



Amazing Python: Guía rápida para principiantes

Vale, este artículo ha sido escrito por una [fan de Python](#) y además [Pylady](#) pero, de verdad que Python mola.



Guía rápida

Python

Presentación

- El Zen de Python

Tipos de datos con súperpoderes

- Número
- String
- Boolean
- Iterables
- Lista
- Tupla
- Rango
- Set & Frozenset
- Dictionary
- Tabla Hash

Variables

Conditionals

Loops

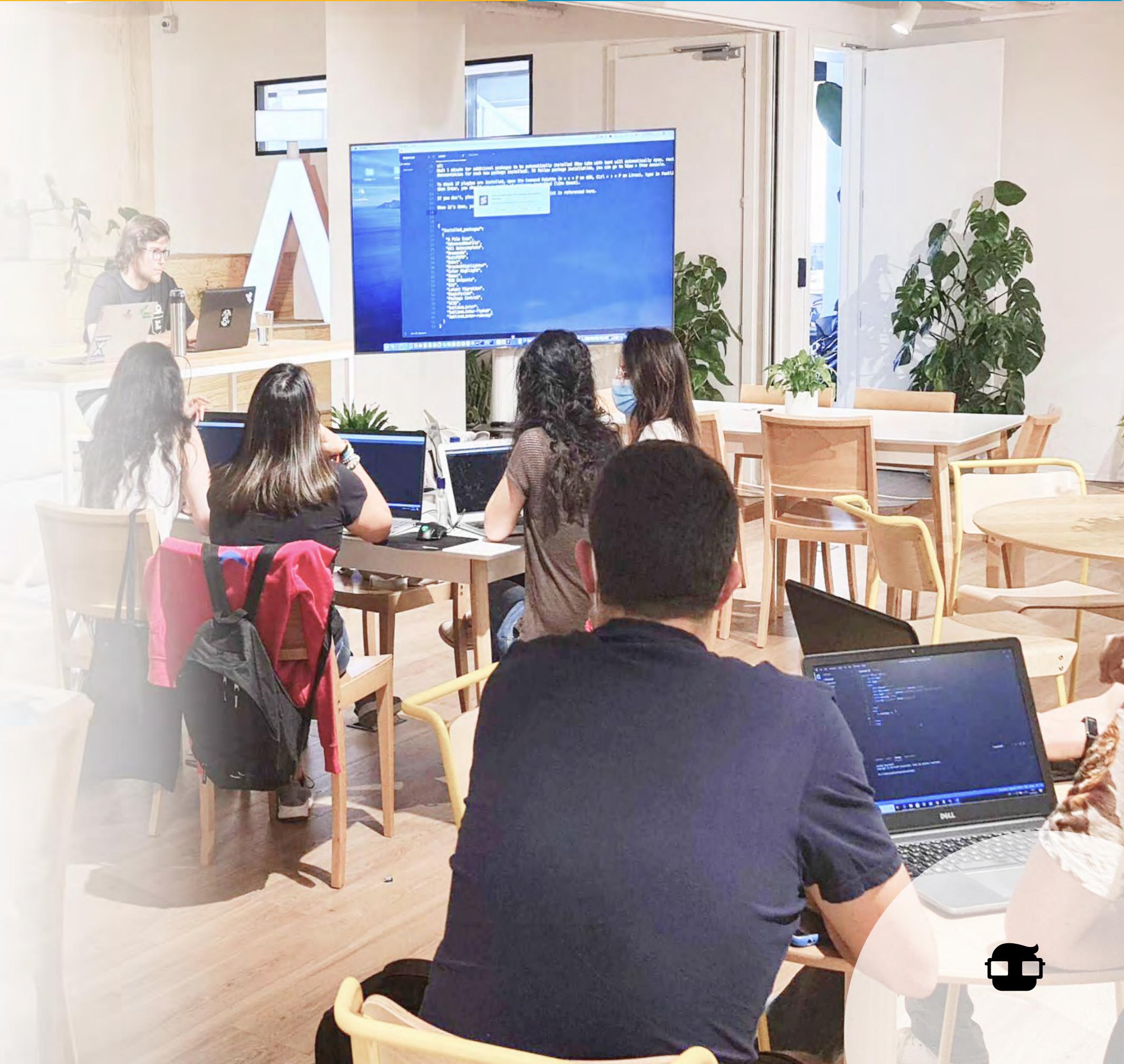
Functions

Decorators

Generators

Virtual environment

Package manager



Presentación

Empecemos con un poco de humor. Escribe en la terminal:

```
$ python  
>>> import antigravity
```

Ahora me entiendes? Probemos otra cosa más:

```
>>> from __future__ import braces
```

¡Eso es, no hay llaves ({})! Ni tampoco puntos y coma (;), lo único que **importa es tabular**.

Ahora ya empiezas a entender que Python mola ¿verdad?

Si quieres aprender desde cero, conoce nuestro programa full-stack

[Descargar programa](#)



El Zen de Python

Bonito es mejor que feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.
Complejo es mejor que complicado.
Plano es mejor que anidado.
Disperso es mejor que denso.
La legibilidad cuenta.
Los casos especiales no son tan especiales como para romper las reglas.
Aunque la practicidad vence a la pureza.
Los errores nunca deberían pasarse por alto.
A menos que esté explícitamente pasado por alto.
En caso de ambigüedad, rechaza la tentación de adivinar.
Tendría que haber un — y preferiblemente únicamente uno — camino obvio para hacerlo.
Aunque ese camino puede no ser obvio la primera vez, a menos que seas holandés.
Ahora es mejor que nunca.
Aunque nunca es, en algunos casos, mejor que ahora mismo.
Si la implementación es difícil de explicar, es una mala idea.
Si la implementación es fácil de explicar, puede que sea una buena idea.
Los namespaces (espacio de nombres) son una gran idea — ¡hagamos más de estos!

Empecemos con un poco de humor. Escribe en la terminal:

```
$ python  
>>> import this
```

Para cumplir con las normas del Zen y hacer de nuestro código lo más pythonico posible siempre podemos recurrir al PEP-8 dónde se reúnen todas las buenas prácticas a la hora de escribir Python.



Tipos de datos con súperpoderes

Número

Tenemos tres tipos numéricos distintos: enteros, números en coma flotante (floats) y números complejos.

```
integer_number = 2
float_number = 2.3456
complex_number = complex (2, 3j)
```

Además de los tipos numéricos básicos tenemos un par de opciones más.

Decimal

```
from decimal import Decimal
decimal_number = Decimal ('7.01') # Decimals use strings for initialization
```

La diferencia entre cuándo usar decimal y cuándo *float* dependerá de si lo que importa es la velocidad o la precisión. Usaremos *float* cuándo importe la velocidad o cuándo la precisión no sea necesaria, por ejemplo, al calcular la media de las reviews; en cambio usaremos decimales cuándo necesitemos precisión cómo, por ejemplo, en las operaciones bancarias.



Consejo: Si vas a trabajar con PostgreSQL usaremos siempre decimales.

Fíjate en este ejemplo:

```
from decimal import Decimal

# Difference between decimal and float
sum_float = .1 + .1 + .1
sum_decimal = Decimal('0.1') + Decimal('0.1') + Decimal('0.1')

print(sum_float) # 0.3000000000000004
print(sum_decimal) # Decimal('0.3')
```

¿Quieres saber más detalles sobre condicionales y su utilidad?

Descarga el programa de estudios de full-stack

[Descargar programa](#)





Fracciones

En el colegio aprendimos que hay dos formas de escribir el resultado de una división, podemos poner 0.8 o 4/5. Python tiene el tipo *fraction* para cuando trabajamos con fracciones y la precisión importa.

```
from fractions import Fraction  
  
my_fraction = Fraction(16, -10)  
my_other_fraction = Fraction ('3/7') # In this way we use a string  
  
print (my_fraction) # Fraction(-8, 5)  
# Fraction(-8, 5) is the same that Fraction(16, -10), but simplifier
```

Hablando de precisión, fíjate en esta curiosidad:

```
from decimal import Decimal  
from fractions import Fraction  
  
decimal_operation = Decimal ('1') / Decimal ('3') * Decimal ('3')  
fraction_operation = Fraction ('1') / Fraction ('3') * Fraction ('3')  
  
print (decimal_operation) # Decimal ('0.99999999999999999999999999999999')  
print (fraction_operation) # Fraction(1, 1)
```



String

Los Strings pueden ser texto o sólo una letra o carácter.

```
string_empty = ""  
string_character = ''  
string_double_quoted = "Hello world"  
string_single_quoted = 'Hello world'  
  
# We don't have multiline comments, but you can use triple quoted for that  
string_triple_quoted = """ Hello world """
```

En Python los Strings deberían ser siempre inmutables, es decir, que no cambien de valor. Por eso para unir dos usamos join que nos genera un nuevo String a partir de los datos.

```
strings = ['do', 're', 'mi']  
.join(strings)  
'do,re,mi'
```

Para generar un String con valores variables usamos format (f).

```
my_var = "
```



Boolean

Los Booleanos son en verdad números, pero tenemos True y False para representarlos.

```
my_true = True #or 1  
my_false = False #or 0
```

Además, existen valores verdaderos (*truthy*) o falsos (*falsy*).

Todos los elementos de esta lista devuelven False.

- `None` y `False`.
- Cero de cualquier tipo numérico: `0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`
- Secuencias y colecciones vacías: `"`, `()`, `[]`, `{}`, `set()`, `range(0)`

Cualquier otro valor devuelve True.



Iterables

En Python no decimos "Array", decimos Lista (List), o Tupla (Tuple) o Rango (Range), y creo que es hermoso.

Los iterables son aquellos objetos que podemos iterar gracias a sus índices. A parte de las listas, tuplas y rangos también podemos iterar los strings.

Además de recorrer estos elementos en bucles usando sus índices podemos hacer otras operaciones como estas:

```
my_iterable = ['a', 'b', 'c']
```

```
'a' in my_iterable # True  
'a' not in my_iterable #False
```

Lista

Las listas se usan para almacenar una secuencia (o lista) de elementos. Puedes almacenar lo que quieras: strings, números, otras listas, etc. Incluso puedes mezclar elementos de diferentes tipos pero, normalmente, vas a necesitar listas homogéneas.

```
list_empty = []  
list_of_strings = ['potatoe', 'hamburguer', 'tomatoe', 'bread']  
list_of_lists = [['hi', 'bye'], [1, 2, [3]], [True, False, 1, 0, None]]
```



Las listas siempre tienen índice y longitud y esto es lo que nos permite trabajar con ellas.

- **Índice:** Empieza en 0 e indica la posición de un elemento en la lista.
- **Longitud :** Total de elementos que contiene la lista.

```
# Length  
print (len (list_empty)) #0  
print (len (list_of_strings)) #4
```

```
# Index  
print (list_of_strings[0]) #potatoe  
print (list_of_strings[1]) #hamburguer  
print (list_of_strings[2]) #tomatoe  
print (list_of_strings[3]) #bread  
print (list_of_strings[4]) #ERROR! Never use the length value
```

```
# With a list of lists  
print (len (list_of_lists)) #3  
print (list_of_lists[0]) #['hi', 'bye']  
print (list_of_lists[1]) #[1, 2, [3]]  
print (list_of_lists[2]) #[True, False, 1, 0, None]
```

```
# Going inside a list of a list  
print (list_of_lists[1][0]) #1  
print (list_of_lists[1][1]) #2  
print (list_of_lists[1][2]) #[3]  
print (list_of_lists[1][2][0]) # 3
```

Otra cosa guay de las listas es que podemos jugar de forma muy fácil con las sublistas usando [posición:posición].

```
list_of_numbers = [1,2,3,4,5]  
  
# From first to last  
print (list_of_numbers[0:]) #[1,2,3,4,5]  
  
# From first to penultimate  
print (list_of_numbers[:-1]) #[1,2,3,4]  
  
# Last element  
print (list_of_numbers[-1:]) #[5]  
  
# Without first and last  
print (list_of_numbers[1:-1]) #[2,3,4]  
  
# Without 3 last elements  
print (list_of_numbers[:-3]) #[1,2]
```

And beyond, just play with it!



Tupla

Las tuplas son colecciones de objetos, normalmente heterogéneos, que no se pueden modificar, es decir, son inmutables.

```
my_empty_tuple = ()  
my_tuple = (' a ', ' b ', ' c ')  
my_other_tuple = tuple (' abc ')
```

Tienen índices y longitud, así que podemos usarlas igual que las listas.

Rango

Los rangos son colecciones inmutables de números y normalmente se usan en bucles para iterar una serie de veces o de una forma específica.

Tenemos siempre que indicar dónde acaban y opcionalmente dónde empiezan y cómo avanzan.

Cómo lo que nos devuelven suele ser un objeto tipo range(0,10), los solemos convertir en listas a la hora de imprimirlos por pantalla para poder leerlos mejor y ¡Ojo! Que aunque sean iterables no tienen longitud.

```
my_empty_range = list (range (0)) #[]  
my_other_range = list (range (-1, -10, 3)) #[0, -3, -6, -9]
```



Set & Frozenset

Un conjunto (Set) es una colección no ordenada de objetos distintos, más o menos es como una lista pero sin elementos repetidos y sin índices. No ordenada significa sin índices.

```
my_set = set ('abc')
my_other_set = {1,2,3,'a','b'}
```

```
''' If we try to enter a duplicated object,
we will not get any error,
but the object will only appear once '''
```

```
my_set = {1,4,4,4}
print (my_set) # {1,4}
```

Frozenset es un conjunto inmutable, no tiene métodos como add() o remove().

```
my_frozenset = frozenset ({1,2,3,'a','b'})
```

Dictionary

```
my_dict = {}
```



Tabla Hash

Las tablas hash son un tipo de diccionario cuyas claves son únicas y tienen un valor semántico.

Fíjate en este ejemplo, es muy fácil crear una.

```
my_dictionaries_list = [
    {
        'name': "Wonder Woman",
        'greeting': "You're the coolest member of Justice League",
    },
    {
        'name': "Batman",
        'greeting': "You're just a boy with money",
    },
]

my_hash_table = {
    "Wonder Woman": "You're the coolest member of Justice League",
    "Batman": "You're just a boy with money",
}
```

En este punto puede que quieras que te diga las funciones que tiene cada tipo, pero la mejor forma de saberlo es con la [Documentación de Python]. (<http://ocs.python.org/3/library/stdtypes.html>).

Vale, soy una buena persona así que te daré un super consejo:

Tip: Para saber el tipo de variable, `print type(mi_variable)`, y para saber todas las funciones que puedes usar con una variable `print dir(mi_variable)`.



Variables

```
# No more var, let, or const  
my_first_variable = 2  
my_second_variable = "This is very cool"  
  
# Swap  
my_first_variable, my_second_variable = my_second_variable, my_first_variable
```

No tenemos constantes, tenemos una convención. Si escribes el nombre de una variable en mayúsculas, consideramos que es una constante, pero ¡ten cuidado! Puedes cambiar el valor de esa variable porque no es de tipo constante.



Conditionals

Existen if <condición>, elif <condición> y else.

Recuerda que if puede funcionar sólo, pero elif y else necesitan antes un if.

```
my_variable = "Wonder Woman"

if my_variable == "Wonder Woman":
    print ("You're the coolest member of Justice League")
elif my_variable == "Batman":
    print("You're just a boy with money")
else:
    print ("Hi regular human")

# We also have ternary conditionals
message = "You're cool" if my_variable == "Wonder Woman" else "Hi regular human"
```

No existe el switch de otros lenguajes, en caso de echarlo de menos no lo sustituyas por un montón de elif y utiliza una tabla hash.

```
my_variable = "Wonder Woman"
my_hash_table = {
    "Wonder Woman": "You're the coolest member of Justice League",
    "Batman": "You're just a boy with money",
    "Superman": "Hi alien",
}

print (my_hash_table.get(my_variable, "Hi regular human"))
```



Loops

```
# With Lists
for item in list_of_items:
    pass
for index, item in enumerate(list_of_items):
    pass

# With Dictionaries
for key, value in dictionary.items():
    pass

# Try this loop with a range
BATCH_SIZE = 3
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for x in range(0, len(numbers), BATCH_SIZE):
    print(numbers[x:x+BATCH_SIZE])
```

Functions

```
def my_function():
    pass

def my_function(param):
    pass

def my_function(*argv):
    pass
```

Los Loops a veces pueden dar dolores de cabeza, aprende en 4Geeks cómo utilizarlos. [Descarga el programa de estudios de full-stack](#)

[Descargar programa](#)





Impulsa tu carrera ¡aprende a programar! Únete a 4Geeks Academy

Si quieres aprender con acompañamiento de por vida, habla con el equipo de Admisiones sobre el bootcamp de 4Geeks Academy

[HABLAR CON ADMISSIONES](#)

