Tipos de Datos Simples

1. Tipos de datos primitivos simples

- **Números (numbers):** Secuencia de dígitos (pueden incluir el - para negativos y el . para decimales) que representan números.
  Ejemplo. 0, -1, 3.1415.
- **Cadenas (strings):** Secuencia de caracteres alfanuméricos que representan texto. Se escriben entre comillas simples o dobles.
  Ejemplo. 'Hola', "Adiós".
- Booleanos (boolean): Contiene únicamente dos elementos `True` y `False` que representan los valores lógicos verdadero y falso respectivamente.

Estos datos son inmutables, es decir, su valor es constante y no puede cambiar.

2. Tipos de datos primitivos compuestos (contenedores)

- **Listas (lists):** Colecciones de objetos que representan secuencias ordenadas de objetos de distintos tipos. Se representan con corchetes y los elementos se separan por comas.
  Ejemplo. [1, "dos", [3, 4], True].
- **Tuplas (tuples).** Colecciones de objetos que representan secuencias ordenadas de objetos de distintos tipos. A diferencia de las listas son inmutables, es decir, que no cambian durante la ejecución. Se representan mediante paréntesis y los elementos se separan por comas.
  Ejemplo. (1, 'dos', 3)
- **Diccionarios (dictionaries):** Colecciones de objetos con una clave asociada. Se representan con llaves, los pares separados por comas y cada par contiene una clave y un objeto asociado separados por dos puntos.
  Ejemplo. 'pi':3.1416, 'e':2.718.

3. Clase de un dato (`type()`)

La clase a la que pertenece un dato se obtiene con el comando `type()`

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> type(1)
<class 'int'>
>>> type("Hola")
<class 'str'>
>>> type([1, "dos", [3, 4], True])
<class 'list'>
>>>type({'pi':3.1416, 'e':2.718})
<class 'dict'>
>>>type((1, 'dos', 3))
<class 'tuple'>
```

---

4. Números (clases `int` y `float`)

Secuencia de dígitos (pueden incluir el - para negativos y el . para decimales) que representan números. Pueden ser enteros (`int`) o reales (`float`).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> type(1)
<class 'int'>
>>> type(-2)
<class 'int'>
>>> type(2.3)
<class 'float'>
```

4.1. Operadores aritméticos

Los operadores aritméticos son + (suma), - (resta), * (producto), /(cociente), // (cociente división entera), % división entera), ** (potencia).

El orden de prioridad de evaluación es:

1. Funciones predefinidas
2. Potencias
3. Productos y cocientes
4. Sumas y restas

Se puede saltar el orden de evaluación utilizando paréntesis ( ).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 2+3
5
>>> 5*-2
-10
>>> 5/2
2.5
>>> 5//2
2
>>> (2+3)**2
25
```

4.2. Operadores lógicos con números

Devuelven un valor lógico o booleano.

Los operadores lógicos son == (igual que), > (mayor que), < (menor que), >= (mayor o igual que), <= (menor o igual que), != (distinto de).

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 3==3
True
>>> 3.1<=3
False
>>> -1!=1
True
```

5. Cadenas (clase `str`)

Es una secuencia de caracteres alfanuméricos que representan texto. Se escriben entre comillas sencillas ó dobles “.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
'Python'
"123"
'True'
#Cadena vacía
''
#Cadena con un espacio en blanco
' '
#Cambio de línea
'\n'
#Tabulador
'\t'
```

5.1. Acceso a los elementos de una cadena

Cada carácter tiene asociado un índice que permite acceder a él.

Cadena	P	y	t	h	o	n
Indice Positivo	0	1	2	3	4	5
Indice Negativo	-6	-5	-4	-3	-2	-1

- `c[i]` devuelve el carácter de la cadena `c` con el índice `i`.

El índice del primer carácter de la cadena es 0. También se pueden utilizar índices negativos para recorrer la cadena del final al principio.

El índice del último carácter de la cadena es -1.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 'Python'[0]
'P'
>>> 'Python'[1]
'y'
>>> 'Python'[-1]
'n'
>>> 'Python'[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

5.2. Subcadenas

- `c[i:j:k]`: Devuelve la subcadena de `c` desde el carácter con el índice `i` hasta el carácter anterior al índice `j`, tomando caracteres cada `k`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 'Python'[1:4]
'yth'
>>> 'Python'[1:1]
''
>>> 'Python'[2:]
'thon'
>>> 'Python'[:-2]
'Pyth'
>>> 'Python'[:]
'Python'
>>> 'Python'[0:6:2]
'Pto'
```

5.3. Operaciones con cadenas

- **`c1 + c2`**: Devuelve la cadena resultado de concatenar las cadenas **`c1`** y **`c2`**.
- **`c * n`**: Devuelve la cadena resultado de concatenar **`n`** copias de la cadena **`c`**.
- **`c1 in c2`**: Devuelve True si **`c1`** es una cadena contenida en **`c2`** y **`False`** en caso contrario.
- **`c1 not in c2`**: Devuelve **`True`** si **`c1`** es una cadena no contenida en **`c2`** y **`False`** en caso contrario.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 'Me gusta ' + 'Python'
'Me gusta Python'
>>> 'Python' * 3
'PythonPythonPython'
>>> 'y' in 'Python'
True
>>> 'tho' in 'Python'
True
>>> 'to' not in 'Python'
True
```

5.4. Operaciones de comparación de cadenas

- **`c1 == c2`**: Devuelve **`True`** si la cadena **`c1`** es igual que la cadena **`c2`** y **`False`** en caso contrario.
- **`c1 > c2`**: Devuelve **`True`** si la cadena **`c1`** sucede a la cadena **`c2`** y **`False`** en caso contrario.
- **`c1 < c2`**: Devuelve **`True`** si la cadena **`c1`** antecede a la cadena **`c2`** y **`False`** en caso contrario.
- **`c1 >= c2`**: Devuelve **`True`** si la cadena **`c1`** sucede o es igual a la cadena **`c2`** y **`False`** en caso contrario.
- **`c1 <= c2`**: Devuelve **`True`** si la cadena **`c1`** antecede o es igual a la cadena **`c2`** y **`False`** en caso contrario.
- **`c1 != c2`**: Devuelve **`True`** si la cadena **`c1`** es distinta de la cadena **`c2`**y **`False`** en caso contrario.

*Utilizan el orden establecido en el **`código ASCII`**.*

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 'Python' == 'python'
False
>>> 'Python' < 'python'
True
>>> 'a' > 'Z'
True
>>> 'A' >= 'Z'
False
>>> '' < 'Python'
True
```

5.5. Funciones de cadenas

- `len(c)`: Devuelve el número de caracteres de la cadena `c`.
- `min(c)`: Devuelve el carácter menor de la cadena `c`.
- `max(c)`: Devuelve el carácter mayor de la cadena `c`.
- `c.upper()`: Devuelve la cadena con los mismos caracteres que la cadena `c` pero en mayúsculas.
- `c.lower()`: Devuelve la cadena con los mismos caracteres que la cadena `c` pero en minúsculas.
- `c.title()`: Devuelve la cadena con los mismos caracteres que la cadena `c` con el primer carácter en mayúsculas y el resto en minúsculas.
- `c.split(delimitador)`: Devuelve la lista formada por las subcadenas que resultan de partir la cadena `c` usando como delimitador la cadena delimitador. Si no se especifica el delimitador utiliza por defecto el espacio en blanco.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> len('Python')
6
>>> min('Python')
'P'
>>> max('Python')
'y'
>>> 'Python'.upper()
'PYTHON'
>>> 'A,B,C'.split(',')
['A', 'B', 'C']
>>> 'I love Python'.split()
['I', 'love', 'Python']
```

5.6. Cadenas formateadas (`format()`)

- `c.format(valores)`: Devuelve la cadena `c` tras sustituir los valores de la secuencia valores en los marcadores de posición de `c`. Los marcadores de posición se indican mediante llaves en la cadena `c`, y el reemplazo de los valores se puede realizar por posición, indicando en número de orden del valor dentro de las llaves, o por nombre, indicando el nombre del valor, siempre y cuando los valores se pasen con el formato `nombre = valor`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 'Un {} vale {} {}'.format('USD', 80, '$')
'Un USD vale 80 $'
>>> 'Un {2} vale {1} {0}'.format('USD', 80, '$')
'Un USD vale 80 $'
>>> 'Un {moneda1} vale {cambio} {moneda2}'.format(moneda1 = 'USD', cambio = 80,
    moneda2 = '$')
'Un USD vale 80 $'
```

Los marcadores de posición, a parte de indicar la posición de los valores de reemplazo, pueden indicar también el formato de estos. Para ello se utiliza la siguiente sintaxis:

- `{:n}`: Alinea el valor a la izquierda rellenando con espacios por la derecha hasta los `n` caracteres.
- `{:>n}`: Alinea el valor a la derecha rellenando con espacios por la izquierda hasta los `n` caracteres.
- `{:n̂}`: Alinea el valor en el centro rellenando con espacios por la izquierda y por la derecha hasta los `n` caracteres.
- `{:nd}`: Formatea el valor como un número entero con `n` caracteres rellenando con espacios blancos por la izquierda.
- `{:n.mf}`: Formatea el valor como un número real con un tamaño de `n` caracteres (incluído el separador de decimales) y `m` cifras decimales, rellenando con espacios blancos por la izquierda.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> 'Hoy es {:^10}, mañana {:10} y pasado {:>10}'.format('lunes', 'martes', 'mié
    rcoles')
'Hoy es   lunes   , mañana martes     y pasado  miércoles'
>>> 'Cantidad {:5d}'.format(12)'
'Cantidad    12'
>>> 'Pi vale {:8.4f}'.format(3.141592)
'Pi vale   3.1416'
```

6. Datos lógicos o booleanos (clase `bool`)

Contiene únicamente dos elementos `True` y `False` que representan los valores lógicos verdadero y falso respectivamente.

`False` tiene asociado el valor 0 y `True` tiene asociado el valor 1.

6.1. Operaciones con valores lógicos

- Operadores lógicos: `==` (igual que), `>` (mayor), `<` (menor), `>=` (mayor o igual que), `<=` (menor o igual que), `!=` (distinto de).
- `not b (negación)`: Devuelve `True` si el dato booleano `b` es `False` , y `False` en caso contrario.
- `b1 and b2`: Devuelve `True` si los datos booleanos `b1` y `b2` son `True`, y `False` en caso contrario.
- `b1 or b2`: Devuelve `True` si alguno de los datos booleanos `b1` o `b2` son `True`, y `False` en caso contrario.

6.2. Tabla de Verdad

x	y	not x	x and y	x or y
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
>>> not True
False
>>> False or True
True
>>> True and False
False
>>> True and True
True
```

7. Conversión de datos primitivos simples

Las siguientes funciones convierten un dato de un tipo en otro, siempre y cuando la conversión sea posible.

- `int()` convierte a entero.
  Ejemplo. `int('12')` → `12`
  `int(True)` → `1`
  `int('c')` → Error
- `float()` convierte a real.
  Ejemplo. `float('3.14')` → `3.14`
  `float(True)` → `1.0`
  `float(ÍII')` → Error
- `str()` convierte a cadena.
  Ejemplo. `str(3.14)` → `'3.14'`
  `str(True)` → `'True'`
- `bool()` convierte a lógico.
  Ejemplo. `bool('0')` → `False`
  `bool('3.14')` → `True`
  `bool('')` → `False`
  `bool('Hola')` → `True`

8. Variables

Una variable es un identificador ligado a algún valor.
Reglas para nombrarlas:

- Comienzan siempre por una letra, seguida de otras letras o números.
- No se pueden utilizarse palabras reservadas del lenguaje.

A diferencia de otros lenguajes no tienen asociado un tipo y no es necesario declararlas antes de usarlas (tipado dinámico).

Para asignar un valor a una variable se utiliza el operador = y para borrar una variable se utiliza la instrucción `del`.

Ejercicio de Inducción: *Pruebe las siguientes líneas de código y verifique los resultados presentados:*

```
x = 3.14
y = 3 + 2
# Asignación múltiple
a1, a2 = 1, 2
# Intercambio de valores
a, b = b, a
# Incremento (equivale a x = x + 2)
x += 2
# Decremento (equivale a x = x - 1)
x -= 1
# Valor no definido
x = None
del x
```
