Lenguajes compilados vs interpretados

TODO: Completar...

Uso interactivo de Python desde el interprete

TODO: Completar...

Variables

En programación, el término **variable** describe un espacio en la memoria donde se guarda información (números, texto, etc). Otra forma de pensar a la variable es como una *etiqueta* para determinada información. Por ejemplo, la en el código que sigue la variable *vidas* guarda el numero 3, que puede indicar la cantidad de vidas restantes en un videojuego.

```
vidas = 3
```

Nombres de variables

A la hora de elegir un nombre para una variable se debe respetar una serie de reglas que son generalmente las mismas en la mayoría de los lenguajes:

- Debe comenzar con una letra
- No debe contener espacios
- Sólo debe contener letras, números y guiones bajos (_), no se pueden utilizar caracteres "raros" como %$#-, letras con acento, etc.
- Son *case-sensitive*, es decir sensibles al uso de mayúsculas y minúsculas. Por lo cual los siguientes son tres nombres distintos: `nivel`, `Nivel`, `NIVEL`

Además, elegir un buen nombre para una variable es importante para que a medida que el código se vuelva más complejo sea sencillo de comprender, por nosotros mismos o por otros.

Algunas recomendaciones a la hora de elegir nombres para una variable son:

- Utilizar un sustantivo que describa **qué** es lo que almacena esa variable.
- Que la primer letra sea minúscula (los nombres que empiezan con mayúscula generalmente se reservan para clases).
- Si el nombre elegido para la variable tiene varias palabras se puede utilizar camelCase o separar por guiones bajos. Por ejemplo: `intentosFallidos` o `intentos_fallidos`.

Variables vs constantes

Las **constantes** son valores que se mantienen siempre igual durante toda la ejecución del programa. Son constantes, por ejemplo, los valores:

- `3`
- `4.85`
- `"Hola usuario"`
- `True`

Estos valores generalmente tienen relación con la lógica del problema.

Tipos de datos

Cada dato que forma parte de un programa, tanto las variables como las constantes, **tienen un tipo asociado** (entero, real, cadena de texto, etc).

Algunos de los tipos de datos principales en Python son

- *int*
- *float*
- *str* (string)
- *bool*

Debajo se muestran ejemplos de variables para cada uno de estos tipos.

```
i = 3 f = 7.1 s = "hola" b = True
```

El tipo de un dato es importante porque indica qué operaciones se pueden hacer con el mismo. Por ejemplo, se puede sumar números enteros o reales, pero no se puede sumar cadenas de texto o valores de verdad (booleanos).

En Python, a diferencia de otros lenguajes como C++, **no es necesario declarar el tipo de una variable**, simplemente se le asigna un valor y el entorno *"adivina"* automáticamente su tipo.

Además, el tipo de una variable puede cambiar durante la ejecución del programa, por ejemplo:

```python var = 3

# en este punto, el tipo de var es int

var = "hola"

# en este punto, el tipo de var es str

```

Por este hecho de que el tipo pueda mutar durante la ejecución, se dice que lenguajes como Python son **débilmente tipados** o **no tipados**, a diferencia de lenguajes **fuertemente tipados** como C++, donde se debe explicitar el tipo de una variable y respetarlo.

En principio puede parecer que utilizar un lenguaje débilmente tipado puede ser más rápido y cómodo (y generalmente lo es), pero también puede resultar confuso cuando existen muchas variables y, ya al no declarar el tipo, el entorno dispone de menos información para ayudarnos a encontrar errores.

Se puede obtener el tipo de una variable en cualquier momento del programa utilizando la función *type()*, por ejemplo:

```
python a = 3 type(a)
```

El operador de asignación

El **operador de asignación** (=) es el operador más importante en cualquier lenguaje de programación. Permite **copiar el valor de la expresión a la derecha del operador dentro de la variable que está a la izquierda**, por ejemplo:

```
a = 3 c = 2 * a
```

En el primer caso se copia el valor de la **constante** 3 dentro de la **variable** `a`. En el segundo caso **primero se resuelve la expresión que está a la derecha del operador** (`2 * a` que arroja el valor entero 6) y luego se coloca dicho valor dentro de la **variable** `a`.

Es importante notar que a la izquierda del operador no puede haber otra cosa que no sea una variable, por ejemplo, la siguiente sentencia no es válida:

```
3 = 2 * a
```

Conversiones

¿Qué ocurre cuando se trata de realizar una operación sobre variables/ constantes que poseen tipos diferentes? Por ejemplo:

```
a = 3/5.0 b = "Hola" + 7.5
```

Conversiones implícitas

Son las conversiones que realiza automáticamente el intérprete.

Para realizar una operación es necesario que todos los datos que intervienen sean del mismo tipo. Si no es así, el intérprete buscará convertir todos los datos a un mismo tipo antes de operar.

Ejemplos:

`a = 3/5 print(a)` Devuelve 0, ya que 5 y 3 son enteros y el resultado de dividirlos es 0 (y sobran 3).

`b = 3 a = b/5.0 print(a)` Muestra 0.6, el valor de b (3, entero) es convertido a *float* para poder operarlo con 5.0 (que es *float*).

`b = "Hola" + 7.5` La expresión arroja el siguiente error.

`TypeError: cannot concatenate 'str' and 'float' objects` El intérprete no puede convertir automáticamente la cadena *"Hola"* en un número, ni tampoco puede traducir el valor *float* 7.5 a una cadena de texto (al menos no automáticamente). Por lo tanto, no se puede aplicar el operador (+) a datos de tipo *str* y *float*.

Conversiones explícitas

A diferencia de los casos anteriores, en los que el intérprete realiza las conversiones de tipo de manera automática sin notificar al programador, el programador puede especificar determinadas conversiones de datos que desee hacer, este tipo de conversiones se llaman **conversiones explícitas** y se hacen mediante funciones nombradas como los tipos y que reciben como argumento la expresión a convertir. Por ejemplo:

```
a = 3/float(5) print(a)
```

En este caso el 5 (**int**) es convertido explicitamente a **float**, por lo cual el intérprete se ve forzado a realizar también una conversión implícita a **float** del valor 3.

```
b = "Hola" + str(7.5)
```

En este caso la conversión explícita del valor 7.5 a string sí funciona, por lo que el resultado del operador (+) es la concatenación de ambas cadenas (es decir: "Hola7.5").

Entrada y salida en Python

Entrada

Para ingresar valores por consola en Python 3 se utiliza la función *input()*. Por ejemplo:

```
a = input()
```

También se puede especificar como argumento de input un texto que se mostrará al usuario. Por ejemplo:

```
a = input("Ingrese algo: ")
```

Muy importante: en Python 3 la función *input()* devuelve lo que ingresó el usuario pero con el tipo *str* (ya que lo que el usuario ingresa desde la

consola es, técnicamente, texto). Por lo tanto, si queremos leer un valor entero, deberemos aplicar una conversión explícita a lo que el usuario ingresó antes de almacenarlo en la variable, por ejemplo:

```
a = int(input("Ingrese un entero: "))
```

ó, en más pasos

```
b = input("Ingrese un entero: ") a = int(b)
```

Salida

En Python 3 la salida por consola se realiza mediante la función *print()*, pasando dentro de los paréntesis. Por ejemplo:

```
print("Hola!!!") a = 5 print(a)
```

Strings con formato

Ver éste artículo

Forma vieja (%)

```

print('El valor de pi es aproximadamente %f.'
% (3.142)) El valor de pi es aproximadamente
3.142. ```
```

Forma nueva (format)

```

print('We are the {} who say
"{}!"'.format('knights', 'Ni')) We are the
knights who say "Ni!" ```
```

Operadores

Ver éste artículo

Estructuras de control

Ver éste artículo

??? Estructuras condicionales

TODO: Completar...

??? Estructuras repetitivas

TODO: Completar...