

{desafío}
latam_

API_



Diccionarios

Diferencias entre listas y diccionarios

Lista	Diccionario
<ul style="list-style-type: none">● Se accede a los elementos por medio de la posición o índice.● Los índices se definen implícitamente.● Los índices siempre son int.	<ul style="list-style-type: none">● Se accede a los elementos por medio de la clave.● Las claves se definen explícitamente.● Las claves suelen ser string.

```
lista = [25, 31, "hola"]  
lista[2] # "hola"  
  
diccionario = {"a": 25, "b": 31, "c": "hola"}  
diccionario["c"] # "hola"
```

Desafíos

Iteración, transformación y filtrado

Se tiene la siguiente lista de productos:

`diccionario_productos = {"celular": 140000, "notebook": 489990, "tablet": 120000, "cargador": 12400}`

Se solicita:

- Si el producto tiene un valor menor a 120.000, se le debe aplicar un 10% de descuento (en el diccionario original).
- En caso contrario, se debe asignar a un nuevo diccionario de filtrado.

```
1  diccionario_productos = {
2      "celular": 140000,
3      "notebook": 489990,
4      "tablet": 120000,
5      "cargador": 12400
6  }
7
8  # Se crea diccionario vacío para el filtrado
9  diccionario_caros = {}
10
11 # Se itera por clave y valor utilizando items()
12 for key, value in diccionario_productos.items():
13
14     if value < 120000:
15         # Caso 1: Transformación
16         diccionario_productos[key] = int(value * 0.9)
17     else:
18         # Caso 2: Filtrado
19         diccionario_caros[key] = value
20
21 print(diccionario_productos)
22 print(diccionario_caros)
```

Desafío

Búsqueda de colores

(versión 1)

```

1  import sys
2
3  colores = {
4      "aliceblue": "#f0f8ff",
5      "antiquewhite": "#faebd7",
6      "aqua": "#00ffff",
7      "aquamarine": "#7fffd4",
8      "azure": "#f0ffff",
9      "darkorchid": "#9932cc",
10     "darkred": "#8b0000",
11     "darksalmon": "#e9967a",
12     "navajowhite": "#ffdead",
13     "navy": "#000080",
14     "orchid": "#da70d6"
15 }
16
17 # Opción del usuario
18 search = sys.argv[1]
19
20 # "Flag" que cambia a True si se encuentra el color
21 found = False
22
23 # Iteramos por clave (nombre) y valor (hexadecimal)
24 for name, hexa in colors.items():
25     if hexa == search and found == False:
26         # Si se encuentra, se imprime y se cambia el flag a True
27         found = True
28         print(name)
29
30 # Si flag permanece falso, se muestra mensaje de no encontrado
31 if not found:
32     print("no-no")
33

```


Desafío

Búsqueda de colores

(versión 2)

```
1 import sys
2
3 colores = {
4     "aliceblue": "#f0f8ff",
5     "antiquewhite": "#faebd7",
6     "aqua": "#00ffff",
7     "aquamarine": "#7fffd4",
8     "azure": "#f0ffff",
9     "darkorchid": "#9932cc",
10    "darkred": "#8b0000",
11    "darksalmon": "#e9967a",
12    "navajowhite": "#ffdead",
13    "navy": "#000080",
14    "orchid": "#da70d6"
15 }
16
17 # Opción del usuario
18 search = sys.argv[1]
19
20 # Se invierte el diccionario
21 colors_inv = {v: k for k, v in colors.items()}
22
23 # Se consulta si el hexadecimal ingresado se encuentra en las claves
24 # del diccionario invertido. Si no es así, muestra el mensaje.
25 if search in colors_inv:
26     print(colors_inv[search])
27 else:
28     print("no-no")
29
```

Desafío

Búsqueda de colores

(versión 3)

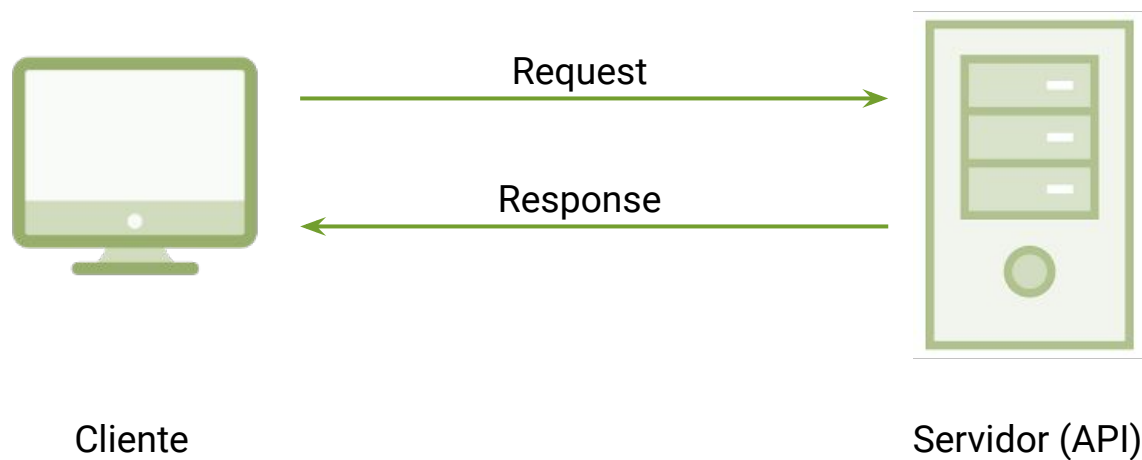
```
1  import sys
2
3  colores = {
4      "aliceblue": "#f0f8ff",
5      "antiquewhite": "#faebd7",
6      "aqua": "#00ffff",
7      "aquamarine": "#7fffd4",
8      "azure": "#f0ffff",
9      "darkorchid": "#9932cc",
10     "darkred": "#8b0000",
11     "darksalmon": "#e9967a",
12     "navajowhite": "#ffdead",
13     "navy": "#000080",
14     "orchid": "#da70d6"
15 }
16
17 # Opciones del usuario
18 searched_hexas = sys.argv[1:]
19
20 # Se invierte el diccionario
21 colors_inv = {v: k for k, v in colores.items()}
22
23 # Se itera por la lista de valores ingresados
24 for search in searched_hexas:
25     # Se consulta por cada opción si existe esa clave en el diccionario invertido
26     if search in colors_inv:
27         print(colors_inv[search])
28     else:
29         print("no-no")
30
```

APIs

Tipos

- Clima y temperatura
- Cambio de monedas
- Indicadores económicos
- Servicios para subir archivos
- Compra y venta de criptomonedas
- Servicios de geolocalización, como Google Maps

Funcionamiento





POSTMAN

Product ▾

Plans & Pricing ▾

Learning Center

Support

SIGN IN

Postman Simplifies API Development.

Get easy, API-First solutions with the industry's
only complete API Development Environment.

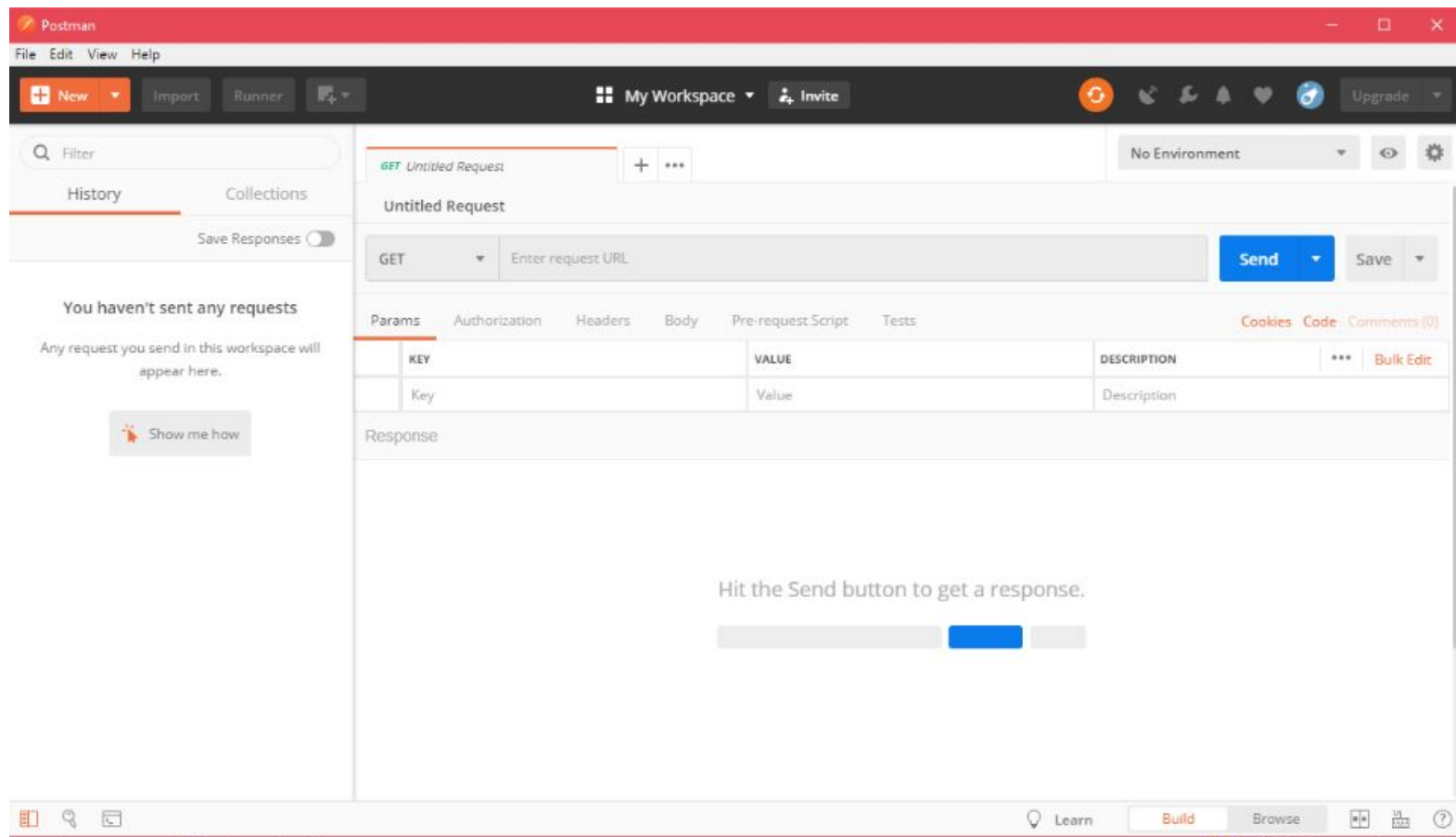
Get Started

Already have the app? [Take the next step!](#)



This website uses cookies to ensure you get the
best experience on our website. [Learn more](#)

Got it!



https://jsonplaceholder.typicode.com/posts

GET

https://jsonplaceholder.typicode.com/posts

Send

Save

Params

Authorization

Headers

Body

Pre-request Script

Tests

Cookies

Code

Comments (0)

1

2

GENERATE CODE SNIPPETS



Python Requests ▾

Copy to Clipboard

```
1 import requests
2
3 url = "https://jsonplaceholder.typicode.com/posts"
4
5 payload = ""
6 headers = {
7     'cache-control': "no-cache",
8     'Postman-Token': "fb2cd1b9-eca0-4b18-9602-fc68c4183d3a"
9 }
10
11 response = requests.request("GET", url, data=payload, headers=headers)
12
13 print(response.text)
```

Códigos de respuesta

- **1xx:** Información (¡Espera!)
- **2xx:** Respuesta correcta (¡Todo bien!)
- **3xx:** Redirección (¡No es aquí!)
- **4xx:** Error del cliente (¡Lo hiciste mal!)
- **5xx:** Error del servidor (¡No eres tu!)

Analizar el body de la respuesta

```
1 import requests
2 import json
3
4 url = "https://jsonplaceholder.typicode.com/posts"
5
6 payload = ""
7 headers = {
8     'cache-control': "no-cache",
9     'Postman-Token': "467ab332-ae53-499a-9ff2-5a7bb7ec1515"
10 }
11
12 response = requests.request("GET", url, data=payload, headers=headers)
13
14 # Código de la respuesta
15 print(response)
16
17 #Cuerpo de la respuesta
18 body = response.text
19
20 # Convertir a estructura de python
21 results = json.loads(body)
22
23 # Es una lista
24 print(type(results))
25
26 # Vemos el primer elemento
27 print(results[0])
28
29 # Y su tipo (un diccionario)
30 print(type(results[0]))
31
32 # Accedemos a cada título para cada elemento de la respuesta
33 for post in results:
34     print(post["title"])
35
```

Ejercicio de integración

```

1  import requests
2  import json
3
4  def request(requested_url):
5      headers = {
6          "cache-control": "no-cache",
7          "Postman-Token": "2467ab332-ae53-499a-9ff2-5a7bb7ec1515",
8      }
9      response = requests.request("GET", requested_url, headers=headers)
10     return json.loads(response.text)
11
12     prices = request("https://api.coindesk.com/v1/bpi/historical/close.json")["bpi"]
13
14     # Solución 1
15     selected_data = [k for k, v in prices.items() if v < 5000]
16     print(selected_data)
17
18     # Solución 2
19     selected_data = []
20     for date, value in prices.items():
21         if value < 5000:
22             selected_data.append(date)
23     print(selected_data)
24
25     # Solución 3
26     under_5000 = [v for v in prices.values() if v < 5000]
27     prices_inv = {v: k for k, v in prices.items()}
28     selected_data = [prices_inv[k] for k in under_5000]
29     print(selected_data)

```

SSL



Alice tiene 2 claves:
Una privada para cifrar su mensaje: **AWETGT**
Una pública para descifrar su mensaje: **EGEUJU**



Alice se junta con Bob y le pasa su clave pública para descifrar

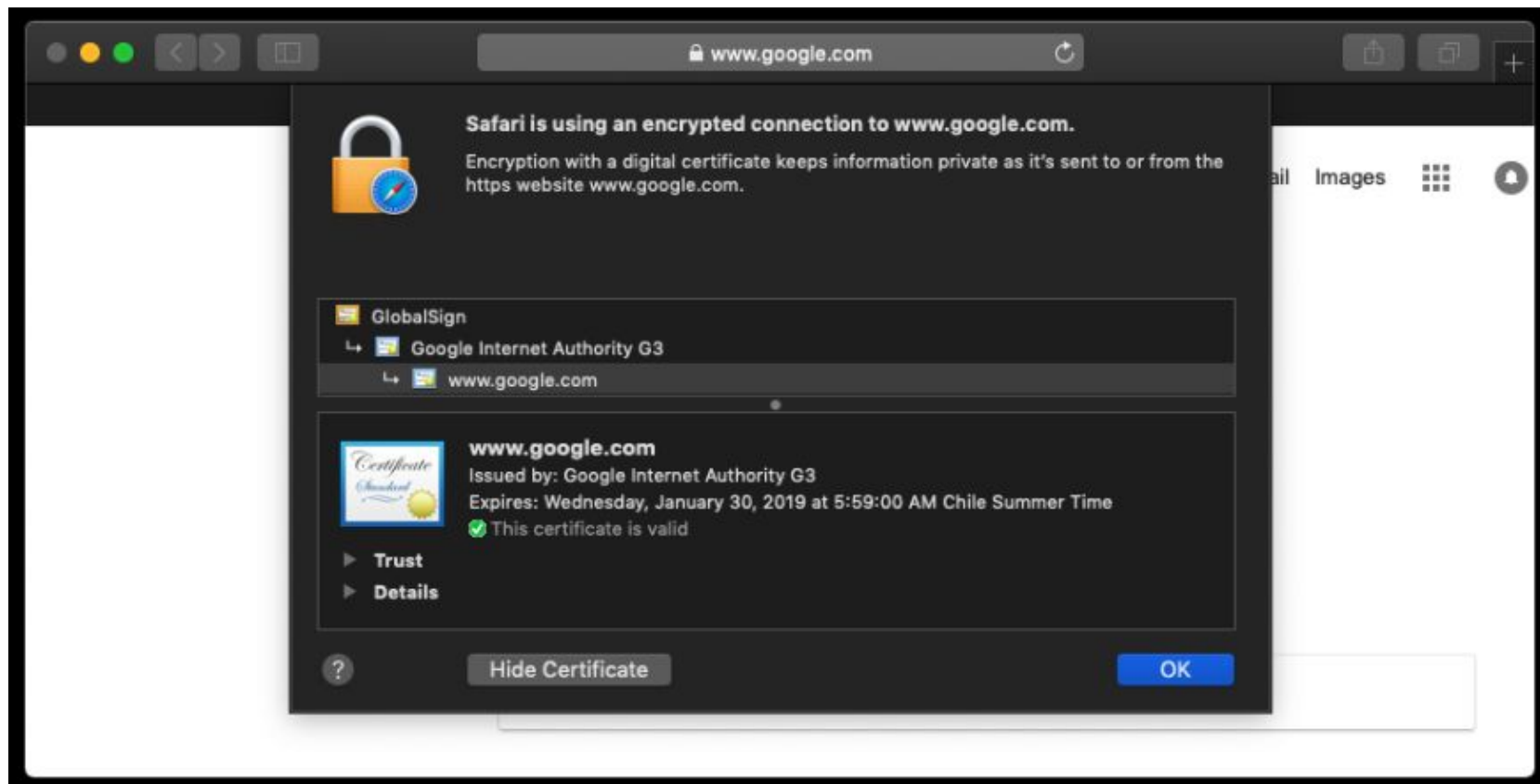


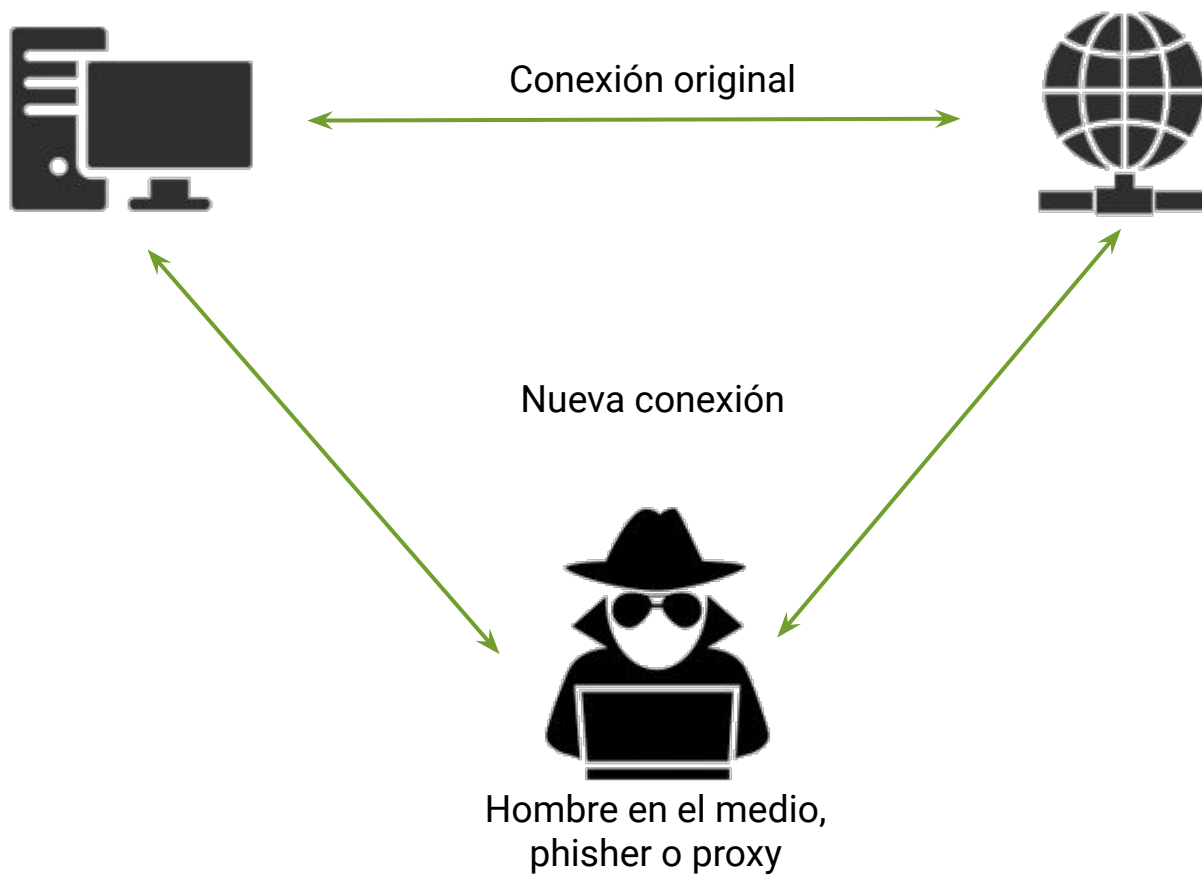
Alice envía a Bob un mensaje cifrado con su clave privada:



Bob utiliza la clave que Alice le pasó en el paso 2 para descifrar el mensaje







Desafío

**Solicitar una palabra a la API
del diccionario de Oxford**

Código

```

1 import json
2 import requests
3
4 def request(requested_url):
5     headers = {
6         "app_id": "XXXXXXXXXX",
7         "app_key": "XXXXXXXXXXXXXXX-XXXXXXXXXXXX-XXXXXXXXXXXX",
8     }
9     response = requests.request("GET", requested_url, headers=headers)
10    return json.loads(response.text)
11
12
13 word = "hedgehog"
14 data = request("https://od-api.oxforddictionaries.com:443/api/v1/entries/en/{0}".format(word))
15
16 print(data['results'][0]['lexicalEntries'][0]['entries'][0]['senses'][0]['definitions'])

```

Salida

```
{
  "metadata": {
    "provider": "Oxford University Press"
  },
  "results": [
    {
      "id": "hedgehog",
      "language": "en",
      "lexicalEntries": [
        {
          "entries": [
            {
              "etymologies": [
                "late Middle English: from  
hedge (from its habitat) + hog  
(from its piglike snout)"
              ],
              "grammaticalFeatures": [
                {
                  "text": "Singular",
                  "type": "Number"
                }
              ],
              "homographNumber": "000",
              "senses": [
                {
                  "definitions": [
                    "a small nocturnal Old  
World mammal with a spiny  
coat and short legs, able
```

Elementos clave

- Un algoritmo es una serie de pasos finitos para resolver un problema.
- Para crear algoritmos disponemos de herramientas como los diagramas de flujo y pseudocódigo.
- Las funciones son importantes para ordenar el código y evitar repetir varias veces lo mismo.
- Los ciclos son clave para evitar repetir código y hacer el programa escalable.
- Un problema puede tener más de una solución, y cuál será la mejor dependerá de cada caso.
- Las estructuras de datos como listas y diccionarios nos permiten almacenar y manejar grandes cantidades de datos en una sola variable.
- Podemos comunicarnos con otros programas de forma segura mediante el uso de una API, y usar estos datos para nuestros propios programas.

Otras cosas para explorar

- Estructuras de datos
- Complejidad algorítmica
- Almacenamiento y recuperación de información
- Bases de datos
- QA y Construcción de pruebas automatizadas
- Arquitectura de software
- Ciencia de datos
- Inteligencia artificial

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com