Curso

Profesional de Python

Facundo García Martoni

@facmartoni


pytho

Curso Básico de Python

★★★★★ 5125 Opiniones | BÁSICO

Inicia en el mundo de la programación con el lenguaje de mayor crecimiento en el planeta: Python. Descubre qué es un algoritmo, y cómo se construye uno. Domina las variables, funciones, estructuras de datos, los condicionales y ciclos.

- Hacer estructuras de datos
- Crear bucles
- Conocer herramientas para programar
- Aprender conceptos básicos de Python

CONTINUAR APRENDIENDO

AGREGAR A MI RUTA

Facundo García Martoni
Technical Mentor en Platzi

Curso Profesional de Git y GitHub

★★★★★ 9589 Opiniones | BÁSICO

Deja de versionar tus proyectos usando tu propio sistema de control de versiones. Mejor usa Git, el sistema de control de versiones por excelencia que utiliza la industria tecnológica. Aprende a trabajar con git, conceptos básicos, clonar un repositorio y gestionar tus proyectos alojándolos en tu repositorio local y en GitHub.

- Llevar un Control de Versiones en tus Proyectos con Git
- Trabajar en Equipos de Forma Colaborativa
- Utilizar Dominios Personalizados con GitHub Pages
- Instalar Git en tu sistema operativo

CONTINUAR APRENDIENDO

AGREGAR A MI RUTA

Curso de Python Intermedio

★★★★★ 1206 Opiniones | INTERMEDIO

Fortalece tus habilidades para profesionalizarte con Python, uno de los lenguajes más utilizados en el mundo en desarrollo backend, ciencia de datos e inteligencia artificial. Aprende conceptos y practica con retos que elevarán tu nivel al programar.

- Utiliza list y dictionary comprehensions, y high order functions.
- Preparar un entorno virtual con pip.
- Maneja archivos de texto con el context manager.
- Aprende qué significan los errores y cómo manejarlos.

CONTINUAR APRENDIENDO

AGREGAR A MI RUTA

Facundo García Martoni
Technical Mentor en Platzi

Curso de Programación Orientada a Objetos: POO

★★★★★ 5480 Opiniones | BÁSICO

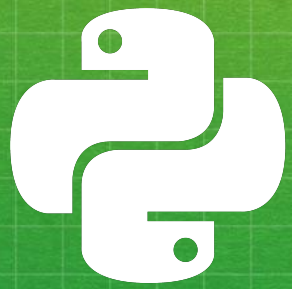
La misión: lograr que la virtualidad sea idéntica a la realidad. El elegido para esta tarea: tú. Sé parte de la generación que logra cambiar el mundo a través de ideas innovadoras, porque la diferencia entre ser testigo del cambio y ser parte de él comienza con Platzi.

- Conocer los diferentes lenguajes de programación orientados a objetos
- Analizar problemas para el desarrollo de proyectos orientado a objetos
- Entender el funcionamiento de los objetos
- Entender el funcionamiento de las clases

CONTINUAR APRENDIENDO

AGREGAR A MI RUTA

Anahí Salgado Díaz de la Vega
Android, Firebase, Java, Geek & Teacher en Platzi

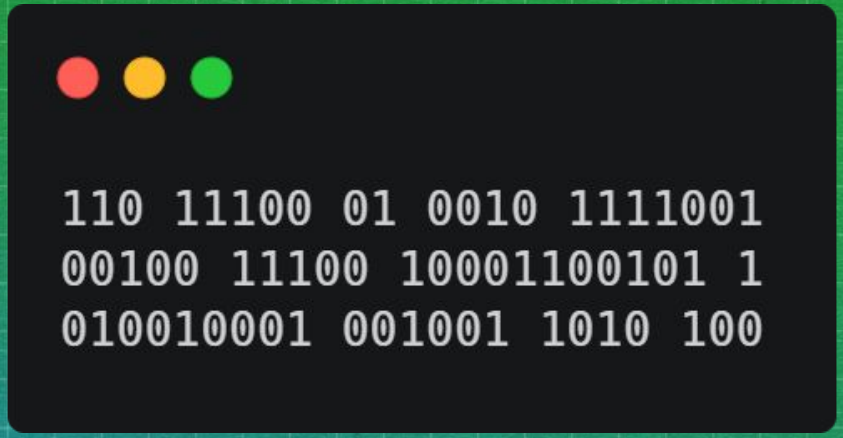


¿Cómo funciona Python?

Desde que escribes una línea,
hasta que el programa se ejecuta

Compilado
vs.
Interpretado

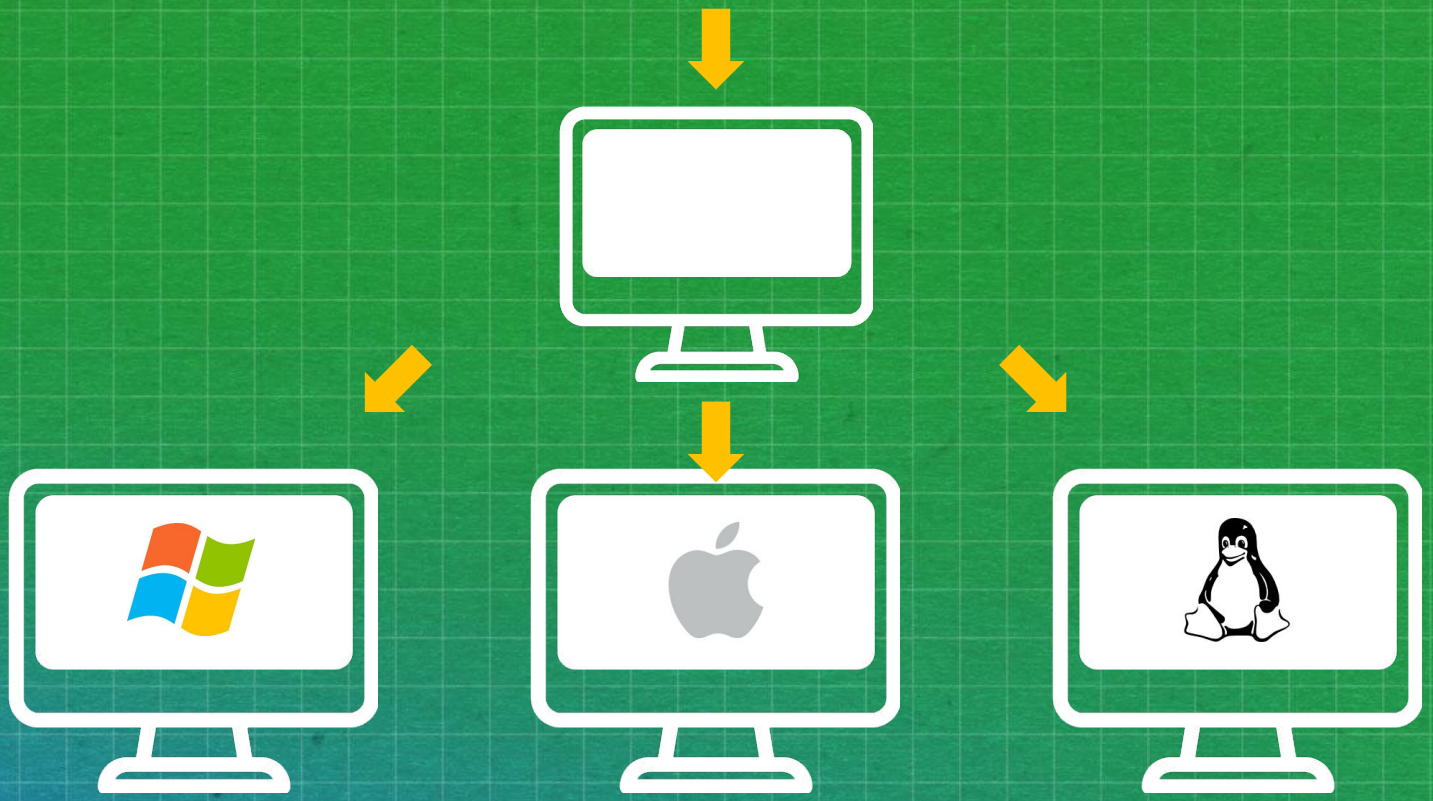

void main() {
cout<<"Hola Mundo";
}


110 11100 01 0010 1111001
00100 11100 10001100101 1
0100010001 001001 1010 100


C++
Computadora
Compilación

```
 1           0 LOAD_GLOBAL              0 (print)
             2 LOAD_CONST               1 ('Hola Mundo')
             4 CALL_FUNCTION            1
             6 POP_TOP
             8 LOAD_CONST               0 (None)
            10 RETURN_VALUE
```

```
 1           0 LOAD_GLOBAL              0 (print)
             2 LOAD_CONST               1 ('Hola Mundo')
             4 CALL_FUNCTION            1
             6 POP_TOP
             8 LOAD_CONST               0 (None)
            10 RETURN_VALUE
```



Python

Computadora

Máquina Virtual

Preguntas frecuentes

¿Los lenguajes interpretados son más lentos?

¿Qué es el “Garbage collector”?

¿Qué es la carpeta __pycache__?

Organiza los archivos de tus proyectos

Paquetes y módulos

Módulo

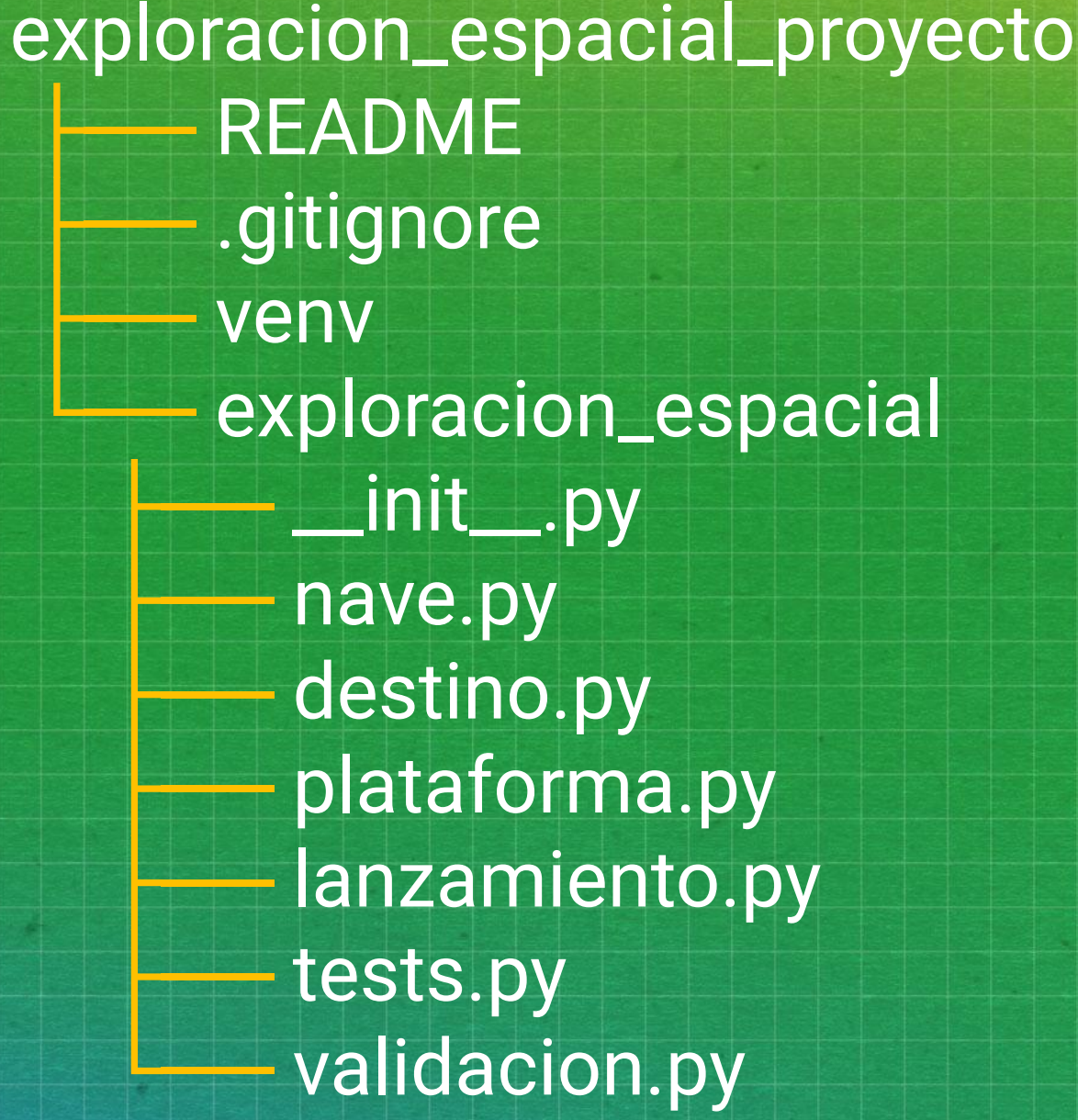
Un módulo es
cualquier archivo de Python.
Generalmente, contiene código
que puedes reutilizar.

Paquete

Un paquete es
una carpeta que contiene módulos.
Siempre posee el archivo __init__.py.

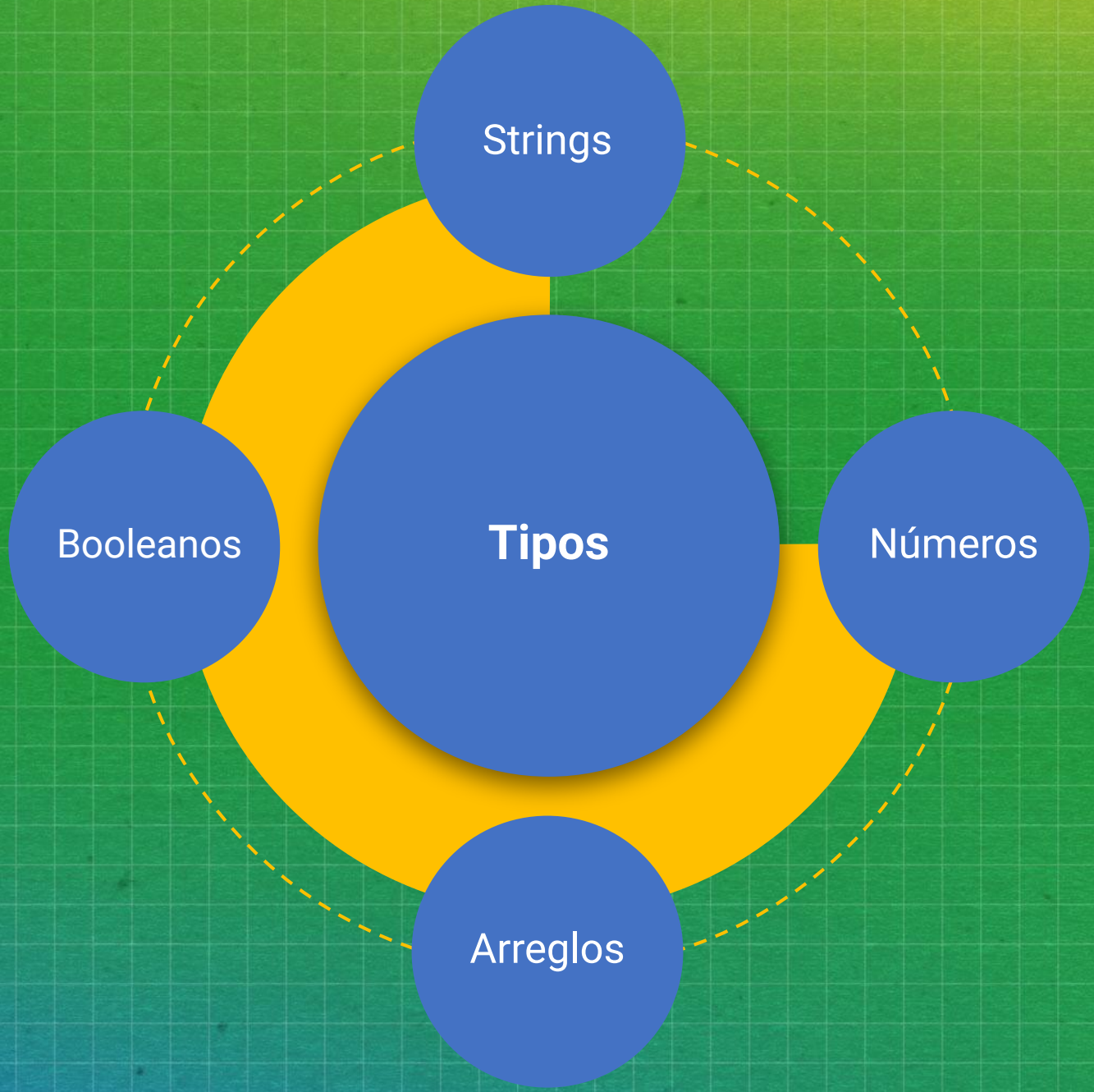

Paquete
Módulo
Módulo
Módulo
Módulo
Módulo
Módulo

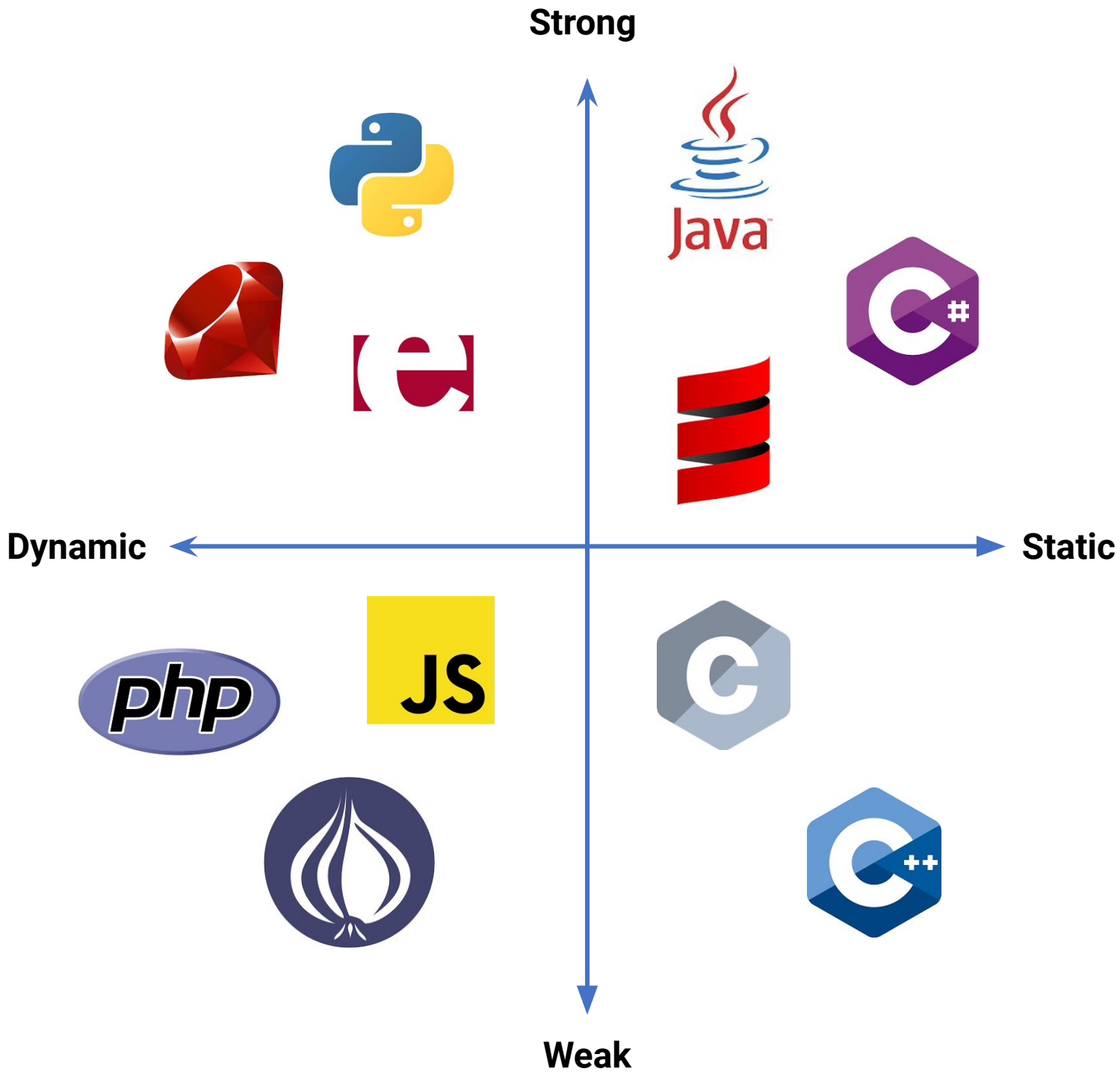

exploracion_espacial
nave.py
destino.py
plataforma.py
lanzamiento.py
tests.py
validacion.py


exploracion_espacial_proyecto
README
.gitignore
venv
exploracion_espacial
__init__.py
nave.py
destino.py
plataforma.py
lanzamiento.py
tests.py
validacion.py

¿Qué son los "tipados"?

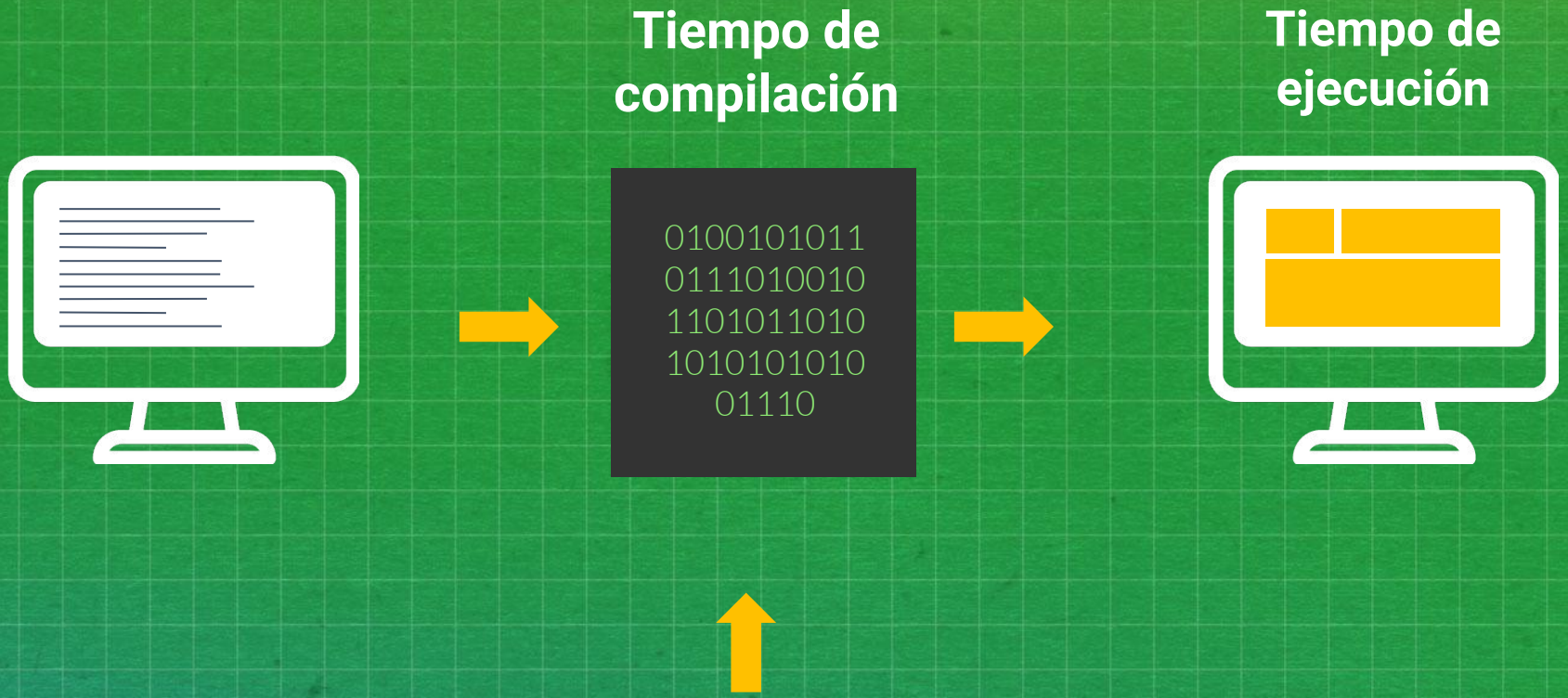
Estático, dinámico, débil, y fuerte


Strings
Booleanos
Tipos
Números
Arreglos


Strong
Java
Dynamic
Static
JS
php
Weak


Tiempo de compilación
Tiempo de ejecución
0100101011
0111010010
1101011010
1010101010
01110

Static


Tiempo de compilación
Tiempo de ejecución
0100101011
0111010010
1101011010
1010101010
01110
Static

```
function foo(a: number) {
    if (typeof(a) === 'number') {
        return 'number';
    } else {
        return 'not number';
    }
}
foo(1);
foo('1'); // this will throw a compiler error, because it is not a number
```

Code by: https://medium.com/@cpave3/understanding-types-static-vs-dynamic-strong-vs-weak-88a4e1f0ed5f

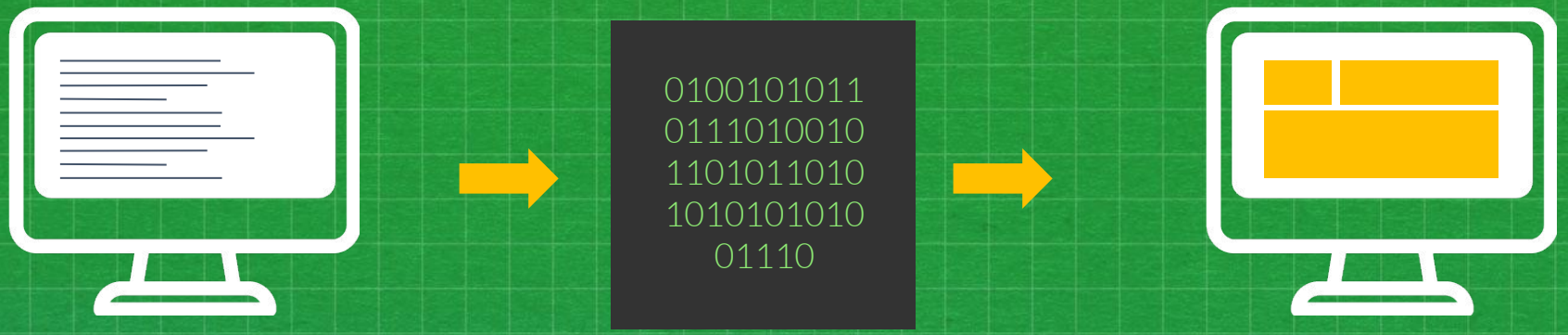
```java
String str = "Hello";  //statically typed as string
str = 5;               //would throw an error since java is statically typed
```

Code by: https://stackoverflow.com/questions/2351190/static-dynamic-vs-strong-weak

Dynamic

Tiempo de compilación
(en Python, de pasaje a bytecode)

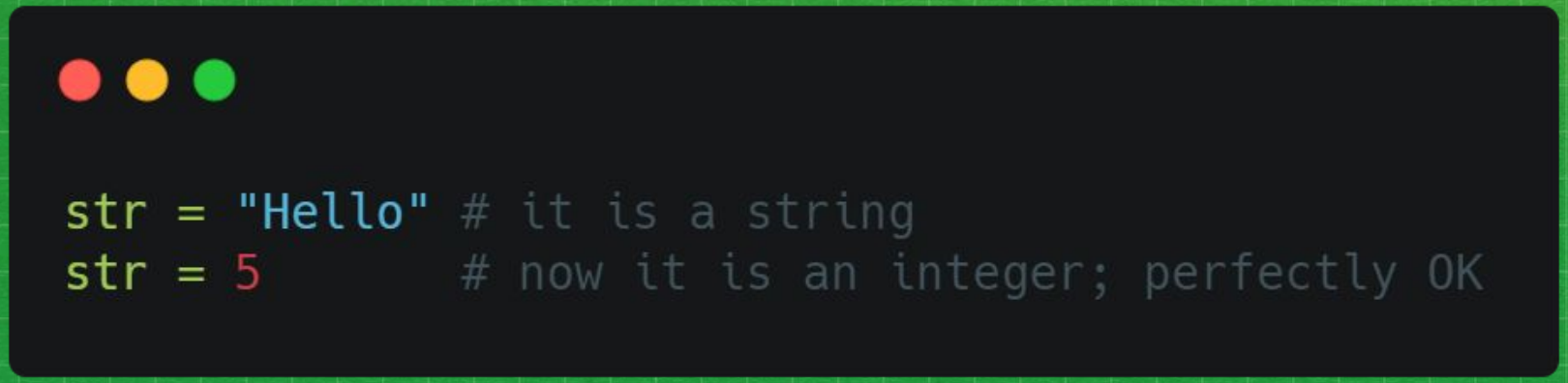
Tiempo de ejecución



Dynamic

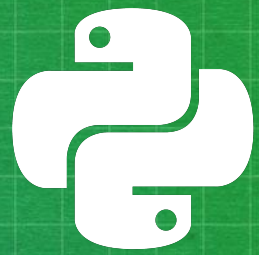
```php
function foo($a) {
    if (gettype($a) === 'integer') {
        return 'integer';
    } else {
        // This will never be evaluated
        return 'not integer';
    }
}
echo foo(1); // integer
```

Code by: https://medium.com/@cpave3/understanding-types-static-vs-dynamic-strong-vs-weak-88a4e1f0ed5f

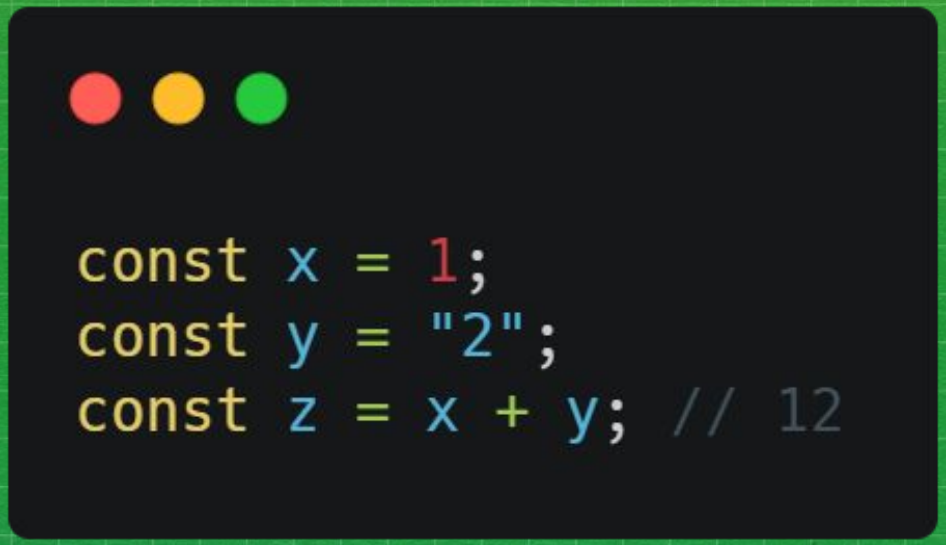


Code by: https://stackoverflow.com/questions/2351190/static-dynamic-vs-strong-weak

¿Strong? ¿Weak?



Code by: https://medium.com/@cpave3/understanding-types-static-vs-dynamic-strong-vs-weak-88a4e1f0ed5f



```
const x = 1;
const y = "2";
const z = x + y; // 12
```

JS

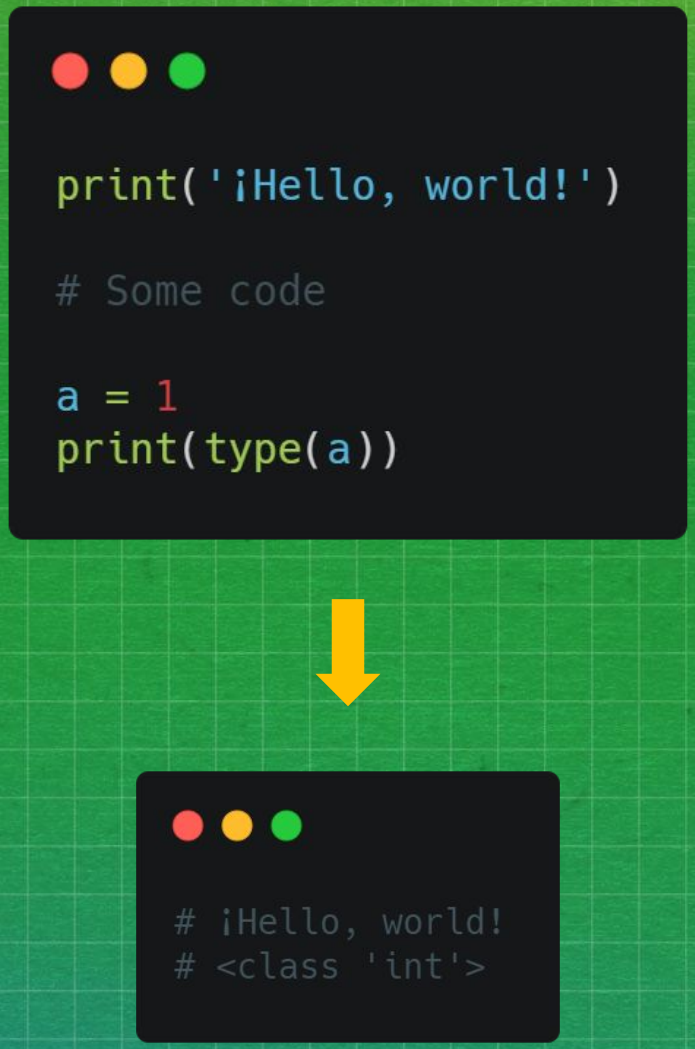
Code by: https://medium.com/@cpave3/understanding-types-static-vs-dynamic-strong-vs-weak-88a4e1f0ed5f

```php
<?php
$str = 5 + "5"; // equals 10 because "5" is implicity casted to 5 as an integer
// PHP is weakly typed, thus is a very forgiving language.
```

Code by: https://medium.com/@cpave3/understanding-types-static-vs-dynamic-strong-vs-weak-88a4e1f0ed5f

Tipado estático en Python

Y cómo implementarlo

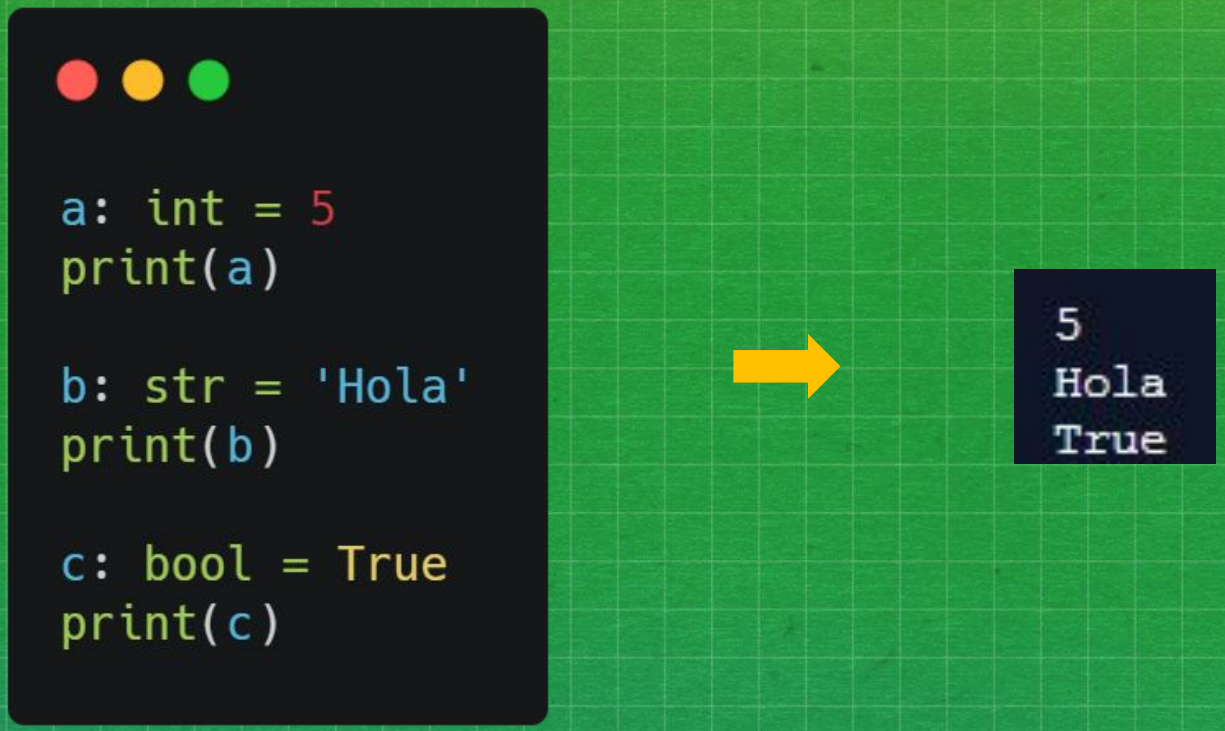

print('¡Hello, world!')
# Some code
a = 1
print(type(a))
# ¡Hello, world!
# <class 'int'>

```python
FALLBACK_PHONE = '+e00000000'

def get_phone():
  phone = input('Give me your phone: ')
  if not phone:
    return FALLBACK_PHONE.round()
  return int(phone)


def run():
  my_phone = get_phone()
  print(f'Your phone is: {my_phone}')


run()
```

```
Give me your phone: 190823901
Your phone is: 190823901
```

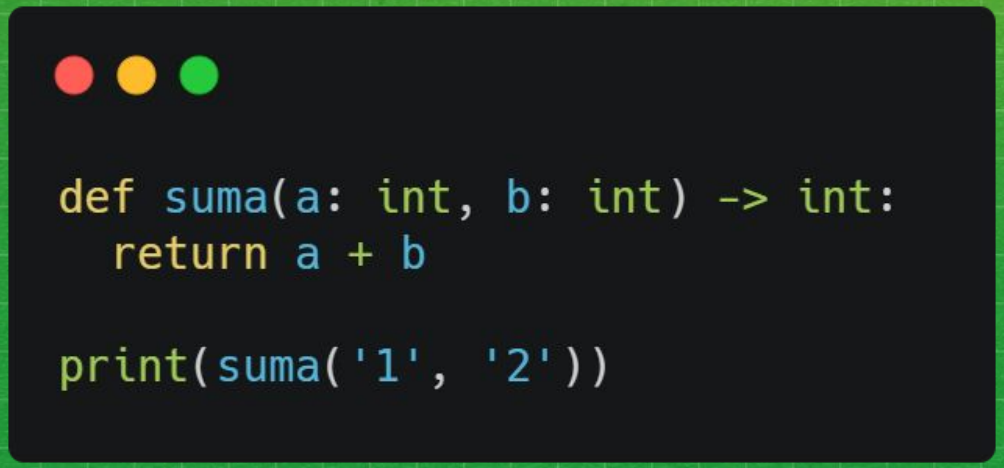
```
Give me your phone:
Traceback (most recent call last):
  File "main.py", line 15, in <module>
    run()
  File "main.py", line 11, in run
    my_phone = get_phone()
  File "main.py", line 6, in get_phone
    return FALLBACK_PHONE.round()
AttributeError: 'str' object has no attribute 'round'
```

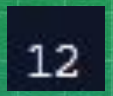
Static Typing


a: int = 5
print(a)
b: str = 'Hola'
print(b)
c: bool = True
print(c)
5
Hola
True

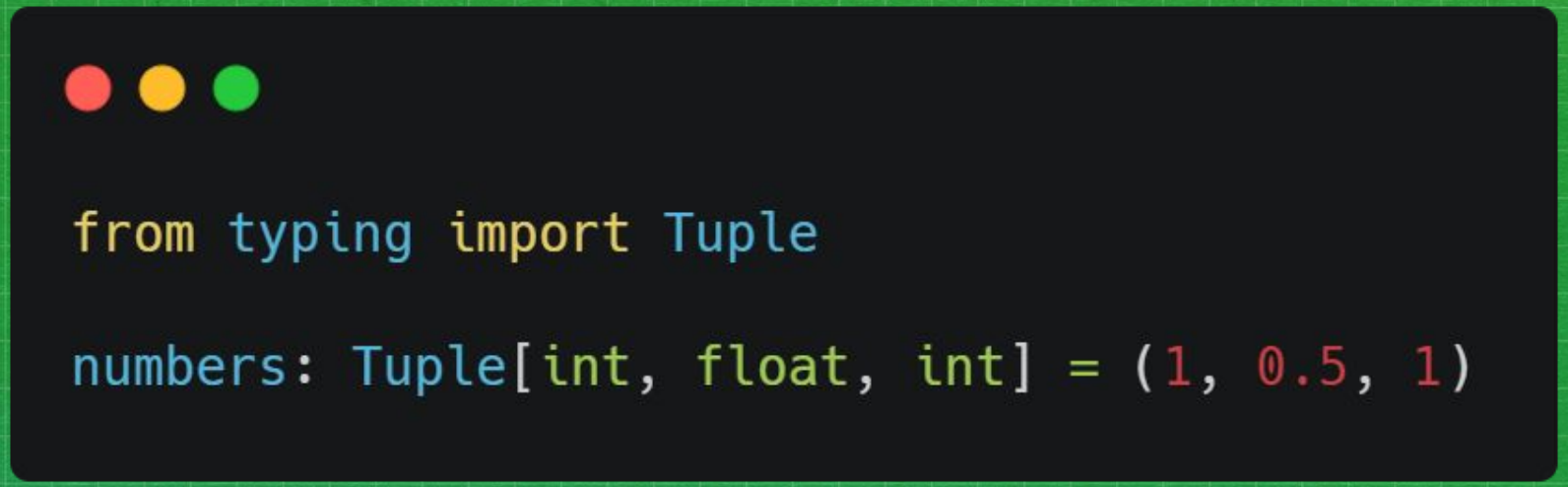
```python
def suma(a: int, b: int) -> int:
  return a + b

print(suma(1, 2))
```

```
3
```


def suma(a: int, b: int) -> int:
return a + b
print(suma('1', '2'))


12

```python
from typing import Dict, List

positives: List[int] = [1, 2, 3, 4, 5]

users: Dict[str, int] = {
  'argentina': 1,
  'mexico': 34,
  'colombia': 45,
}

countries: List[Dict[str, str]] = [
  {
    'name': 'Argentina',
    'people': '450000',
  },
  {
    'name': 'México',
    'people': '9000000',
  },
  {
    'name': 'Colombia',
    'people': '999999999999',
  },
]
```


from typing import Tuple
numbers: Tuple[int, float, int] = (1, 0.5, 1)

```python
from typing import Tuple, Dict, List

CoordinatesType = List[Dict[str, Tuple[int, int]]]

coordinates: CoordinatesType = [
  {
    'coord1': (1, 2),
    'coord2': (3, 5)
  },
  {
    'coord1': (0, 1),
    'coord2': (2, 5)
  },
]
```

mypy

¿Ventajas?

Practicando el tipado estático

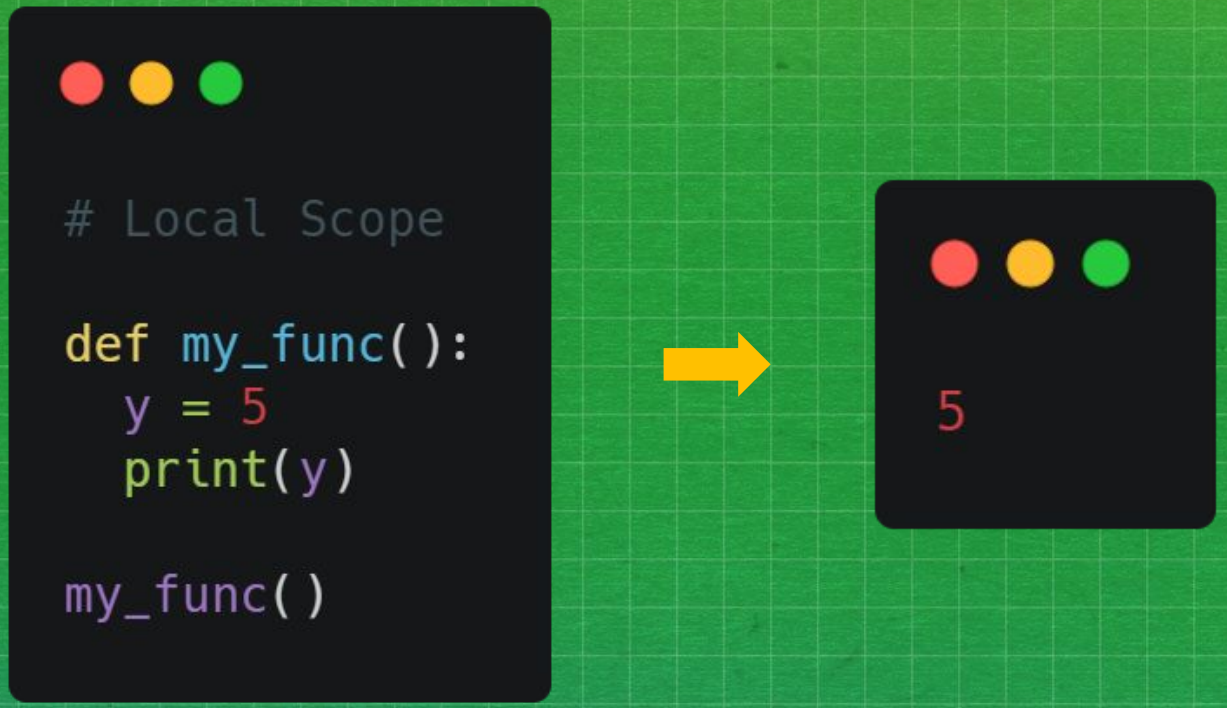
¿Listo para escribir código?

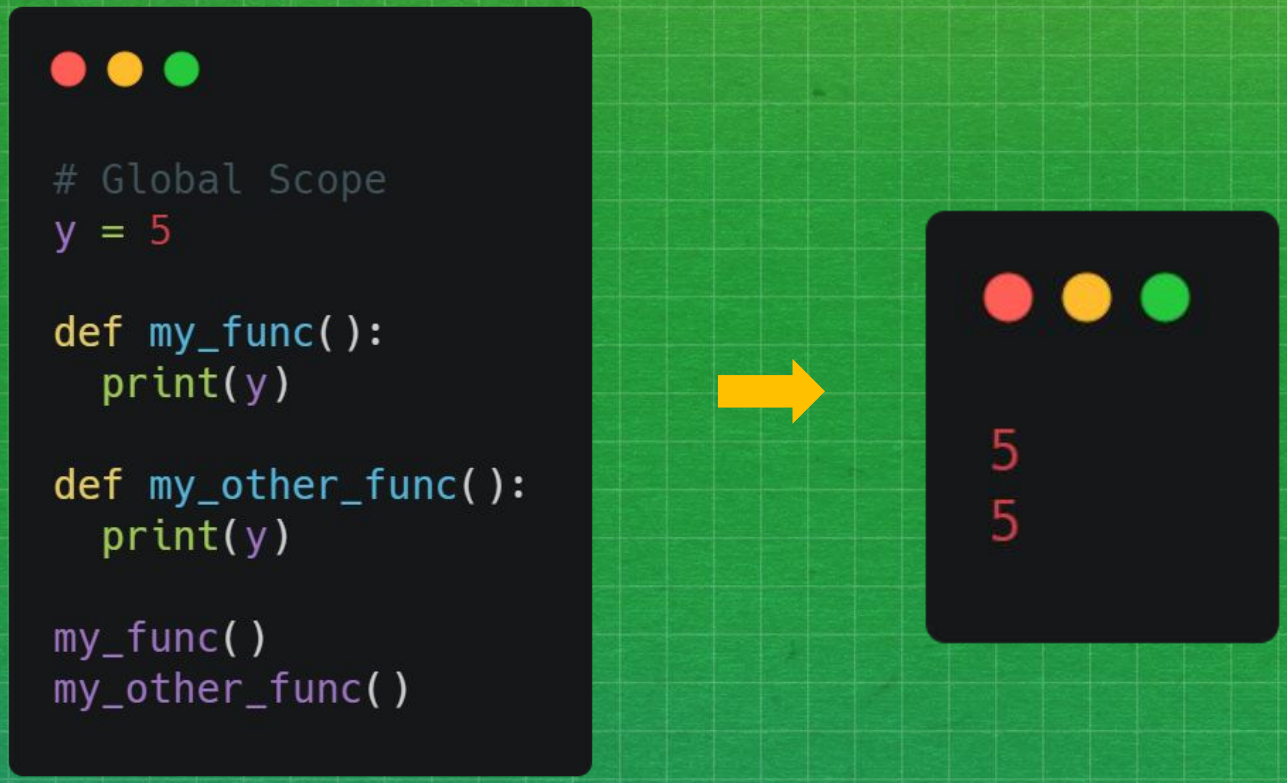
Reto

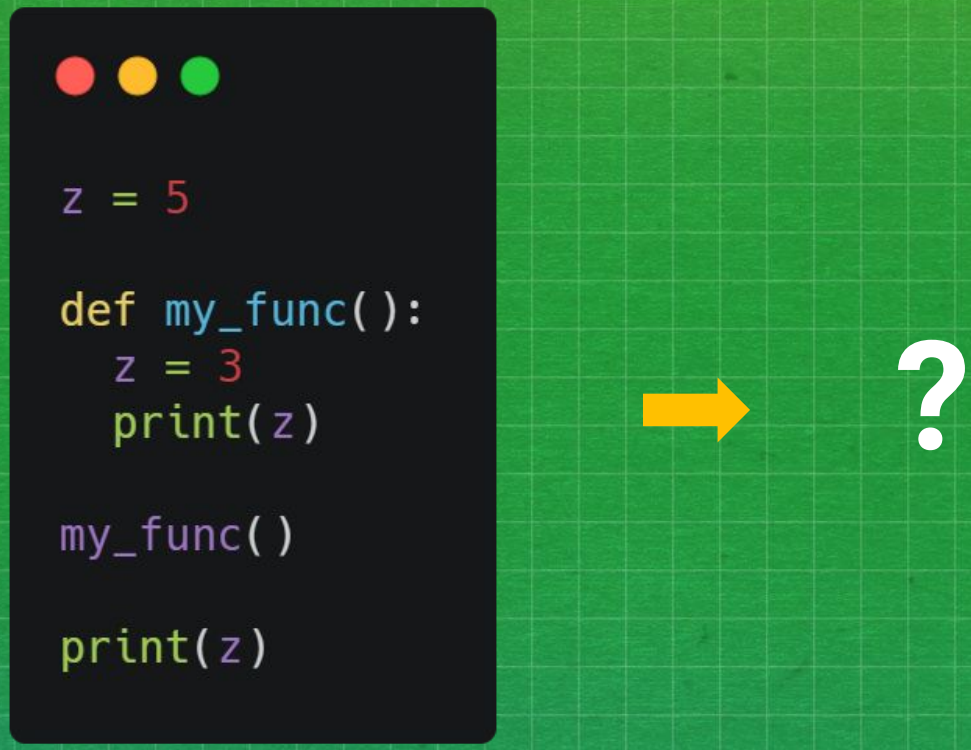
Crea un programa que verifique si un número es primo o no, pero hazlo con tipado estático.

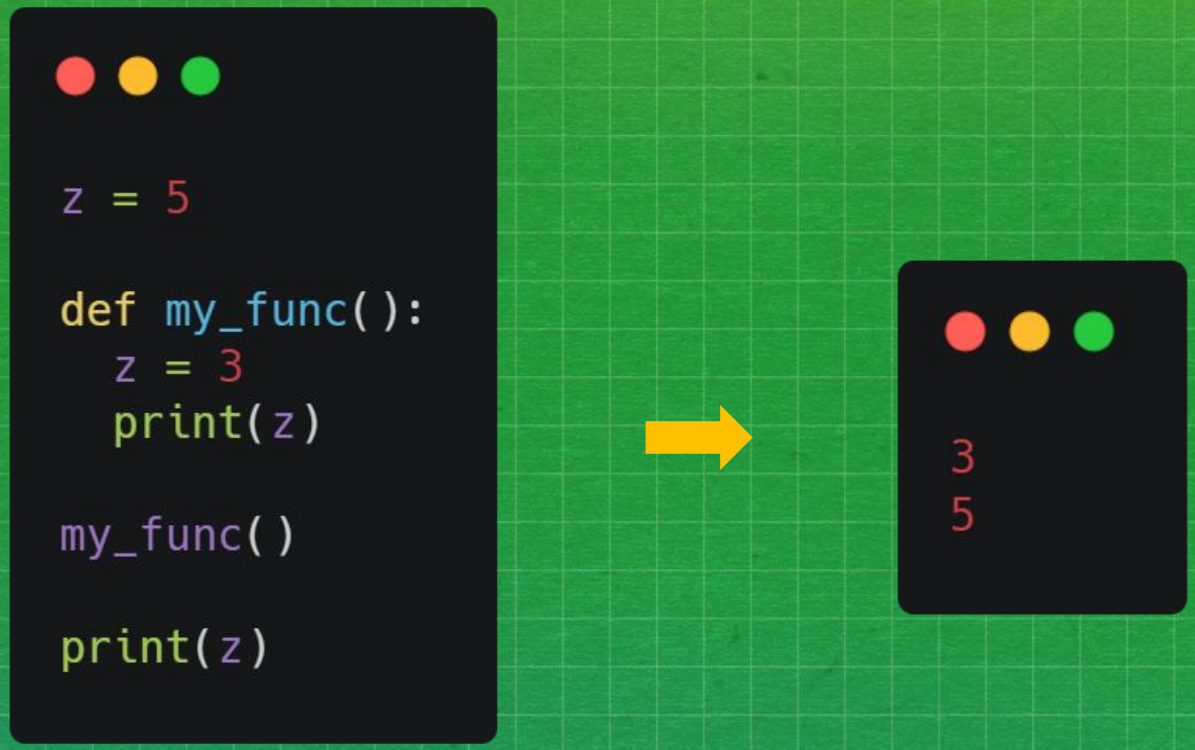
Scope: alcance de las variables

Una variable solo está disponible dentro de la región donde fue creada.


# Local Scope
def my_func():
y = 5
print(y)
my_func()
5

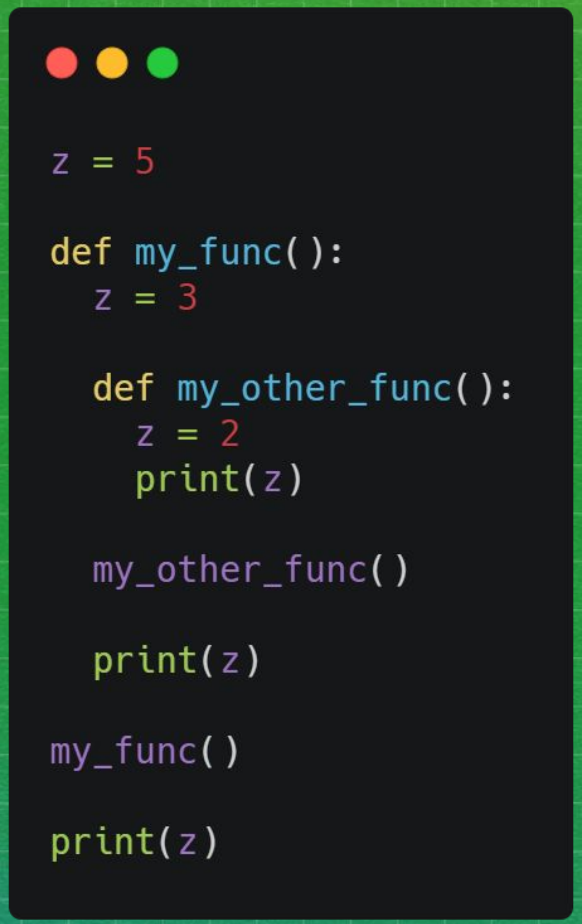

# Global Scope
y = 5
def my_func():
print(y)
def my_other_func():
print(y)
my_func()
my_other_func()
5
5


z = 5
def my_func():
z = 3
print(z)
my_func()
print(z)
?


z = 5
def my_func():
z = 3
print(z)
my_func()
print(z)
3
5

```python
z = 5

def my_func():
  z = 3

  def my_other_func():
    z = 2
    print(z)

  my_other_func()

  print(z)

my_func()

print(z)
```

?



```python
z = 5

def my_func():
  z = 3

  def my_other_func():
    z = 2
    print(z)

  my_other_func()

  print(z)

my_func()

print(z)
```

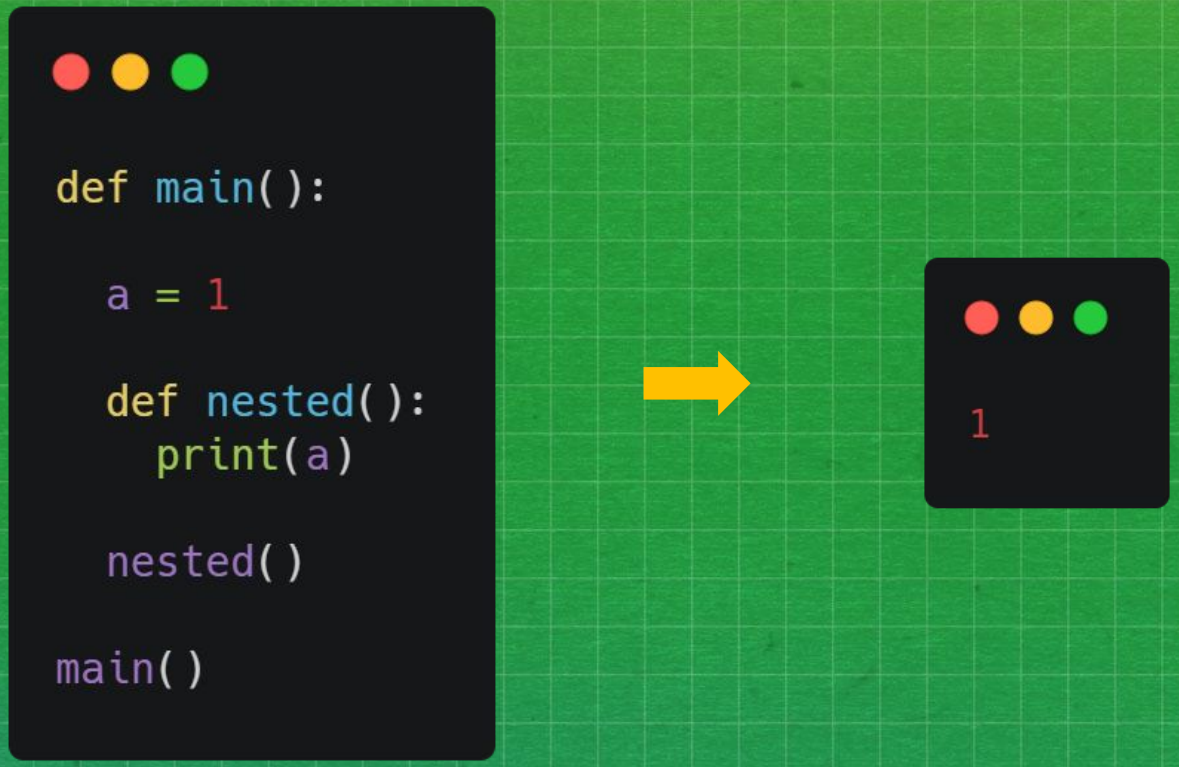
```
2
3
5
```

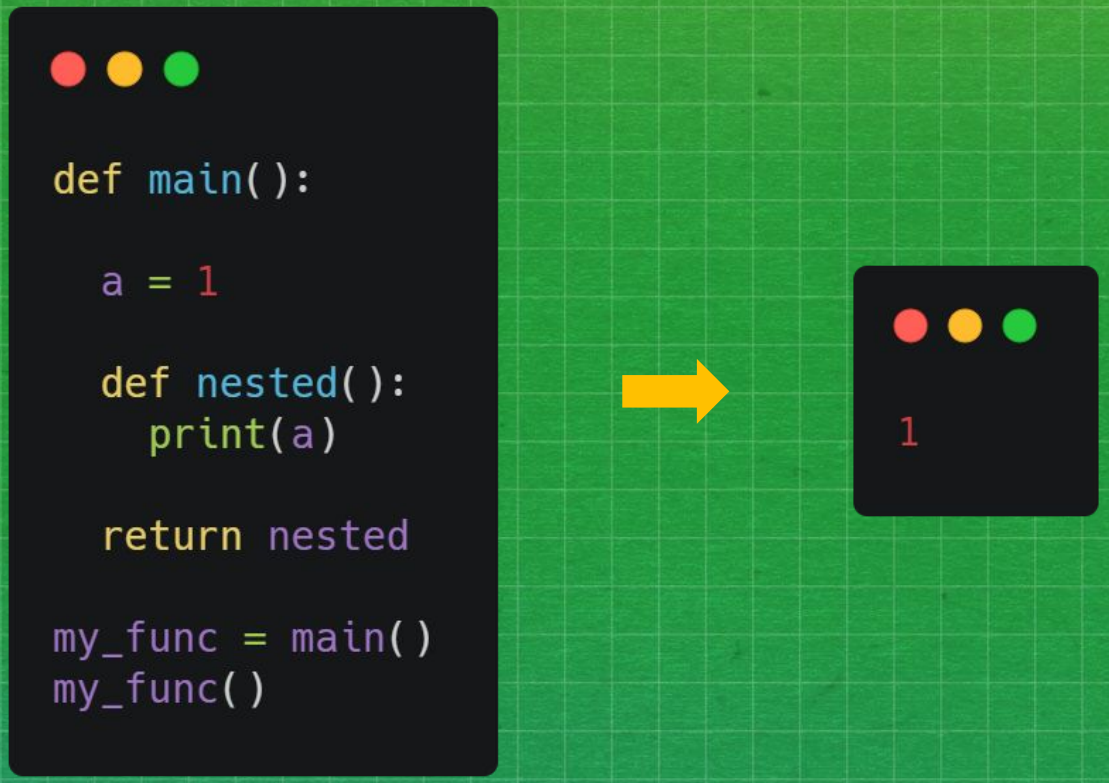
Closures

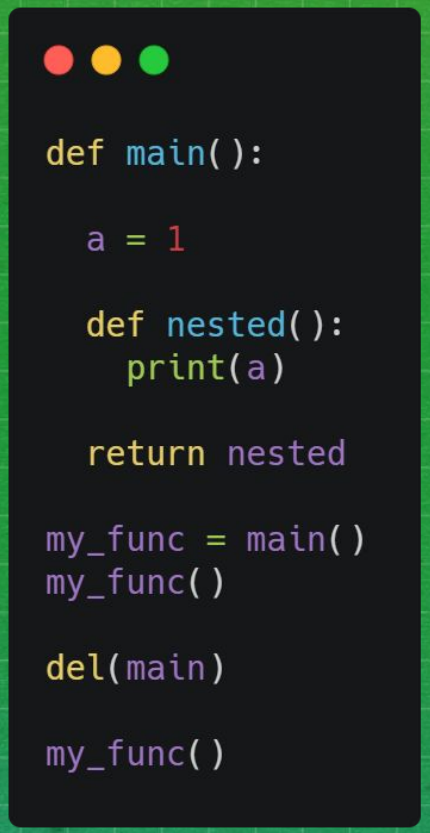
Qué son y dónde se utilizan

Nested functions

En español, “funciones anidadas”


def main():
a = 1
def nested():
print(a)
nested()
main()
1

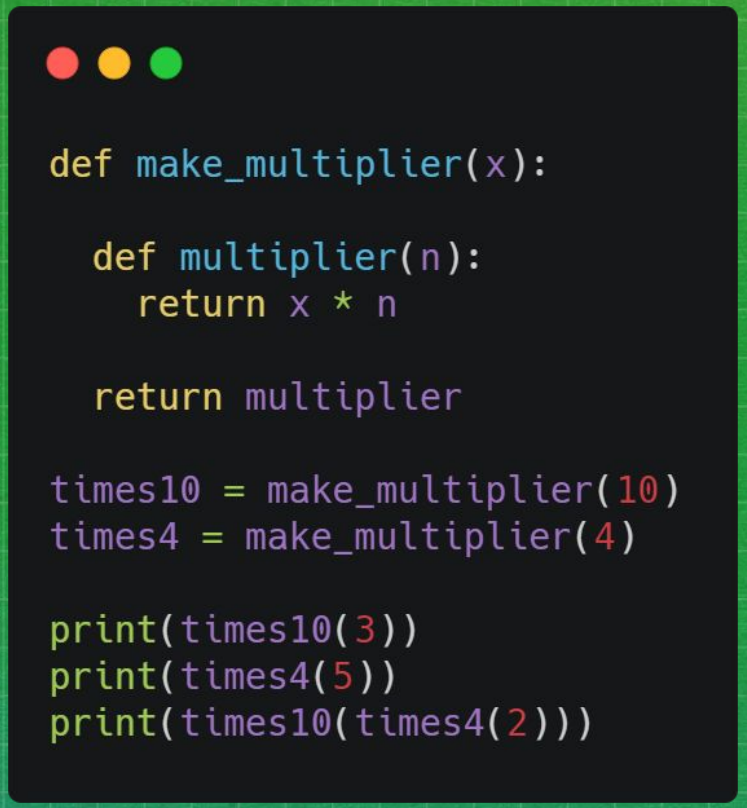

def main():
a = 1
def nested():
print(a)
return nested
my_func = main()
my_func()
1

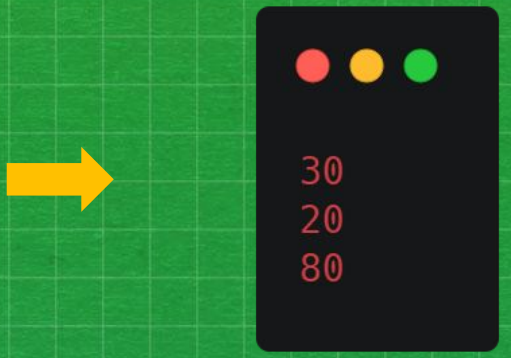

def main():
a = 1
def nested():
print(a)
return nested
my_func = main()
my_func()
del(main)
my_func()


1
1

Reglas para encontrar un closure

- Debemos tener una nested function.
- La nested function debe referenciar un valor de un scope superior.
- La función que envuelve a la nested function debe retornarla también.


def make_multiplier(x):
def multiplier(n):
return x * n
return multiplier
times10 = make_multiplier(10)
times4 = make_multiplier(4)
print(times10(3))
print(times4(5))
print(times10(times4(2)))


30
20
80

¿Dónde aparecen los closures?

Programando closures

Es hora de poner en práctica lo aprendido.

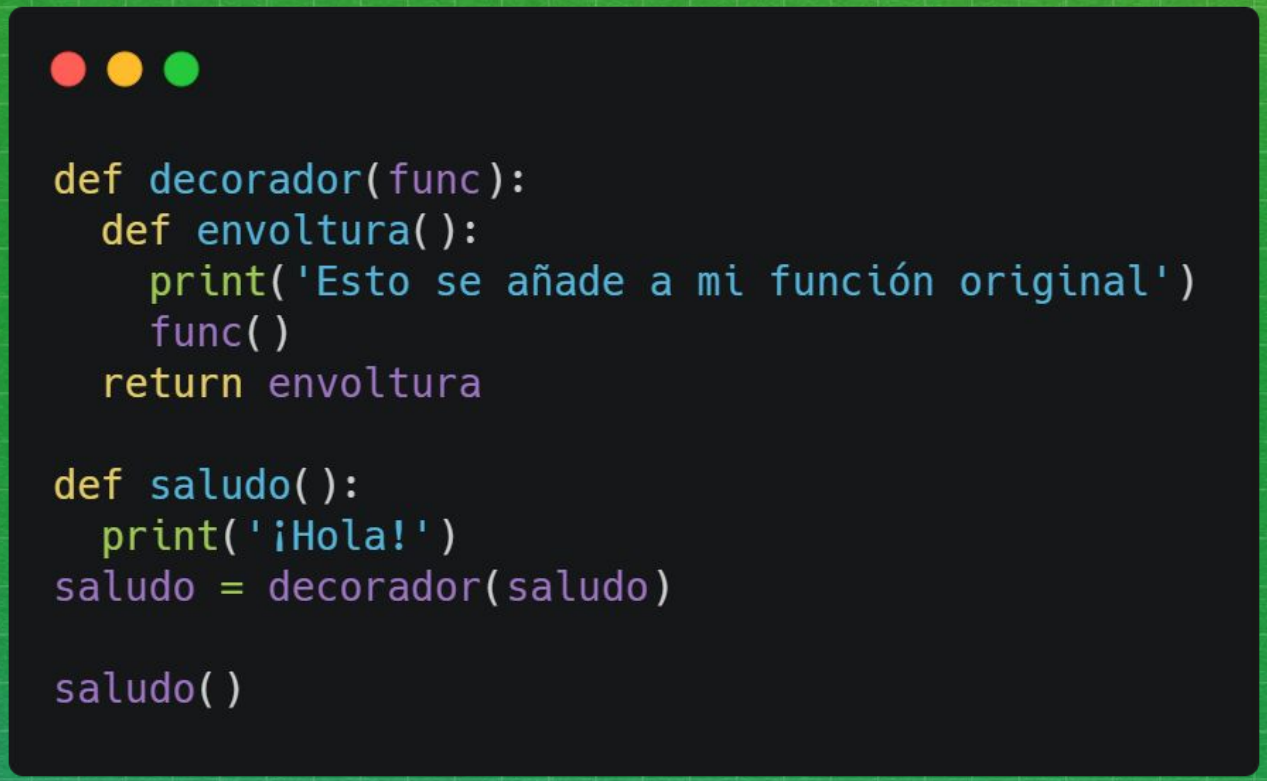
```python
def make_division_by(n):
    """This closure returns a function that returns the division
       of an x number by n
    """
    # You have to code here!
    pass

division_by_3 = make_division_by(3)
print(division_by_3(18))  # The expected output is 6

division_by_5 = make_division_by(5)
print(division_by_5(100))  # The expected output is 20

division_by_18 = make_division_by(18)
print(division_by_18(54))  # The expected output is 3
```

Decoradores

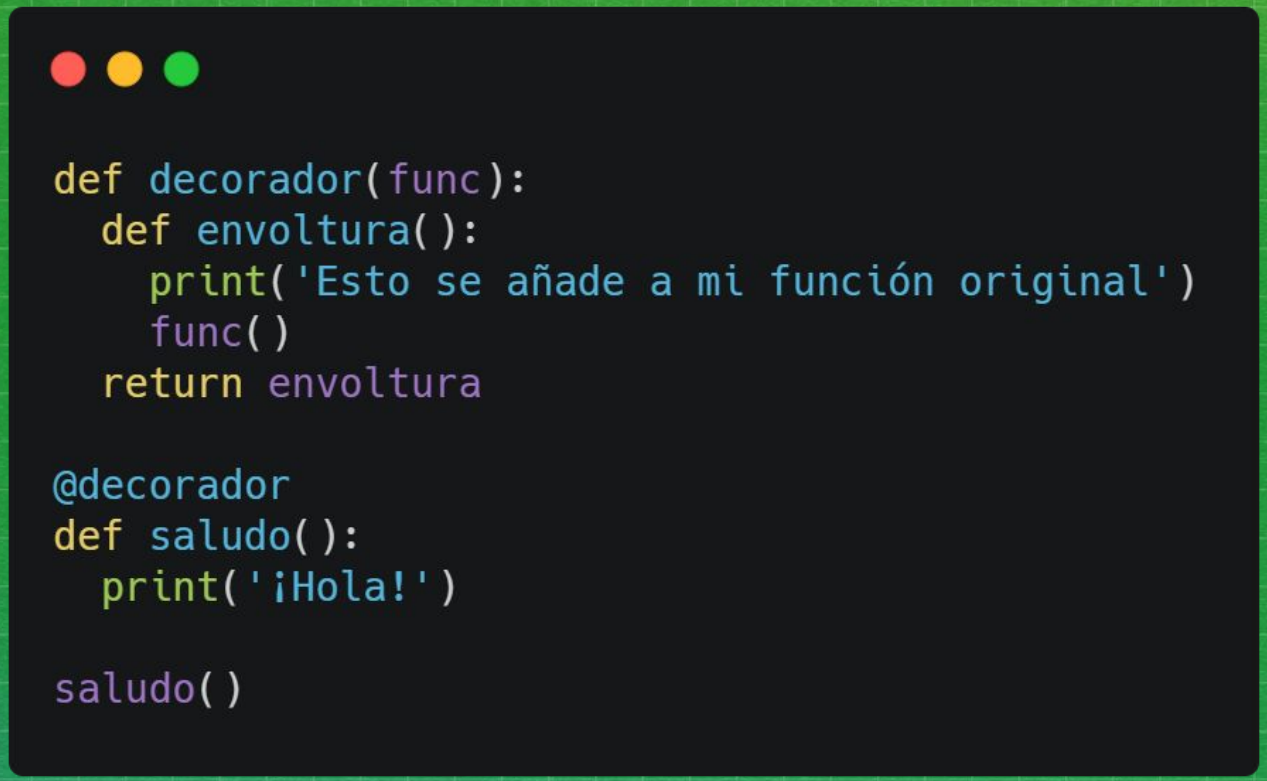
El concepto más avanzado de funciones.

Función que recibe como parámetro otra función, le añade cosas, y retorna una función diferente.

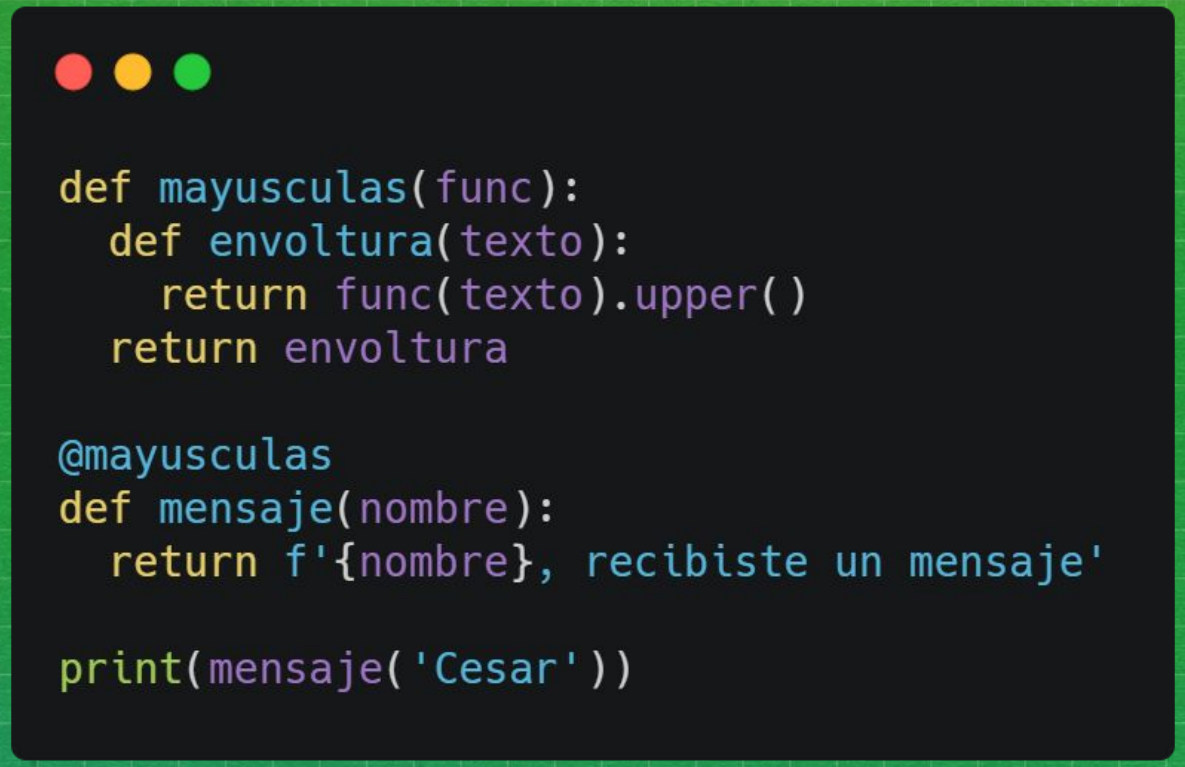
```python
def decorador(func):
  def envoltura():
    print('Esto se añade a mi función original')
    func()
  return envoltura

def saludo():
  print('¡Hola!')

saludo()
# Output 👇
# ¡Hola!

saludo = decorador(saludo)
saludo()
# Output 👇
# Esto se añade a mi función original
# ¡Hola!
```



```
def decorador(func):
  def envoltura():
    print('Esto se añade a mi función original')
    func()
  return envoltura

def saludo():
  print('¡Hola!')
saludo = decorador(saludo)

saludo()
```



```
def decorador(func):
  def envoltura():
    print('Esto se añade a mi función original')
    func()
  return envoltura

@decorador
def saludo():
  print('¡Hola!')

saludo()
```



```python
def mayusculas(func):
  def envoltura(texto):
    return func(texto).upper()
  return envoltura

@mayusculas
def mensaje(nombre):
  return f'{nombre}, recibiste un mensaje'

print(mensaje('Cesar'))
```

Programando decoradores

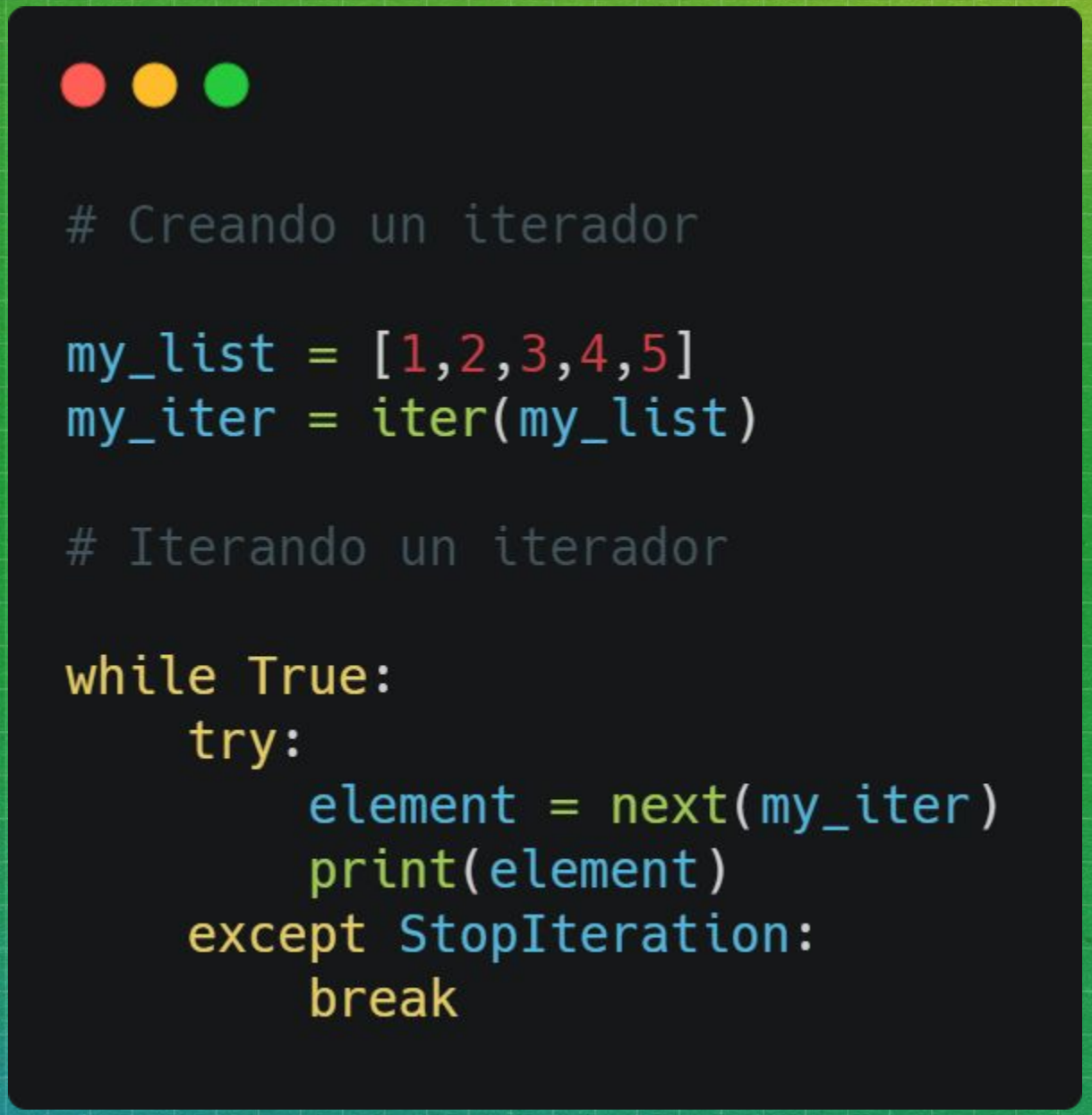
Es hora de poner en práctica lo aprendido

Reto libre

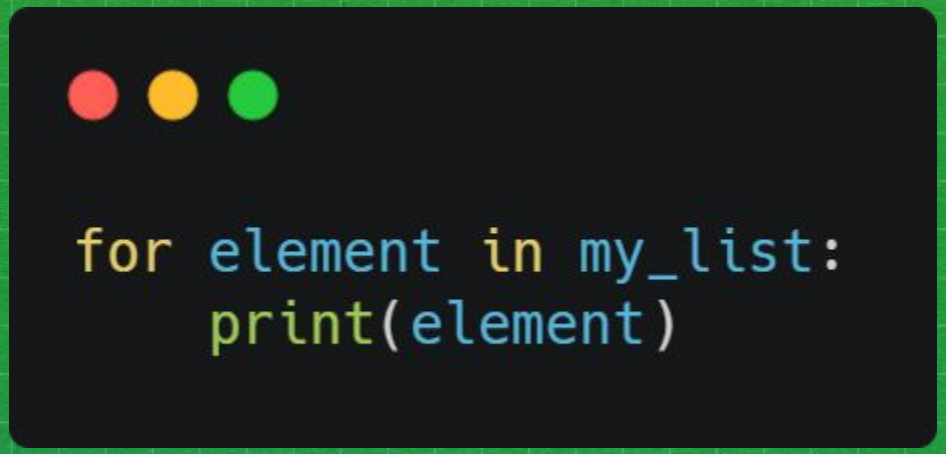
Iteradores

Una estructura de datos para guardar datos infinitos

```python
# Creando un iterador

my_list = [1,2,3,4,5]
my_iter = iter(my_list)

# Iterando un iterador

print(next(my_iter))

# Cuando no quedan datos, la excepción StopIteration es elevada
```



```python
# Creando un iterador

my_list = [1,2,3,4,5]
my_iter = iter(my_list)

# Iterando un iterador

while True:
    try:
        element = next(my_iter)
        print(element)
    except StopIteration:
        break
```


for element in my_list:
print(element)

¿Cómo construyo un iterador?

```python
class EvenNumbers:

    """Clase que implementa un iterador
    de todos los números pares, o los números
    pares hasta un máximo"""

    def __init__(self, max=None):
        self.max = max

    def __iter__(self):
        self.num = 0
        return self

    def __next__(self):
        if not self.max or self.num <= self.max:
            result = self.num
            self.num += 2
            return result
        else:
            raise StopIteration
```

Las ventajas de usar iteradores

La Sucesión de Fibonacci

Un ejemplo clave para entender iteradores

0 1 1 2 3 5 8 13
21 34 55 89 144...

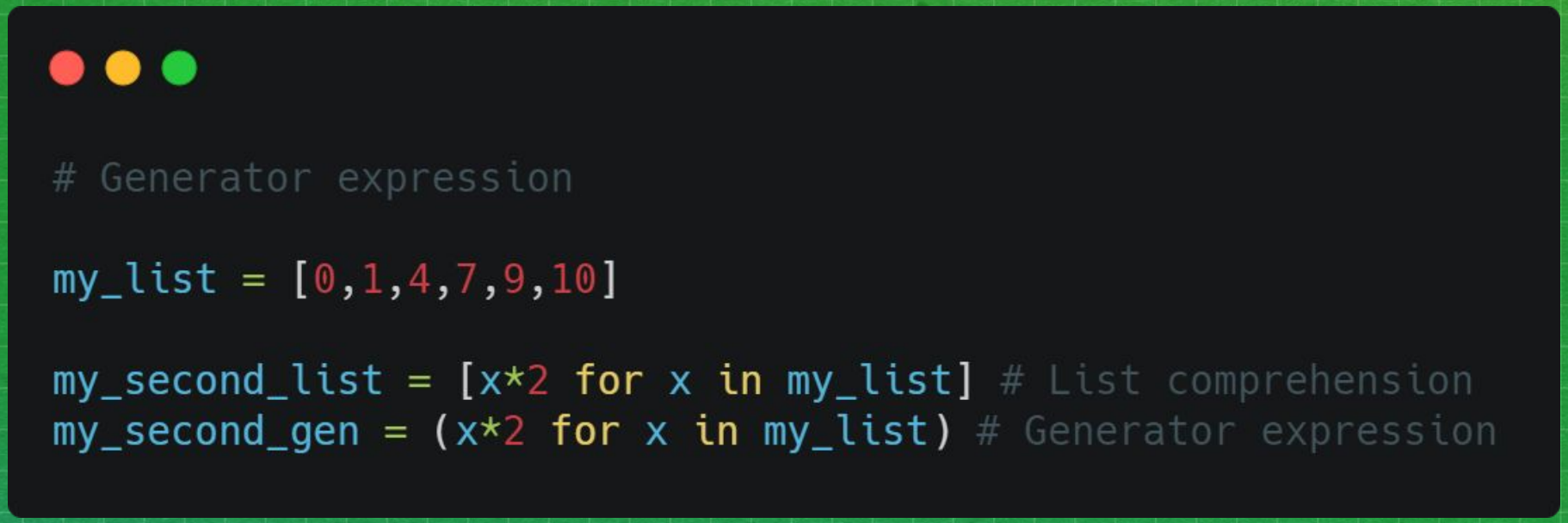
Reto

Modifica el iterador que acabamos de crear, pero que en lugar de devolver los infinitos números de la sucesión de Fibonacci, solo devuelva los números hasta un máximo. Una vez que esto pase, eleva el error StopIteration.

Generadores

“Sugar syntax” de los iteradores

```python
def my_gen():

    """Un ejemplo de generadores"""

    print('Hello world!')
    n = 0
    yield n

    print('Hello heaven!')
    n = 1
    yield n

    print('Hello hell!')
    n = 2
    yield n

a = my_gen()
print(next(a)) #Hello world!
print(next(a)) #Hello heaven!
print(next(a)) #Hello hell!
print(next(a)) StopIteration
```


Generator expression

my_list = [0,1,4,7,9,10]

my_second_list = [x*2 for x in my_list] # List comprehension
my_second_gen = (x*2 for x in my_list) # Generator expression

Las ventajas de usar generadores

Mejorando nuestra Sucesión de Fibonacci

Generadores en la práctica

Reto

Modifica el generador que acabamos de crear, pero que en lugar de devolver los infinitos números de la sucesión de Fibonacci, solo devuelva los números hasta un máximo.

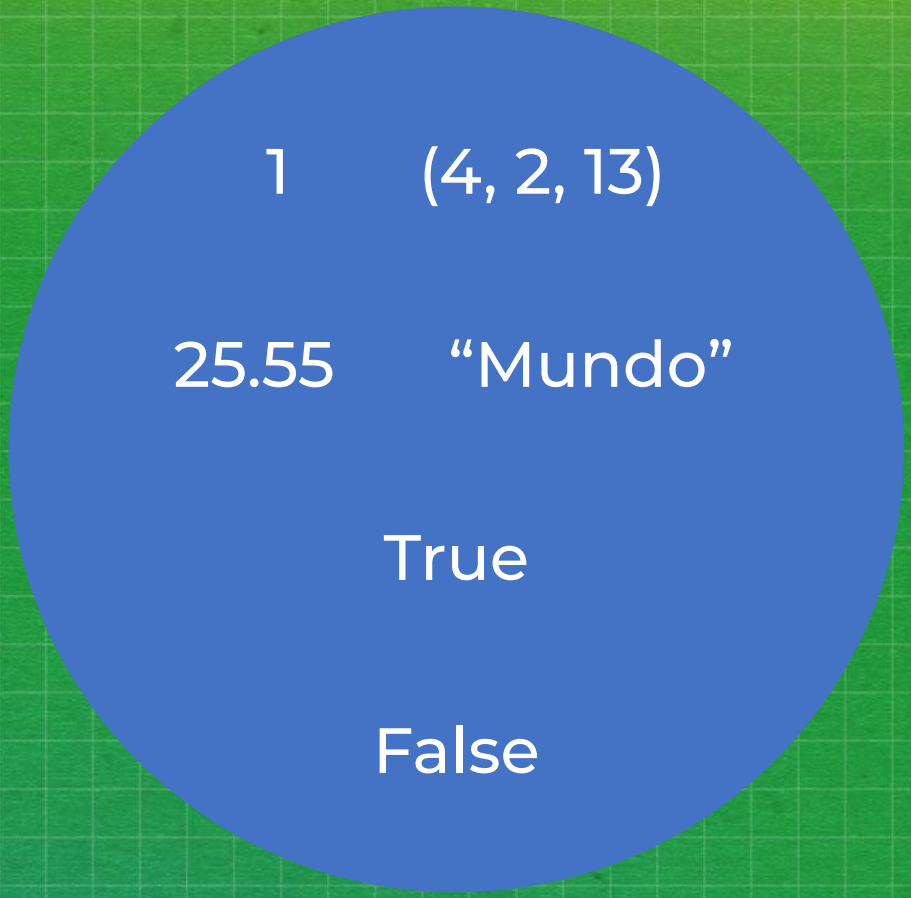
Sets

¿Qué son los conjuntos?

Sets

Una colección desordenada de elementos únicos e inmutables.

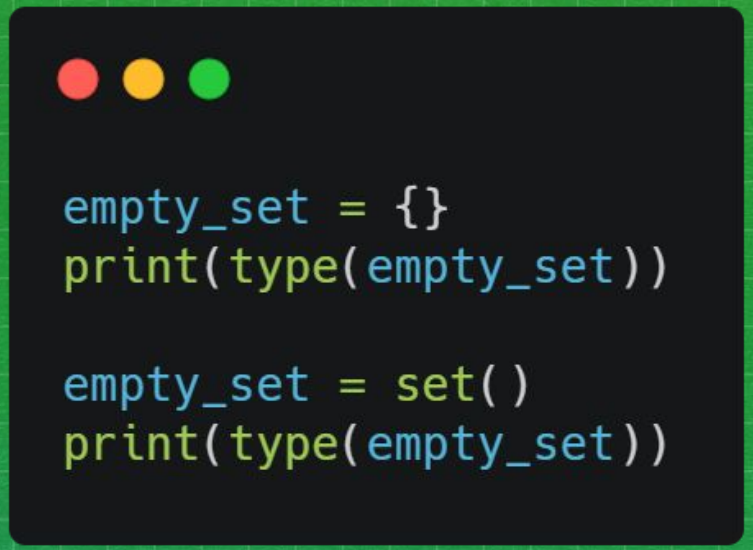

1
(1, 2, 3)
23.3
“Hola”
True
[1, 2]
1


1
(4, 2, 13)
25.55
“Mundo”
True
False

Creación de sets

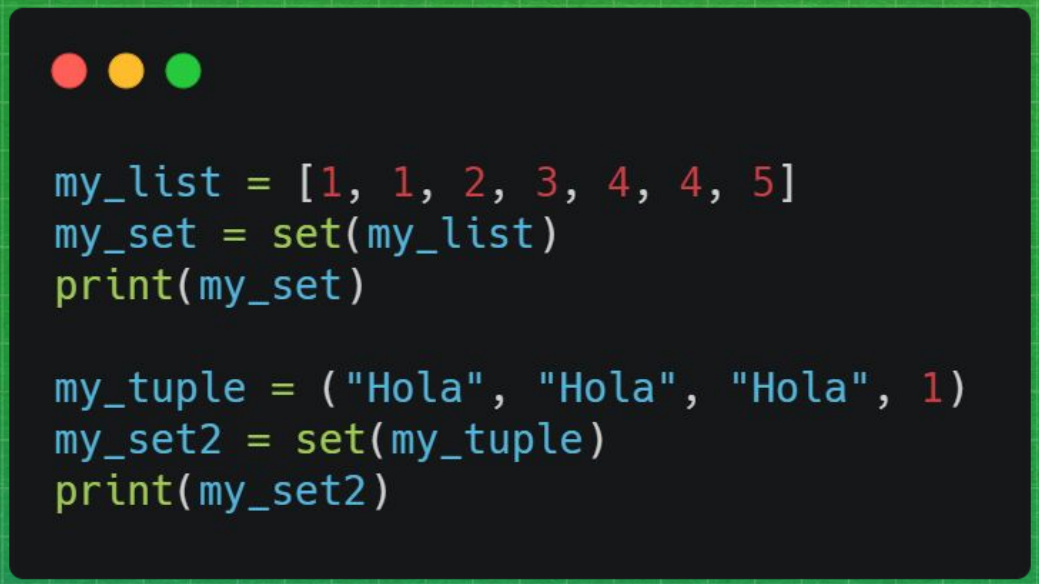
```python
my_set = {3, 4, 5}
print("my_set =", my_set)

my_set2 = {"Hola", 23.3, False, True}
print("my_set2 =", my_set2)

my_set3 = {3, 3, 2}
print("my_set3 =", my_set3)

my_set4 = {[1, 2, 3], 4}
print("my_set4 =", my_set4)
```

```
my_set = {3, 4, 5}
my_set2 = {False, True, 'Hola', 23.3}
my_set3 = {2, 3}

Traceback (most recent call last):
  File "main.py", line 10, in <module>
    my_set4 = {[1, 2, 3], 4}
TypeError: unhashable type: 'list'
```

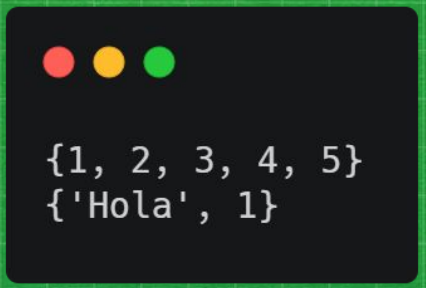

empty_set = {}
print(type(empty_set))
empty_set = set()
print(type(empty_set))


<class 'dict'>
<class 'set'>

Casting con sets



```python
my_list = [1, 1, 2, 3, 4, 4, 5]
my_set = set(my_list)
print(my_set)

my_tuple = ("Hola", "Hola", "Hola", 1)
my_set2 = set(my_tuple)
print(my_set2)
```



```
{1, 2, 3, 4, 5}
{'Hola', 1}
```

Añadir elementos a un set

```python
my_set = {1, 2, 3}
print(my_set)

# Añadir un elemento
my_set.add(4)
print(my_set)

# Añadir múltiples elementos
my_set.update([1, 2, 5])
print(my_set)

# Añadir múltiples elementos
my_set.update((1, 7, 2), {6, 8})
print(my_set)
```

```
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6, 7, 8}
```

Borrar elementos de un set

```python
my_set = {1, 2, 3, 4, 5, 6, 7}
print(my_set)

# Borrar un elemento existente
my_set.discard(1)
print(my_set)

# Borrar un elemento existente
my_set.remove(2)
print(my_set)

# Borrar un elemento inexistente
my_set.discard(10)
print(my_set)

# Borrar un elemento inexistente
my_set.remove(12)
print(my_set)
```

```
{1, 2, 3, 4, 5, 6, 7}
{2, 3, 4, 5, 6, 7}
{3, 4, 5, 6, 7}
{3, 4, 5, 6, 7}

Traceback (most recent call last):
    File "main.py", line 17, in <module>
        my_set.remove(12)
KeyError: 12
```

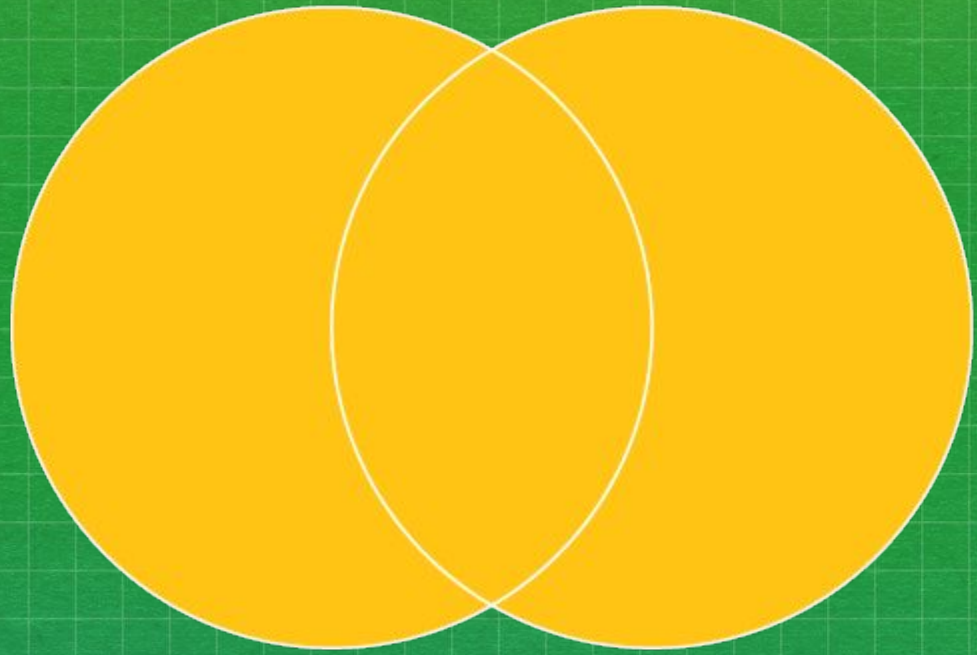
```python
my_set = {1, 2, 3, 4, 5, 6, 7}
print(my_set)

# Borrar un elemento aleatorio
my_set.pop()
print(my_set)

# Limpiar el set
my_set.clear()
print(my_set)
```

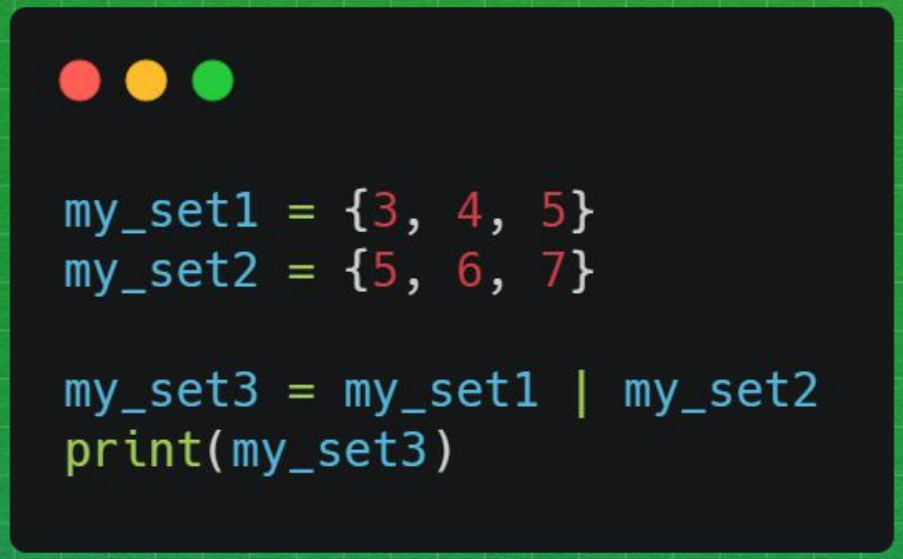
```
{1, 2, 3, 4, 5, 6, 7}
{2, 3, 4, 5, 6, 7}
set()
```

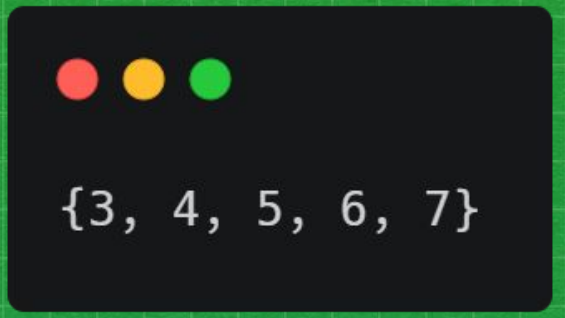
Operaciones con sets

Unión, intersección y diferencia

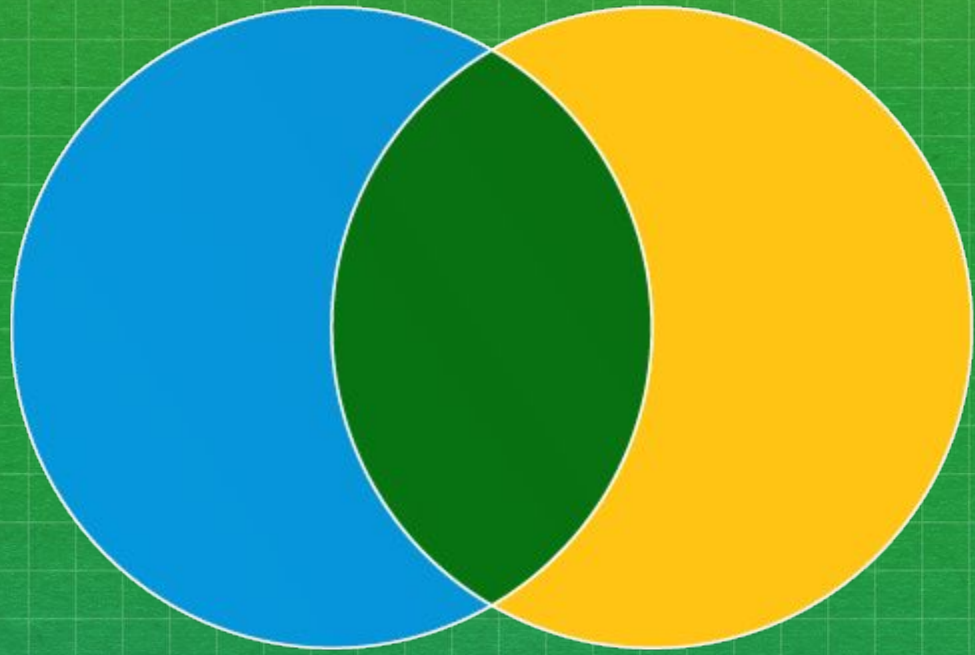
Unión

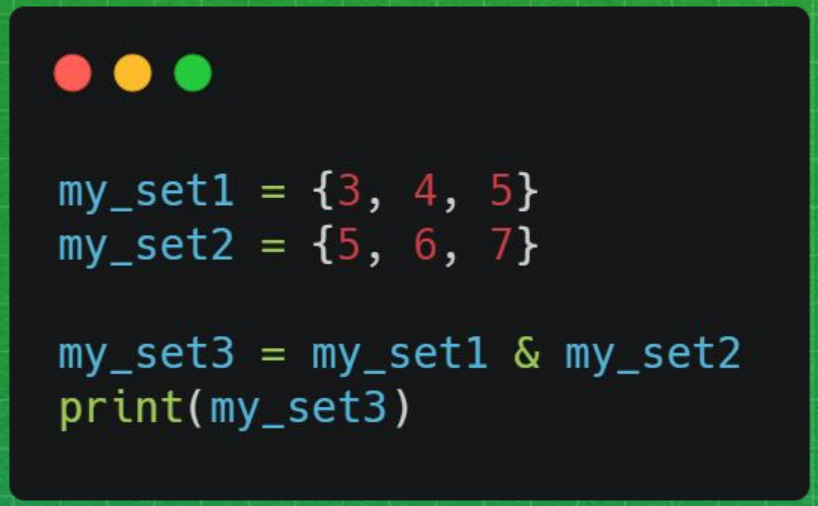



my_set1 = {3, 4, 5}
my_set2 = {5, 6, 7}
my_set3 = my_set1 | my_set2
print(my_set3)


{3, 4, 5, 6, 7}

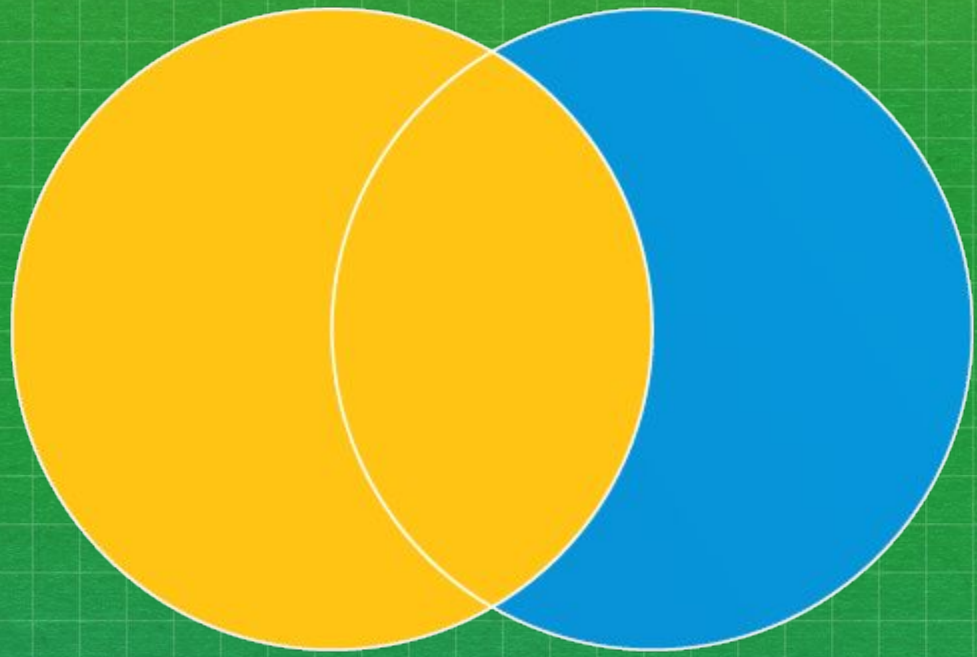
Intersección

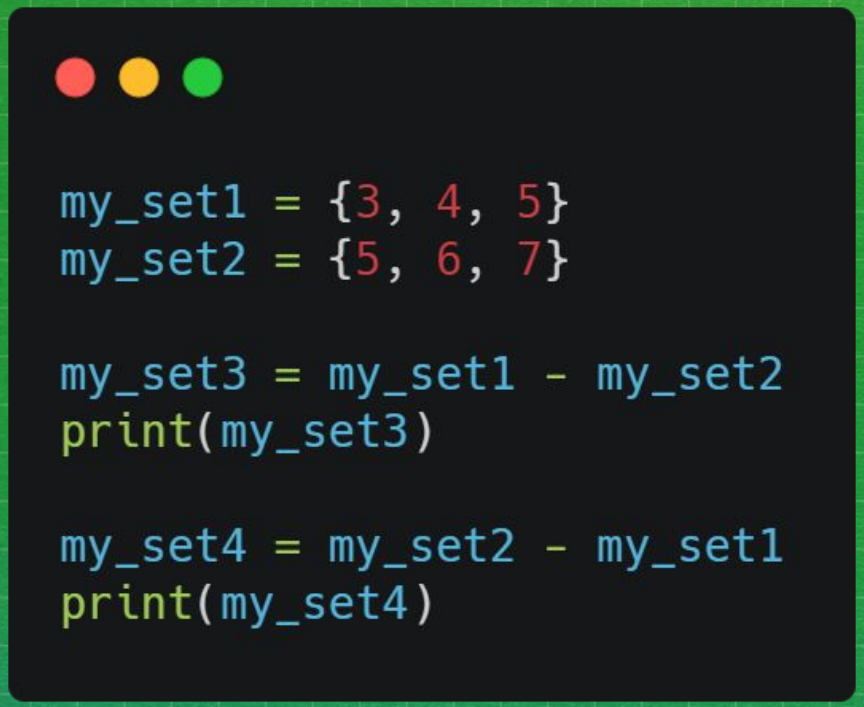



my_set1 = {3, 4, 5}
my_set2 = {5, 6, 7}
my_set3 = my_set1 & my_set2
print(my_set3)


{5}

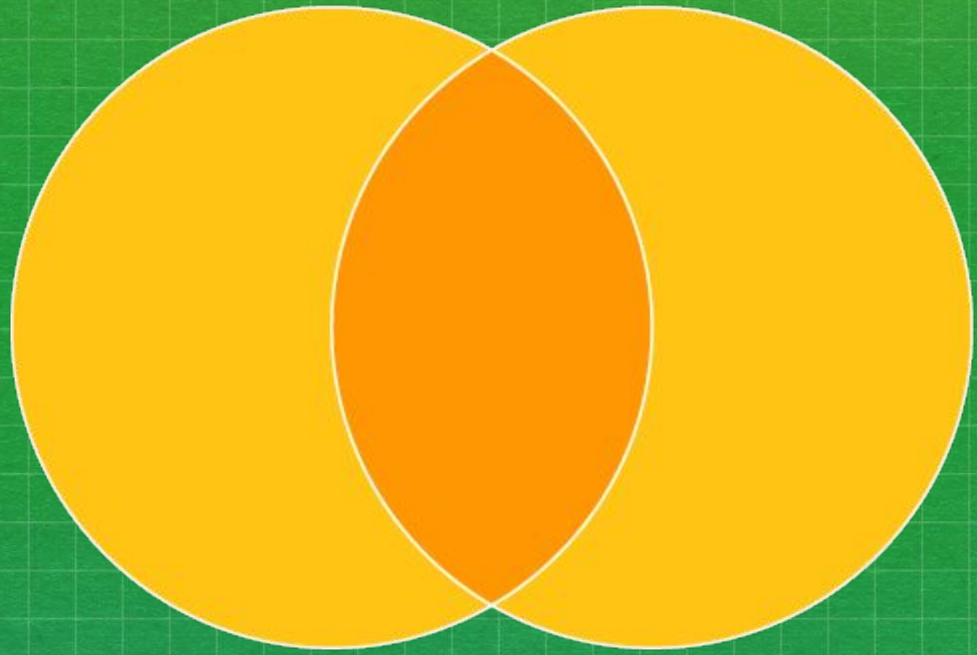
Diferencia

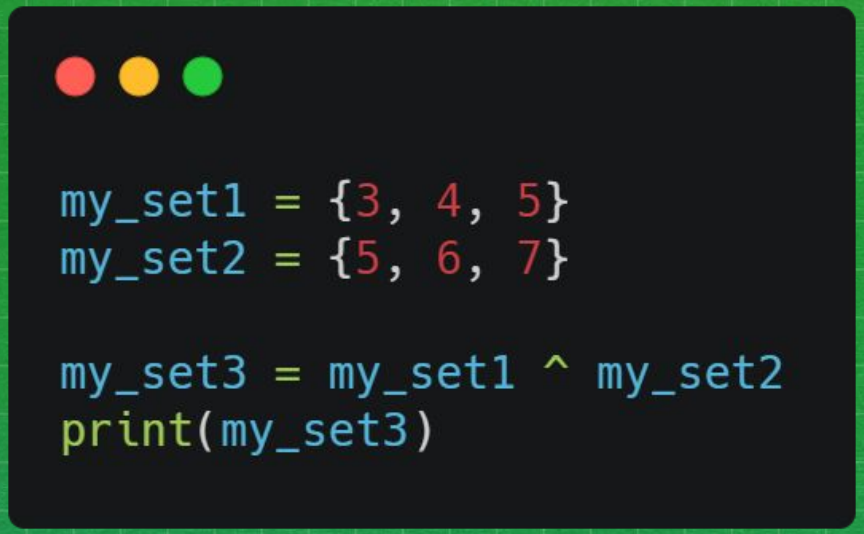


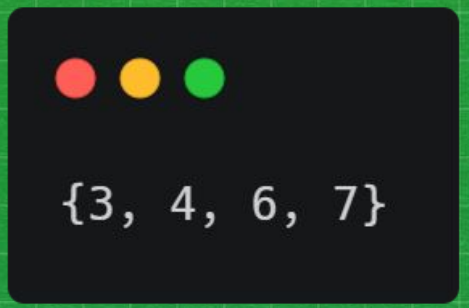

my_set1 = {3, 4, 5}
my_set2 = {5, 6, 7}
my_set3 = my_set1 - my_set2
print(my_set3)
my_set4 = my_set2 - my_set1
print(my_set4)


{3, 4}
{6, 7}

Diferencia simétrica




my_set1 = {3, 4, 5}
my_set2 = {5, 6, 7}
my_set3 = my_set1 ^ my_set2
print(my_set3)


{3, 4, 6, 7}

Eliminando los repetidos de una lista

Practicando el uso de conjuntos

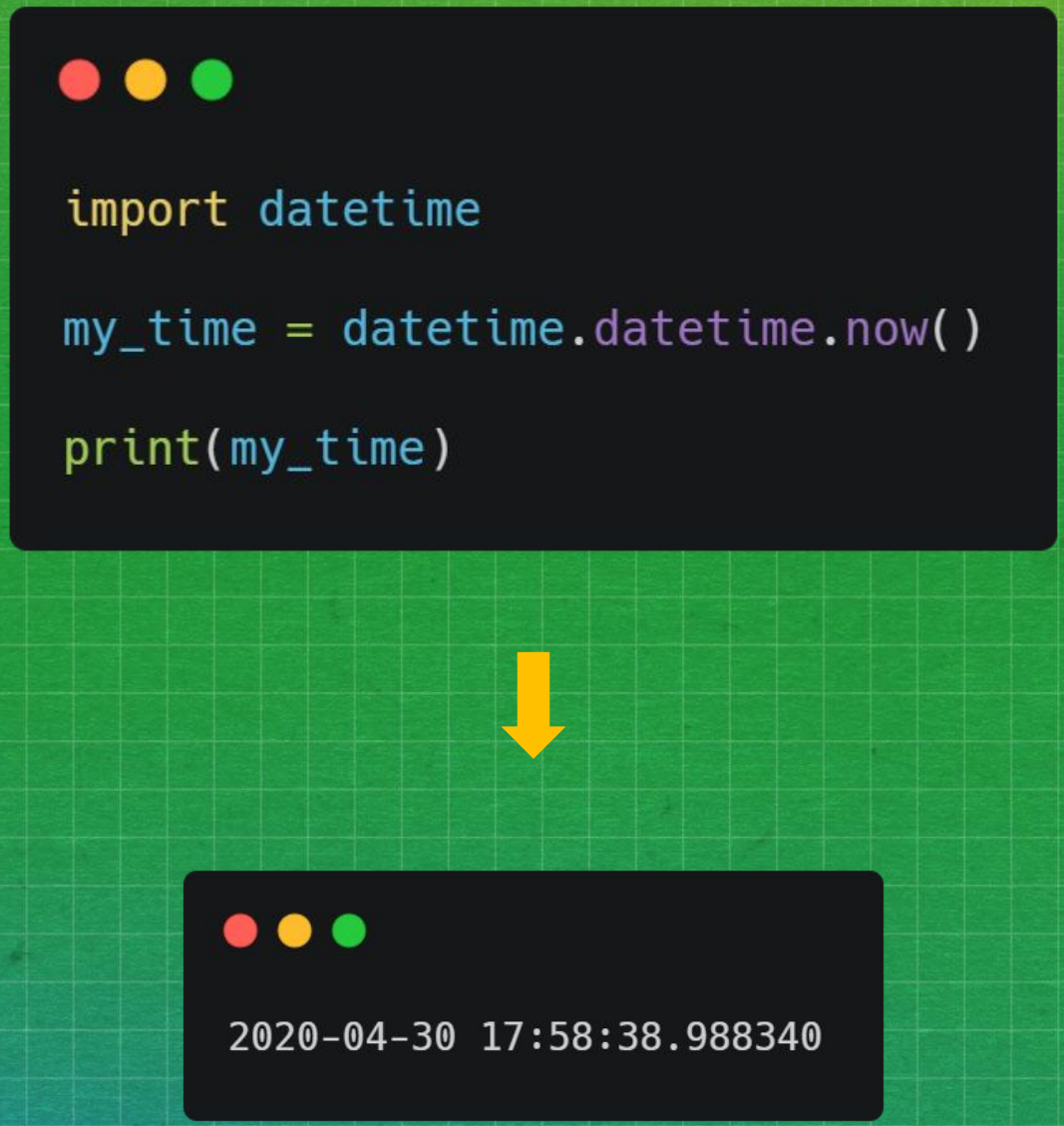
Reto

Crea un programa que elimine los elementos repetidos de una lista, pero en lugar de utilizar un ciclo for utiliza sets.

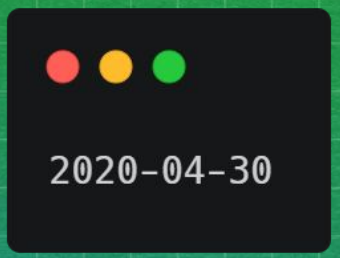
Manejo de fechas

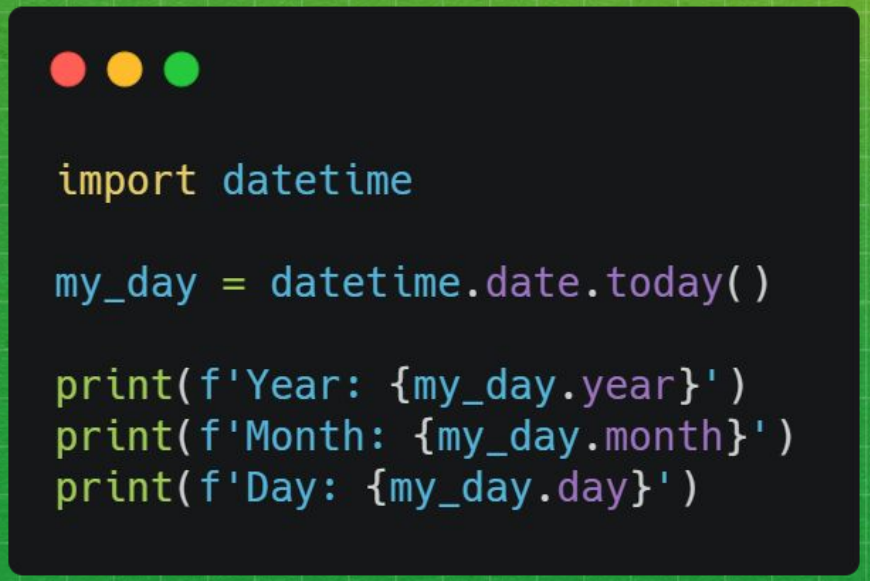
¿Cómo manejar el tiempo en tu aplicación?

datetime


import datetime
my_time = datetime.datetime.now()
print(my_time)
2020-04-30 17:58:38.988340


import datetime
my_day = datetime.date.today()
print(my_day)


2020-04-30


import datetime
my_day = datetime.date.today()
print(f'Year: {my_day.year}')
print(f'Month: {my_day.month}')
print(f'Day: {my_day.day}')


Year: 2020
Month: 4
Day: 30

Formateo de fechas

mm/dd/yyyy
dd/mm/yyyy

%Y	Year
%m	Month
%d	Day
%H	Hour
%M	Minute
%S	Second

```python
from datetime import datetime

my_datetime = datetime.now()
print(my_datetime)

my_str = my_datetime.strftime('%d/%m/%Y')
print(f'Formato LATAM: {my_str}')

my_str = my_datetime.strftime('%m/%d/%Y')
print(f'Formato USA: {my_str}')

my_str = my_datetime.strftime('Estamos en el año %Y')
print(f'Formato Random: {my_str}')
```

```
2020-04-30 19:57:19.904103
Formato LATAM: 30/04/2020
Formato USA: 04/30/2020
Formato Random: Estamos en el año 2020
```

Time zones

Sabemos que en cada país la hora es diferente...

00:00

22:00

22:00

pytz

Desafío

¿Quieres preguntarme algo?

@facmartoni

facundonicolas.com