Clase 1 - Variables y funciones

March 28, 2017

1 Introducción

Esta es la primera clase de un cursillo de Python 3 que se dictó en marzo del 2017 en la Universidad Nacional de Colombia, sede Manizales. El cursillo está orientado a matemáticos y estudiantes de matemáticas que ya están ligeramente familiarizados con varios conceptos en programación (como, por ejemplo, qué es un condicional o un ciclo y cómo se usan). Todos los apuntes se hicieron en jupyter notebook, y todo en todo el curso se usó la distribución de anaconda. El cursillo fue orientado (y las notas fueron hechas por) Miguel González Duque.

En este jupyter notebook se pueden encontrar los contenidos de la primera clase del curso introductiorio a python. En esta clase se trataron los siguientes temas:

1. Usar Jupyter Notebook para procesar python.
2. Python como calculadora.
3. Tipos de variables y asignación.
4. Flujo de control: condicionales y ciclos.
5. Definición de funciones.

2 Usando Jupyter Notebook y Spyder

Jupyter notebook es algo así como un editor de python. Las ventajas que tiene son las siguientes:

1. Es modular por celdas, y las celdas pueden contener no solo código, sino texto. Esto permite comentar nuestros códigos de una forma más limpia.
2. En las partes de texto admite Markdown, un lenguaje de marcas (como HTML), y adentro de Markdown podemos escribir código TeX (por ejemplo $F(x, y, z) = (e^{x+y}, e^y + z)$ o $\sum_{i=1}^{n} a_i$)
3. Si trabajamos en Linux (o si tenemos instalado Pandoc), podemos descargar nuestro trabajo como un pdf que podemos enviar a colegas para revisiones.

2.1 Atajos usuales

En Jupyter tenemos dos modos (que nombraremos por conveniencia *modo azul* y *modo edición*). Mientras estamos en modo azul, podemos movernos entre celdas y ejecutar los siguientes atajos con el teclado:

- *ctrl* + *enter* para compilar una celda.
- *a* para insertar una celda arriba de la celda actual.

- *b* para insertar una celda debajo de la celda actual.
- *d d* para eliminar una celda.
- *i i* para interrumpir el código (por si quedamos en un ciclo infinito o por si nuestro computador se está quemando).

Un resumen más completo de los atajos se puede encontrar en el pequeño teclado al lado de *CellToolbar*, en la barra superior.

Podemos entrar a *modo edición* haciendo clic en una celda una o dos veces o presionando *enter*. Podemos salir de modo edición (y entrar a modo azul) presionando *esc*.

2.2 Un poco de Markdown

Markdown permite poner texto en *cursiva* al rodearlo de asteriscos, en **negrilla** al rodearlo entre dobles asteriscos. Podemos hacer listas usando el guión usual y hacer listas numeradas usando 1., 2., ...; podemos inclusive poner hipervínculos poniendo entre llaves [,] el texto y justo después entre paréntesis el link. Por ejemplo esto. Por último, podemos poner títulos, subtítulos y así usando numeral de forma iterada (es decir, # Título, ## Subtítulo...)

2.3 Sobre IPython

IPython es un emulador de consola de python. La única diferencia en la que nos vamos a concentrar es su manejo de la ayuda: usando '?' después de una función o paquete podemos obtener ayuda *in situ*.

2.4 Sobre Spyder

Como ya vimos, podemos usar Jupyter Notebook para experimentar y documentar el código. Una vez tenemos todos los experimentos hechos y la idea clara, podemos pasar a Spyder y escribir el script como un archivo .py. Spyder permite también experimentar un poco al ofrecer una (o más) terminales de IPython.

3 Python como calculadora

Python se puede usar como calculadora, y almacena los enteros cuán grandes sean.

```
In [1]: 2+2

Out[1]: 4

In [2]: (463456*23425)**250

Out[2]: 835655238362011361648381696169681259176238980722311968966295444824203985793
```

Sin embargo hay que tener cuidado: la aritmética de punto flotante en python no es la más precisa.

```
In [3]: 100 - 99.95

Out[3]: 0.04999999999999716
```

```
In [4]: 0.1 + 0.1 + 0.1 == 0.3

Out[4]: False

In [5]: 1/10 + 1/10 + 1/10 == 3/10

Out[5]: False

In [6]: 1 + 1 + 1 == 3

Out[6]: True
```

4 Tipos de variables en python

En python hay enteros, reales, cadenas de caracteres, booleanos, listas, conjuntos y diccionarios.

```
In [7]: print(type('Hola mundo'))
        print(type(2))
        print(type(2.0))
        print(type(False))
        print(type([1,2,3]))
        print(type({1,2,3}))
        print(type({1: 1}))

<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'list'>
<class 'set'>
<class 'dict'>
```

4.1 Operaciones entre variables

4.1.1 Operaciones entre strings

Recordemos que podemos acceder a todos los atributos y métodos de una clase (o tipo de variable) escribiendo por ejemplo *dir(str)*.

```
In [8]: 'Hola' + 'Mundo' # Concatenación

Out[8]: 'HolaMundo'

In [9]: 'hola'.upper()

Out[9]: 'HOLA'

In [10]: 'hola'.islower()
```

```
Out[10]: True
```

Podemos acceder a cierto caracter en un string en python. Para hacerlo, ponemos entre llaves la posición a la que queremos acceder:

```
In [11]: 'hola'[0]

Out[11]: 'h'
```

Como pueden notar, python comienza indexando en 0: la primera posición está rotulada con 0, la segunda con 1 y así.

```
In [12]: 'hola'[1]

Out[12]: 'o'
```

4.1.2 Operaciones entre enteros

```
In [13]: 1 + 2 #Suma

Out[13]: 3

In [14]: 1*3*4 #Multiplicación

Out[14]: 12

In [15]: 3 ** 4 #Exp.

Out[15]: 81

In [16]: 4 / 3 #División normal

Out[16]: 1.3333333333333333

In [17]: 4 // 3 #División entera

Out[17]: 1

In [18]: 21 % 6 #Módulo

Out[18]: 3
```

4.1.3 Operaciones entre floats

```
In [19]: 0.1 + 0.2

Out[19]: 0.30000000000000004

In [20]: 1.4 * 3

Out[20]: 4.199999999999999

In [21]: 4.5 ** 2.3

Out[21]: 31.7971929089206

In [22]: 4.5.as_integer_ratio()

Out[22]: (9, 2)

In [23]: 0.1.as_integer_ratio()

Out[23]: (3602879701896397, 36028797018963968)
```

4.1.4 Operaciones entre bools

```
In [24]: False or True

Out[24]: True

In [25]: True and not True

Out[25]: False

In [26]: not True

Out[26]: False
```

4.1.5 Operaciones entre listas

```
In [27]: [1,2] + [3,4,5] # Concatenación

Out[27]: [1, 2, 3, 4, 5]

In [28]: A = [1,2,3]
         A.append(4) # Añadir un elemento al final
         A

Out[28]: [1, 2, 3, 4]

In [29]: A = [1,2,3]
         A[0] # Acceder al elemento en la posición 0.

Out[29]: 1
```

En la clase entrante veremos más propiedades y operaciones que se pueden hacer con listas. Además, veremos los diccionarios y cómo se trabaja con ellos.

5 Asignación de variables y comparación.

La asignación de variables se hace con un "=", por ejemplo:

```
In [30]: verdad = True
         hola = 'hola'
         numero_favorito = 42

In [31]: print(type(verdad))
         print(type(hola))
         print(type(numero_favorito))

<class 'bool'>
<class 'str'>
<class 'int'>
```

```
In [32]: a = numero_favorito - 1
         a

Out[32]: 41

In [33]: b = 7

In [34]: b += 1
         b

Out[34]: 8

In [35]: b = b + 1
         b

Out[35]: 9
```

Podemos comprarar dos valores usando "==", por ejemplo:

```
In [36]: numero_favorito = 42
         numero_favorito == 24

Out[36]: False

In [37]: numero_favorito == 42

Out[37]: True
```

6 Flujo de control

En python están los flujos de control usuales (condicionales y ciclos), además de uno no tan normal llamado try-except. El alcance de cada flujo de control está indicado por el nivel de indentación.

6.1 Condicionales

```
In [38]: a = 9
         if a < 0:
             print('a es negativo')
         elif a == 0:
             print('a vale 0')
         else:
             print('a es positivo')

a es positivo
```

6.2 Ciclos

el ciclo for tiene una filosofía diferente al resto: mientras que en MATLAB y en C++ se itera cambiando el valor de una variable cierta cantidad de veces, en python se itera sobre objetos **secuenciables** (strings, listas, entre otros).

```
In [39]: for letter in 'Miguel':
             print(letter)

M
i
g
u
e
l
```

Lo que va justo después de la palabra *for* es una variable muda.

```
In [40]: for x in [1,2,3]:
             print(x+1)

2
3
4
```

Es una buena práctica ser tan explísitos como podamos con las variables, de tal forma que sea más legible cuando volvamos al código 80 años después o cuando se lo presentemos a un colega.

```
In [41]: for number in [1,2,3]:
             print(number+1)

2
3
4
```

Para iterar un número entero en un rango (práctica usual en el resto de lenguajes de programación), usamos los objetos *range*:

```
In [42]: for i in range(0,10):
             print(i)

0
1
2
3
4
5
```

```
6
7
8
9
```

Como podemos notar, los objetos *range* nunca toman el límite superior.
Podemos alterar el paso ingresándolo como un tercer parámetro:

```
In [43]: for j in range(0,10,2):
             print(j)

0
2
4
6
8
```

Los ciclos for admiten siempre **un** objeto para iterar sobre él. Sin embargo, éste objeto puede contener más información: puede ser por ejemplo la tupla (i,j,k), y estar iterando sobre una lista [(1,1,1), (1,1,2)...]. Por ejemplo:

```
In [44]: for i,j in [(1,1), (2,2), (2,1)]:
             print('i: ' + str(i) + ', ' + 'j: ' + str(j))

i: 1, j: 1
i: 2, j: 2
i: 2, j: 1
```

Y está también el ciclo while:

```
In [45]: a = 10
         while a > 0:
             a = a - 1
             print(a)

9
8
7
6
5
4
3
2
1
0
```

Por último, está *try-except*. La estructura del try-except es simple: se ejecuta lo que está en la parte de *try*. Si no ocurre ningún error o excepción, se omite la parte del *except*. Si sí hay un error, la ejecución para y se pasa al *except*.

```
In [46]: a = 4
         while a>-5:
             try:
                 print(1/a)
             except:
                 print('hubo un error')
             a -= 1
```

```
0.25
0.3333333333333333
0.5
1.0
hubo un error
-1.0
-0.5
-0.3333333333333333
-0.25
```

7 Definiendo funciones en python

La estructura de las funciones en python es la siguiente:

```
def nombre(parámetros):
    código
    return valor #no necesariamente.
```

Por ejemplo, definamos una función que pida un número n y devuelva el factorial $\prod_{k=1}^{n} k = n!$

```
In [47]: def factorial(n):
             acumulador = 1
             for k in range(1,n + 1):
                 acumulador *= k

             return acumulador
```

```
In [49]: n = factorial(5)
         print(n)
```

```
120
```

8 Ejercicios

8.1 Primer ejercicio

Escriba una función que tome dos listas A y B y devuelva una lista con todos las parejas ordenadas con primera componente en A y segunda en B (es decir, una *lista* $A \times B$).

8.2 Segundo ejercicio

Supongamos que construimos el objeto *función conjuntista* en python como una lista de parejas ordenadas, es decir

```
A = [1,2,3]
B = ['a', 'b', 'c']
f = [(1, 'a'), (2, 'b'), (3, 'c')]
```

Escriba una función que, dada una lista f de parejas ordenadas y dos listas A y B, determine si f es en efecto una función de A en B y retorne *True* o *False* dependiendo del caso. Recuerde que un conjunto (en este caso una lista) de parejas ordenadas se dice función si

1. La lista es un subconjunto de $A \times B$.
2. Todo elemento de A es primera componente de una pareja.
3. A cada elemento del dominio le corresponde una única imagen.

Aunque no hemos hablado formalmente de las variables de tipo *tuple* (es decir, parejas ordenadas, triplas, etc), su comportamiento es prácticamente idéntico al de las listas. Podemos acceder al primer elemento de la tupla

```
pareja = (1, 'a')
```

diciendo

```
pareja[0]
```

Es decir

```
pareja[0] == 1 # es True
pareja[1] == 'a' # también es True
```

Pista: Uno puede evaluar pertenencia con la palabra clave *in*. Por ejemplo, si definimos f como en el enunciado, la evaluación

```
(1, 'a') in f
```

sería *True*.

8.3 Tercer ejercicio

Escriba una función que retorne *True* si un número es par y *False* si un número es impar.

Pista: si esta función le toma más de una línea, es probable que esté haciendo algo mal.