

Introducción a Python



CodeLab

FAQ

Frequently Asked Questions

¿DE QUÉ VA A TRATAR?

Programación

Python

¿DE QUÉ NO VA A TRATAR?

Otras librerías*

Uso de frameworks

Inteligencia Artificial

¿QUÉ TENGO QUE SABER?

Fundamentos de la

programación (opcional)

* Si que se va a enseñar a importar código y gestionar dependencias



Presentación

Sígueme en LinkedIn ;)



Alejandro Gómez González 

Ingeniero Informático, Analista-Programador
Zaragoza, Aragón, España · [Información de contacto](#)

 hiberus

 Universidad de Zaragoza

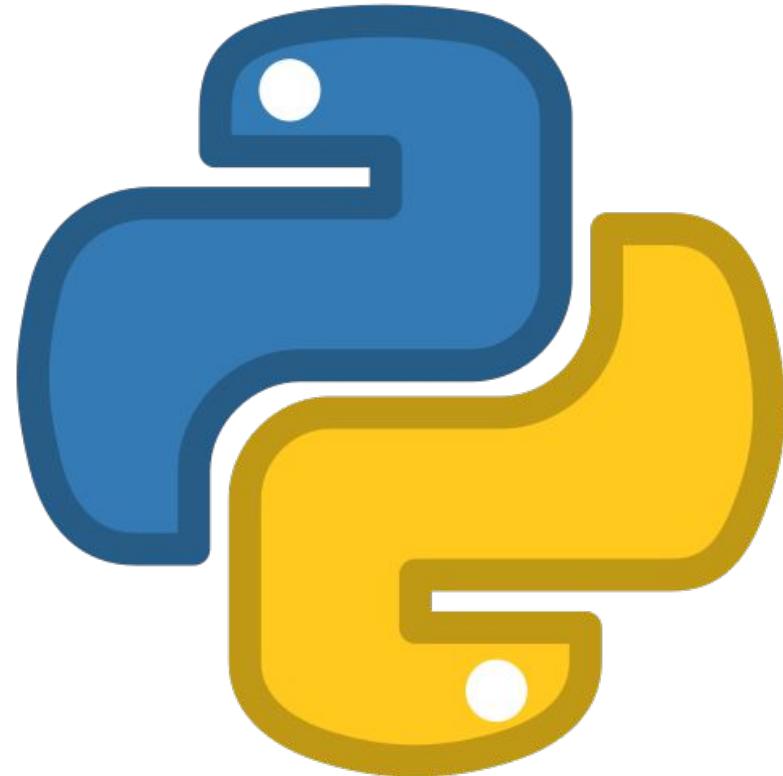


CodeLab

Índice

Duración aproximada 2-3h

1. Historia y filosofía
2. Sintaxis y ortografía
3. Importación de código
4. Entresijos
5. Anuncio



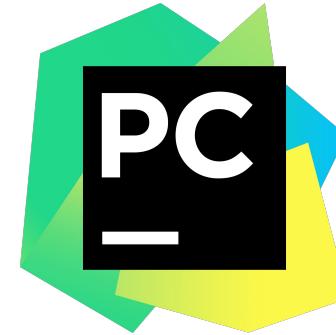
Entornos de desarrollo

Se vienen cositas

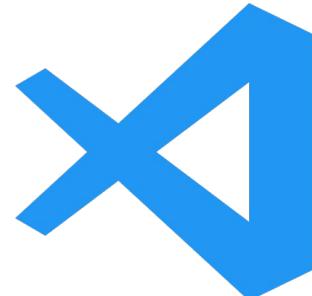


Google Colab

<https://colab.research.google.com/#create=true>



PyCharm Community Edition



Virtual Studio Code

<https://vscode.dev/>



CodeLab

Historia y filosofía



CodeLab

Historia y filosofía

Origen



Van Rossum

En los 80 fue parte del equipo de desarrollando el lenguaje de programación ABC en el CWI

An example function to collect the set of all `words` in a document:

```
HOW TO RETURN words document:  
  PUT {} IN collection  
  FOR line IN document:  
    FOR word IN split line:  
      IF word not.in collection:  
        INSERT word IN collection  
  RETURN collection
```



Historia y filosofía

Origen



Van Rossum



Los caballeros de la mesa cuadrada (1975)



Versión 0.9.0 en 1991



Historia y filosofía

¿Qué tipo de lenguaje de programación es Python?

Según su nivel:

- Bajo nivel
- Medio nivel
- Alto nivel

Según su ejecución:

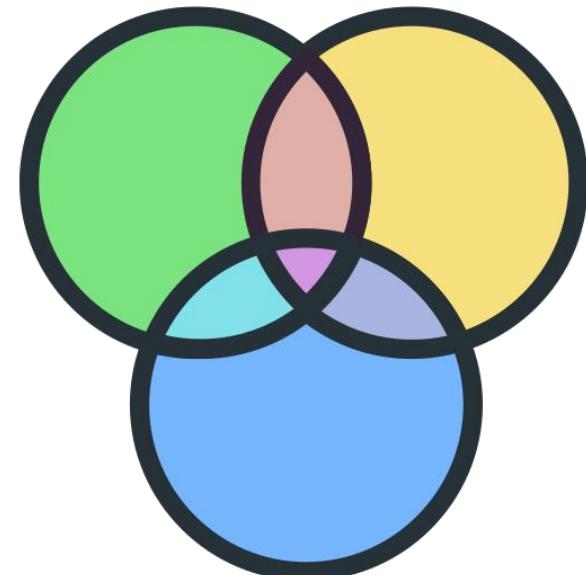
- Compilado
- Interpretado

Según su tipado:

- Fuertemente tipado
- Débilmente tipado

Según su programación:

- Programación orientada a objetos (POO)
- Programación funcional
- Programación imperativa
- Programación declarativa



Historia y filosofía

Historial de versiones

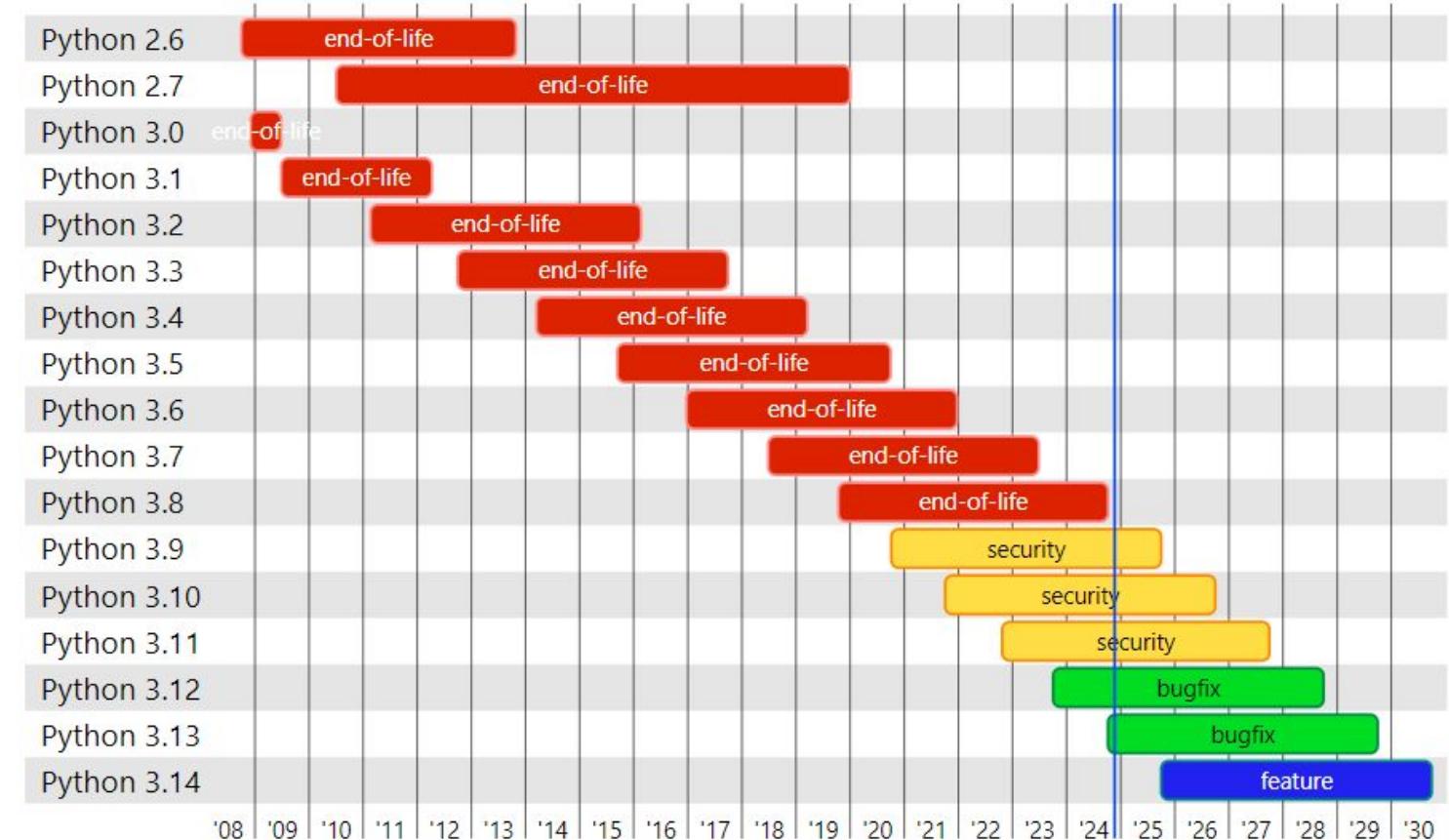
Versión 0.9.0 en 1991

Versión 1.0.0 en 1994

Versión 2.0.0 en 2000

Versión 3.0.0 en 2008

Python release cycle



A Noviembre de 2024

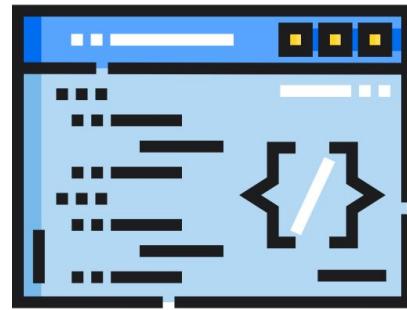


Historia y filosofía

Usos



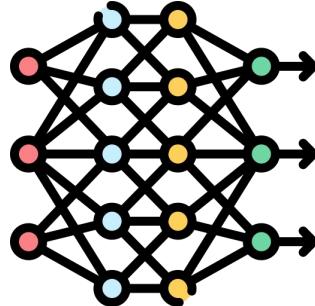
Desarrollo
Web



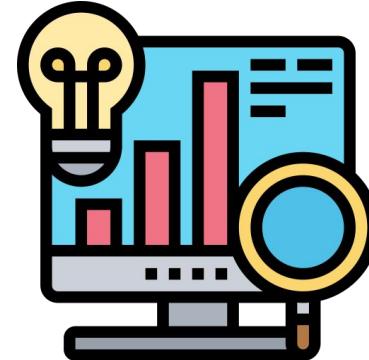
Automatización y
Tareas de
Scripting



Aplicaciones
de Escritorio



Inteligencia Artificial y
Aprendizaje Automático



Ciencia de
Datos y Análisis



Aplicaciones
Científicas y de
Ingeniería



Historia y filosofía

Puntos débiles



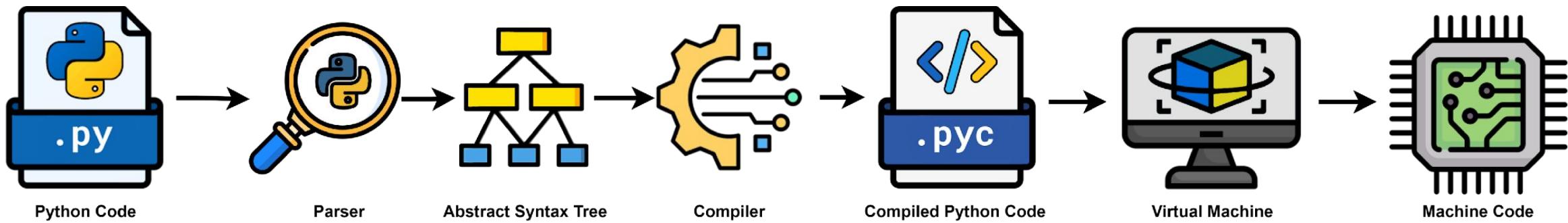
Rendimiento extremo



CodeLab

Historia y filosofía

Ejecución de Python



Sintaxis y ortografía: Lo básico



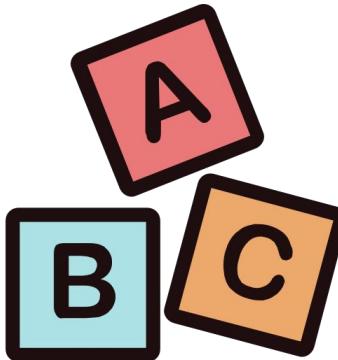
CodeLab

Sintaxis y ortografía

¿Qué hay en el código?



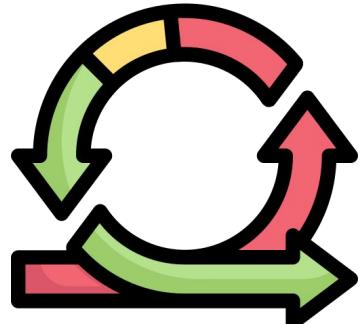
Comentarios



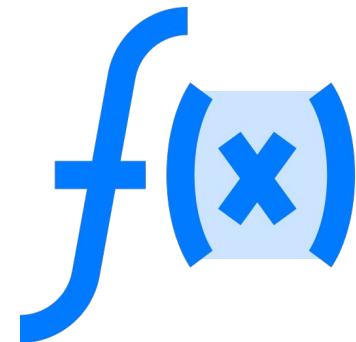
Variables



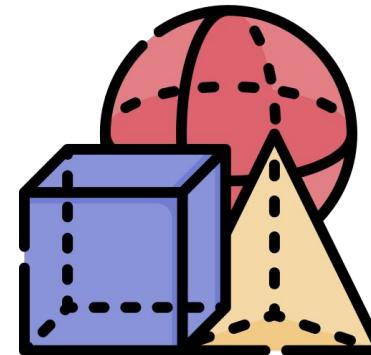
Operaciones



Estructuras de control



Funciones

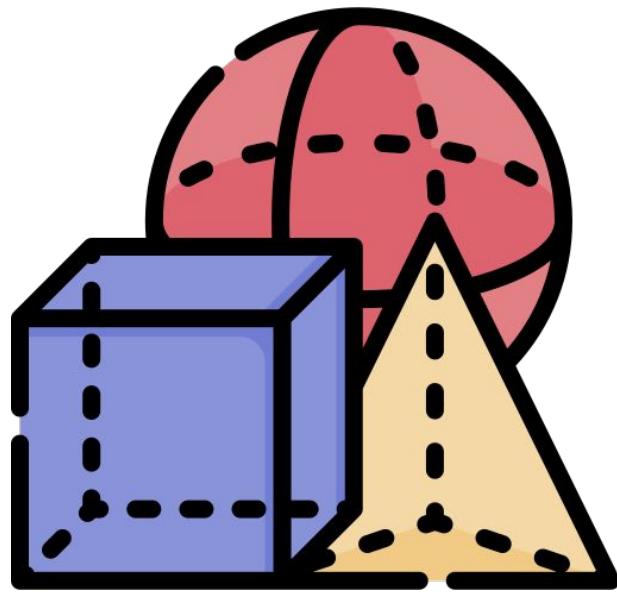


Clases

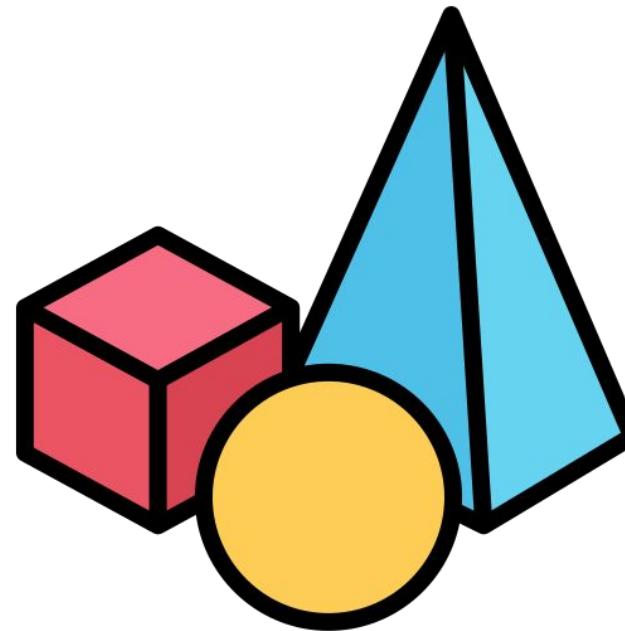


Sintaxis y ortografía

Las clases se instancian en objetos



Clases



Objetos



Sintaxis y ortografía

Comentarios



example.py

```
1 # Comment: This is a single-line comment
2
3 """
4 Multi-line comment:
5 This is a comment that spans multiple lines.
6 """
```



Sintaxis y ortografía

Declaración de variables



example.py

```
1 # Integers (int)
2 integer_number = 42
3
4 # Floats (float)
5 decimal_number = 3.14
6
7 # Strings (str)
8 name = "John"
9 message = 'Hello, World!'
10
11 # Booleans (bool)
12 is_true = True
13 is_false = False
14
15 # Multiple variables
16 x, y, z = 1, 2, 3
17
18 # Variables with names containing underscores (snake_case)
19 user_name = "user123"
```



CodeLab

Sintaxis y ortografía

Sugerencias de tipado (Type Hints)



example.py

```
1 # Integers (int)
2 integer_number: int = 42
3
4 # Floats (float)
5 decimal_number: float = 3.14
6
7 # Strings (str)
8 name: str = "John"
9 message: str = 'Hello, World!'
10
11 # Booleans (bool)
12 is_true: bool = True
13 is_false: bool = False
14
15 # Multiple variables
16 # Only single target be annotated!!
17
18 # Variables with names containing underscores (snake_case)
19 user_name = "user123"
```



CodeLab

Sintaxis y ortografía

¿Se puede hacer esto?



example.py

```
1 # Variable initialized annotated as an integer
2 x: int = 42
3
4 # Variable reassigned as an string
5 x: str = "The answer to all things"
6
```



CodeLab

Sintaxis y ortografía

Entrada y salida de datos por terminal



example.py

```
1 # Get user input from terminal
2 name = input("Enter your name: ")
3
4 # Display content in terminal
5 print(f"{name} is learning python")
6
```



CodeLab

Sintaxis y ortografía

Operaciones básicas de strings



example.py

```
1 # String Formatting
2 name = "Alice"
3 age = 30
4 formatted_message = f"Hello, my name is {name} and I am {age} years old."
5 print(formatted_message)
6 # Hello, my name is Alice and I am 30 years old.
7
8 # Concatenation
9 message = "Python" + " is " + "cool!"
10 print(message)
11 # Python is cool!
12
13 # String Methods
14 uppercase_message = message.upper()
15 print(f"Uppercase: {uppercase_message}")
16 # Uppercase: PYTHON IS COOL!
17
18 lowercase_message = message.lower()
19 print(f"Lowercase: {lowercase_message}")
20 # Lowercase: python is cool!
21
22 # String Length
23 message = "Python is amazing"
24 length = len(message)
25 print(f"The length of the message is {length} characters")
26 # The length of the message is 17 characters
```



CodeLab

Sintaxis y ortografía

Operaciones básicas

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False



example.py

```
1 # Boolean variables
2 is_true = True
3 is_false = False
4
5 # Logical AND
6 result_and = is_true and is_false
7 # result_and = False
8
9 # Logical OR
10 result_or = is_true or is_false
11 # result_or = True
12
13 # Logical NOT
14 result_not_true = not is_true
15 # result_not_true = False
16
```



Sintaxis y ortografía

Operaciones básicas

a es igual que b	a == b
a no es igual que b	a != b
a es mayor que b	a > b
a es menor que b	a < b
a es igual o mayor que b	a >= b
a es igual o menor que b	a <= b



example.py

```
1 # Comparisons
2 a = 5
3 b = 10
4
5 # Equality
6 equal = a == b
7 # equal = False
8
9 # Inequality
10 not_equal = a != b
11 # not_equal = True
12
13 # Greater than
14 greater_than = a > b
15 # greater_than = False
16
17 # Less than
18 less_than = a < b
19 # less_than = True
20
21 # Greater than or equal to
22 greater_equal = a >= b
23 # greater_equal = False
24
25 # Less than or equal to
26 less_equal = a <= b
27 # less_equal = True
28
```



Sintaxis y ortografía

Operaciones básicas numéricas



example.py

```
1 # Basic Arithmetic Operations in Python (I)
2
3 # Addition
4 num1 = 10
5 num2 = 3
6 sum_result = num1 + num2
7 print("Sum:", sum_result)
8 # Sum: 13
9
10 # Subtraction
11 difference = num1 - num2
12 print("Difference:", difference)
13 # Difference: 7
14
15 # Multiplication
16 product = num1 * num2
17 print("Product:", product)
18 # Product: 30
19
```



example.py

```
1 # Basic Arithmetic Operations in Python (II)
2
3 # Division
4 quotient = num1 / num2
5 print("Quotient:", quotient)
6 # Quotient: 3.3333333333333335
7
8 # Integer Division (Floor Division)
9 integer_quotient = num1 // num2
10 print("Integer Quotient:", integer_quotient)
11 # Integer Quotient: 3
12
13 # Modulo (Remainder)
14 remainder = num1 % num2
15 print("Remainder:", remainder)
16 # Remainder: 1
17
18 # Exponentiation
19 power_result = num1 ** num2
20 print("Power:", power_result)
21 # Power: 1000
22
```



Sintaxis y ortografía

Declaración de condicionales



example.py

```
1 # Conditional Statements (if, elif, else)
2 x = 10
3 y = 5
4 if x > y:
5     print(f"{x} is greater than {y}")
6 elif x < y:
7     print(f"{x} is less than {y}")
8 else:
9     print(f"{x} is equal to {y}")
10 # 10 is greater than 5
11
```



Sintaxis y ortografía

Declaración de bucles while



example.py

```
1 # While loop
2 i = 0
3 while i < 5:
4     print(f"Countdown: {i}")
5     i += 1
6 """
7 Countdown: 0
8 Countdown: 1
9 Countdown: 2
10 Countdown: 3
11 Countdown: 4
12 """
```



Sintaxis y ortografía

Declaración de bucles for



example.py

```
1 for i in [0, 1, 2, 3, 4]:  
2     print(f"Iteration {i}")  
3  
4 """  
5 Iteration 0  
6 Iteration 1  
7 Iteration 2  
8 Iteration 3  
9 Iteration 4  
10 """  
11
```



Sintaxis y ortografía

Declaración de bucles for



example.py

```
1 # For loop
2 for i in range(5):
3     print(f"Iteration {i}")
4 """
5 Iteration 0
6 Iteration 1
7 Iteration 2
8 Iteration 3
9 Iteration 4
10 """
11
```



example.py

```
1 # For loop
2 for i in range(0, 5):
3     print(f"Iteration {i}")
4 """
5 Iteration 0
6 Iteration 1
7 Iteration 2
8 Iteration 3
9 Iteration 4
10 """
11
```



Sintaxis y ortografía

Declaración de bucles for



example.py

```
1 # For loop
2 for i in range(-2, 3):
3     print(f"Iteration: {i}")
4 """
5 Iteration: -2
6 Iteration: -1
7 Iteration: 0
8 Iteration: 1
9 Iteration: 2
10 """
11
```



example.py

```
1 # For loop
2 for i in range(0, 10, 2):
3     print(f"Iteration {i}")
4 """
5 Iteration 0
6 Iteration 2
7 Iteration 4
8 Iteration 6
9 Iteration 8
10 """
11
```



Sintaxis y ortografía

Sentencia pass

pass = no hacer nada



example.py

```
1 if (number > 100):  
2     # TODO: Decide how to treat this case  
3     pass  
4
```

Se suele utilizar para marcar algo que hay que hacer



CodeLab

Sintaxis y ortografía

Sentencia pass



example.py

```
1 if evaluate_certain_condition():
2     # Nothing to be done here
3     pass
4 else:
5     # Need to do something here
6     print("Im doing things...")
7
```

**SE FUSILA
MUY POCO**



ICLA-
CHA!

No hagáis estas cosas...



CodeLab

Sintaxis y ortografía

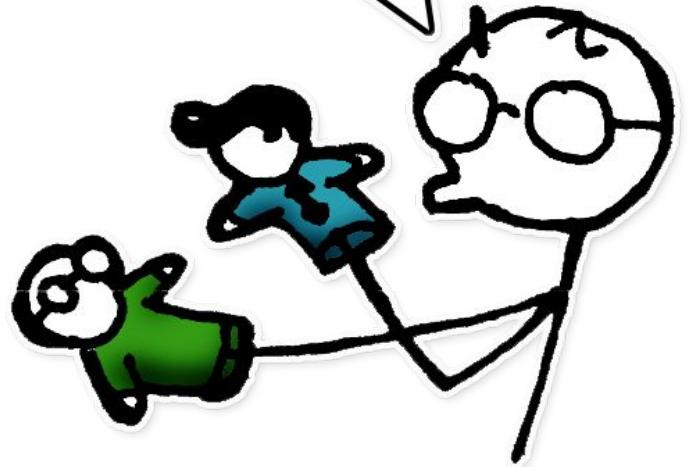
Sentencia `pass`



example.py

```
1 if not evaluate_certain_condition( ):  
2     # Need to do something here  
3     print("I'm doing things...")  
4
```

**¿Te lo explico
con marionetas?**



... hacer esto mejor



CodeLab

Sintaxis y ortografía

Sentencia continue

continue = saltar a la siguiente iteración del bucle



example.py

```
1 for i in range(5):  
2     if i == 2:  
3         continue  
4     print(i)  
5 """  
6 0  
7 1  
8 3  
9 4  
10 """  
11
```

Cuando llega al `continue`, salta a la siguiente iteración y no pasa por el



CodeLab

Sintaxis y ortografía

Sentencia break

break = salir del bucle



example.py

```
1 for number in [10, 11, 12, 13, 14, 15]:  
2     if (number % 3) == 0:  
3         divisible_by_3 = number  
4 # divisible_by_3 = 15  
5
```



example.py

```
1 for number in [10, 11, 12, 13, 14, 15]:  
2     if (number % 3) == 0:  
3         divisible_by_3 = number  
4     break  
5 # divisible_by_3 = 12  
6
```

Itera sobre toda la lista

Cuando llega al break, sale del bucle



CodeLab

Sintaxis y ortografía

Declaración de funciones sin argumentos



example.py

```
1 # Function with Arguments and a Return Value
2 def greet():
3     print("Hello world!")
4
5 # Calling the function
6 greet()
7 # Hello world!
8
```



Sintaxis y ortografía

Declaración de funciones con argumentos



example.py

```
1 # Function with Arguments and a Return Value
2 def add(a, b):
3     result = a + b
4     return result
5
6 # Calling the function
7 sum_result = add(5, 3)
8 print(f"The sum is: {sum_result}")
9 # The sum is: 8
10
```



Sintaxis y ortografía

Declaración de funciones con argumentosopcionales



example.py

```
1 # Function with Optional Arguments and a Return Value
2 def calculate_price(total, discount=0):
3     discounted_price = total - (total * discount)
4     return discounted_price
5
6 # Calling the function with and without a discount
7 total_price = calculate_price(100)
8 discounted_price = calculate_price(100, 0.1)
9
10 print(f"Total price: {total_price}")
11 # Total price: 100
12 print(f"Discounted price: {discounted_price}")
13 # Discounted price: 90.0
14
```



Sintaxis y ortografía

Declaración de funciones con argumentos variables



example.py

```
1 # Function with Variable-Length Arguments
2 def average(*args):
3     total = sum(args)
4     return total / len(args)
5
6 # Calling the function with different numbers of arguments
7 average1 = average(4, 5, 6)
8 average2 = average(10, 20, 30, 40, 50)
9
10 print(f"Average 1: {average1}")
11 # Average 1: 5.0
12 print(f"Average 2: {average2}")
13 # Average 2: 30.0
14
```



Sintaxis y ortografía

Declaración de funciones con sugerencias de tipado



example.py

```
1 def add_integers(x: int, y: int):  
2     return x + y  
3  
4  
5 z = add_integers(3, 2)  
6
```



CodeLab

Sintaxis y ortografía

Declaración de funciones con sugerencias de tipado



example.py

```
1 def add_integers(x: int, y: int):  
2     return x + y  
3  
4  
5 z = add_integers("eggs", "potatoes")  
6
```



CodeLab

Sintaxis y ortografía

Funciones documentadas



example.py

```
1 def add_integers(x: int, y: int) -> int:
2     """
3     This function adds two integers
4
5     :param int x: the first variable of the sum operation
6     :param int y: the second variable of the sum operation
7     :return int: the sum of variable x and y
8     """
9     return x + y
10
```



Sintaxis y ortografía

Funciones como funciones como parámetro



example.py

```
1 # Function that takes another function as an argument
2 def apply_function(func, x):
3     result = func(x)
4     return result
5
6 # Example function to be passed as an argument
7 def square(number):
8     return number ** 2
9
10 # Another example function to be passed as an argument
11 def half(number):
12     return number / 2
13
14 # Calling apply_function with the square function
15 result = apply_function(square, 5)
16 # result = 25
17
18 # Calling apply_function with the half function
19 result = apply_function(half, 5)
20 # result = 2.5
21
22 # Calling apply_function with a lambda function
23 result = apply_function(lambda x: x*2, 5)
24 # result = 10
25
```



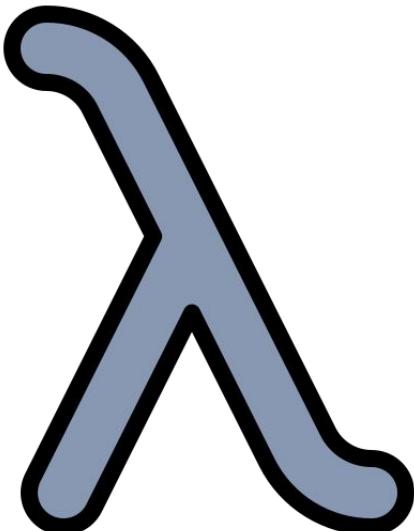
Sintaxis y ortografía

Funciones lambda



example.py

```
1 add_lambda = lambda x, y: x + y
2
3 def add_function(x, y):
4     return x + y
```



example.py

```
1 # Single parameter lambda
2 duplicate = lambda x: x * 2
3 print(duplicate(2))
4 # 4
5
6 # Two parameter lambda
7 add = lambda x, y: x + y
8 print(add(2, 3))
9 # 5
10
11 # Conditional return
12 find_max = lambda x, y: x if x > y else y
13 print(find_max(2, 3))
14 # 3
15
16 # Boolean return
17 is_even = lambda x: x % 2 == 0
18 print(is_even(4))
19 # True
20
```



CodeLab

Sintaxis y ortografía

Declaración e instanciación de clases

```
example.py

1 # Class declaration
2 class FormulaOneCar:
3     def __init__(self, model, driver):
4         self.model = model
5         self.driver = driver
6         self.is_running = False
7
8     def start_engine(self):
9         self.is_running = True
10
11    def stop_engine(self):
12        self.is_running = False
13
14 # Class instantiation
15 my_ferrari = FormulaOneCar("Ferrari SF23", "Carlos Sainz")
16
17 # Accessing class attributes
18 print(f"My F1 car is a {my_ferrari.model} driven by {my_ferrari.driver}.")
19 # My F1 car is a Ferrari SF23 driven by Carlos Sainz.
20
21 print(f"Is the engine running? {my_ferrari.is_running}")
22 # Is the engine running? False
23
24 # Calling class methods
25 my_ferrari.start_engine()
26 print(f"Is the engine running? {my_ferrari.is_running}")
27 # Is the engine running? True
28
```



¡Hagamos una calculadora!



CodeLab

Volvemos a...

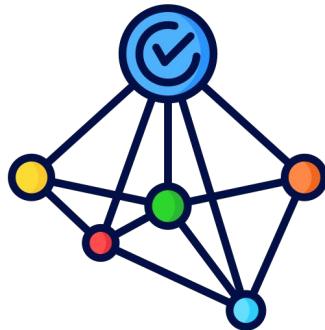
Sintaxis y ortografía: Estructuras de datos



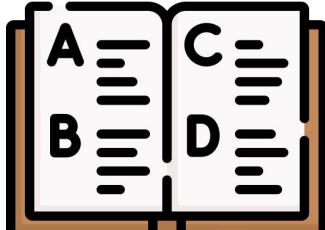
CodeLab

Sintaxis y ortografía

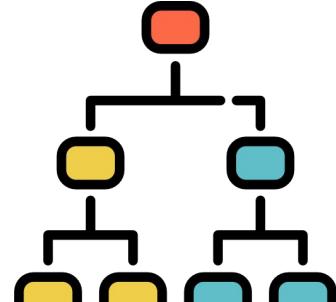
Estructuras de datos



Grafos

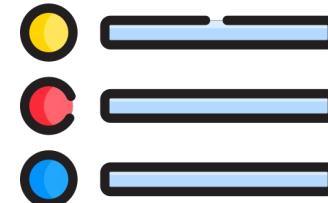


Diccionarios

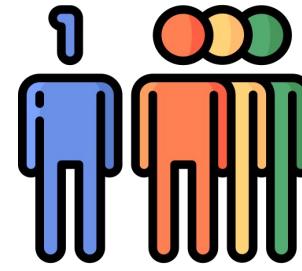


Árboles

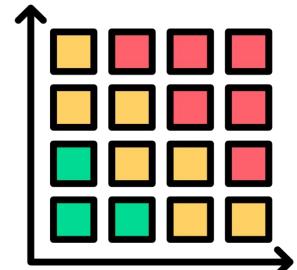
No lineales



Listas



Colas



Matrices

Lineales



Sintaxis y ortografía

Estructuras de datos: lista, tupla, set, diccionario



example.py

```
1 # Lists are versatile and commonly used for storing collections of items of varying types.
2 my_list = [1, 2, 3, 'apple', 'banana']
3
4 # Tuples are used to store INMUTABLE sequences of elements, often for data that should not be
5 # changed.
6 my_tuple = (1, 2, 3, 'apple', 'banana')
7
8 # Sets are used for storing UNIQUE elements and are useful for tasks like removing duplicates
9 # from a list.
10 my_set = {1, 2, 3, 4, 5}
11
12 # Dictionaries are used for mapping keys to values and are helpful for representing
13 # structured data.
14 my_dict = {
15     'name': 'Alice',
16     'age': 30,
17     'city': 'New York'
18 }
```



Sintaxis y ortografía

Estructuras de datos: diccionarios anidados



example.py

```
1 hiberus_ecosystem = {
2     'location': {
3         'city': 'Zaragoza',
4         'postal_code': 50009,
5         'address': 'P. de Isabel la Católica, 6'
6     }
7     'workers': {'Alejandro G', 'Alejandro P', 'Raúl J'}
8 }
9
```



Sintaxis y ortografía

Operaciones con listas



example.py

```
1 # Creating a list
2 my_list = [1, 2, 3, 4, 5]
3
4 # Accessing elements by index
5 first_element = my_list[0]
6 # first_element = 1
7 last_element = my_list[-1]
8 # last_element = 5
9
10 # Slicing a list
11 subset = my_list[1:4]
12 # subset = [2, 3, 4]
13
14 # Modifying elements
15 my_list[2] = 99 # Modifies the third element
16 # my_list = [1, 2, 99, 4, 5]
17
18 # Append element to the end of the list
19 my_list.append(6)
20 # my_list = [1, 2, 99, 4, 5, 6]
21
22 # Extend list with other list
23 my_list.extend([7, 8, 9])
24 # my_list = [1, 2, 99, 4, 5, 6, 7, 8, 9]
25
26 # Remove element by value
27 my_list.remove(99)
28 # my_list = [1, 2, 4, 5, 6, 7, 8, 9]
29
30 # Remove element by list position
31 popped_element = my_list.pop(2)
32 # my_list = [1, 2, 5, 6, 7, 8, 9]
33
```



example.py

```
1 my_list = [1, 2, 3, 4, 5]
2
3 # Finding elements
4 element_index = my_list.index(5)

5 # element_index = 4
6
7 # Checking membership
8 is_3_present = 3 in my_list
9 # is_3_present = True
10 is_9_present = 9 in my_list
11 # is_9_present = False
12
13 # Sort list in ascending order
14 my_list.sort()
15 # my_list = [1, 2, 3, 4, 5]
16
17 # Sort list in reverse order
18 my_list.reverse()
19 # my_list = [5, 4, 3, 2, 1]
20
21 # Shallow copying a list
22 new_list = my_list.copy()
23
24 # Length of a list
25 list_length = len(my_list)
26 # list_length = 5
27
28 # Removes all elements
29 my_list.clear()
30 # my_list = []
31
```



Sintaxis y ortografía

Operaciones con tuplas



example.py

```
1 # Creating a tuple
2 fruits = ('apple', 'banana', 'cherry',
3           'date')
4
5 # Accessing elements by index
6 first_fruit = fruits[0]
7 # first_fruit = 'apple'
8 last_fruit = fruits[-1]
9 # last_fruit = 'date'
10
11 # Slicing a tuple
12 selected_fruits = fruits[1:3]
13 # selected_fruits = ('banana', 'cherry')
14
15 # Checking membership
16 is_melon_in_fruits = 'melon' in fruits
17 # is_melon_in_fruits = False
18
19 # Counting elements
20 count_cherry = fruits.count('cherry')
21 # count_cherry = 1
22
23 # Finding the index of an element
24 index_banana = fruits.index('banana')
25 # index_banana = 1
26
27 # Length of a tuple
28 num_fruits = len(fruits)
29 # num_fruits = 4
```



Sintaxis y ortografía

Operaciones con sets

```
example.py

1 # Creating a Set
2 fruits = {"apple", "banana", "cherry"}
3
4 # Adding Elements
5 fruits.add("orange")
6
7 # Removing Elements
8 fruits.remove("banana")
9
10 # Checking Membership
11 is_melon_in_fruits = "melon" in fruits
12 # is_melon_in_fruits = False
13
14 set1 = {1, 2, 3}
15 set2 = {3, 4, 5}
16
17 # Union of sets
18 union_set = set1.union(set2)
19 # union_set = {1, 2, 3, 4, 5}
20
21 # Intersection of sets
22 intersection_set = set1.intersection(set2)
23 # intersection_set = {3}
24
25 # Set difference
26 difference_set = set1.difference(set2)
27 # difference_set = {1, 2}
28
29 # Length of a Set
30 num_fruits = len(fruits)
31
```



Sintaxis y ortografía

Operaciones con diccionarios



example.py

```
1 # Dictionary Creation
2 person = {
3     "name": "Alex",
4     "age": 24,
5     "city": "Zaragoza"
6 }
7
8 # Accessing Elements
9 # Accessing information about Alex
10 print(f"{person['name']} is {person['age']} years old and lives in {person['city']}")
11 # Alex is 24 years old and lives in Zaragoza
12
13 # Modifying Elements
14 person["age"] = 25
15
16 # Adding new element
17 person["height"] = 184
18
19 # Checking Key Existence
20 if "height" in person:
21     print("Alex's height is:", person["height"])
22
23 # Deleting Elements
24 person.pop("age")
25
26 # Dictionary Length (number of elements in the dictionary)
27 num_elements = len(person)
28
```



CodeLab

Sintaxis y ortografía

Inicialización de lista, tupla, set y diccionario



example.py

```
1 # Empty initialization
2 empty_list = list()
3 empty_tuple = tuple()
4 empty_set = set()
5 empty_dictionary = dict()
6
```



example.py

```
1 # Regular initialization
2 my_list = [1, 2, 3]
3 my_tuple = (1, 2, 3)
4 my_set = {1, 2, 3}
5 my_dict = {1: False, 2: True, 3: False}
6
```



CodeLab

Sintaxis y ortografía

Conversiones de lista a otros tipos



example.py

```
1 # Transform a tuple into a list
2 my_list_from_tuple = list(my_tuple)
3 print(my_list_from_tuple)
4 # [1, 2, 3]
5
6 # Transform a set to list
7 my_list_from_set = list(my_set)
8 print(my_list_from_set)
9 # [1, 2, 3]
10
11 # Transform a dictionary into a list of keys and values
12 keys_list = list(my_dict.keys())
13 print(keys_list)
14 # [1, 2, 3]
15
16 values_list = list(my_dict.values())
17 print(values_list)
18 # [False, True, False]
19
```



Sintaxis y ortografía

Conversiones de tupla a otros tipos



example.py

```
1 # Transform a list into a tuple
2 my_tuple_from_list = tuple(my_list)
3 print(my_tuple_from_list)
4 # (1, 2, 3)
5
6 # Transform a set into a tuple
7 my_tuple_from_set = tuple(my_set)
8 print(my_tuple_from_set)
9 # (1, 2, 3)
10
11 # Transform a dictionary into a tuple of keys and values
12 keys_tuple = tuple(my_dict.keys())
13 print(keys_tuple)
14 # (1, 2, 3)
15
16 values_tuple = tuple(my_dict.values())
17 print(values_tuple)
18 #(False, True, False)
19
```



Sintaxis y ortografía

Conversiones de set a otros tipos



example.py

```
1 # Transform a list into a set
2 my_set_from_list = set(my_list)
3 print(my_set_from_list)
4 # {1, 2, 3}
5
6 # Transform a tuple into a set
7 my_set_from_set = set(my_tuple)
8 print(my_set_from_set)
9 # {1, 2, 3}
10
11 # Transform a dictionary into a set of keys and values
12 keys_set = set(my_dict.keys())
13 print(keys_set)
14 # {1, 2, 3}
15
16 values_tuple = set(my_dict.values())
17 print(values_tuple)
18 # {False, False}
19
```



Sintaxis y ortografía

Conversiones de diccionario a otros tipos



example.py

```
1 person = dict(name = "Alex", age = 24, city = "Zaragoza")
2 print(person)
3 # {'name': 'Alex', 'age': 24, 'city': 'Zaragoza'}
4
5 # Transform a list of lists into a dictionary
6 list_of_lists = [['name', 'Alex'], ['age', 24], ['city', 'Zaragoza']]
7 dict_from_list_of_lists = dict(list_of_lists)
8 print(dict_from_list_of_lists)
9 # {'name': 'Alex', 'age': 24, 'city': 'Zaragoza'}
10
11 # Transform a list of tuples into a dictionary
12 list_of_tuples = [('name', 'Alex'), ('age', 24), ('city', 'Zaragoza')]
13 dict_from_list_of_tuples = dict(list_of_tuples)
14 print(dict_from_list_of_tuples)
15 # {'name': 'Alex', 'age': 24, 'city': 'Zaragoza'}
16
17 # Transform a tuple of lists into a dictionary
18 tuple_of_lists = ([('name', 'Alex'), ('age', 24), ('city', 'Zaragoza')])
19 dict_from_tuple_of_lists = dict(tuple_of_lists)
20 print(dict_from_tuple_of_lists)
21 # {'name': 'Alex', 'age': 24, 'city': 'Zaragoza'}
22
23 # Transform a tuple of tuples into a dictionary
24 tuple_of_tuples = (('name', 'Alex'), ('age', 24), ('city', 'Zaragoza'))
25 dict_from_tuple_of_tuples = dict(tuple_of_tuples)
26 print(dict_from_tuple_of_tuples)
27 # {'name': 'Alex', 'age': 24, 'city': 'Zaragoza'}
28
```



¡Añadamos un histórico a la calculadora!



CodeLab

Sintaxis y ortografía: Comprensión de listas, tuplas, dict



CodeLab

Sintaxis y ortografía

Sintaxis de comprensiones



example.py

```
1 list = [value for element in iterable]
2 set = {value for element in iterable}
3 dictionary = {key: value for element in iterable}
4
```



Sintaxis y ortografía

Bucle vs Comprensión



example.py

```
1 # Traditionally
2 result = []
3 for character in 'text':
4     result.append(character)
5 print(result)
6 # ['t', 'e', 'x', 't']
7
8 # With list comprehension
9 result = [character for character in 'text']
10 print(result)
11 # ['t', 'e', 'x', 't']
12
```



Sintaxis y ortografía

Bucle vs Comprensión



example.py

```
1 # Traditionally
2 squared_numbers = []
3 for number in range(0,11):
4     squared_numbers.append(number**2)
5 print(squared_numbers)
6 # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
7
8 # With list comprehension
9 squared_numbers = [number**2 for number in range(0,11)]
10 print(squared_numbers)
11 # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
12
```



CodeLab

Sintaxis y ortografía

Bucle vs Comprensión



example.py

```
1 # Traditionally
2 even_numbers = []
3 for number in range(0,11):
4     if number % 2 == 0:
5         even_numbers.append(number)
6 print(even_numbers)
7 # [0, 2, 4, 6, 8, 10]
8
9 # With list comprehension
10 even_numbers = [number for number in range(0,11) if
11     number % 2 == 0 ]
12 print(even_numbers)
13 # [0, 2, 4, 6, 8, 10]
```



Sintaxis y ortografía

Bucle vs Comprepción



example.py

```
1 # Traditionally
2 squared_numbers = []
3 for number in range(0,11):
4     squared_numbers.append(number**2)
5
6 squared_even_numbers = []
7 for squared_number in squared_numbers:
8     if squared_number % 2 == 0:
9         squared_even_numbers.append(squared_number)
10 print(squared_even_numbers)
11 # [0, 4, 16, 36, 64, 100]
12
13 # With list comprehension
14 squared_numbers = [number**2 for number in range(0, 11)]
15 squared_even_numbers = [squared_number for squared_number in squared_numbers if squared_number % 2 == 0]
16 print(squared_even_numbers)
17 # [0, 4, 16, 36, 64, 100]
18
```



CodeLab

Sintaxis y ortografía

Compreensión de listas



example.py

```
1 # With list comprehension
2 squared_even_numbers = [number**2 for number in range(0, 11) if number**2 % 2 == 0]
3 print(squared_even_numbers)
4 # [0, 4, 16, 36, 64, 100]
5
```



CodeLab

Sintaxis y ortografía

Bucle vs Comprensión



example.py

```
1 module_of_7 = set()
2 for number in [7, 10, 11, 14]:
3     module_of_7.add(number % 7)
4 print(module_of_7)
5 # {0, 3, 4}
6
7 # Set comprehension
8 module_of_7 = {number % 7 for number in [7, 10, 11, 14]}
9 print(module_of_7)
10 # {0, 3, 4}
11
```



Sintaxis y ortografía

Bucle vs Comprepción



example.py

```
1 # Traditionally
2 is_even_by_number = {}
3 for number in range(0,6):
4     is_even_by_number[number] = number % 2 == 0
5 print(is_even_by_number)
6 # {0: True, 1: False, 2: True, 3: False, 4: True, 5: False}
7
8 # With list comprehension
9 is_even_by_number = {number: number % 2 == 0 for number in range(0,6) }
10 print(is_even_by_number)
11 # {0: True, 1: False, 2: True, 3: False, 4: True, 5: False}
12
```



CodeLab

Sintaxis y ortografía

Bucle vs Comprensión



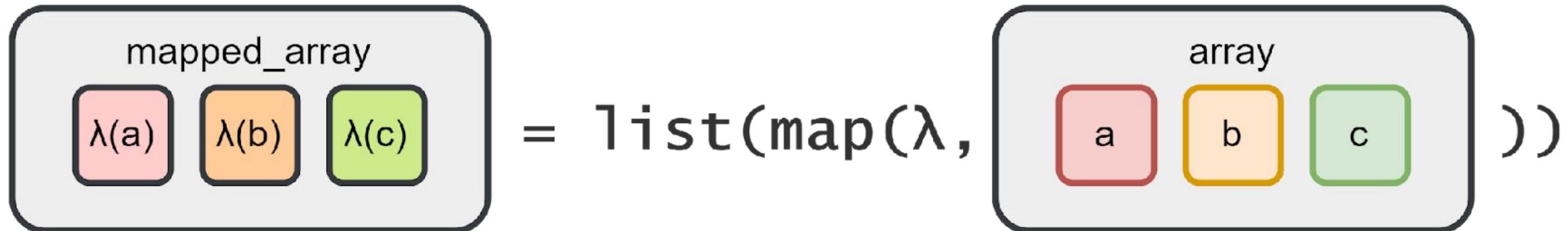
Sintaxis y ortografía: Map y Filter



CodeLab

Sintaxis y ortografía

Map



Sintaxis y ortografía

Map



example.py

```
1 numbers = [1, 2, 3, 4, 5]
2 squared = list(map(lambda x: x**2, numbers))
3 # squared = [1, 4, 9, 16, 25]
```



Sintaxis y ortografía

Map



example.py

```
1 numbers_1 = [1, 7, 5, 9, 6]
2 numbers_2 = [2, 3, 8, 4, 10]
3
4 # Lambda function that takes two arguments
5 find_max = lambda x, y: x if x > y else y
6
7 # map() with a 2 parameter lambda function process 2 lists
8 max_result = list(map(find_max, numbers_1, numbers_2))
9 print(max_result)
10 # [2, 7, 8, 9, 10]
11
```



Sintaxis y ortografía

Map



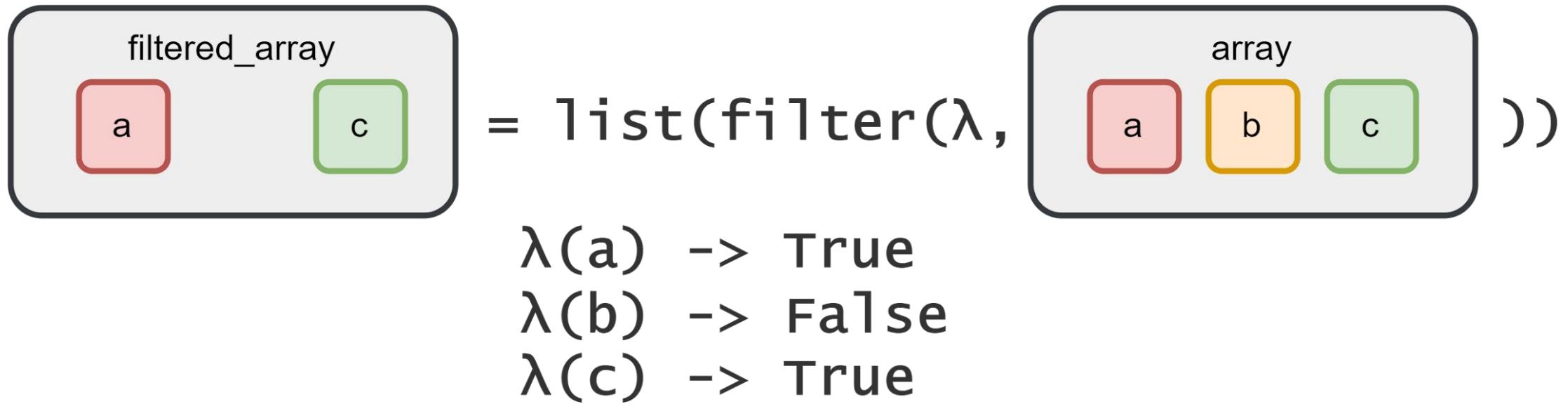
example.py

```
1 numbers = [7, 10, 11, 14]
2 module_of_7 = set(map(lambda x: x % 7, numbers))
3 # module_of_7 = {0, 3, 4}
4
```



Sintaxis y ortografía

Filter



CodeLab

Sintaxis y ortografía

Filter



example.py

```
1 numbers = [-3, -1, 0, 2, 5, -8, 9]
2 positive_numbers = list(filter(lambda x: x > 0, numbers))
3 # positive_numbers = [2, 5, 9]
4
```



CodeLab

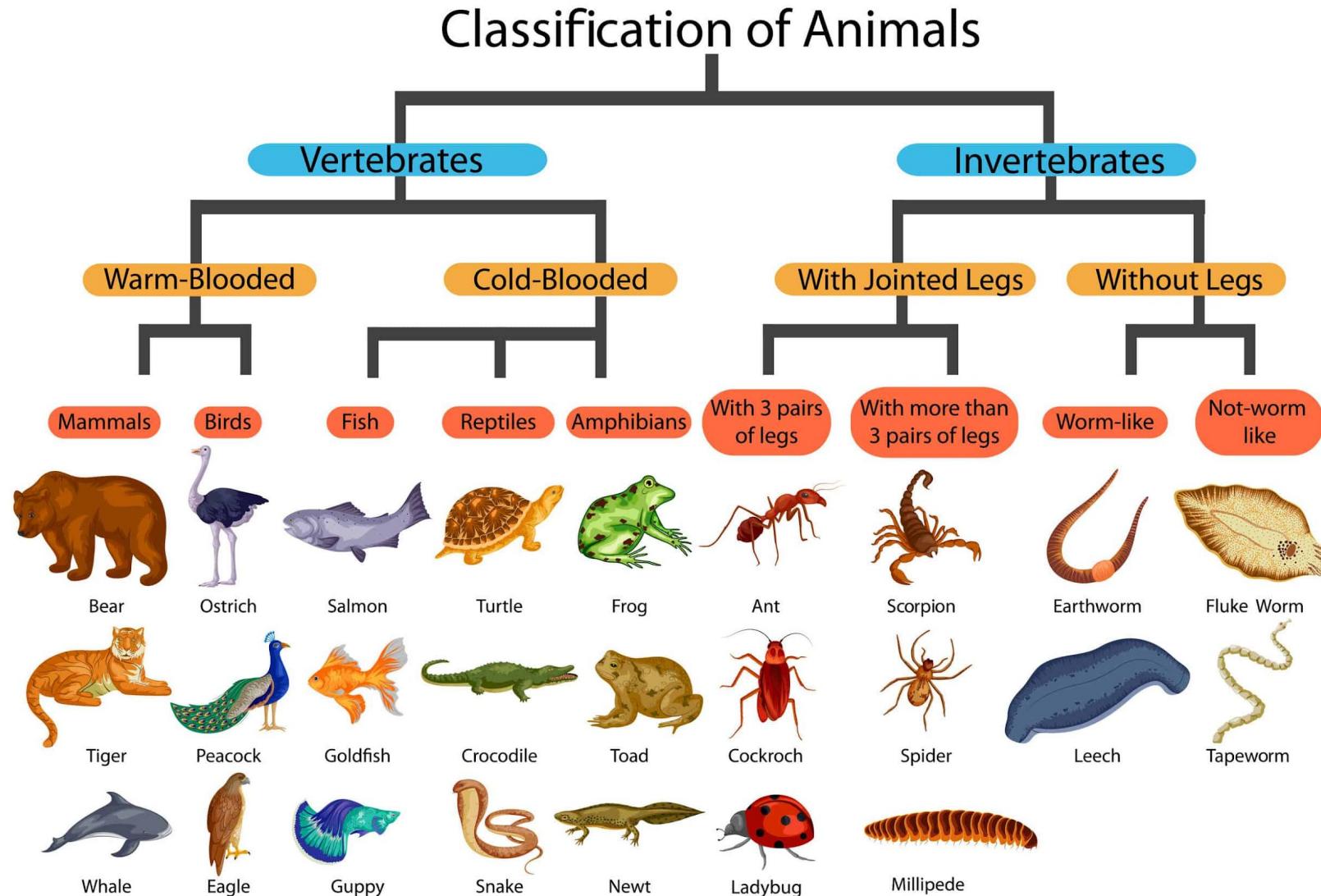
Sintaxis y ortografía: Herencia de clases



CodeLab

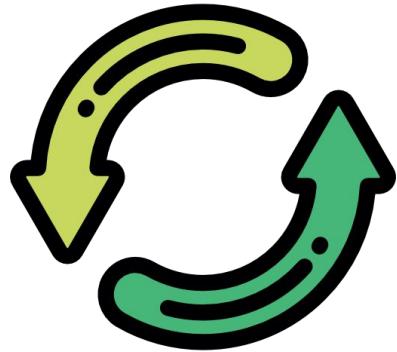
Sintaxis y ortografía

Definición herencia de clases

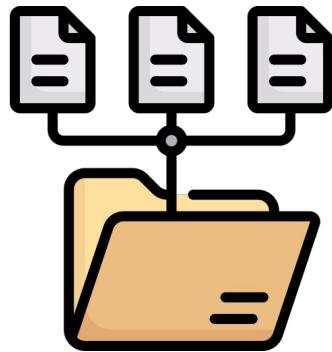


Sintaxis y ortografía

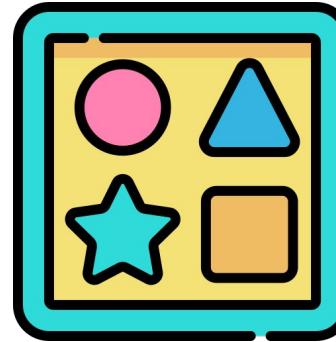
Utilidad herencia de clases



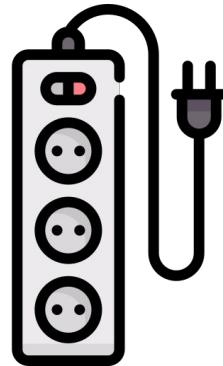
Reutilizar
código



Estructuración



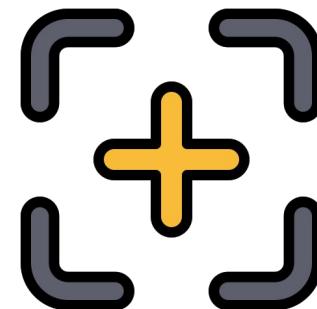
Polimorfismo



Extensibilidad



Abstracción



Especialización

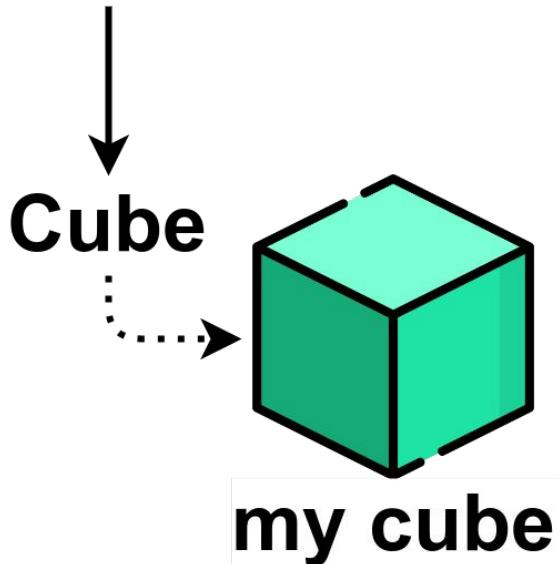


CodeLab

Sintaxis y ortografía

Herencia simple

MaterialObject



example.py

```
1 class MaterialObject:
2     def __init__(self, volume, mass):
3         self.volume = volume
4         self.mass = mass
5
6     def calculate_volume(self):
7         return self.mass / self.volume
8
9 class Cube(MaterialObject):
10
11     def __init__(self, side, mass):
12         self.side = side
13         return super().__init__(side**3, mass)
14
15     def calculate_area(self):
16         return self.side**2 * 6
17
18 my_cube = Cube(2, 10)
19 print(f"My cube volume is: {my_cube.calculate_volume()}")
20 # My cube volume is: 1.25
21 print(f"My cube area is: {my_cube.calculate_area()}")
22 # My cube area is: 24
23
```



CodeLab

Sintaxis y ortografía

Herencia múltiple

Swimmer **Walker**

Platypus



Perry



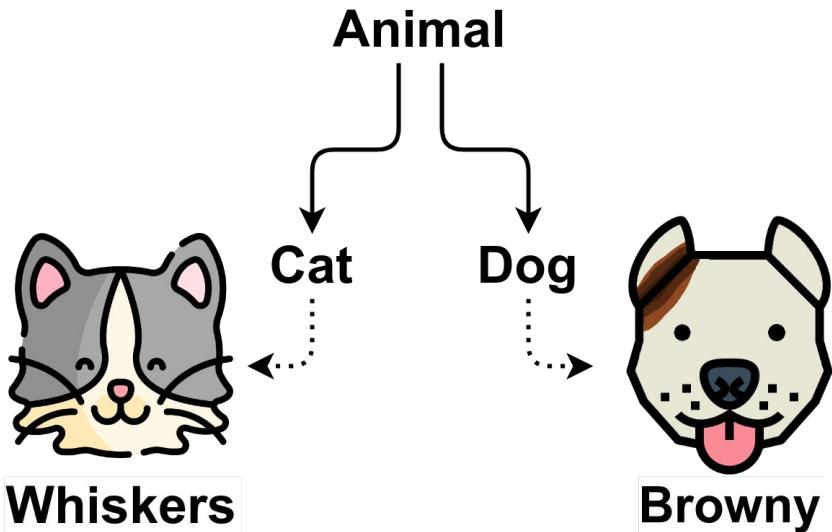
example.py

```
1 # Base class 1
2 class Swimmer:
3     def swim(self):
4         return "Swimming gracefully."
5
6 # Base class 2
7 class Walker:
8     def walk(self):
9         return "Walking on land."
10
11 # Derived class inheriting from Swimmer and Walker
12 class Platypus(Swimmer, Walker):
13     def __init__(self, name):
14         self.name = name
15
16     def description(self):
17         return f"A platypus? {self.name} the platypus!"
18
19 # Create an instance of the Platypus
20 perry = Platypus("Perry")
21
22 # Use methods from both base classes
23 print(perry.swim())
24 # Swimming gracefully.
25 print(perry.walk())
26 # Walking on land.
27
28 # Use the method of the derived class
29 print(perry.description())
30 # A platypus? Perry the platypus!
31
```



Sintaxis y ortografía

Sobreescritura de métodos mediante herencia



```
example.py
```

```
1 # Base class
2 class Animal:
3     def __init__(self, name):
4         self.name = name
5
6     def speak(self):
7         return f"{self.name} can speak!"
8
9 # Derived classes with method overriding
10 class Dog(Animal):
11     def speak(self):
12         return f"{self.name} the dog says Woof!"
13
14 class Cat(Animal):
15     def speak(self):
16         return f"{self.name} the cat says Meow!"
17
18
19 # Creating instances of derived classes
20 dog = Dog("Browny")
21 cat = Cat("Whiskers")
22
23 # Using method overridden in derived classes
24 print(dog.speak())
25 # Browny the dog says Woof!
26 print(cat.speak())
27 # Whiskers the cat says Meow!
28
```



Sintaxis y ortografía

Ejemplo herencia

Animal



Pet



Dog



example.py

```
1 # Base class
2 class Animal:
3     def __init__(self, name):
4         self.name = name
5
6     def speak(self):
7         return f"{self.name} can speak!"
8
9 # Intermediate class with shared behavior
10 class Pet(Animal):
11     def __init__(self, name, owner):
12         super().__init__(name)
13         self.owner = owner
14
15     def introduce(self):
16         return f"{self.name} is owned by {self.owner}."
17
18 # Derived classes with method overriding and new attributes
19 class Dog(Pet):
20     def __init__(self, name, owner, breed):
21         super().__init__(name, owner)
22         self.breed = breed
23
24     def speak(self):
25         return f"{self.name} the dog ({self.breed}) says Woof!"
26
27 # Creating instances of derived classes with new attributes
28 dog = Dog("Browny", "Carlos", "Pitbull")
29
30 # Using methods and accessing new attributes
31 print(dog.introduce())
32 # Browny is owned by Carlos
33 print(dog.speak())
34 # Browny the dog (Pitbull) says Woof!
35
```



Sintaxis y ortografía

Clases abstractas



example.py

```
1 from abc import ABC, abstractmethod
2
3 # Abstract class
4 class Animal(ABC):
5     def __init__(self, name, species):
6         self.name = name
7         self.species = species
8
9     @abstractmethod
10    def speak(self):
11        pass
12
13 class Dog(Animal):
14
15    def __init__(self, name):
16        super().__init__(name, "Dog")
17
18    def speak(self):
19        return "Woof!"
20
21 # The next line will give a TypeError (Can't instantiate abstract class)
22 abstract_animal = Animal("Name", "Species")
23 # Creating instance of the Dog class
24 browny_boy = Dog("Browny")
```



Sintaxis y ortografía: Decoradores



CodeLab

Sintaxis y ortografía

Decoradores



decorators.py

```
1 def my_decorator(func):  
2     def wrapper_my_decorator(*args, **kwargs):  
3         # Here goes whatever you want  
4         print("You are using a decorator")  
5         return func(*args, **kwargs)  
6     return wrapper_my_decorator  
7  
8 @my_decorator  
9 def hello_world():  
10    print("Hello world!")  
11  
12 hello_world()  
13 # You are using a decorator  
14 # Hello world!  
15
```



CodeLab

Sintaxis y ortografía

Crisis de identidad con decoradores



decorators.py

```
1 name = hello_world.__name__
2 # name = wrapper_my_decorator
3
4 help(hello_world)
5 # Help on function wrapper_my_decorator in module decorators:
6 # wrapper_my_decorator(*args, **kwargs)
7
```



CodeLab

Sintaxis y ortografía

Resolver crisis de identidad con decoradores



A screenshot of a terminal window titled "decorators.py". The window has three colored window control buttons (red, yellow, green) in the top-left corner. The code inside the terminal is as follows:

```
decorators.py

1 import functools
2
3
4 def my_decorator(func):
5     @functools.wraps(func)
6     def wrapper_my_decorator(*args, **kwargs):
7         # Here goes whatever you want
8         print("You are using a decorator")
9         return func(*args, **kwargs)
10
11     return wrapper_my_decorator
12
13
14 @my_decorator
15 def hello_world():
16     print("Hello world!")
17
18
19 hello_world()
20 # You are using a decorator
21 # Hello world!
22
23 name = hello_world.__name__
24 # name = hello_world
25
26 help(hello_world)
27 # Help on function hello_world in module decorators:
28 # hello_world()
29
```



Sintaxis y ortografía

Múltiples decoradores

```
● ● ● decorators.py

1 import functools
2
3
4 def decorator_a(func):
5     @functools.wraps(func)
6     def inner_a():
7         print("Decorator a called")
8         return "Decorator a: " + func()
9     return inner_a
10
11 def decorator_b(func):
12     @functools.wraps(func)
13     def inner_b():
14         print("Decorator b called")
15         return "Decorator b: " + func()
16     return inner_b
17
18 @decorator_a
19 @decorator_b
20 def hello_world_with_decorators():
21     return "Hello world!"
22
23
24 result = hello_world_with_decorators()
25 ## Decorator a called
26 ## Decorator b called
27 # result = Decorator a called: Decorator b called: Hello world!
# = "Decorator a: " + ("Decorator b: " + ("Hello world!"))
28
```



Sintaxis y ortografía

Decoradores con parámetros



decorators.py

```
1 import functools
2
3 def multiply_by(x):
4     def inner_multiply_by(func):
5         @functools.wraps(func)
6         def wrapper_multiply_by(*args, **kwargs):
7             return x * func(*args, **kwargs)
8             return wrapper_multiply_by
9     return inner_multiply_by
10
11 @multiply_by(3)
12 def give_me_a_number():
13     return 2
14
15 result = give_me_a_number()
16 # result = 6
17
```



CodeLab

Sintaxis y ortografía

Ejemplo práctico con decoradores



decorators.py

```
1 import functools
2 import time
3
4
5 # Decorator to measure execution time
6 def calculate_execution_time(func):
7     @functools.wraps(func)
8     def wrapper(*args, **kwargs):
9         # Record the start time
10        start_time = time.time()
11        # Execute the original function
12        result = func(*args, **kwargs)
13        # Record the end time
14        end_time = time.time()
15        # Calculate the execution time
16        execution_time = end_time - start_time
17        print(f"The function '{func.__name__}' took {execution_time:.4f} seconds to execute.")
18        return result
19
20    return wrapper
21
```



main.py

```
1 import math
2
3 from decorators import calculate_execution_time
4
5 # Gets n first prime numbers
6 @calculate_execution_time
7 def get_prime_numbers(n: int):
8     prime_numbers = [2, 3]
9
10    def evaluate_number(number):
11        for prime_number in prime_numbers:
12            if prime_number * 3 > number:
13                prime_numbers.append(number)
14                break
15            if number % prime_number == 0:
16                break
17
18    def uneven_numbers(min_uneven=0, max_uneven=math.inf):
19        if min_uneven % 2 == 0:
20            min_uneven += 1
21
22        i = min_uneven
23        while i < max_uneven:
24            yield i
25            i += 2
26
27    for i in uneven_numbers(5):
28        evaluate_number(i)
29        if len(prime_numbers) >= n:
30            break
31    return prime_numbers
32
33
34 get_prime_numbers(10000)
35 # The function 'get_prime_numbers' took 0.8821 seconds to execute.
36
```



CodeLab

Sintaxis y ortografía: Gestores de contexto



CodeLab

Sintaxis y ortografía

Gestores de contexto



example.py

```
1 file = open('some_file', 'r')
2 text_in_file = file.read()
3 print(text_in_file)
4 file.close()
5
```



example.py

```
1 with open('some_file', 'r') as file:
2     text_in_file = file.read()
3     print(text_in_file)
4
```



CodeLab

Sintaxis y ortografía

Modos de abrir un fichero

Modo	Descripción
'r'	Modo de lectura (read): Abre el archivo para lectura. Si el archivo no existe, se genera un error.
'w'	Modo de escritura (write): Abre el archivo para escritura. Si el archivo ya existe, trunca su contenido; si no existe, crea uno nuevo.
'a'	Modo de agregado (append): Abre el archivo para escritura, pero agrega datos al final sin truncar. Si el archivo no existe, se crea uno nuevo.
'r+'	Modo de lectura y escritura (read-write): Abre el archivo para lectura y escritura. Si el archivo no existe, se genera un error.
'w+'	Modo de escritura y lectura (write-read): Abre el archivo para escritura y lectura. Si el archivo ya existe, trunca su contenido. Si no existe, crea uno nuevo.
'a+'	Modo de agregado y lectura (append-read): Abre el archivo para lectura y escritura. Si el archivo no existe, se crea uno nuevo. Permite agregar datos al final.
'x'	Modo de creación exclusiva (exclusive creation): Abre el archivo para escritura, generando un error si ya existe. Útil para crear nuevos archivos de forma segura.
'rb'	Modo binario de lectura (binary read): Abre el archivo en modo binario para lectura. Utilizado para leer archivos binarios como imágenes.
'wb'	Modo binario de escritura (binary write): Abre el archivo en modo binario para escritura. Utilizado para escribir archivos binarios.



Sintaxis y ortografía

Concurrencia con ThreadPool



example.py

```
1 from concurrent.futures import ThreadPoolExecutor, as_completed
2
3 CONCURRENCY = 4
4
5
6 def calculate_square(number):
7     return number ** 2
8
9
10 tasks = []
11 with ThreadPoolExecutor(CONCURRENCY) as executor:
12     for number in range(10):
13         tasks.append(executor.submit(calculate_square, number))
14     for task in as_completed(tasks):
15         result = task.result()
16         print(result)
```



CodeLab

Sintaxis y ortografía

Gestores de contexto



example.py

```
1 # Class definition:  
2 class DatabaseConnection(object):  
3     def __init__(self, db_name):  
4         self.db_name = db_name  
5  
6     def __enter__(self):  
7         self.db_resource = open_db_resource(self.db_name)  
8  
9         return self.db_resource  
10  
11    def __exit__(self, *args):  
12        self.db_resource.close()  
13  
14  
15  
16 # Use example:  
17 with DatabaseConnection('my_database') as db_connection:  
18     db_connection.save_entity(my_entity)  
19
```



Sintaxis y ortografía: Control de excepciones



CodeLab

Sintaxis y ortografía

Sintaxis control de excepciones



example.py

```
1 try:
2     # Try to execute code that may raise an exception
3     number = int(input("Enter a number: "))
4     result = 10 / number
5     print("The result is:", result)
6
7 except ZeroDivisionError:
8     # Catch the ZeroDivisionError exception if it occurs
9     print("Error: You can't divide by zero.")
10
11 except ValueError as e:
12     # Catch the ValueError exception if it occurs, and display the error message
13     print(f"Error: {e}")
14
15 except Exception as e:
16     # Catch any other unhandled exceptions
17     print(f"Unexpected error: {e}")
18
19 else:
20     # This block executes if no exception is raised
21     print("Operation successful.")
22
23 finally:
24     # This block always executes, regardless of whether there was an exception
25     print("End of the program.")
26
```



Sintaxis y ortografía

Tipos de excepciones

BaseException

- BaseExceptionGroup
- GeneratorExit
- KeyboardInterrupt
- SystemExit
- Exception
 - ArithmetError
 - FloatingPointError
 - OverflowError
 - ZeroDivisionError
- AssertionError
- AttributeError
- BufferError
- EOFError
- ExceptionGroup [BaseExceptionGroup]
- ImportError
 - ModuleNotFoundError
- LookupError
 - IndexError
 - KeyError
- MemoryError
- NameError
 - UnboundLocalError
- .
- .
- .

BaseException

- Exception
 - OSError
 - BlockingIOError
 - ChildProcessError
 - ConnectionError
 - BrokenPipeError
 - ConnectionAbortedError
 - ConnectionRefusedError
 - ConnectionResetError
 - FileExistsError
 - FileNotFoundException
 - InterruptedError
 - IsADirectoryError
 - NotADirectoryError
 - PermissionError
 - ProcessLookupError
 - TimeoutError
 - ReferenceError
 - RuntimeError
 - NotImplementedError
 - RecursionError
 - StopAsyncIteration
 - StopIteration
 - SyntaxError
 - IndentationError
 - TabError

BaseException

- Exception
 - SystemError
 - TypeError
 - ValueError
 - UnicodeError
 - UnicodeDecodeError
 - UnicodeEncodeError
 - UnicodeTranslateError
 - Warning
 - BytesWarning
 - DeprecationWarning
 - EncodingWarning
 - FutureWarning
 - ImportWarning
 - PendingDeprecationWarning
 - ResourceWarning
 - RuntimeWarning
 - SyntaxWarning
 - UnicodeWarning
 - UserWarning



Sintaxis y ortografía

Ejemplo de control de excepciones



example.py

```
1 # Custom exception class for Shrek-related errors
2 class ShrekException(Exception):
3     def __init__(self, message="Oops! Something Shrektastic happened."):
4         super().__init__(message)
5
6 # Function that may raise the custom ShrekException
7 def is_it_swampy():
8     swamp_decision = input("Is it swampy? (yes/no): ").lower().strip()
9     if swamp_decision != "yes":
10         raise ShrekException("This is not a swamp! Shrek would not approve.")
11
12 def shrek_color():
13     what_colour = input("What's your favourite colour?: ").lower().strip()
14     if what_colour != "green":
15         raise ShrekException("Why not green! Shrek would not approve.")
16
17 try:
18     is_it_swampy()
19     shrek_color()
20     print("If you can see this, shrek is happy and no exception was raised")
21 except ShrekException as e:
22     print(f"Shrek Exception: {e}")
23
```



Importación de código: Módulos, paquetes e importaciones



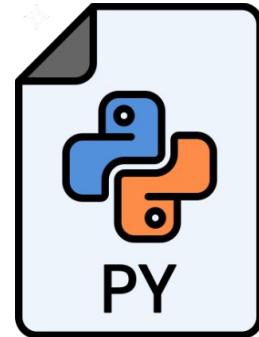
CodeLab

Importación de código

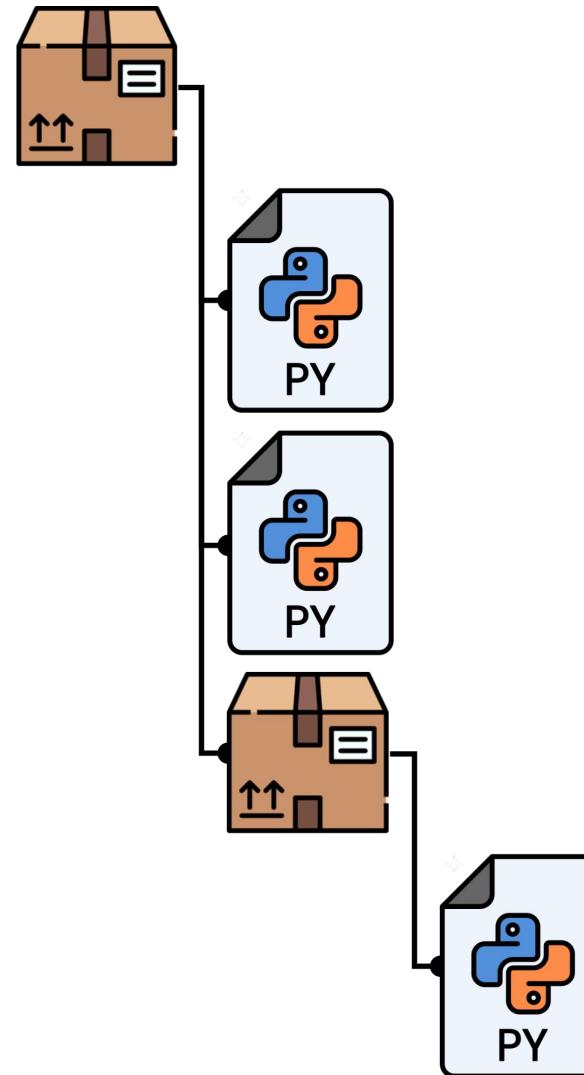
Módulos y paquetes



package



module.py



Organización de paquetes y modulos



CodeLab

Importación de código

Como importar código



example.py

```
1 import module
2 import package.module
3 import package.subpackage.module
4 import module_1, module_2
5 import module as custom_name
6
7 from module import object
8 from package.module import object
9 from module import object_1, object_2
10
```



CodeLab

Importación de código

Módulos y paquetes

```
animals/
  • animal.py
  • dog.py
  • pet.py
main.py
```



animal.py

```
1 class Animal:
2     def __init__(self, name):
3         self.name = name
4
5     def speak(self):
6         pass
7
```



Importación de código

Módulos y paquetes

```
animals/
  • animal.py
  • dog.py
  • pet.py
main.py
```



pet.py

```
1 from .animal import Animal
2
3
4 class Pet(Animal):
5     def __init__(self, name,
6                  owner):
7         super().__init__(name)
8         self.owner = owner
9
10    def introduce(self):
11        return f"{self.name}"
12        is owned by {self.owner}."
```



Importación de código

Módulos y paquetes

```
animals/
  • animal.py
  • dog.py
  • pet.py
main.py
```



dog.py

```
1 from .pet import Pet
2
3
4 class Dog(Pet):
5     def __init__(self, name,
6                  owner, breed):
7         super().__init__(name,
8                           owner)
9         self.breed = breed
10
11    def speak(self):
12        return f"{self.name}
13        the dog ({self.breed}) says
14        Woof!"
```

11



CodeLab

Importación de código

Módulos y paquetes

```
animals/
  • animal.py
  • dog.py
  • pet.py
main.py
```

```
main.py

1 from animals.dog import Dog
2
3
4 # Creating instances of derived classes with new attributes
5 dog = Dog("Browny", "Carlos", "Pitbull")
6
7 # Using methods and accessing new attributes
8 print(dog.introduce())
9 # Browny is owned by Carlos.
10 print(dog.speak())
11 # Browny the dog (Pitbull) says Woof!
12
```



Importación de código: Librerías externas



CodeLab

Importación de código

Python Package Index (PyPI)



Python Package Index (PyPI)



CodeLab

Importación de código

Instalación de dependencias



The screenshot shows the Python Package Index (PyPI) homepage. The header features a blue navigation bar with a white icon of stacked cubes on the left, and links for Help, Sponsors, Log in, and Register on the right. The main content area has a dark blue background with white text. It displays the tagline: "Find, install and publish Python packages with the Python Package Index". Below this is a search bar with the placeholder "Search projects" and a magnifying glass icon. At the bottom, there is a link "Or [browse projects](#)".

Help Sponsors Log in Register

Find, install and publish Python packages with the Python Package Index

Search projects

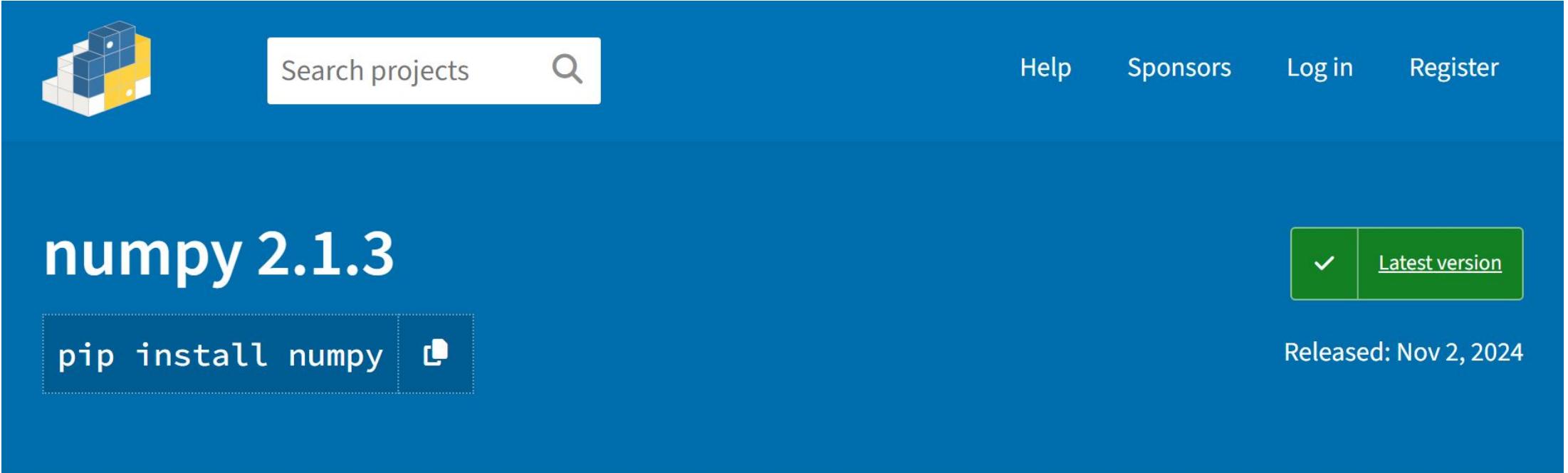
Or [browse projects](#)



CodeLab

Importación de código

Instalación de dependencias



The screenshot shows the PyPI search results for the term "numpy". The top navigation bar includes a logo, a search bar with the placeholder "Search projects", and links for Help, Sponsors, Log in, and Register. The search results list "numpy 2.1.3" as the top result. The numpy page features a large image of a 3D cube composed of smaller blocks, a "Latest version" button with a checkmark, and a "pip install numpy" button with a clipboard icon. The page also displays the release date "Released: Nov 2, 2024".

Search projects

Help Sponsors Log in Register

numpy 2.1.3

pip install numpy

✓ [Latest version](#)

Released: Nov 2, 2024

Fundamental package for array computing in Python



CodeLab

Importación de código

Instalación de dependencias



terminal

```
pip install numpy
```



example.py

```
1 import numpy as np
2
3 a = np.array([[1, 2, 3], [4, 5, 6]])
```



CodeLab

Librerías externas

Gestionar dependencias – requirements.txt

```
● ● ● requirements.txt

1 # Framework
2 fastapi==0.115.0
3 uvicorn==0.30.6
4
5 # Database connection
6 pymongo==4.9.1
7
8 # Operation
9 numpy==2.1.1
10
11 # Testing
12 pytest==8.3.3
13
```

Para instalar las dependencias:

```
● ● ● terminal
```

```
pip install -r requirements.txt
```



Importación de código

Gestionar dependencias – pipenv

Para instalar las dependencias:



terminal

```
pipenv install fastapi==0.115.0
```



Pipfile

```
1 [[source]]
2 url = "https://pypi.org/simple"
3 verify_ssl = true
4 name = "pypi"
5
6 [packages]
7 fastapi = "==0.115.0"
8 uvicorn = "==0.30.6"
9 pymongo = "==4.9.1"
10 numpy = "==2.1.1"
11
12 [dev-packages]
13 pytest = "==8.3.3"
14
15 [requires]
16 python_version = "3.13"
17 python_full_version = "3.13.0"
18
```



CodeLab

Entresijos: Referencia o valor



CodeLab

Sintaxis y ortografía

Los argumentos por referencia o por valor

pass by reference

cup =



fillCup()

pass by value

cup =



fillCup()



Sintaxis y ortografía

Los argumentos por referencia o por valor



example.py

```
1 def modify_number(num):  
2     num += 10  
3     return num  
4  
5 x = 5  
6 result = modify_number(x)  
7 # x = 5 (unchanged)  
8 # result = 15 (modified and returned inside the function)  
9
```



Sintaxis y ortografía

Los argumentos por referencia o por valor



example.py

```
1 def modify_list(my_list):  
2     my_list.append(4)  
3     return my_list  
4  
5 my_list = [1, 2, 3]  
6 result = modify_list(my_list)  
7 # my_list = [1, 2, 3, 4] (modified inside the function)  
8 # result = [1, 2, 3, 4] (same list)  
9
```



CodeLab

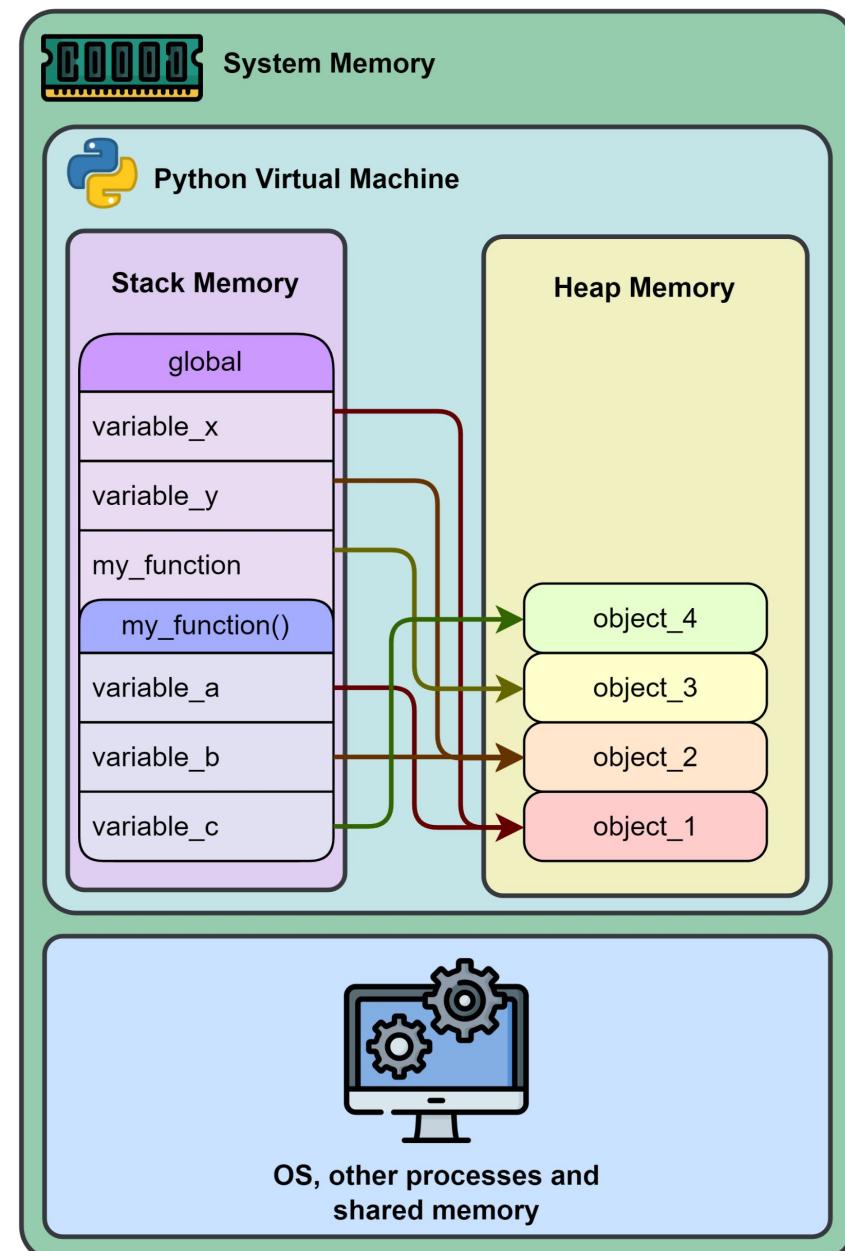
Sintaxis y ortografía

Los argumentos por referencia o por valor

No es lo mismo reasignar que modificar...

```
example.py

1 # Reassign
2 variable_a = "red"
3
4 variable_a = "blue"
5
6
7 # Modify
8 variable_b = [ "eggs" ]
9
10 variable_b.append( "potatoes" )
```



Entresijos: Funcionamiento de los import



CodeLab

Entresijos

Fucionamiento de import



module_a.py

```
1 import module_b
2
3
4 def function_a( ):
5     # Do some stuff...
6     print("function_a called")
7     module_b.function_b( )
8
9
10 function_a( )
11
```



module_b.py

```
1 def function_b( ):
2     # Do some stuff...
3     print("function_b called")
4
5
6 function_b( )
7
```



terminal

```
$ python module_a.py
function_b called
function_a called
function_b called
```



CodeLab

Entresijos

Fucionamiento de import



module_a.py

```
1 from module_b import function_b
2
3
4 def function_a():
5     # Do some stuff...
6     print("function_a called")
7     function_b()
8
9
10 function_a()
11
```



module_b.py

```
1 def function_b():
2     # Do some stuff...
3     print("function_b called")
4
5
6 function_b()
7
```



terminal

```
$ python module_a.py
function_b called
function_a called
function_b called
```



CodeLab

Entresijos

```
if __name__ == '__main__':
```



module_a.py

```
1 import module_b
2
3
4 def function_a():
5     # Do some stuff
6     print("function_a called")
7     module_b.function_b()
8
9
10 if __name__ == '__main__':
11     function_a()
12
```



module_b.py

```
1 def function_b():
2     # Do some stuff
3     print("function_b called")
4
5
6 if __name__ == '__main__':
7     function_b()
8
```



terminal

```
$ python module_a.py
function_a called
function_b called
```



CodeLab

¡Muchas gracias!



CodeLab