



Universidad Mayor  
de San Andrés  
SPE Student Chapter

# CURSO PYTHON BÁSICO CLASE 2

# Operador

Un operador puede estar conformado por un carácter o un conjunto de caracteres que se ejecuta sobre una, dos o más variables para ejecutar una operación que lleve a un resultado determinado.

Algunos ejemplos de operadores comunes son los aritméticos como + (suma), - (resta) o \* (producto). Sin embargo, en lenguaje Python existen otros operadores.



# Operadores

Operador	Descripción	Ejemplo
+	Suma	<code>&gt;&gt;&gt; 3 + 2</code> 5
-	Resta	<code>&gt;&gt;&gt; 4 - 7</code> -3
-	Negación	<code>&gt;&gt;&gt; -7</code> -7
*	Multiplicación	<code>&gt;&gt;&gt; 2 * 6</code> 12
**	Exponente	<code>&gt;&gt;&gt; 2 ** 6</code> 64
/	División	<code>&gt;&gt;&gt; 3.5 / 2</code> 1.75
//	División entera	<code>&gt;&gt;&gt; 3.5 // 2</code> 1.0
%	Módulo	<code>&gt;&gt;&gt; 7 % 2</code> 1



# Expresión

Una expresión se puede definir como un conjunto de código que devuelve un valor y está conformada por una composición de operandos y operadores. A continuación, mostramos un ejemplo. Cabe destacar que cada línea es una expresión diferente:

5 + 2 # Suma del 5 y el 2

a < 10 # Compara si el valor de la variable a es menor que 10

b is None # Compara si la identidad de la variable b es None

3 \* (200 - c) # Resta a 200 el valor de c y lo multiplica por 3.



# Sentencia

Una sentencia en el lenguaje Python, es una instrucción que establece una acción. Una sentencia podría estar conformada por una o varias expresiones, claro está que no siempre es así.

En resumen, las sentencias son las normas que conforman el programa y definen su comportamiento.

Algunos ejemplos de sentencias son la asignación = o, las instrucciones if, if, else, for o while, etc.



# Sentencia

**Sentencias de más de una línea:** Generalmente, las sentencias son de una sola línea, ejemplo; `a = 2 + 3 # Asigna a la variable <a> el resultado de 2 + 3`

Pero, las sentencias muy largas pueden abarcar más de una línea. Aquí recomendamos una longitud máxima de 72 caracteres por línea. Si queremos dividir una sentencia en varias líneas, debemos utilizar el carácter (`\`).

**Ejemplo:**

`a = 2 + 3 + 5 + \`

`7 + 9 + 4 + \`

`6`



# Sentencia

En el lenguaje Python, además de la separación que se realiza con el carácter \, la continuación de línea es entendida siempre y cuando la expresión esté dentro de los caracteres {}, [] y () .

Ejemplo, podemos iniciar una lista de la siguiente manera:

```
a = [1, 2, 7,  
     3, 8, 4,  
     9]
```



# Variables

## Números

En Python, los números son un tipo de dato importante y se utilizan para realizar operaciones matemáticas. Los números pueden ser enteros, plataformas flotantes y complejos. Los números enteros representan valores enteros, mientras que las plataformas flotantes representan números con decimales. Los números complejos representan números que tienen una parte real y una imaginaria.

```
entero = 5  
flotante = 2.5  
complejo = 3 + 2j
```



# Variables

## Cadenas de texto

Las cadenas de texto son un tipo de dato que es utilizado para representar texto y se encierran en comillas simples o dobles. Las cadenas de texto son inmutables, lo que significa que una vez que se ha creado una cadena de texto, no se puede modificar.

```
nombre = "Juan"  
mensaje = "Hola, bienvenido"
```



# Variables

## Booleanos

Los valores booleanos se utilizan para representar verdadero o falso. Los valores booleanos son importantes en la toma de decisiones y en la evaluación de condiciones.

```
verdadero = True  
falso = False
```

## Listas

Las listas son un tipo de dato importante en Python que se utilizan para almacenar una colección de elementos. Los elementos de una lista pueden ser de diferentes tipos de datos. Las listas son mutables, lo que significa que se pueden agregar, eliminar y modificar elementos.

```
lista_numeros = [1, 2, 3, 4, 5]  
lista_palabras = ["hola", "adios", "bienvenido"]
```



# Variables

## Tuplas

Las tuplas son similares a las listas, pero son inmutables, lo que significa que una vez que se han creado, no se pueden modificar. Las tuplas se utilizan para almacenar elementos que no cambiarán durante la ejecución del programa.

```
tupla_numeros = (1, 2, 3, 4, 5)
tupla_palabras = ("hola", "adios", "bienvenido")
```

## Conjuntos

Los conjuntos son un tipo de dato que se utiliza para almacenar elementos únicos y no ordenados. Los elementos en un conjunto no pueden repetirse y se utilizan para realizar operaciones matemáticas como la unión, intersección y diferencia.

```
conjunto_numeros = {1, 2, 3, 4, 5}
conjunto_letras = {"a", "b", "c", "d"}
```

# Variables

## Diccionarios

Los diccionarios son un tipo de dato importante en Python que se utilizan para almacenar pares de clave-valor. Las claves se utilizan para indexar y recuperar valores. Los diccionarios son mutables y se pueden actualizar agregando y eliminando pares clave-valor.

```
diccionario = {"nombre": "Juan", "edad": 25, "direccion": "Calle 10 #20-30"}
```



# Estructuras de control

Una estructura de control, es un bloque de código que permite agrupar instrucciones de manera controlada, veremos dos tipos:

- Estructuras de control condicionales
- Estructuras de control iterativas

Pero antes de abordar estos temas debemos tener claros los siguientes conceptos:

**IDENTACIÓN**

**ASIGNACIÓN  
MÚLTIPLE**

**ENCODING**



# IDENTACIÓN

Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, requieren una identación.

No todos los lenguajes de programación, necesitan de una identación, aunque sí, es bueno usarla a fin de otorgar mayor legibilidad al código fuente. Pero en el caso de Python, la identación es obligatoria, ya que de ella, dependerá su estructura.

Una estructura de control, entonces, se define de la siguiente forma:

inicio de la estructura de control:  
expresiones



# ENCODING

El encoding no es más que una directiva que se coloca al inicio de un archivo Python, a fin de indicar al sistema, la codificación de caracteres utilizada en el archivo.

utf-8 podría ser cualquier codificación de caracteres. Si no se indica una codificación de caracteres, Python podría producir un error si encontrara caracteres extraños:

```
print "En el Nágara encontré un Nandú"
```

Producirá un error de sintaxis:

```
SyntaxError: Non-ASCII character[...]
```

En cambio, indicando el encoding correspondiente, el archivo se ejecutará con éxito:

```
# -*- coding: utf-8 -*-
print "En el Nágara encontré un Nandú"
```

Producido la siguiente salida:

```
En el Nágara encontré un Nandú
```

# ASIGNACIÓN MULTIPLE

Otra de las ventajas que Python nos provee, es la de poder asignar en una sola instrucción, múltiples variables:

```
a, b, c = 'string', 15, True
```

En una sola instrucción, estamos declarando tres variables: a, b y c y asignándoles un valor concreto a cada una:

```
>>> print a  
string  
>>> print b  
15  
>>> print c  
True
```



La asignación múltiple de variables, también puede darse utilizando como valores, el contenido de una tupla:

```
>>> mi_tupla = ('hola mundo', 2014)  
>>> texto, anio = mi_tupla  
>>> print texto  
hola mundo  
>>> print anio  
2014
```

O también, de una lista:

```
>>> mi_lista = ['Argentina', 'Buenos Aires']  
>>> pais, provincia = mi_lista  
>>> print pais  
Argentina  
>>> print provincia  
Buenos Aires
```



# ESTRUCTURAS DE CONTROL CONDICIONALES

Las estructuras de control condicionales, son aquellas que nos permiten evaluar si una o más condiciones se cumplen, para decir qué acción vamos a ejecutar. La evaluación de condiciones, solo puede arrojar 1 de 2 resultados: verdadero o falso (True o False).

Para describir la evaluación a realizar sobre una condición, se utilizan operadores relacionales (o de comparación):

Símbolo	Significado	Ejemplo	Resultado
<code>==</code>	Igual que	<code>5 == 7</code>	<code>False</code>
<code>!=</code>	Distinto que	<code>rojo != verde</code>	<code>True</code>
<code>&lt;</code>	Menor que	<code>8 &lt; 12</code>	<code>True</code>
<code>&gt;</code>	Mayor que	<code>12 &gt; 7</code>	<code>True</code>
<code>&lt;=</code>	Menor o igual que	<code>12 &lt;= 12</code>	<code>True</code>
<code>&gt;=</code>	Mayor o igual que	<code>4 &gt;= 5</code>	<code>False</code>

Y para evaluar más de una condición simultáneamente, se utilizan operadores lógicos:

Operador	Ejemplo	Explicación	Resultado
and	<code>5 == 7 and 7 &gt; 12</code>	False and False	False
and	<code>9 &lt; 12 and 12 &gt; 7</code>	True and True	True
and	<code>9 &lt; 12 and 12 &gt; 15</code>	True and False	False
or	<code>12 == 12 or 15 &lt; 7</code>	True or False	True
or	<code>7 &gt; 5 or 9 &lt; 12</code>	True or True	True
xor	<code>4 == 4 xor 9 &gt; 3</code>	True o True	False
xor	<code>4 == 4 xor 9 &lt; 3</code>	True o False	True



## Veamos un ejemplo:

Si gasto hasta 100 bs, pago con dinero en efectivo. Si no, si gasto más de 100 bs pero menos de 300 bs, pago con tarjeta de débito. Si no, pago con tarjeta de crédito.

```
if compra <= 100:  
    print "Pago en efectivo"  
  
elif compra > 100 and compra < 300:  
    print "Pago con tarjeta de débito"  
  
else:  
    print "Pago con tarjeta de crédito"
```

# ESTRUCTURAS DE CONTROL ITERATIVAS

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

**BUCLES:**

**WHILE**

**FOR**



# WHILE

Este bucle, se encarga de ejecutar una misma acción "mientras que" una determinada condición se cumpla.

Ejemplo: Mientras que año sea menor o igual a 2012, imprimir la frase "Informes del Año año".

Este código generará la siguiente salida:

```
# -*- coding: utf-8 -*-
anio = 2001
while anio <= 2012:
    print "Informes del Año", str(anio)
    anio += 1
```

```
Informes del año 2001
Informes del año 2002
Informes del año 2003
Informes del año 2004
Informes del año 2005
Informes del año 2006
Informes del año 2007
Informes del año 2008
Informes del año 2009
Informes del año 2010
Informes del año 2011
Informes del año 2012
```

Podrás notar que en cada iteración, incrementamos el valor de la variable que condiciona el bucle (anio). Si no lo hicéramos, esta variable siempre sería igual a 2001 y el bucle se ejecutaría de forma infinita, ya que la condición (anio <= 2012) siempre se estaría cumpliendo.

Pero ¿Qué sucede si el valor que condiciona la iteración no es numérico y no puede incrementarse? En ese caso, podremos utilizar una estructura de control condicional, anidada dentro del bucle, y frenar la ejecución cuando el condicional deje de cumplirse, con la palabra clave reservada break:

```
while True:  
    nombre = raw_input("Indique su nombre: ")  
    if nombre:  
        break
```

El bucle incluye un condicional anidado que verifica si la variable nombre es verdadera (solo será verdadera si el usuario tipea un texto en pantalla cuando el nombre le es solicitado). Si es verdadera, el bucle para (break). Sino, seguirá ejecutándose hasta que el usuario, ingrese un texto en pantalla.

# FOR

El bucle **for**, en Python, es aquel que nos permitirá iterar sobre una variable compleja, del tipo lista o tupla:

1) Por cada nombre en mi\_lista, imprimir nombre

```
mi_lista = ['Juan', 'Antonio', 'Pedro', 'Herminio']

for nombre in mi_lista:

    print nombre
```

2) Por cada color en mi\_tupla, imprimir color:

```
mi_tupla = ('rosa', 'verde', 'celeste', 'amarillo')

for color in mi_tupla:

    print color
```

En los ejemplos anteriores, nombre y color, son dos variables declaradas en tiempo de ejecución (es decir, se declaran dinámicamente durante el bucle), asumiendo como valor, el de cada elemento de la lista (o tupla) en cada iteración.

Otra forma de iterar con el bucle for, puede emular a while:

3) Por cada año en el rango 2001 a 2013, imprimir la frase "Informes del Año año":

```
# -*- coding: utf-8 -*-
for anio in range(2001, 2013):
    print "Informes del Año", str(anio)
```