Facultad de Ciencias de la Administración

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

PROGRAMACIÓN I

Unidad III – Tipos de Datos

Tipos de datos elementales. Primera parte.

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración

Universidad Nacional de Entre Ríos

- **Objetivos**
  - Comprender cómo se definen variables, contantes se realizan operaciones con ellas.
  - Entender el concepto de tipo de datos.
  - Identificar las diferencias entre los tipos de datos elementales.
  - Inspeccionar cómo se representa en memoria cada tipo.
- **Temas a desarrollar:**
  - Variables y constantes. Definición. Alcance.
  - Expresiones. Operadores lógicos. Operadores aritméticos. Operadores de cadenas de texto. Precedencia.
  - Tipos de datos elementales: booleano, entero, punto flotante, cadena de texto.
  - Conversión de tipos.
  - Representación de datos en memoria.

Tipos de datos

- En la vida cotidiana estamos rodeados de datos de diferentes tipos:
  - Existen datos numéricos:
    - Estadísticas de compras y ventas de los productos.
    - Kilómetros por hora de un automóvil.
    - Latitud y longitud.
  - En forma de cadenas de caracteres:
    - Pequeñas, como siglas.
    - Enormes como un libro completo.
  - En forma de fechas, que ayudan a organizar calendarios.
- *Un **Tipo de Datos** define una representación interna, un conjunto de valores válidos y las operaciones que se pueden realizar sobre esos valores.*
- En esta asignatura se verán los tipos de datos presentes en el núcleo de **Python** o integrados.

Tipos de datos (2)

- **Python** provee un amplio abanico de tipos de datos fácilmente instanciables e intuitivos que permiten que cualquier desarrollador pueda enfocarse en qué construir y no en cómo hacerlo.

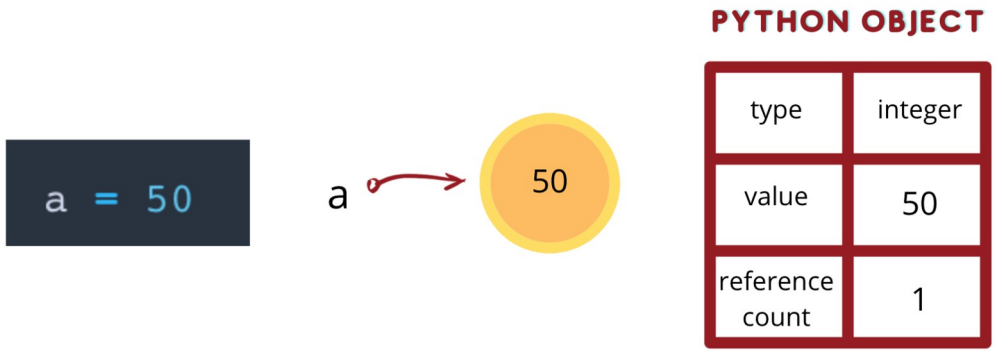
Grupo	Nombre	Tipo	Ejemplo
Numéricos	Entero	`int`	`34, 1_999, -12, -98`
	Punto flotante	`float`	`1.62, 5.7e8`
	Complejo	`complex`	`5j, 2 + 8j`
Secuencias	Listas	`list`	`[1, 2, 3]` `[3.14, False, 'c']`
	Tuplas	`tuple`	`(3, 4, True)`
	Secuencias numéricas	`range`	`range(5)`
Secuencias de texto	Cadenas de caracteres	`str`	`'casa', "color", '''tecla''', """gato"""`
Secuencias Binarias	Cadenas binarias	`bytes`	`b'coche'`
	Cadenas binarias mutables	`bytearray`	`bytearray(b'hola')`
Conjuntos	Conjunto	`set`	`set([3, true, 2]), {4, False, 12}`
	Conjunto estático	`frozenset`	`frozenset([2, 'hola', True, 3])`
Mapas	Diccionarios	`dict`	`{'x': 1, 'y': 2}, dict(x=90, y=20)`

Literales, variables y tipos de datos

- Cuando los valores se intentan trasladar al ámbito de la computación y de los lenguajes de programación, se hace uso de la ***memoria*** de un sistema, que es la herramienta que permite almacenarlos y manejarlos de forma eficiente.
- En **Python** los tipos de datos son **objetos** y es el propio intérprete el que se encarga de saber dónde se ubican en la memoria y cómo acceder a ellos.
- Por este motivo, a la hora de desarrollar aplicaciones no es necesario dedicarle mucho esfuerzo a la gestión de memoria.

Literales, variables y tipos de datos (2)

- Conceptualmente los objetos en **Python** se pueden ver como dos componentes:
  - **Una referencia**, que guarda la dirección de memoria en la que está alojado el dato.
  - **Un contenedor** que representa el almacenamiento del objeto guardado en sí, el cual contiene información relevante como el tipo de dato y la información que alberga.



Constantes literales

- Cuando hablamos de ***constantes literales*** estamos haciendo referencia al resultado de las **expresiones** o las propias formas primitivas de cada dato que ocupan un espacio en la memoria y pueden ser accedidas por el intérprete.

- Ejemplos de constantes literales:
  - Números: `1, 24, 54.`
  - Cadenas de caracteres: `'cadena de prueba', 'Python', 'Martes'`
- Todos los objetos en **Python** tienen un ***identificador*** que define dónde se encuentra en memoria.
- Esta relación de **identificador** con posición o posiciones específicas de memoria es interna al intérprete de **Python**, por lo que, a pesar de tener un **identificador**, no se puede saber con facilidad en qué zona de la memoria está alojado el valor que representa.

Literales, variables e identificadores

- Si dos **valores** tienen el mismo **identificador**, entonces, ambos valores son exactamente el mismo, lo que permite ahorrar espacio en la memoria. No obstante, esto conlleva un inconveniente:
  - Si se cambia el valor de referencia del identificador, el cambio afectará a todos los objetos que lo usen.
- Es decir si escribimos `'Hola!'` en múltiples lugares del programa el valor propiamente dicho se almacenará en la misma dirección de memoria.
  - Para saber el identificador de un objeto se puede usar la función **`id()`**.
  - Para acceder a la información que contiene el objeto con ese identificador, se puede hacer uso de la librería **`ctypes`**.

Tipos Booleanos

- En **Python**, los conceptos de ***verdadero*** y ***falso*** están presentes y modelados como **booleanos** con dos valores constantes: `True` y `False` (con la primera letra en mayúscula).
- La función **`bool()`** permite convertir cualquier valor en un valor booleano:
  - `>>> print(bool(True), bool(False))`
  - `True False`
  - `>> print(bool(0), bool(0j), bool(''), bool(None), bool(set())`
  - `False False False False False`
  - `>> print(bool(1), bool(-1), bool('casa'), bool(24))`
  - `True True True True`
- Todos los valores en **Python** tienen asociada una noción de verdad. Algunos son evaluados como **falso**:
  - Las constantes `None` y `False`
  - Los valores numéricos interpretados por cero: `0, 0.0, 0j`
  - Los objetos vacíos: `''`, `""`, `()`, `dict()`, `set()` `range(0)`, `[]`

Tipos Booleanos - Operaciones

- Las operaciones lógicas con booleanos son tres: `or`, `and` y `not`, como se muestra en la siguiente tabla:

Operador	Ejemplo	Resultado
`or`	`» x or y`	Si x es False, entonces y, de otro modo, x
`and`	`» x and y`	Si x es False, entonces x; de otro modo, y
`not`	`» not x`	Si x es False, entonces True; de otro modo, False

- Las operaciones `and` y `or` siempre devuelven uno de los operandos, no simplemente `True` o `False.`
- La operación `and` devuelve el último de sus elementos y la operación `or`, el primero que cumpla con el cortocircuito lógico:
  - » `324 and 89 and 2` dá como resultado `2`
  - » `False and 21` dá como resultado `False`
  - » `23 or 'casa'` dá como resultado `23`
  - » `True and 0 and 90` dá como resultado `0`

Tipos Booleanos – Operadores – Cortocircuito lógico

- El ***cortocircuito lógico*** es una propiedad que implementa **Python** para la evaluación de expresiones y que, además, ayuda a hacerlas más eficientes puesto que no necesita evaluar las expresiones completas
- Durante la evaluación de una expresión `and` se encuentra un elemento que es `False`, se detiene la ejecución (cortocircuita la ejecución) y devuelve ese valor.
- Hace lo mismo con las expresiones `or`, pero con los elementos que devuelvan `True`.

Tipos Numéricos

- En el núcleo de **Python** existen tres tipos numéricos definidos que permiten expresar los números enteros, los números de coma flotante (números reales) y los números complejos en forma sencilla, así como operar con ellos.
- Operadores:
  - Existen operaciones numéricas compartidas por todos los tipos numéricos de **Python**, Además, se pueden combinar valores de tipos numéricos diferentes en cuyo caso prevalecerá el más general:

Operador	Descripción	Ejemplo	Resultado
`+`	Suma	`5 + 8`	`13`
`-`	Sustracción	`90 - 10`	`80`
`*`	Multiplicación	`4 * 7`	`28`
`/`	División tradicional	`7 / 2`	`3,5`
`//`	División entera	`7 // 2`	`3`
`%`	Módulo	`7 % 3`	`1`
`**`	Exponenciación	`3 ** 4`	`81`

Enteros

- Los enteros son valores numéricos más simples e intuitivos son del tipo **`int`**.
- Ejemplos:
  - **`-1`**
  - **`100`**
  - **`+123`**
  - **`5_000_000`**
- La expresión: **`05`** (anteponiendo cero) arroja error de sintaxis.
- En **Python** los enteros pueden ser tan grandes como sea necesario, dado que no tienen un número máximo fijo asignado.
- La función **`int()`** toma un argumento de entrada y nos devuelve un valor entero equivalente.
  - **`» int(True) → 1`**
  - **`» int(False) → 0`**
- La conversión de números de punto flotante retorna un entero perdiéndose la parte fraccionaria.
  - **`» int(4.8) → 4`**

Números de punto flotante

- Los números de punto o coma flotante (o números reales) forman parte del conjunto de tipos numéricos implementado en **Python** y permite realizar operaciones de forma fácil y sencilla gracias a las operaciones que hay disponibles en el núcleo.
- Ejemplos:
  - `» 5.` → `5.0`
  - `» 5.0` → `5.0`
  - `» 05.0` → `5.0`
- Podemos escribir en notación científica:
  - `» 5e0` → `5.0`
  - `» 5.0e1` → `50.0`
  - `» 5.0e10` → `50000000000.0`
- La función que nos permite convertir cualquier valor en un número de punto flotante es `float()`.

Comparaciones

- En **Python** los objetos se pueden comprar entre sí con los ocho tipos de operadores básicos:

Operador	Ejemplo	Descripción
`>`	`» x > y`	x es mayor que y
`>=`	`» x >= y`	x es mayor o igual que y
`<`	`» x < y`	x es menor que y
`<=`	`» x <=`	X es menor o igual que y
`==`	`» x == y`	X es igual a y
`!=`	`» x != y`	X es distinto de y
`is`	`» x is y`	X es un objeto idéntico a y
`is not`	`» x is not y`	X no es un objeto idéntico a y

Bibliografía

- Óscar Ramírez Jiménez: ***"Python a fondo"*** 1era Edición. Ed. Marcombo S.L.. 2021.
- Allen Downey. ***"Think Python"***. 2Da Edición. Green Tea Press. 2015.
- Bill Lubanovic. ***"Introducing Python"***. 2Da Edición. O' Reilly. 2020.
- Eirc Matthes: ***"Python Crash Course"***. 1era Edición. Ed. No Starch Press. 2016.
- Zed A. Shaw: ***"Learn Python 3 the Hard Way"***. 1era Edición. Ed. Addison-Wesley. 2017.