Fundamentos de Programación

INTRODUCCIÓN A LOS FUNDAMENTOS DE PROGRAMACIÓN 2
Semana 02

OBJETIVO DEL LABORATORIO

Aplica los operadores aritméticos y prioridades en una sentencia de código de programación.

MARCO TEÓRICO

Todos los lenguajes de programación más o menos convencionales nos permiten almacenar datos en la memoria del sistema, y ello a través de una suerte de "cajas" en donde guardamos información. Estas cajas las llamamos comúnmente variables. Pero una de las diferencias fundamentales entre un lenguaje u otro es el modo en que maneja esas variables. De algunos decimos que son estáticos, por cuanto una caja solamente puede almacenar datos de un tipo en particular; mientras que de otros se dice que son dinámicos, cuando una misma variable puede contener un número al comienzo del programa y luego albergar una cadena de caracteres, por ejemplo.

Según esta clasificación, diremos que Python es un lenguaje dinámico. No obstante, la forma en la que entiende las "variables" es un tanto diferente respecto de otros lenguajes. En lugar de concebirlas como "cajas", las entiende como nombres o referencias. Por ello, en lugar de "variable" generalmente se emplea el término objeto. De cualquier forma, no constituye una cuestión fundamental para este curso introductorio. Para profundizar, véase Diferencia entre variables en Python y otros lenguajes.

RECURSOS

a. Hardware
- Pc Pentium IV a superior
- Conexión de red

b. Software
- Sistema Operativo Windows XP a superior
- Navegador Chrome o Firefox
- Edube Sandbox de Python desde Netacad

PROCEDIMIENTO

1. ELEMENTOS DE UN PROGRAMA

   Un programa de Python es un fichero de texto (normalmente guardado con el juego de caracteres UTF-8) que contiene expresiones y sentencias del lenguaje Python. Esas expresiones y sentencias se consiguen combinando los elementos básicos del lenguaje.

   El lenguaje Python está formado por elementos (tokens) de diferentes tipos:

   - palabras reservadas (keywords)
   - funciones integradas (built-in functions)
   - literales
   - operadores
   - delimitadores
   - identificadores

2. LITERALES

Los literales son los datos simples que Python es capaz de manejar:

- números: valores lógicos, enteros, decimales y complejos, en notación decimal, octal o hexadecimal
- cadenas de texto

3. OPERADORES

Los operadores son los caracteres que definen operaciones matemáticas (lógicas y aritméticas). Son los siguientes:

```
+       -       *       **      /       //      %       @
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=
```

4. DELIMITADORES

Los delimitadores son los caracteres que permiten delimitar, separar o representar expresiones. Son los siguientes:

```
'       "       #       \
(       )       [       ]       {       }
,       :       .       ;       @       =       ->
+=      -=      *=      /=      //=     %=      @=
&=      |=      ^=      >>=     <<=     **=
```

5. ENTEROS Y DECIMALES

- Python distingue entre números enteros y decimales. Al escribir un número decimal, el separador entre la parte entera y la parte decimal es un punto.

```
>>> 3
3
>>> 4.5
4.5
```

Nota:

Si se escribe una coma como separador entre la parte entera y la decimal, Python no lo entiende como separador, sino como una pareja de números

```
>>> 3,5
(3, 5)
```

- Si se escribe un número con parte decimal 0, Python considera el número como número decimal.

```
>>> 3.0
3.0
```

- Se puede escribir un número decimal sin parte entera, pero lo habitual es escribir siempre la parte entera:

```
>>> .3
```

```
0.3
```

6. LAS CUATRO OPERACIONES BÁSICAS

Las cuatro operaciones aritméticas básicas son la suma (+), la resta (-), la multiplicación (*) y la división (/).

- Al hacer operaciones en las que intervienen números enteros y decimales, el resultado es siempre decimal. En el caso de que el resultado no tenga parte decimal, Python escribe 0 como parte decimal para indicar que el resultado es un número decimal:

```
>>> 4.5 * 3
13.5
>>> 4.5 * 2
9.0
```

- Al sumar, restar o multiplicar números enteros, el resultado es entero.

```
>>> 1 + 2
3
>>> 3 - 4
-1
>>> 5 * 6
30
```

- Al dividir números enteros, el resultado es siempre decimal, aunque sea un número entero. Cuando Python escribe un número decimal, lo escribe siempre con parte decimal, aunque sea nula.

```
>>> 9 / 2
4.5
>>> 9 / 3
3.0
```

- Dividir por cero genera un error:

```
>>> 5 / 0
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    5 / 0
ZeroDivisionError: division by zero
>>>
```

- Cuando en una fórmula aparecen varias operaciones, Python las efectúa aplicando las reglas usuales de prioridad de las operaciones (primero multiplicaciones y divisiones, después sumas y restas).

```
>>> 1 + 2 * 3
```

```
7
>>> 10 - 4 * 2
2
```

- En caso de querer que las operaciones se realicen en otro orden, se deben utilizar paréntesis.

```
>>> (5+8) / (7-2)
2.6
```

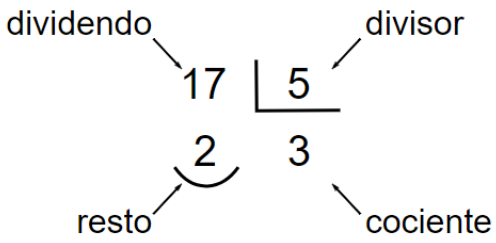
- Debido a los errores de redondeo, dos operaciones que debieran dar el mismo resultado pueden dar resultados diferentes:

```
>>> 4 * 3 / 5
2.4
>>> 4 / 5 * 3
2.4000000000000004
```

7. COCIENTE Y RESTO DE UNA DIVISIÓN

El cociente y resto de una división están relacionados con el dividendo y divisor mediante la fórmula:

dividendo = divisor * cociente + resto



El cociente es siempre un número entero, pero el resto puede ser entero o decimal. En Python, el resto y el divisor tienen siempre el mismo signo y en valor absoluto (sin signo) el resto es siempre inferior al divisor.

8. COCIENTE DE UNA DIVISIÓN

El cociente de una división se calcula en Python con el operador //. El resultado es siempre un número entero, pero será de tipo entero o decimal dependiendo del tipo de los números empleados (en caso de ser decimal, la parte decimal es siempre cero). Por ejemplo:

```
>>> 10 // 3
3
>>> 10 // 4
2
>>> 20.0 // 7
2.0
>>> 20 // 6.0
3.0
```

- El operador cociente // tiene la misma prioridad que la división:

```
>>> 26 // 5 / 2
```

```
2.5
>>> (26//5) / 2
2.5
>>> 26 // (5/2)
10.0
>>> 26 / 5 // 2
2.0
>>> (26/5) // 2
2.0
>>> 26 / (5//2)
13.0
```

9. RESTO DE UNA DIVISIÓN

El resto de una división se calcula en Python con el operador %. El resultado tendrá tipo entero o decimal, de acuerdo con el resultado de la operación.

```
>>> 10 % 3
1
>>> 10 % 4
2
>>> 10 % 5
0
>>> 10.5 % 3
1.5
```

- El operador resto % tiene la misma prioridad que la división:

```
>>> 26 % 5 / 2
0.5
>>> (26%5) / 2
0.5
>>> 26 % (5/2)
1.0
```

10. POTENCIAS Y RAÍCES

Las potencias se calculan con el operador **, teniendo en cuenta que x ** y = x^y.

Las potencias tienen prioridad sobre las multiplicaciones y divisiones.

Utilizando exponentes negativos o decimales se pueden calcular potencias inversas o raíces n-ésimas.

$$a^{-b} = \frac{1}{a^b} \qquad a^{\frac{1}{b}} = \sqrt[b]{a}$$

```
>>> 2 ** 3
8
>>> 10 ** -4
0.0001            # Recuerde que a^-b= 1/a^b
>>> 9 ** 0.5
3.0               # Recuerde que a^1/b es la raíz b-ésima de a
>>> (-1) ** 0.5   # Esto va a dar error porque es la raíz cuadrada de -1
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in ?
    (-1)**0.5 ValueError: negative number cannot be raised to a
fractional power
>>> -9 ** 0.5     # Esto no da error porque hace primero la raíz y luego le pone el -
-3
```

También se pueden calcular potencias o raíces mediante la función integrada pow(x,y). Si se da un tercer argumento, pow(x, y, z), la función calcula primero x elevado a y después calcula el resto de la división por z.

```
>>> pow(2, 3)
8
>>> pow(4, 0.5)
2.0
>>> pow(2, 3, 5)
3
```

11. CADENAS DE TEXTO

Una cadena es una secuencia inmutable de caracteres Unicode, delimitada por comillas.

12. COMILLAS SIMPLES Y DOBLES

Las cadenas de texto se pueden delimitar con comillas simples (') o con comillas dobles (").

```
>>> print('Esto es una cadena')
Esto es una cadena
>>> print("Esto es una cadena")
Esto es una cadena
```

Las cadenas se deben cerrar con las mismas comillas con las que se abrieron, de lo contrario estaremos cometiendo un error de sintaxis:

```
>>> print("Esto es una cadena')
SyntaxError: EOL while scanning string literal
>>>
```

13. COMILLAS TRIPLES

Las comillas triples permiten que las cadenas ocupen más de una línea:

```
>>> print("""Esto es una cadena
que ocupa
```

```
  varias líneas""")
Esto es una cadena
que ocupa
  varias líneas
```

14. COMILLAS DENTRO DE COMILLAS

Se pueden escribir comillas simples en cadenas delimitadas con comillas dobles y viceversa:

```
>>> print("Las comillas simples ' delimitan cadenas.")
Las comillas simples ' delimitan cadenas.
>>> print('Las comillas dobles " delimitan cadenas.')
Las comillas dobles " delimitan cadenas.
```

Pero no se pueden escribir en el interior de una cadena comillas del mismo tipo que las comillas delimitadoras:

```
>>> print("Las comillas dobles " delimitan cadenas")
SyntaxError: invalid syntax
>>>
>>> print('Las comillas simples ' delimitan cadenas')
SyntaxError: invalid syntax
>>>
```

Otra forma de escribir comillas en una cadena es utilizar los caracteres especiales \" y \' que representan los caracteres comillas dobles y simples respectivamente y que Python no interpreta en ningún caso como delimitadores de cadena:

```
>>> print('Las comillas simples \' delimitan cadenas.')
Las comillas simples ' delimitan cadenas.
>>> print("Las comillas dobles \" delimitan cadenas.")
Las comillas dobles " delimitan cadenas.
```

Se pueden utilizar ambos caracteres especiales independientemente del delimitador utilizado

```
>>> print('Las comillas simples \' y las comillas dobles \" delimitan
cadenas.')
Las comillas simples ' y las comillas dobles " delimitan cadenas.
>>> print("Las comillas simples \' y las comillas dobles \" delimitan
cadenas.")
Las comillas simples ' y las comillas dobles " delimitan cadenas.
```

15. CARACTERES ESPECIALES

Los caracteres especiales empiezan por una contrabarra (\).

- Comilla doble: \"

```
>>> print("Las comillas dobles \" delimitan cadenas.")
Las comillas dobles " delimitan cadenas.
```

- Comilla simple: \'

```
>>> print('Las comillas simples \' delimitan cadenas.')
Las comillas simples ' delimitan cadenas.
```

- Salto de línea: \n

```
>>> print("Una línea\nOtra línea")
Una línea
Otra línea
```

CONCLUSIONES Y RECOMENDACIONES DE LA EXPERIENCIA

- Sabiendo cómo funcionan los operadores en un algoritmo, podremos utilizarlos de la manera adecuada; para mostrar la salida esperada.
- Se recomienda usar el Edube SandBox para realizar nuestros ejercicios sin necesidad de instalar algún software adicional.
- Así como en algún momento se solicita cálculos con números enteros, sería factible buscar realizar lo mismo; pero con números enteros.

ACTIVIDAD VIRTUAL

1. Observa y analiza el siguiente video: https://www.youtube.com/watch?v=rRZI3kzPDV0, luego responde las siguientes preguntas:

- ¿Qué tipos de datos mencionan en el video?
- En base a esa clasificación, menciona algunos ejemplos de estos tipos de datos (Desarróllalo en Python)