres etapes per a dibuixar dues línies. Noteu que la finestra de treball de *turtle* (que s'anomena el *playground*) s'obre de forma separada al Notebook, i us pot quedar darrera el navegador.

Executeu l'exemple següent per veure el resultat.

```
In [11]: import turtle                # Importem la llibreria turtle

         finestra = turtle.Screen()   # Creem una finestra (playground) per a les tortugues.
                                      # De moment la creem amb les opcions per defecte, no donem
                                      # cap paràmetre

         tortuga = turtle.Turtle()    # Creem una tortuga virtual. Noteu que l'assignem
```

```
                                # a la variable tortuga, però podeu usar qualsevol nom


tortuga.speed(1)                # Fixem la velocitat de moviment de la tortuga

tortuga.shape("turtle")         # La forma en què la veurem a la finestra (forma de tortuga)
tortuga.forward(50)             # Moviment endavant de 50 unitats
tortuga.left(90)                # Gir a la esquerra de 90 graus
tortuga.forward(30)             # Moviment endavant de 30 unitats

turtle.mainloop()               # Espera a que l'usuari tanqui la finestra sense fer res més
```

La llibreria inclou moltes funcions que permeten controlar des de la mida i color de la finestra als detalls del comportament de les tortugues. En aquesta introducció només veurem les funcions bàsiques i durant la pràctica a l'aula d'informàtica veureu funcions més avençades.

1.6.2 Funcions de control de la posició de les tortugues

Quan es crea una tortuga virtual, assignant-la a una variable, tenim a la nostra disposició diverses funcions per controlar-la. Ja hem vist un primer exemple en la instrucció de control de la seva velocitat de moviment:
`tortuga.speed(1)`
La llista següent descriu les funcions de control. Tingueu en compte que aquestes funcions fan referència al moviment i posició de la tortuga en el sistema de coordenades de la finestra que s'hagi creat.

- `forward()` | `fd()`: Avençar el número de unitats donat
- `backward()` | `bk()` | `back()`: Retrocedir el número de unitats donat
- `right()` | `rt()`: Girar a la dreta l'angle donat (en graus)
- `left()` | `lt()`: Girar a l'esquerra l'angle donat (en graus)
- `goto()` | `setpos()` | `setposition()`: Moure la tortuga a les coordenades especificades
- `setx()`: Fixar la coordenada x de la tortuga
- `sety()`: Fixar la coordenada y de la tortuga
- `setheading()` | `seth()`: Fixar la orientació de la tortuga, angle respecte l'eix X
- `home()`: Moure la tortuga a l'origen de coordenades
- `circle()`: Dibuixar un circle amb el radi donat
- `dot()`: Dibuixar un punt en la posició actual de la tortuga
- `stamp()`: Dibuixar una tortuga en la posició actual
- `undo()`: Desfer la última acció
- `speed()`: Fixar la velocitat de moviment de la tortuga. El valor 0 indica no animació (les línies es dibuixen instantàniament), mentre que els valors de 1 a 10 indiquen velocitats de lenta a ràpida.

També hi ha funcions que permenten obtenir informació sobre l'estat actual de la tortuga:

- `position()` : Retorna les coordenades actuals de la tortuga (x,y)
- `heading()` : Retorna l'angle d'orientació de la tortuga, graus respecte l'eix X

Podeu veure més detalls usant l'ajuda de Python, per exemple:
`help( turtle.Turtle.color )`
El següent exemple us algunes d'aquestes funcions per crear un cercle i un semicercle. Useu-lo per a experimentar amb la tortuga.

```
In [12]: import turtle

         wn = turtle.Screen()          # Incicialitzem la finestra
         tortuga = turtle.Turtle()     # Creem la tortgua
```

```
        tortuga.speed(1)
        tortuga.shape("turtle")

        tortuga.home()                # Enviem la tortuga a l'origen (centre de la finestra)

        print(tortuga.position())     # Imprimim per consola la posició de la tortuga (0,0)
        print(tortuga.heading())      # Imprimim per consola la orientació de la tortuga 0

        tortuga.circle(50)            # Dibuixem un cercle de radi 50
        print(tortuga.position())     # i tornem a consultar la posició i orientació
        print(tortuga.heading())

        tortuga.stamp()               # Dibuixem una tortuga a la posició actual

        tortuga.circle(120, 180)      # Dibuixem un semicercle: radi 120 i angle 180
        print(tortuga.position())     # i tornem a consultar la posició i orientació
        print(tortuga.heading())

        turtle.mainloop()
```

```
(0.00,0.00)
0.0
(-0.00,0.00)
0.0
(0.00,240.00)
180.0
```

Exercici: modifiqueu el programa anterior per tal que la tortuga dibuixi un 8

1.6.3 Funcions de control del color i la forma

- `color()`: permet canviar el color de la tortuga i de les línies que traça color.
- `shape()`: canvia la forma de la tortuga. Les opcions són "arrow", "turtle", "circle", "square", "triangle", "classic"

El programa següent és un exemple d'ús d'aquestes funcions. Noteu que en aquest cas li hem posat un nom més familiar a la tortuga.

```
In [13]: import turtle

        wn = turtle.Screen()

        paula = turtle.Turtle()
        paula.color("blue")           # Canviem el color de la tortuga Paula a blau
        paula.shape("circle")         # Canviem la seva forma a la opció "circle" (rodona)

        paula.forward(50)             # Traçem una línia (blava)
        paula.left(120)

        paula.color("red")            # Canviem de color (vermell) i traçem una altra línia
        paula.forward(50)

        turtle.mainloop()
```

1.6.4 Funcions de control del traç

Les tortugues també inclouen funcions per al control de les característiques del traçat que fan. Es pot canviar el gruix del traçat i també "aixecar" o "baixar" el llàpis per aconseguir que quan es mou la tortuga es traçi o no una línia:

- `pendown()` | `pd()` | `down()`: "Baixa" el llapis. Quan la tortuga es mou es dibuixa una línia.
- `penup()` | `pu()` | `up()`: "Aixeca" el llapis. Quan la tortuga es mou no es dibuixa res.
- `pensize()` | `width()`: Defineix el gruix del traç (en unitats de la finestra).
- `pen()`: Retorna l'estat actual del llapis
- `isdown()`: Permet consultar si el llapis està baixat (retorna TRUE o FALSE)

El programa següent és un exemple d'ús d'aquestes funcions.

```
In [14]: import turtle

         wn = turtle.Screen()

         joan = turtle.Turtle()          # Ara tenim una tortuga virtual mascle
         joan.color("blue")
         joan.shape("turtle")

         joan.forward(50)                # Moviment endavant. D'entrada el llapis esta abaixat
                                         # de manera que es traçarà una línia

         joan.penup()                    # Aixequem el llapis. A partir d'ara encara que la
                                         # tortuga es mogui no es dibuixarà res
         joan.forward(20)

         joan.pendown()                  # Tornem a abaixar el llapis i dibuixem un punt
         joan.dot()

         joan.penup()                    # Aixequem el llapis
         joan.forward(20)

         joan.pendown()                  # El tornem a abaixar i traçem
         joan.forward(50)

         turtle.mainloop()
```

1.6.5 Tortugues múltiples

Un programa que usi la llibreria turtle pot usar varies tortugues, cadascuna dibuixant a la pantalla pel seu compte.

En l'exemple seguent la tortuga Paula i la tortuga Joan dibuixen respectivament un quadrat i un hexagon.

```
In [15]: import turtle

         wn = turtle.Screen()            # Creem la finestra amb alguns detalls
         wn.bgcolor("lightgreen")
         wn.title("Paula i Joan")

         paula = turtle.Turtle()         # Creem la Paula de color vermell i amb traç gruixut
         paula.color("red")
         paula.pensize(5)
```

```
joan = turtle.Turtle()        # Create el Joan amb les opcions per defecte

paula.forward(80)             # La Paula comença a dibuixar l'hexàgon
paula.left(60)
paula.forward(80)
paula.left(60)
paula.forward(80)
paula.left(60)
paula.forward(80)

joan.forward(50)              # El Joan dibuixa el quadrat
joan.left(90)
joan.forward(50)
joan.left(90)
joan.forward(50)
joan.left(90)
joan.forward(50)
joan.left(90)

paula.left(60)                # La Paula completa l'hexagon
paula.forward(80)
paula.left(60)
paula.forward(80)
paula.left(60)
paula.right(180)
paula.forward(80)


turtle.mainloop()
```

Exercici: feu un programa on una tortuga es mogui dibuixant la funció *sin(x)* i una altra la funció *cos(x)*