Algunos apuntes sobre expresiones en Python

Las expresiones válidas en python son las que puede evaluar y obtener un resultado. Además de las expresiones aritméticas, Python reconoce otras expresiones. Las que usaremos en este curso son: expresiones lógicas, expresiones formadas por cadenas de caracteres y expresiones que involucran listas (en algunos casos también tuplas como pares ordenados). En programación decimos que las expresiones tienen un **tipo**, las expresiones artiméticas son de tipo numérico, las lógicas de tipo booleano, las cadenas de caracteres son de tipo string, y las secuencias de elementos son de tipo lista. Ejemplos:

(2 + 1) * 5 es una expresión de tipo numérico, en este caso **Integer** (entero)
(2.0 + 1.75) * 5.5 es una expresión de tipo numérico, en este caso **Real** (real)
2*(3+1) > 4 es una expresión de tipo booleano

Qué es lo que determina el tipo, o sea, ¿por qué es booleana la última expresión y no aritmética? El tipo depende del resultado de evaluar la expresión: si el resultado es numérico (entero o real) ese será el tipo, mientras que si el resultado es True o False (verdadero o falso) la expresión es booleana.
Observar que 2*(3+1) > 4 evalúa a True (es verdadero). Evaluar significa determinar el resultado de realizar las operaciones. En el intérprete se lo preguntamos a Python que, si puede, nos da el resultado, si no, nos da un mensaje de error.

Caracteres, strings y listas
A las letras (y símbolos en general, por ejemplo los del teclado) se les llama *caracteres.* Las cadenas de caracteres son expresiones de tipo *string* y se denotan entre comillas. Por ejemplo ‘Sylvia’ es la cadena formada por los caracteres ‘*S*’, ‘*y*’, ‘*l*’, ‘*v*’, ‘*i*’, ‘*a*’.

Las listas son secuencias de elementos de cualquier tipo, es decir son cadenas de elementos de cualquier tipo, incluidos caracteres y/o strings. Por ejemplo, [1,2,3] es una lista de enteros y [True, 4 > 2, 5 < 1] es una lista de valores booleanos.

Observar que para cada tipo, hay expresiones simples y compuestas.
Las simples: cada número es una expresión de tipo numérico (entero o real), los valores True y False son expresiones booleanas y cada carácter es una

expresión de tipo string, por ejemplo ‘a’, ‘#’ ‘0’ (observar que ‘0’ es un carácter y 0 es un entero). Las listas de un sólo elemento también son expresiones de tipo lista por ejemplo [1], [True], [‘a’]. Existe la lista vacía, que no tiene elementos y se denota []. Ejemplo:

```
>>> [] + [1] + [2]
[1, 2]
```

Las expresiones compuestas son como los ejemplos vistos anteriormente, donde las expresiones simples se combinan con operadores (+, >, etc). Observar que si los elementos de la lista son expresiones compuestas, python las reduce a la expresión simple. Por ejemplo:

```
>>> [4 + 2 * 6, 5/2, 4]
[16, 2, 4]
>>> [True, 4 > 2, 5 < 1]
[True, True, False]
>>> ['hola' + 'sylvia']
['holasylvia']
```

Para las listas y strings tenemos algunas operaciones predefinidas también. Por ejemplo, para cada lista tenemos:
Obtener los elementos de una lista o string usando *[posición]* (observar que la primera posición es el 0):

```
>>> [1, 'hola', True][0]
1                          # 1 es el elemento que ocupa la primera posición
>>> [1, 'hola', True][1]
'hola'                     # ‘hola’ es el elemento que ocupa la segunda posición
>>> [1, 'hola', True][2]
True                       # True es el elemento que ocupa la tercera posición
>>> 'hola'[1]
'o'                        # ‘o’ es el elemento que ocupa la segunda posición
```

Concatenar listas o strings usando +:

```
>>> [1, 'hola', True] + ['a', 'b']
[1, 'hola', True, 'a', 'b']
>>> 'hola' + 'sylvia'
'holasylvia'
```

También está definida la relación de orden lexicográfico entre strings (la que usa el diccionario). Por ejemplo:
>>> 'abc' < 'abd'
True

Reflexiones finales

Un lenguaje como el natural o el matemático, es manejado por humanos, mientras que un lenguaje de programación es manejado por un computador. Esto plantea algunas consideraciones, la primera de las cuales es que ambos tipos de lenguajes han sido diseñados por humanos y por lo tanto pueden contener inexactitudes o ambigüedades. Ahora bien, el lenguaje natural o el matemático *es usado* por humanos, quienes pueden advertir las inexactitudes y/o ambigüedades. Sin embargo, un lenguaje de programación *es usado* por un computador, que ejecutará o realizará las instrucciones sin detenerse a reflexionar ni en la pertinencia ni en el significado de las mismas, actuando siempre como si estuvieran completas y fueran exactas. Por ello, el programar exige rigurosidad en el uso del lenguaje, lo cual debería adquirirse desde cursos iniciales. Las prácticas que no se adquieren en el sistema educativo son más difíciles de adquirir fuera del mismo (o no se adquieren nunca). Por otro lado la rigurosidad exigida en programación es beneficiosa para todos los estudiantes, no sólo para aquellos que seguirán estudios en computación. Citamos palabras de Gilles Dowek en “Quelle informatique enseigner au lycée?” (2005), sobre algunos de los aportes fundamentales del aprendizaje de programación para todos los ciudadanos:

- Construir un puente entre el lenguaje y la acción: sabemos que los estudiantes perciben a veces una diferencia entre el conocimiento “libresco” que se enseña en la escuela secundaria y la ideología a la que están sujetos fuera de la escuela que valoriza la acción y la iniciativa. La idea de una dicotomía entre el mundo del lenguaje y el de la acción es particularmente perjudicial para los estudiantes cuando llegan a la conclusión de que lo que dicen es irrelevante o que el pensamiento es de poco valor para la acción. Un programa informático tiene como primera característica la de pertenecer a un lenguaje de programación, es decir, de ser un texto. Pero este texto tiene una segunda característica que es ser ejecutable, es decir, ser agente de una acción. El aprendizaje de informática enseña que decir "x = 1 ó x = 2" no es de ninguna manera lo mismo. Y también que la acción no puede llevarse a cabo sin la construcción previa de un discurso.
- Utilidad del rigor científico: en informática, los errores de los estudiantes no son castigados por un profesor, sino por una computadora inanimada y por tanto imparcial. Un programa que contiene un error de sintaxis es simplemente rechazado por el compilador. Un programa que contiene un error de semántica no da el resultado esperado. La rigurosidad no aparece como una norma impuesta desde fuera, sino simplemente como una condición de una buena comunicación entre el estudiante y la máquina. El aprendizaje de informática muestra que las computadoras, como la Naturaleza, son menos benevolentes que los profesores y que un programa puede ser incorrecto tanto porque la idea central se ha comprendido mal, como por un detalle que se ha descuidado.