

Tema 0 - Introducción a python

Semestre 2018-2

M. en C. Gustavo Contreras Mayén

M. en C. Abraham Lima Buendía

Facultad de Ciencias - UNAM

1 de febrero de 2018



1. Usando `python` con linux
2. Anaconda y python
3. python como una calculadora
4. Operadores relaciones en python
5. Operadores booleanos en python
6. Las variables en `python`

7. Tipos de Datos Estándar

8. Identificadores en `python`

1. Usando python con linux

1.1 Trabajo en la terminal

2. Anaconda y python

3. python como una calculadora

4. Operadores relaciones en python

5. Operadores booleanos en python

6. Las variables en python

La terminal de comandos en linux

Dentro de un entorno linux, es necesario la operación de comandos, programas, etc. dentro de una terminal.

Que es una ventana en donde debemos de escribir los comandos necesarios para que el sistema operativo ejecute la tarea.

Abrir una terminal en linux

Con la siguiente combinación de teclas, tendremos una terminal en la pantalla de nuestro equipo:

Ctrl + Alt + t

La terminal común en linux

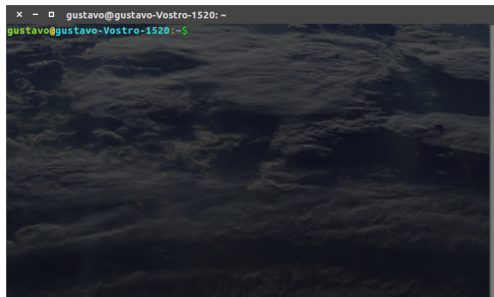



Figura: Pantalla de una terminal en linux.

El prompt de la terminal

En la terminal vemos el llamado “prompt”, que es el símbolo de dinero ~\$

A partir de este momento, linux está en la espera de las instrucciones que ingresemos con el teclado.

Precauciones

Tengan en cuenta que linux es un sistema operativo diferente, ya que una vez que tecleemos  (la tecla **Enter**) no se nos pide alguna confirmación, sencillamente se ejecuta la instrucción.

Usar python desde la terminal

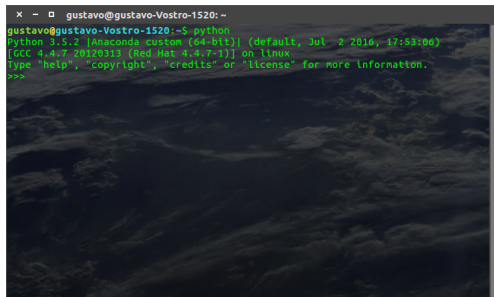
Para usar `python` desde la terminal, basta con que ingresemos la siguiente instrucción en la terminal:

```
~$ python ↵
```

Usar python desde la terminal

Para usar `python` desde la terminal, basta con que ingresemos la siguiente instrucción en la terminal:

`~$ python` 



```
x - gustavo@gustavo-Vostro-1520: ~
gustavo@gustavo-Vostro-1520:~$ python
Python 3.5.2 [Anaconda custom (64-bit)] (default, Jul 2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura: Vemos cierta información sobre la versión de python instalada en nuestro equipo.

El entorno de python en la terminal

Una vez que ya ingresamos en `python` desde la terminal, destacamos el hecho de que el prompt ya cambió: ahora se presenta como `>>>`

Y de nueva cuenta, ahora las instrucciones se deben de escribir en la línea de comandos, pero son instrucciones del lenguaje `python`.

Todo listo para trabajar con python

Para ver un saludo en pantalla, escribimos lo siguiente:

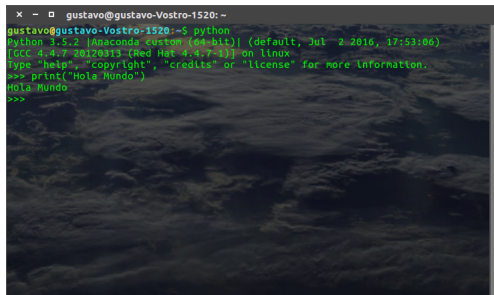
```
>>> print("Hola mundo!")
```



Todo listo para trabajar con python

Para ver un saludo en pantalla, escribimos lo siguiente:

```
>>> print("Hola mundo!")
```

A screenshot of a Linux terminal window. The window title is 'gustavo@gustavo-Vostro-1520: ~'. The prompt is 'gustavo@gustavo-Vostro-1520:~\$'. The user has entered 'python', which has started the Python 3.5.2 interpreter. The interpreter shows its version and GCC version, then prompts for a command. The user has entered 'print("Hola Mundo")', and the interpreter has printed 'Hola Mundo' and returned to the prompt '>>>'.

```
x - gustavo@gustavo-Vostro-1520: ~
gustavo@gustavo-Vostro-1520:~$ python
Python 3.5.2 [Anaconda Custom (64-bit)] (default, Jul 2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hola Mundo")
Hola Mundo
>>>
```

Figura: Obtenemos la respuesta en la siguiente línea de la terminal de python.

Salir del entorno de `python` en la terminal


Par salir del entorno de `python` en la terminal de linux, basta con escribir:

```
>>> exit ()
```



Salir del entorno de `python` en la terminal

Par salir del entorno de `python` en la terminal de linux, basta con escribir:

`>>> exit()` 



```
x - gustavo@gustavo-Vostro-1520: ~
gustavo@gustavo-Vostro-1520:~$ python
Python 3.5.2 [Anaconda Custom (64 bit)] (default, Jul 2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hola Mundo")
Hola Mundo
>>> exit()
gustavo@gustavo-Vostro-1520:~$
```

Figura: Regresamos al entorno de linux, revisa el prompt de la terminal

Entorno más agradable de trabajo

El entorno de trabajo para `python` en la terminal se vuelve en ocasiones muy tedioso, no hay manera de mejorar más allá de la tipografía y fondo de la terminal.

Pero podemos aprovechar al máximo otra herramienta para el curso: `qtConsole`, que está integrada en Anaconda¹.

¹Revisa la guía de instalación de la suite Anaconda

Usaremos ahora qtConsole

Para trabajar con **qtConsole**, con la terminal llamamos a la suite anaconda, por lo que escribimos:

```
~$ anaconda-navigator ↵
```

Veremos ahora la pantalla de Anaconda

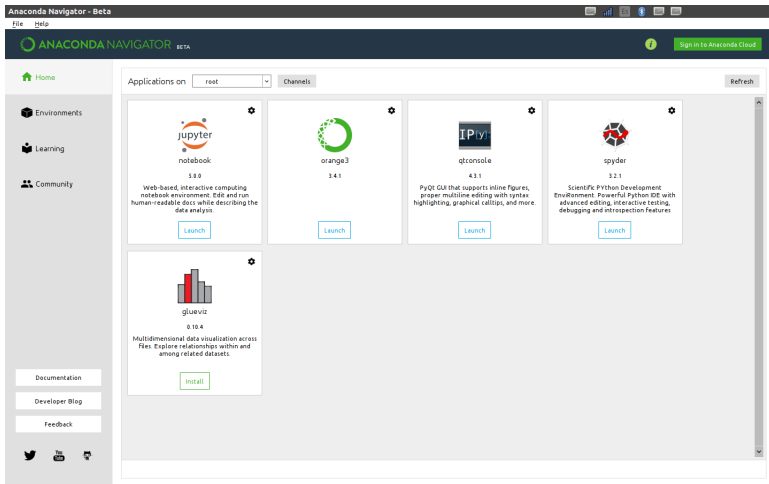
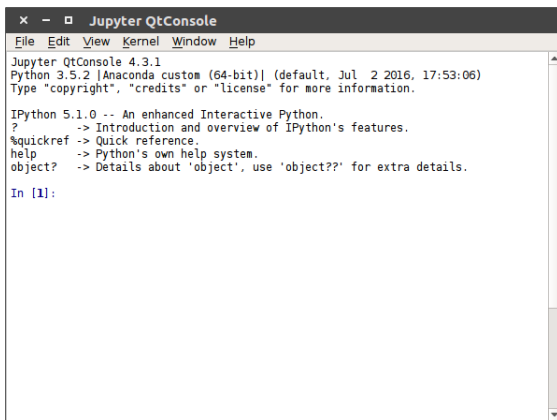


Figura: La suite Anaconda

Elegimos qtConsole



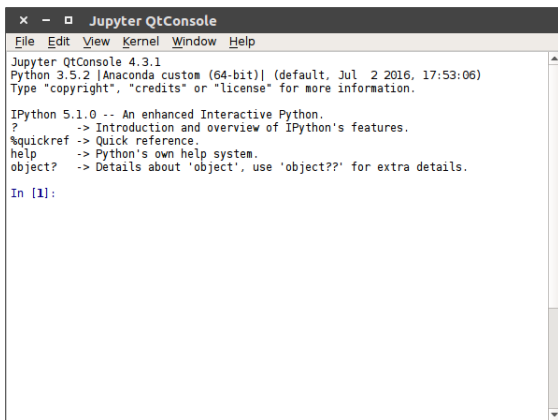
```
Jupyter QtConsole 4.3.1
Python 3.5.2 |Anaconda custom (64-bit)| (default, Jul 2 2016, 17:53:06)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Figura: La ventana de trabajo de qtConsole

La ventana de qtConsole



```
Jupyter QtConsole 4.3.1
Python 3.5.2 |Anaconda custom (64-bit)| (default, Jul 2 2016, 17:53:06)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Figura: La ventana de trabajo de qtConsole

1. Usando `python` con linux

2. Anaconda y python

2.1 Usando Anaconda

3. python como una calculadora

4. Operadores relaciones en python

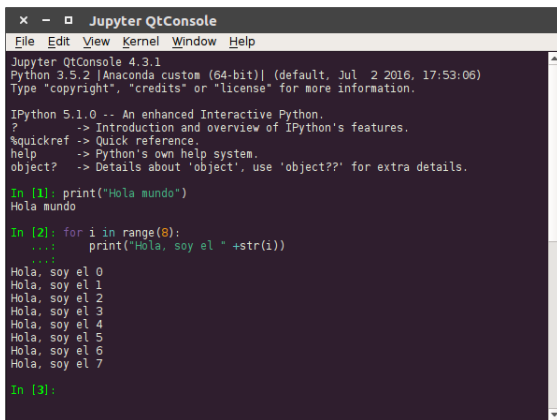
5. Operadores booleanos en python

6. Las variables en `python`

Usando `python` como una calculadora

Una vez abierta la sesión en `python`, podemos aprovechar al máximo este lenguaje: contamos con una calculadora a la mano, sólo hay que ir escribiendo las operaciones en la línea de comandos.

Personalización de qtConsole

A screenshot of the Jupyter QtConsole application window. The window has a title bar with standard OS controls and a menu bar with options: File, Edit, View, Kernel, Window, and Help. The main area is a dark-themed text editor displaying the following content:

```
Jupyter QtConsole 4.3.1
Python 3.5.2 [Anaconda custom (64-bit)] (default, Jul 2 2016, 17:53:06)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

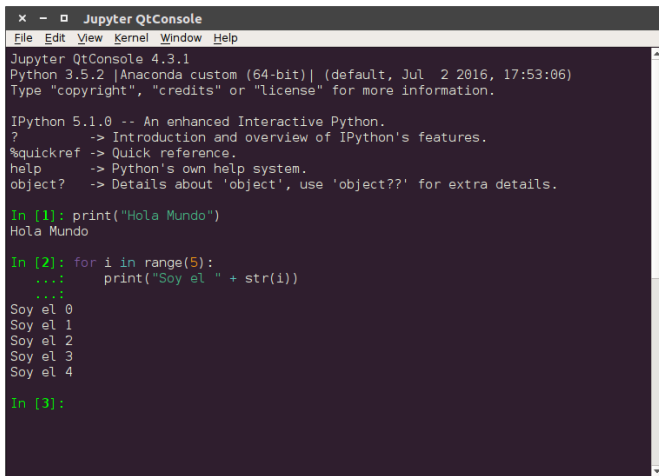
In [1]: print("Hola mundo")
Hola mundo

In [2]: for i in range(8):
...:     print("Hola, soy el " +str(i))
...:
Hola, soy el 0
Hola, soy el 1
Hola, soy el 2
Hola, soy el 3
Hola, soy el 4
Hola, soy el 5
Hola, soy el 6
Hola, soy el 7

In [3]:
```

Figura: Podemos personalizar la ventana de trabajo

Combinación de estilos



```
Jupyter QtConsole
File Edit View Kernel Window Help

Jupyter QtConsole 4.3.1
Python 3.5.2 |Anaconda custom (64-bit)| (default, Jul  2 2016, 17:53:06)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: print("Hola Mundo")
Hola Mundo

In [2]: for i in range(5):
...:     print("Soy el " + str(i))
...:
Soy el 0
Soy el 1
Soy el 2
Soy el 3
Soy el 4

In [3]:
```

Figura: Cambio en la tipografía y tamaño de letra en qtConsole

1. Usando `python` con linux

2. Anaconda y python

3. python como una calculadora

3.1 Operadores aritméticos

3.2 Tabla de operadores

4. Operadores relaciones en python

5. Operadores booleanos en python

6. Las variables en python

python como calculadora

Una vez abierta la sesión en `python`, podemos aprovechar al máximo este lenguaje: contamos con una calculadora a la mano, sólo hay que ir escribiendo las operaciones en la línea de comandos.

Algunas operaciones

Podemos hacer una suma

```
>>>3 + 200
```

Algunas operaciones

Podemos hacer una suma

```
>>>3 + 200  
203
```

División entre enteros

```
>>>30 / 1234
```

División entre enteros

```
>>>30 / 1234  
0.024311183144246355
```

Una división entre reales

```
>>>3.0 / 4.0
```


Una división entre reales

```
>>>3.0 / 4.0  
0.75
```

Una división entera

Devuelve el cociente sin decimales

```
>>>30 // 4
```

Una división entera

Devuelve el cociente sin decimales

```
>>>30 // 4  
7
```

Una división entera

Otro ejemplo de un cociente sin decimales

```
>>>4 // 3
```

Una división entera

Otro ejemplo de un cociente sin decimales

```
>>>4 // 3
```

```
1
```

Combinación de operadores aritméticos

Combinando operadores

```
>>>5.0 / 10 * 2 + 5
```

Combinación de operadores aritméticos

Combinando operadores

```
>>>5.0 / 10 * 2 + 5  
6.0
```

Combinación de operadores aritméticos

Combinando operadores

```
>>>5.0 / 10 * 2 + 5  
6.0
```

¿por qué obtenemos este resultado??

El resultado cambia cuando agrupamos con paréntesis

```
>>>5.0 / (10 * 2 + 5)
```

El resultado cambia cuando agrupamos con paréntesis

```
>>>5.0 / (10 * 2 + 5)  
0.2
```

El resultado cambia cuando agrupamos con paréntesis

```
>>>5.0 / (10 * 2 + 5)  
0.2
```

Como podemos ver, el uso de paréntesis en las expresiones tiene una particular importancia sobre la manera en que se evalúan las expresiones.

Potenciación de un número

Podemos elevar a una potencia en particular, cualquier número

```
>>>2 ** 3 ** 2
```

Potenciación de un número

Podemos elevar a una potencia en particular, cualquier número

```
>>>2 ** 3 ** 2
```

```
512
```

Potenciación de un número

Podemos elevar a una potencia en particular, cualquier número

```
>>>2 ** 3 ** 2
```

512

¿de qué manera se evaluó esta expresión?

Orden en que se evalúan las potencias

Vemos que elevar a una potencia, la manera en que se ejecuta la expresión se realiza en un sentido en particular: de derecha a izquierda.

Potenciación de un número

Podemos elevar a una potencia en particular, cualquier número >>> (2 ** 3) ** 2

Orden en que se evalúan las potencias

Vemos que elevar a una potencia, la manera en que se ejecuta la expresión se realiza en un sentido en particular: de derecha a izquierda.

Potenciación de un número

Podemos elevar a una potencia en particular, cualquier número >>> (2 ** 3) ** 2

64

Orden en que se evalúan las potencias

Vemos que elevar a una potencia, la manera en que se ejecuta la expresión se realiza en un sentido en particular: de derecha a izquierda.

Potenciación de un número

Podemos elevar a una potencia en particular, cualquier número `>>> (2 ** 3) ** 2`

64

El uso de paréntesis nos indica que la expresión contenida dentro de ellos, es la que se evalúa primero, posteriormente se sigue la regla de precedencia de operadores.

Operador módulo

El operador módulo % nos devuelve el residuo del cociente.

```
>>>17 // 3
```

Operador módulo

El operador módulo % nos devuelve el residuo del cociente.

```
>>>17 // 3  
2
```

Tabla de los operadores aritméticos

| Operador | Operación | Ejemplo | Resultado |
|-----------|----------------|-----------|-----------|
| ** | Potencia | $2 * 3$ | 8 |
| * | Multiplicación | $7 * 3$ | 21 |
| / | División | $10.5/2$ | 5.25 |
| // | Div. entera | $10.5//2$ | 5.0 |
| + | Suma | $3 + 4$ | 7 |
| - | Resta | $6 - 8$ | -2 |
| % | Módulo | $15 \% 6$ | 3 |

Precedencia de los operadores aritméticos

- 1 Las expresiones contenidas dentro de pares de paréntesis son evaluadas primero. En el caso de expresiones con paréntesis anidados, los operadores en el par de paréntesis más interno son aplicados primero.

Precedencia de los operadores aritméticos

- 1 Las expresiones contenidas dentro de pares de paréntesis son evaluadas primero. En el caso de expresiones con paréntesis anidados, los operadores en el par de paréntesis más interno son aplicados primero.
- 2 Las operaciones de exponentes son aplicadas después. Si una expresión contiene muchas operaciones de exponentes, los operadores son aplicados de derecha a izquierda.

Precedencia de los operadores aritméticos

- 3 La multiplicación, división y módulo son las siguientes en ser aplicadas. Si una expresión contiene muchas multiplicaciones, divisiones u operaciones de módulo, los operadores se aplican de izquierda a derecha.

Precedencia de los operadores aritméticos

- 3 La multiplicación, división y módulo son las siguientes en ser aplicadas. Si una expresión contiene muchas multiplicaciones, divisiones u operaciones de módulo, los operadores se aplican de izquierda a derecha.
- 4 Suma y resta son las operaciones que se aplican por último. Si una expresión contiene muchas operaciones de suma y resta, los operadores son aplicados de izquierda a derecha. La suma y resta tienen el mismo nivel de precedencia.

1. Usando `python` con `linux`
2. Anaconda y `python`
3. `python` como una calculadora
4. Operadores relaciones en `python`
 - 4.1 Operadores relacionales
5. Operadores booleanos en `python`
6. Las variables en `python`

Operadores relacionales

Cuando se comparan dos (o más expresiones) mediante un operador, el tipo de dato que se devuelve es lógico: **True** o **False**, que también tienen una representación de tipo numérico:

- **True** = 1
- **False** = 0

Operaciones aritméticas y relacionales

Podemos extender el manejo con `python`, al combinar las operaciones aritméticas y relaciones, nótese que siempre tendremos un valor booleano que se devuelve.

```
>>> 1 + 2 > 7 - 3
```

Operaciones aritméticas y relacionales

Podemos extender el manejo con `python`, al combinar las operaciones aritméticas y relaciones, nótese que siempre tendremos un valor booleano que se devuelve.

```
>>> 1 + 2 > 7 - 3  
False
```

```
>>> 1 < 2 < 3
```

```
>>> 1 < 2 < 3  
True
```

El operador de comparación de igualdad

El doble signo igual (==) es el operador de igualdad, a diferencia del operador = que es el operador de asignación.

```
>>> 1 > 2 == 2 < 3
```

El operador de comparación de igualdad

El doble signo igual (==) es el operador de igualdad, a diferencia del operador = que es el operador de asignación.

```
>>> 1 > 2 == 2 < 3  
False
```



```
>>> 3 > 4 < 5
```

```
>>> 3 > 4 < 5
```

```
False
```

```
>>> 3 > 4 < 5
```

```
False
```

```
>>> 1.0 / 3 < 0.3333
```

```
>>> 3 > 4 < 5
```

```
False
```

```
>>> 1.0 / 3 < 0.3333
```

```
False
```

```
>>> 3 > 4 < 5
```

```
False
```

```
>>> 1.0 / 3 < 0.3333
```

```
False
```

```
>>> 5.0 / 3 >= 11 / 7.0
```

```
>>> 3 > 4 < 5  
False
```

```
>>> 1.0 / 3 < 0.3333  
False
```

```
>>> 5.0 / 3 >= 11 / 7.0  
True
```

Las expresiones se pueden complicar cada vez más, por lo que hay que mantener atención al momento de escribirlas.

Tabla de operadores relacionales

| Operador | Operación | Ejemplo | Resultado |
|--------------------|---------------|---------------------------|-----------|
| <code>==</code> | Igual a | <code>4 == 5</code> | False |
| <code>!=</code> | Diferente | <code>2 != 3</code> | True |
| <code><</code> | Menor que | <code>10 < 4</code> | False |
| <code>></code> | Mayor que | <code>5 > -4</code> | True |
| <code><=</code> | Menor o igual | <code>7 <= 7</code> | True |
| <code>>=</code> | Mayor o igual | <code>3.5 >= 10</code> | False |

1. Usando `python` con `linux`
2. Anaconda y `python`
3. `python` como una calculadora
4. Operadores relaciones en `python`
5. Operadores booleanos en `python`
 - 5.1 Operadores booleanos
6. Las variables en `python`

Operadores booleanos

En el caso del operador booleano `and` y el operador `or` evalúan una expresión compuesta por dos (o más términos).

Si ambas expresiones tienen el valor `True`, el valor que devuelve la evaluación de la expresión que involucra a las dos primeras, es `True`.

Como se verá en la tabla de verdad, se necesita una condición particular para que el valor que devuelva la comparación, sea **False**.

| Operador | Operación | Ejemplo | Resultado |
|----------|------------|------------------------------|--------------|
| and | Conjunción | False and True | False |
| or | Disyunción | False or True | True |
| not | Negación | not True | False |

Tabla de verdad de los operadores booleanos

| A | B | A and B | A or B | not A |
|-------|-------|---------|--------|-------|
| True | True | True | True | False |
| True | False | False | True | False |
| False | True | False | True | True |
| False | False | False | False | True |

1. Usando `python` con linux
2. Anaconda y python
3. python como una calculadora
4. Operadores relaciones en python
5. Operadores booleanos en python
6. Las variables en `python`
 - 6.1 Tipos de variables

Tipos de variables

Las variables en `python` sólo son ubicaciones de memoria reservadas para almacenar valores.

Esto significa que cuando se crea una variable, se reserva un poco de espacio disponible en la memoria.

Basándose en el tipo de datos de una variable, el intérprete asigna memoria y decide qué se puede almacenar en la memoria reservada.

Por lo tanto, al asignar diferentes tipos de datos a las variables, se pueden almacenar **enteros**, **decimales** o **caracteres (cadenas)** en estas variables.

Asignando valores a variables

Las variables de `python` no necesitan una declaración explícita para reservar espacio de memoria.

La declaración ocurre automáticamente cuando se asigna un valor a una variable. *El signo igual (=) se utiliza para asignar valores a las variables.*

El término a la izquierda del operador = es el *nombre de la variable* y el término a la derecha del operador = es el *valor almacenado* en la variable.

Ejemplos

Los comentarios en `python` se indican con el símbolo `#`, el texto no se interpreta como una instrucción.

```
>>>contador = 100 # Asignacion de tipo entero
>>>distancia = 1000.0 # De punto flotante
>>>nombre = "Chucho" # Una cadena de caracteres

>>>print(contador)
>>>print(distancia)
>>>print(nombre)
```

Resultado

100

1000.0

Chucho

Asignación múltiple de valores

En `python` podemos asignar un valor único a varias variables simultáneamente.

```
>>> A = b = c = 1
```

```
>>> print(A)
```

```
>>> print(b)
```

```
>>> print(c)
```

Asignación múltiple de valores

En `python` podemos asignar un valor único a varias variables simultáneamente.

```
>>> A = b = c = 1
```

```
>>> print(A)
```

```
>>> print(b)
```

```
>>> print(c)
```

```
1
```

```
1
```

```
1
```

Asignación múltiple de valores

En `python` podemos asignar un valor único a varias variables simultáneamente.

```
>>> A = b = c = 1
```

```
>>> print(A)
```

```
>>> print(b)
```

```
>>> print(c)
```

```
1
```

```
1
```

```
1
```

En el ejemplo, se crea un objeto entero con el valor 1, y las tres variables se asignan a la misma ubicación de memoria.

Asignación múltiple a varias variables

También puede asignar varios objetos a varias variables.

```
>>> A, b, c = 1, 2, "Alicia"  
>>> print (A)  
>>> print (b)  
>>> print (c)
```

Asignación múltiple a varias variables

También puede asignar varios objetos a varias variables.

```
>>> A, b, c = 1, 2, "Alicia"
```

```
>>> print (A)
```

```
>>> print (b)
```

```
>>> print (c)
```

```
1
```

```
2
```

```
Alicia
```

Asignación múltiple a varias variables

También puede asignar varios objetos a varias variables.

```
>>> A, b, c = 1, 2, "Alicia"  
>>> print (A)  
>>> print (b)  
>>> print (c)
```

```
1  
2  
Alicia
```

Aquí, dos objetos enteros con valores 1 y 2 se asignan a las variables *A* y *b* respectivamente, y un objeto de cadena con el valor **Alicia** se asigna a la variable *c*.

1. Usando `python` con linux
2. Anaconda y python
3. python como una calculadora
4. Operadores relaciones en python
5. Operadores booleanos en python
6. Las variables en `python`

7. Tipos de Datos Estándar

7.1 Los tipos de datos en `python`

7.2 Tipos de datos numéricos

7.3 Cadenas

7.4 Listas

7.5 Tuplas

7.6 Diccionarios

8. Identificadores en `python`

Los datos almacenados en la memoria pueden ser de varios tipos. Por ejemplo, la edad de una persona se almacena como un valor numérico y su dirección se almacena como caracteres alfanuméricos.

En `python` se cuenta con varios tipos de datos estándar que se utilizan para definir las operaciones posibles entre ellos y el método de almacenamiento para cada uno de ellos.

Tipos de datos

Los tipos de datos que se utilizan en `python` son cinco:

- 1 Números.

Tipos de datos

Los tipos de datos que se utilizan en `python` son cinco:

- 1 Números.
- 2 Cadena.

Tipos de datos

Los tipos de datos que se utilizan en `python` son cinco:

- 1 Números.
- 2 Cadena.
- 3 Lista.

Tipos de datos

Los tipos de datos que se utilizan en `python` son cinco:

- 1 Números.
- 2 Cadena.
- 3 Lista.
- 4 Tupla.

Tipos de datos

Los tipos de datos que se utilizan en `python` son cinco:

- 1 Números.
- 2 Cadena.
- 3 Lista.
- 4 Tupla.
- 5 Diccionario.

Números

Los tipos de datos numéricos almacenan valores numéricos.

Los objetos numéricos se crean cuando se les asigna un valor.

Declaración en variables

```
>>> Var1 = Var2 = 10  
>>> print (Var1)  
>>> print (Var2)
```

Declaración en variables

```
>>> Var1 = Var2 = 10
```

```
>>> print (Var1)
```

```
>>> print (Var2)
```

10

10

Eliminar variables en python

También se puede eliminar la referencia a un objeto numérico utilizando la sentencia `del`

La sintaxis de la sentencia `del` es:

```
del var1 [, var2 [, var3 [... , varN]]]
```

Se puede eliminar un solo objeto o varios objetos utilizando la sentencia `del`

Por ejemplo:

```
del var
```

```
del variable1, variable2
```

Tipos de datos numéricos

En python se soportan tres tipos numéricos diferentes:

- 1 Int (enteros con signo)

Tipos de datos numéricos

En python se soportan tres tipos numéricos diferentes:

- ❶ Int (enteros con signo)
- ❷ Flotante (valores reales de punto flotante)

Tipos de datos numéricos

En python se soportan tres tipos numéricos diferentes:

- 1 Int (enteros con signo)
- 2 Flotante (valores reales de punto flotante)
- 3 Complejos (números complejos)

Números enteros

Los números enteros son aquellos que no tienen decimales, tanto positivos como negativos (además del cero). En `python` se representan mediante el tipo `int` (de integer, entero).

Todos los números enteros en `python3` se representan como enteros largos.

Números reales o flotantes

Los números reales son los que tienen decimales. En `python` se expresan mediante el tipo `float`.

En `python` se implementa su tipo `float` utilizando 64 bits, en concreto se sigue el estándar IEEE 754²: 1 bit para el signo, 11 para el exponente, y 52 para la mantisa.

²En el Tema 1, ampliaremos esta información

Números complejos

Un número complejo consiste en un par ordenado de números reales de coma flotante denotados por

$$x + y j$$

donde x e y son números reales, $y j$ es la unidad imaginaria.

Ejemplos de tipos de datos numéricos

| int | float | complex |
|--------|--------------|----------------|
| 10 | 0.0 | $3.14j$ |
| 100 | 15.20 | $45.j$ |
| 100 | -15.20 | $23.15 + 7.5j$ |
| 080 | $32.3 + e18$ | $0.876j$ |
| -0490 | -90. | $-0.645 + 0j$ |
| -0x260 | $-32.54e100$ | $3e + 26j$ |
| 0x69 | $70.2 - E12$ | $4.53e - 7j$ |

Cadenas en python

Las cadenas en `python` se identifican como un conjunto contiguo de caracteres representados en las comillas.

Con `python` se permite cualquier par de 'comillas simples' o comillas "dobles".

Operación con cadenas

Los subconjuntos de cadenas pueden ser tomados usando el operador de corte `[]` y `[:]` con los índices comenzando en 0 al inicio de la cadena hasta llegar a `-1` al final de la misma.

Operación con cadenas

Los subconjuntos de cadenas pueden ser tomados usando el operador de corte `[]` y `[:]` con los índices comenzando en 0 al inicio de la cadena hasta llegar a `-1` al final de la misma. El signo más `+` es el operador de concatenación de cadenas y el asterisco `*` es el operador de repetición.

Ejemplos

```
>>> cadena = 'Hola Mundo!'

>>> print (cadena)
>>> print (cadena[0])
>>> print (cadena[2:5])
>>> print (cadena[2:])
>>> print (cadena * 2)
>>> print (cadena + "PUMAS")
```


Resultados

```
>>> print (cadena[0])
```

Resultados

```
>>> print (cadena[0])
```

Presenta el primer caracter de la cadena

H

Resultados

```
>>> print (cadena[0])
```

Presenta el primer caracter de la cadena

H

```
>>> print (cadena[2:5])
```

Resultados

```
>>> print (cadena[0])
```

Presenta el primer caracter de la cadena

H

```
>>> print (cadena[2:5])
```

Presenta los caracteres de la 3a a la 5a posicion

la

Resultados

```
>>> print (cadena[2:])
```

Resultados

```
>>> print (cadena[2:])
```

Presenta la cadena que inicia a partir del 3er caracter

la Mundo!

Resultados

```
>>> print (cadena[2:])
```

Presenta la cadena que inicia a partir del 3er caracter

la Mundo!

```
>>> print (cadena * 2)
```

Resultados

```
>>> print (cadena[2:])
```

Presenta la cadena que inicia a partir del 3er caracter

la Mundo!

```
>>> print (cadena * 2)
```

Presenta dos veces la cadena

Hola Mundo!Hola Mundo!

Resultados

```
>>> print (cadena + "PUMAS")
```

Resultados

```
>>> print (cadena + "PUMAS")
```

Presenta la cadena y concatena la segunda cadena

Hola Mundo!PUMAS

Lista in `python`

Las listas es el tipo de dato más versátil de los tipos de datos compuestos de `python`.

Una lista contiene elementos separados por comas y entre corchetes `[]`.

En cierta medida, las listas son similares a los arreglos (arrays) en el lenguaje C.

Una de las diferencias entre ellos es que todos los elementos pertenecientes a una lista pueden ser de tipo de datos diferente.

Los valores almacenados en una lista se pueden acceder utilizando el operador de división `[]` y `[:]` con índices que empiezan en 0 al principio de la lista y opera hasta el final con `-1`.

El signo más `+` es el operador de concatenación de lista y el asterisco `*` es el operador de repetición.

Ejemplos con listas

```
milista = [ 'abcd', 786 , 2.23, 'salmon', 70.2 ]
```

```
listabreve = [123, 'pizza']
```

Operaciones con las listas

```
>>> print (milista)
```

Operaciones con las listas

```
>>> print (milista)
```

```
['abcd', 786, 2.23, 'salmon', 70.2]
```


Operaciones con las listas

```
>>> print (milista)
```

```
['abcd', 786, 2.23, 'salmon', 70.2]
```

```
>>> print (milista[0])
```

Operaciones con las listas

```
>>> print (milista)
```

```
['abcd', 786, 2.23, 'salmon', 70.2]
```

```
>>> print (milista[0])
```

```
abcd
```

Operaciones con las listas

```
>>> print (milista)
```

```
['abcd', 786, 2.23, 'salmon', 70.2]
```

```
>>> print (milista[0])
```

```
abcd
```

```
>>> print (milista[1:3])
```

Operaciones con las listas

```
>>> print (milista)
```

```
['abcd', 786, 2.23, 'salmon', 70.2]
```

```
>>> print (milista[0])
```

```
abcd
```

```
>>> print (milista[1:3])
```

```
[786, 2.23]
```

Operaciones con las listas 2

```
>>> print (milista[2:])
```

Operaciones con las listas 2

```
>>> print (milista[2:])
```

```
[2.23, 'salmon', 70.2]
```

Operaciones con las listas 2

```
>>> print (milista[2:])
```

```
[2.23, 'salmon', 70.2]
```

```
>>> print (listabreve * 2)
```

Operaciones con las listas 2

```
>>> print (milista[2:])
```

```
[2.23, 'salmon', 70.2]
```

```
>>> print (listabreve * 2)
```

```
[123, 'pizza', 123, 'pizza']
```


Operaciones con las listas 2

```
>>> print (milista[2:])
```

```
[2.23, 'salmon', 70.2]
```

```
>>> print (listabreve * 2)
```

```
[123, 'pizza', 123, 'pizza']
```

```
>>> print (milista + listabreve)
```

Operaciones con las listas 2

```
>>> print (milista[2:])
```

```
[2.23, 'salmon', 70.2]
```

```
>>> print (listabreve * 2)
```

```
[123, 'pizza', 123, 'pizza']
```

```
>>> print (milista + listabreve)
```

```
['abcd', 786, 2.23, 'salmon', 70.2, 123, '
```

Tuplas en python

Una tupla es otro tipo de datos de secuencia que es similar a la lista.

Una tupla consiste en un número de valores separados por comas. Sin embargo, a diferencia de las listas, las tuplas se incluyen entre paréntesis.

Diferencias entre listas y tuplas

Las principales diferencias entre las listas y las tuplas son:

- 1 Las listas están entre corchetes `[]` y sus elementos y tamaño pueden cambiarse.

Diferencias entre listas y tuplas

Las principales diferencias entre las listas y las tuplas son:

- 1 Las listas están entre corchetes [] y sus elementos y tamaño pueden cambiarse.
- 2 Las tuplas están entre paréntesis () y *no se pueden actualizar*.

Diferencias entre listas y tuplas

Las principales diferencias entre las listas y las tuplas son:

- 1 Las listas están entre corchetes [] y sus elementos y tamaño pueden cambiarse.
- 2 Las tuplas están entre paréntesis () y *no se pueden actualizar*.

Diferencias entre listas y tuplas

Las principales diferencias entre las listas y las tuplas son:

- 1 Las listas están entre corchetes [] y sus elementos y tamaño pueden cambiarse.
- 2 Las tuplas están entre paréntesis () y *no se pueden actualizar*.

Las tuplas pueden ser consideradas como listas de sólo lectura.

Ejemplos con tuplas

```
mitupla = ( 'abcd', 786 , 2.23, 'arena', 7  
tuplabreve = (123, 'playa')
```


Ejemplos con tuplas

```
mitupla = ( 'abcd', 786 , 2.23, 'arena', 7  
tuplabreve = (123, 'playa')
```

```
>>> print (mitupla)
```

Ejemplos con tuplas

```
mitupla = ( 'abcd', 786 , 2.23, 'arena', 70.2 )  
tuplabreve = (123, 'playa')
```

```
>>> print (mitupla)
```

```
('abcd', 786, 2.23, 'arena', 70.2)
```

Ejemplos con tuplas

```
mitupla = ( 'abcd', 786 , 2.23, 'arena', 70.2)  
tuplabreve = (123, 'playa')
```

```
>>> print (mitupla)
```

```
('abcd', 786, 2.23, 'arena', 70.2)
```

```
>>> print (mitupla[0])
```

Ejemplos con tuplas

```
mitupla = ( 'abcd', 786 , 2.23, 'arena', 70.2 )  
tuplabreve = (123, 'playa')
```

```
>>> print (mitupla)
```

```
('abcd', 786, 2.23, 'arena', 70.2)
```

```
>>> print (mitupla[0])
```

```
abcd
```

Ejemplos con tuplas 2

```
>>> print (mitupla[1:3])
```

Ejemplos con tuplas 2

```
>>> print (mitupla[1:3])
```

```
(2.23, 'arena', 70.2)
```

Ejemplos con tuplas 2

```
>>> print (mitupla[1:3])
```

```
(2.23, 'arena', 70.2)
```

```
>>> print (tuplabreve * 2)
```

Ejemplos con tuplas 2

```
>>> print (mitupla[1:3])
```

```
(2.23, 'arena', 70.2)
```

```
>>> print (tuplabreve * 2)
```

```
(123, 'playa', 123, 'playa')
```


Ejemplos con tuplas 2

```
>>> print (mitupla[1:3])
```

```
(2.23, 'arena', 70.2)
```

```
>>> print (tuplabreve * 2)
```

```
(123, 'playa', 123, 'playa')
```

```
>>> print (mitupla + tuplabreve)
```

Ejemplos con tuplas 2

```
>>> print (mitupla[1:3])
```

```
(2.23, 'arena', 70.2)
```

```
>>> print (tuplabreve * 2)
```

```
(123, 'playa', 123, 'playa')
```

```
>>> print (mitupla + tuplabreve)
```

```
('abcd', 786, 2.23, 'arena', 70.2, 123, 'p
```

Errores con el manejo de tuplas

El siguiente código es inválido con la tupla, porque intentamos actualizar una tupla, ya que la acción de incluir un elemento en la tupla no está permitida.

```
>>> mitupla = ( 'abcd', 786 , 2.23, 'edifi  
>>> mitupla[2] = 1000
```

TypeError Traceback (most recent call last)

```
<ipython-input-19-a99f473d7b8f> in <module>()  
    1 mitupla = ( 'abcd', 786 , 2.23, 'edificio', 70.2 )  
    2 mitupla[2] = 1000
```

TypeError: 'tuple' object does not support item assignment

Agregar elementos a la lista

Pero en la lista podemos agregar nuevos elementos que se colocan al final de la misma:

```
>>> print(milista)
```

```
[ 'abcd', 786 , 2.23, 'salmon', 70.2 ]
```

```
>>> milista.append('hola')
```

```
>>> print(milista)
```

```
[ 'abcd', 786 , 2.23, 'salmon', 70.2, 'hola'
```

Los diccionarios de python son de tipo tabla-hash.

Funcionan como arrays asociativos y consisten en pares *clave-valor*.

Elementos del diccionario

La clave de diccionario puede ser casi cualquier tipo de `python`, pero suelen ser números o cadenas.

Los valores, por otra parte, pueden ser cualquier objeto arbitrario de `python`.

Ejemplo de diccionarios

```
>>> fisicos = dict()
>>> fisicos = {
    1 : "Eistein",
    2 : "Bohr",
    3 : "Pauli",
    4 : "Schrodinger",
    5 : "Hawking"
}
```

Ejemplo de diccionarios

```
>>> print(fisicos) }  
>>> print (fisicos.keys())  
>>> print (fisicos.values())  
>>> fisicos[6] = "Planck"  
>>> print(fisicos)
```


1. Usando `python` con `linux`
2. Anaconda y `python`
3. `python` como una calculadora
4. Operadores relaciones en `python`
5. Operadores booleanos en `python`
6. Las variables en `python`

7. Tipos de Datos Estándar

8. Identificadores en `python`

8.1 Reglas para los identificadores

Reglas para los identificadores

Los identificadores son nombres que hacen referencia a los objetos que componen un programa: **constantes**, **variables**, **funciones**, etc.

Reglas para construir identificadores:

- El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)

Reglas para construir identificadores:

- El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.

Reglas para construir identificadores:

- El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.
- No pueden utilizarse espacios en blanco, ni símbolos de puntuación.

Reglas para construir identificadores:

- El primer carácter debe ser una letra o el carácter de subrayado (guión bajo)
- El primer carácter puede ir seguido de un número variable de dígitos numéricos, letras o caracteres de subrayado.
- No pueden utilizarse espacios en blanco, ni símbolos de puntuación.
- En python se distingue de las mayúsculas y minúsculas.

Palabras reservadas

No pueden utilizarse las palabras reservadas del lenguaje para ningún tipo de identificador.

| | | | | |
|-------|----------|--------|--------|--------|
| del | for | is | raise | assert |
| elif | global | else | or | yield |
| from | lamda | return | break | system |
| not | try | class | except | if |
| while | continue | exec | import | pass |
| def | finally | in | print | del |