



# Historia de Python



# Historia Python

Este lenguaje fue creado a principios de los noventa por Guido van Rossum en los Países Bajos.

En una navidad de 1989, Guido Van Rossum, quien trabajaba en el CWI (un centro de investigación holandés), decidió empezar un proyecto como pasatiempo dándole continuidad a ABC, un lenguaje de programación que se desarrolló en el CWI.

ABC fue desarrollado a principios de los 80s como alternativa a BASIC, fue pensado para principiantes por su facilidad de aprendizaje y uso. Su código era compacto pero legible.



# Historia Python

El proyecto no trascendió ya que el hardware disponible en la época hacía difícil su uso. Así que Van Rossum le dio una segunda vida creando Python.

A Guido Van Rossum le gustaba mucho el grupo Monty Python, por esta razón escogió el nombre del lenguaje. Actualmente Van Rossum sigue ejerciendo el rol central decidiendo la dirección de Python.

Uno de los proyectos más importantes escritos en Python es el servidor de Dropbox donde hoy en día trabaja Guido.



# Características Python

- Es un lenguaje interpretado, no compilado, usa tipado dinámico, fuertemente tipado.
- Es multiplataforma, lo cual es ventajoso para hacer ejecutable su código fuente entre varios sistemas operativos.
- Es un lenguaje de programación multiparadigma, el cual soporta varios paradigmas de programación como orientación a objetos, estructurada, programación imperativa y, en menor medida, programación funcional.
- El formato del código es estructural.



# Python: fuertemente tipado

Fuertemente tipado significa que el tipo de valor no cambia repentinamente. Un string que contiene solo dígitos no se convierte mágicamente en un número. Cada cambio de tipo requiere una conversión explícita. Por ejemplo:

```
# variable "valor1" es integer, variable "valor2" es string  
valor1, valor2 = 2, "5"  
# el metodo int() es para convertir a integer  
total = valor1 + int(valor2)  
# el metodo str() es para convertir a string  
print "El total es: " + str(total)
```



# Python: tipado dinámico

El tipado dinámico significa que los objetos en tiempo de ejecución (valores) tienen un tipo, a diferencia del tipado estático donde las variables tienen un tipo. A continuación un ejemplo de este concepto:

```
# "variable" guarda un valor integer  
  
variable = 11  
  
print variable, type(variable)  
  
# "variable" guarda un valor string  
  
variable = "activo"  
  
print (variable), type(variable)
```



# Python: multiplataforma

Multiplataforma hace ejecutable el código fuente entre varios sistema operativos, eso quiere decir, soporta las siguientes plataformas para su ejecución:





# Variables y constantes



# Python: declaración de variables

En Python, a diferencia de muchos otros lenguajes, no se declara el tipo de la variable al crearla.

```
>>> c = "Hola Mundo" # cadenas de caracteres
```

```
>>> type(c) # comprobar tipo de dato
```

```
<type 'str'>
```

```
>>> e = 23 # número entero
```

```
>>> type(e) # comprobar tipo de dato
```

```
<type 'int'>
```



# Python: múltiples valores a múltiples variables

Se creará múltiples variables (entero, coma flotante, cadenas de caracteres) asignando múltiples valores:

```
>>> a, b, c = 5, 3.2, "Hola"
```

```
>>> print a
```

```
5
```

```
>>> print b
```

```
3.2
```

```
>>> print c
```

```
'Hola'
```



# Python: múltiples valores a múltiples variables

Si quieres asignar el mismo valor a múltiples variables al mismo tiempo:

```
>>> x = y = z = True
```

```
>>> print x
```

```
True
```

```
>>> print y
```

```
True
```

```
>>> print z
```

```
True
```



# Python: múltiples valores a múltiples variables

Una constante es un tipo de variable la cual no puede ser cambiada.

```
IP_DB_SERVER = "127.0.0.1"
```

```
PORT_DB_SERVER = 3307
```

```
USER_DB_SERVER = "root"
```

```
PASSWORD_DB_SERVER = "123456"
```

```
DB_NAME = "nomina"
```



# Python: variable global

La sentencia `global` es una declaración que se mantiene para todo el bloque de código actual. Eso significa que los identificadores listados son interpretados como globales.

```
>>> variable1 = "variable original"

>>> def variable_global():

...     global variable1

...     variable1 = "variable global modificada"

...

>>> print variable1

variable original

>>> variable_global()

>>> print variable1

variable global modificada
```



# Operadores de asignación



# Python: Operadores de asignación

Existe en Python todo un grupo de operadores los cuales le permiten básicamente asignar un valor a una variable, usando el operador “=”. Con estos operadores pueden aplicar la técnica denominada asignación aumentada con +=

Operador =, +=, -=, \*=, /=, \*\*=, //=, %=







**GeeksHubs**  
academy \_

/ Desarrollamos { talento }

# Operadores aritméticos



# Python: Operadores aritméticos

Operador suma, resta, negación (-7), multiplicación, exponente ( $2^{**}6 = 64$ ), división, división entera ( $3.5 // 22 = 1.0$ ), módulo (devolver el resto de la división entre los dos operandos).

orden de precedencia:

Exponente: \*\*

Negación: -

Multiplicación, División, División entera, Módulo: \*, /, //, %

Suma, Resta: +, -



# Operadores relacionales



# Python: Operadores relacionales

Operador ==, !=, <, >, <= y >=

```
>>> 5 != 3
```

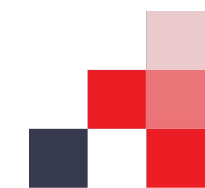
```
True
```

```
>>> "Plone" != 5
```

```
True
```

```
>>> "Plone" != False
```

```
True
```



# Tipos de números



# Python: Tipo de números

Clase	Tipo	Notas	Ejemplo
int	Números	Número entero con precisión fija.	42
long	Números	Número entero en caso de overflow.	42L ó 456966786151987643L
float	Números	Coma flotante de doble precisión.	3.1415927
complex	Números	Parte real y parte imaginaria j.	(4.5 + 3j)



# Python: Convertir a numéricos

La función **int()** devuelve un tipo de datos *número entero*.

La función **long()** devuelve un tipo de datos *número entero long*.

La función **float()** devuelve un tipo de datos *número entero float*.

La función **complex()** devuelve un tipo de datos *número complejo*.



# Cadena de caracteres





# Python: Cadenas

Son caracteres encerrado entre comillas simples (') o dobles (").

Las cadenas largas son caracteres encerrados entre grupo comillas triples simples (') o dobles ("""), están son generalmente son referenciadas como cadenas de triple comillas.



# Python: Formateo de cadenas

El carácter módulo % es un operador integrado en Python. Ese es conocido como el operador de interpolación.

El % seguido por el tipo que necesita ser formateado o convertido. El operador % entonces sustituye la frase '%tipodato' con cero o más elementos del tipo de datos especificado.

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print "el resultado de %s es %.8f" % (tipo_calculo, valor)

el resultado de raíz cuadrada de dos es 1.41421356
```



# Python: Formateo de cadenas

Con esta sintaxis hay que determinar el tipo del objeto:

`%c` = str, simple carácter.

`%s` = str, cadena de carácter.

`%d` = int, enteros.

`%f` = float, coma flotante.

`%o` = octal.

`%x` = hexadecimal.



# Tipo listas y tuplas



# Python: Listas

La lista en Python son variables que almacenan arrays, internamente cada posición puede ser un tipo de datos distinto.

- **heterogéneas:** pueden estar conformadas por elementos de distintos tipo, incluidos otras listas.
- **mutables:** sus elementos pueden modificarse.

```
>>> factura = ['pan', 'huevos', 100, 1234]
```

```
>>> factura
```

```
['pan', 'huevos', 100, 1234]
```



# Python: Listas

Los elementos de una lista pueden accederse mediante su índice, siendo 0 el índice del primer elemento.

```
>>> factura[0]
```

```
'pan'
```

```
>>> factura[3]
```

```
1234
```

La función `len()` devuelve la longitud de la lista (su cantidad de elementos).

```
>>> len(factura)
```

```
4
```



# Python: Métodos en las listas

`append()`: Este método agrega un elemento al final de una lista.

`count()`: Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista.

`extend()`: Este método extiende una lista agregando un iterable al final.

```
>>> vec = [2.1, 2.5, 3.6]
>>> vec.extend([4])
>>> print vec
[2.1, 2.5, 3.6, 4]
```

`index()`: Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la lista.

```
>>> vec = [2.1, 2.5, 3.6, 4, 5, 6, 4]
>>> print vec.index(4)
3
```



# Python: Métodos en las listas

`insert()`: Este método inserta el elemento `x` en la lista, en el índice `i`.

```
>>> vec = [2.1, 2.5, 3.6, 4, 5, 6]
```

```
>>> print vec
```

```
[2.1, 2.5, 3.6, 4, 5, 6]
```

```
>>> vec.insert(2, 3.7)
```

```
>>> print vec
```

```
[2.1, 2.5, 3.7, 3.6, 4, 5, 6]
```

`pop()`: Este método devuelve el último elemento de la lista, y lo borra de la misma.

`remove()`: Este método recibe como argumento un elemento, y borra su primera aparición en la lista.

`reverse()`: Este método invierte el orden de los elementos de una lista.

`sort()`: Este método ordena los elementos de una lista.





# Python: Tuplas

Las tuplas son objetos de tipo secuencia, específicamente es un tipo de dato lista inmutable. Esta no puede modificarse de ningún modo después de su creación.

```
>>> valores = ("Python", True, "Zope", 5)
>>> print "True ->", valores.count(True)
True -> 1
>>> print "'Zope' ->", valores.count('Zope')
'Zope' -> 1
>>> print "5 ->", valores.count(5)
5 -> 1
```



# Tipo diccionarios



# Python: Diccionesarios

El diccionario, define una relación uno a uno entre claves y valores.

Un objeto mapping mapea valores hashable a objetos arbitrariamente. Los objetos mapeos son objetos mutable. El diccionario es el único tipo de mapeo estándar actual.

Los diccionarios pueden ser creados colocando una lista separada por coma de pares “key:value” entre {}, por ejemplo: “{'python': 27, 'plone': 51}” o “{27:'python', 51:'plone'}”, o por el constructor “dict()”.



# Estructuras de control



# Python: if

```
if numero < 0:
    numero = 0
    print 'El número ingresado es negativo cambiado a cero.\n'
elif numero == 0:
    print 'El número ingresado es 0.\n'
elif numero == 1:
    print 'El número ingresado es 1.\n'
else:
    print 'El número ingresado es mayor que uno.\n'
```



# Python: if

```
a, b = 10, 20
```

```
if (a and b):
```

```
    print "Las variables 'a' y 'b' son VERDADERO."
```

```
else:
```

```
    print "O bien la variable 'a' no es VERDADERO " + \
```

```
    "o la variable 'b' no es VERDADERO."
```



# Python: while

```
suma, numero = 0, 1
```

```
while numero <= 10:
```

```
    suma = numero + suma
```

```
    numero = numero + 1
```

```
print "La suma es " + str(suma)
```



# Python: iter

Un iterador es un objeto adherido al iterator protocol, básicamente esto significa que tiene una función `next()`, es decir, cuando se le llama, devuelve la siguiente elemento en la secuencia, cuando no queda nada para ser devuelto, lanza la excepción `StopIteration`.

```
>>> frase = 'HO'

>>> letra = iter(frase)

>>> letra.next()

'H'

>>> letra.next()

'O'

>>> letra.next()

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

StopIteration
```





# Funciones



# Python: funciones

La sentencia **def** es una definición de función usada para crear objetos funciones definidas por el usuario.

Una definición de función es una sentencia ejecutable. Su ejecución enlaza el nombre de la función en el namespace local actual a un objeto función (un envoltorio alrededor del código ejecutable para la función).

```
def NOMBRE (LISTA_DE_PARAMETROS) :
```

```
    """DOCSTRING_DE_FUNCION"""
```

```
    SENTENCIAS
```

```
    RETURN [EXPRESION]
```



# Python: funciones

```
>>> def hola(arg):  
...     """El docstring de la función"""  
...     print "Hola", arg, "!"  
...  
  
>>> hola("Plone")  
  
Hola Plone !
```



# Python: funciones

```
>>> def resta(a, b):  
...     return a - b  
...  
  
>>> (b=30, a=10)  
  
-20
```



# Python: funciones recursivas

Las funciones recursivas son funciones que se llaman a sí mismas durante su propia ejecución.

```
>>> def cuenta_regresiva(numero):  
...     numero -= 1  
...     if numero > 0:  
...         print numero  
...         cuenta_regresiva(numero)  
...     else:  
...         print "Booooooooooom!"  
...     print "Fin de la función", numero  
...  
>>> cuenta_regresiva(5)
```

## RESULTADO:

```
4  
3  
2  
1  
Booooooooooom!  
Fin de la función 0  
Fin de la función 1  
Fin de la función 2  
Fin de la función 3  
Fin de la función 4
```



**CEO - Chaume Sánchez**  
**@sansanal**

[www.geekshubsacademy.com](http://www.geekshubsacademy.com)  
@geekshubs

