2. Objetos

October 31, 2018

1 Introducción a Python para ciencias e ingenierías (notebook 3)

Ing. Martín Gaitán

Links útiles

Repositorio del curso:

1.0.1 http://bit.ly/cursopy

Python "temporal" online:

1.0.2 http://try.jupyter.org

- Descarga de Python "Anaconda"
- Resumen de sintaxis markdown

1.1 Módulos y paquetes

Buenísimo estos *notebooks* pero £qué pasa si quiero reusar código?

Hay que crear **módulos**.

Por ejemplo, creemos un módulo para guardar la función que encuentra raíces de segundo grado.

Podemos abrir cualquier editor (incluído el que trae el propio jupyter), o alternativamente podemos preceder la celda con la "función magic" (que aporta Jupyter y se denotan por empezar con % o %%), en este caso `%%writefile`

El resultado es dejar el un archivo llamado `cuadratica.py` con el código de nuestra función en el mismo directorio donde tenemos el notebook (el archivo .ipynb)

```
In [2]: %%writefile cuadratica.py

        def raices(a, b=0, c=0):
            """dados los coeficientes, encuentra los valores de x tal que ax^2 + bx + c = 0"""

            discriminante = (b**2 - 4*a*c)**0.5
            x1 = (-b + discriminante)/(2*a)
            x2 = (-b - discriminante)/(2*a)
```

```
            return (x1, x2)


Writing cuadratica.py
```

Lo hemos guardado en un archivo `cuadratica.py` en el directorio donde estamos corriendo esta consola (notebook), entonces directamente podemos **importar** ese modulo.

```
In [3]: import cuadratica
```

El módulo `cuadratica` importado funciona como *"espacio de nombres"*, donde todos los objetos definidos dentro son atributos

```
In [4]: cuadratica.raices

Out[4]: <function cuadratica.raices(a, b=0, c=0)>

In [5]: cuadratica.raices(3, 2, -1)

Out[5]: (0.3333333333333333, -1.0)

In [6]: cuadratica.raices(3, 2, 1)

Out[6]: ((-0.3333333333333333+0.47140452079103173j),
         (-0.3333333333333333-0.47140452079103173j))
```

Importar un modulo es importar un "espacio de nombres", donde todo lo que el modulo contenga (funciones, clases, constantes, etc.) se accederá como un atributo del modulo de la forma `modulo.<objeto>`

Cuando el nombre del espacio de nombres es muy largo, podemos ponerle un alias

```
In [63]: import cuadratica as cuad   # igual que la primera forma pero poniendole un alias (ma

         cuad.raices(24,6,-5)

Out[63]: (0.3482423621500228, -0.5982423621500228)
```

Si **sólo queremos alguna unidad de código y no todo el módulo**, entonces podemos hacer una importación selectiva

```
In [8]: from cuadratica import raices  # sólo importa el "objeto" que indequemos y lo deja
                                       # en el espacio de nombres desde el que estamos importa

In [9]: raices?
```

Si, como sucede en general, el módulo definiera más de una unidad de código (una función, clase, constantes, etc.) podemos usar una tupla para importar varias cosas cosas al espacio de nombres actual. Por ejemplo:

```
from cuadratica import raices, integral, diferencial
```

Por último, si queremos importar todo pero no usar el prefijo, podemos usar el `*`. **Esto no es recomendado**

```
from cuadratica import *
```

1.1.1 Ejercicios

1. Cree un módulo `circunferencia.py` que defina una constante `PI` y una función `area(r)` que calcula el área para una circunferencia de radio `r`.

2. Desde una celda de la sesión intereactiva, importe todo el módulo como un alias `circle` y verifique `circle.PI` y `circle.area()`. Luego importe utilizando la forma `from circunferencia import ...` que importe también la función y la constante

3. verifique que circle.area y area son el mismo objeto

```
In [2]: %mkdir paquete             # creamos un directorio  "paquete"
```

```
In [3]: %%writefile paquete/__init__.py
```

```
Writing paquete/__init__.py
```

```
In [6]: %%writefile paquete/modulo.py

        def funcion_loca(w=300,h=200):
            _                                       =   (
                                                  255,
                                                lambda
                                       V         ,B,c
                                     :c   and Y(V*V+B,B,  c
                                       -1)if(abs(V)<6)else
                        (             2+c-4*abs(V)**-.4)/i
                          )  ;v,      x=w,h; C = range(-1,v*x
                           +1);import  struct; P = struct.pack;M, \
                    j  =b'<QIIHHHH',open('M.bmp','wb').write; k= v,x,1,24
            for X in C or 'Mandelbrot. Adapted to Python3 by @tin_nqn_':
                j(b'BM' + P(M, v*x*3+26, 26, 12,*k)) if X==-1 else 0; i,\
                    Y=_;j(P(b'BBB',*(lambda T: map(int, (T*80+T**9
                          *i-950*T  **99,T*70-880*T**18 + 701*
                         T  **9     ,T*i**(1-T**45*2))))(sum(
                        [             Y(0,(A%3/3.+X%v+(X/v+
                                      A/3/3.-x/2)/1j)*2.5
                                     /x   -2.7,i)**2 for  \
                                       A       in C
                                              [:9]])
                                                /9)
                                               )   )
```

```
Overwriting paquete/modulo.py
```