Tipos de datos

Python

Literales

Literales enteros: 3, 12, -23
Literales reales: 12.3, 45.6
Literales cadenas:

```
'hola que tal!'
"Muy bien"
'''Podemos \n
ir al cine'''
```

- Los literales nos permiten representar valores.
- Estos valores pueden ser de diferentes tipos → diferentes tipos de literales:

  - **Literales numéricos**
    - Para representar números enteros utilizamos cifras enteras (Ejemplos: 3, 12, -23).
    - Para los números reales utilizamos un punto para separar la parte entera de la decimal (12.3, 45.6). Podemos indicar que la parte decimal es 0, por ejemplo 10., o la parte entera es 0, por ejemplo .001.

  - **Literales cadenas**

Nos permiten representar cadenas de caracteres.
Para delimitar las cadenas podemos usar el carácter ‘ o el carácter “.
También podemos utilizar la combinación ‘’’ cuando la cadena ocupa más de una línea.
Ejemplos:

'hola que tal!’
"Muy bien“
'''Podemos \n
ir al cine'''

El caracter \n es el retorno de carro (los siguientes caracteres se escriben en una nueva línea).

Variables

- Una variables es un identificador que referencia a un valor. Para que una variable referencia a un valor se utiliza el operador de asignación =.

- El nombre de una variable, ha de empezar por una letra o por el carácter guión bajo, seguido de letras, números o guiones bajos.

```
>>> var = 5
>>> var
5
```

- python distingue entre mayúsculas y minúsculas en el nombre de una variable, pero se recomienda usar sólo minúsculas.

Expresiones

- Una expresión es una combinación de variables, literales, operadores, funciones y expresiones, que tras su evaluación o cálculo nos devuelven un valor de un determinado tipo.
- Ejemplos:

```
a + 7
(a ** 2) + b
```

Operadores

- Los operadores que podemos utilizar se clasifican según el tipo de datos con los que trabajen.
- Ejemplos:
  - Operadores aritméticos: +, -, *, /, //, %, **.
  - Operadores de cadenas: +, *
  - Operadores de asignación: =
  - Operadores de comparación: ==, !=, >=, >, <=, <
  - Operadores lógicos: and, or, not
  - Operadores de pertenencia: in, not in

Operadores- precedencia

1. Los paréntesis rompen la precedencia.
2. La potencia (**)
3. Operadores unarios (+ -)
4. Multiplicar, dividir, módulo y división entera (* % //)
5. Suma y resta (+ -)
6. Operador binario AND (&)
7. Operadores binario OR y XOR (^ |)
8. Operadores de comparación (<= < > >=)
9. Operadores de igualdad (<> == !=)
10. Operadores de asignación (=)
11. Operadores de pertenencia (in, in not)
12. Operadores lógicos (not, or, and)

Tipos de datos

- Numéricos
  - Tipo entero (int)
  - Tipo real (float)
- Booleanos (bool)
- Secuencia
  - Tipo lista (list)
  - Tipo tuplas (tuple)
- Cadenas de caracteres
  - Tipo cadena (str)
- Mapas o diccionario (dict)

Función *type()*

- La función *type()* nos devuelve el tipo de dato de un objeto dado.
- Ejemplos:

```
>>> type(5)
<class 'int'>
>>> type(5.5)
<class 'float'>
>>> type("hola")
<class 'str'>
>>> type([1,2])
<class 'list'>
```

Datos numéricos- TIPOS

- Enteros (int): Representan todos los números enteros (positivos, negativos y 0), sin parte decimal. En python3 este tipo no tiene limitación de espacio.
- Reales (float): Sirve para representar los números reales, tienen una parte decimal y otra decimal. Normalmente se utiliza para su implementación un tipo double de C.

```
>>> entero = 7
>>> type(entero)
<class 'int'>
>>> real = 7.2
>>> type (real)
<class 'float'
```

Operadores aritméticos

- +: Suma dos números
- -: Resta dos números
- *: Multiplica dos números
- /: Divide dos números, el resultado es float.
- //: División entera
- %: Módulo o resto de la división
- **: Potencia
- +, -: Operadores unarios positivo y negativo

Funciones predefinidas que trabajan con números

- abs(x): Devuelve al valor absoluto de un número.
- divmod(x,y): Toma como parámetro dos números, y devuelve una tubla con dos valores, la división entera, y el módulo o resto de la división.
- hex(x): Devuelve una cadena con la representación hexadecimal del número que recibe como parámetro.
- bin(x): Devuelve una cadena con la representación binaria del número que recibe como parámetro.
- pow(x,y): Devuelve la potencia de la base x elevado al exponente y. Es similar al operador**`.
- round(x,[y]): Devuelve un número real (float) que es el redondeo del número recibido como parámetro, podemos indicar un parámetro opcional que indica el número de decimales en el redondeo.

```
>>> abs(-7)
7
>>> divmod(7,2)
(3, 1)
>>> hex(255)
'0xff'
>>> pow(2,3)
8
>>> round(7.567,1)
7.6
```

Conversión de tipos

- int(x): Convierte el valor a entero.
- float(x): Convierte el valor a float.

```
>>> a=int(7.2)
>>> a
7
>>> type(a)
<class 'int'>
>>> a=int("345")
>>> a
345
>>> type(a)
<class 'int'>
>>> b=float(1)
>>> b
1.0
>>> type(b)
<class 'float'>
>>> b=float("1.234")
>>> b
1.234
>>> type(b)
<class 'float'>
```

- Si queremos convertir una cadena a entero, la cadena debe estar formada por caracteres numéricos, sino es así, obtenemos un error:

```
a=int("123.3")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '123.3'
```

Otras operaciones

- En el módulo math encontramos muchas de estas operaciones. Para utilizarlas vamos a importar el módulo, por ejemplo para realizar una raíz cuadrada:

```
>>> import math
>>> math.sqrt(9)
3.0
```

Datos booleanos- TIPOS

- El tipo booleano o lógico se considera como un subtipo del tipo entero.
- Se puede representar dos valores: verdadero o falso (True, False).
- Cuando se evalúa una expresión, hay determinados valores que se interpretan como False:
  - False
  - Cualquier número 0. (0, 0.0)
  - Cualquier secuencia vacía ([], (), ‘’)
  - Cualquier diccionario vacío ({})

Operaciones de comparación

- Las expresiones lógicas utilizan operadores de comparación
- Permiten comparar dos valores y devuelven un valor booleano, dependiendo de lo que este comparando.
  - ==: Igual que
  - !=: Distinto que
  - >: Mayor que
  - <: Menor que
  - <=: Menor o igual
  - >=: Mayor o igual

Operadores booleanos o lógicos

- Los operadores booleanos se utilizan para operar sobre expresiones booleanas
- Se suelen utilizar en las estructuras de control alternativas (if, while):
  - x or y: Si x es falso entonces y, sino x. Este operados sólo evalúa el segundo argumento si el primero es False.
  - x and y: Si x es falso entonces x, sino y. Este operados sólo evalúa el segundo argumento si el primero es True.
  - not x: Si x es falso entonces True, sino False.

VARIABLE

- Una variables es un identificador que referencia a un valor.
- No hay que declarar la variable antes de usarla, el tipo de la variable será el mismo que el del valor al que hace referencia.
- El tipo de una variable puede cambiar en cualquier momento:

```
>>> var = 5
>>> type(var)
<class 'int'>
>>> var = "hola"
>>> type(var)
<class 'str'>
```

Creación, borrado y ámbito de variables

- Para crear una variable simplemente tenemos que utilizar un operador de asignación, el más utilizado = para que referencia un valor.
- Si queremos borrar la variable utilizamos la instrucción del.
- El ámbito de una variable se refiere a la zona del programa donde se ha definido y existe esa variable.
- Las variables creadas dentro de funciones o clases tienen un ámbito local, es decir no existen fuera de la función o clase.

```
>>> a = 5
>>> a
5
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Modificación del valor de una variable

- En cualquier momento podemos cambiar el valor de una variable, asignándole un nuevo valor:

```
>>> a = 5
>>> a
5
>>> a = 8
>>> a
8
```

- También podemos modificar el valor de una variable, por ejemplo si queremos incrementarla en uno, podríamos usar:

```
>>> a = a + 1
```

- Aunque también podemos utilizar otro operador de asignación:

```
>>> a+=1
```

- Otros operadores de asignación: +=, -=, *=, /=, %=, **=, //=.

FUNCION *INPUT*

- No permite leer por teclado información.
- Devuelve una cadena de caracteres y puede tener como argumento una cadena que se muestra en pantalla.
- Ejemplos:

```
>>> nombre=input("Nombre:")
Nombre:jose
>>> nombre
'jose'
>>> edad=int(input("Edad:"))
Edad:23
>>> edad
23
```

FUNCION *PRINT*

- No permite escribir en la salida estándar.
- Podemos indicar varios datos a imprimir, que por defecto serán separado por un espacio.
- Podemos también imprimir varias cadenas de texto utilizando la concatenación.

```
>>> print(1,2,3)
1 2 3

>>> print("Hola son las",6,"de la tarde")
Hola son las 6 de la tarde

>>> print("Hola son las "+str(6)+" de la tarde")
Hola son las 6 de la tarde
```

Formateo cadenas de caracteres

- Con la función *print* podemos indicar el formato con el que se va a mostrar los datos
- Ejemplo:

```
>>> print("%d %f %s" % (2.5,2.5,2.5))
2 2.500000 2.5

>>> print("El producto %s cantidad=%d precio=%.2f"%
("cesta",23,13.456))
El producto cesta cantidad=23 precio=13.46
```

CADENAS de Caracteres- Definición

- Las cadenas de caracteres (str): Me permiten guardar secuencias de caracteres.
- Podemos definir una cadena de caracteres de distintas formas:

```
>>> cad1 = "Hola"
>>> cad2 = '¿Qué tal?'
>>> cad3 = '''Hola,
    que tal?'''
```

Operaciones básicas

- **Concatenación**: +: El operador + me permite unir datos de tipos secuenciales, en este caso dos cadenas de caracteres.

```
>>> "hola " + "que tal"
'hola que tal'
```

- **Repetición**: *: El operador * me permite repetir un dato de un tipo secuencial, en este caso de cadenas de caracteres.

```
>>> "abc" * 3
'abcabcabc'
```

- **Indexación**: Puedo obtener el dato de una secuencia indicando la posición en la secuencia. En este caso puedo obtener el carácter de la cadena indicando la posición (empezando por la posición

```
>>> cadena = "josé"
>>> cadena[0]
'j'
>>> cadena[3]
'é'
```

- Para obtener la longitud de un caracteres que tiene), utilizamos la función len:

```
>>> cadena = "josé"
>>> len(cadena)
4
```

Comparación de cadenas

- Las cadenas se comparan carácter a carácter, en el momento en que dos caracteres no son iguales se compara alfabéticamente (es decir, se convierte a código unicode y se comparan).
- Ejemplos

```
>>> "a">"A"
True

>>> "informatica">"informacion"
True

>>> "abcde">"abcdef"
False
```