# Estructuras de datos y funciones

## Introducción a APIs

{desafío} latam_

***Utilizar estructuras de datos apropiadas para la elaboración de un algoritmo que resuelve un problema acorde al lenguaje Python.***

***Codificar un programa utilizando funciones para la reutilización de código acorde al lenguaje Python.***

- Unidad 1:
  Introducción a Python
- Unidad 2:
  Sentencias condicionales e iterativas
- Unidad 3:
  Estructuras de datos y funciones

***Te encuentras aquí***

{desafío} latam_

# ¿Qué aprenderás en esta sesión?

- *Conecta a APIs mediante Python para extraer información relevante para su aplicación.*
- *Aplica los distintos verbos REST para interactuar con la API de manera apropiada.*

# ¿Cómo se puede organizar un proyecto en Python?

{desafío} latam_

# /* API */

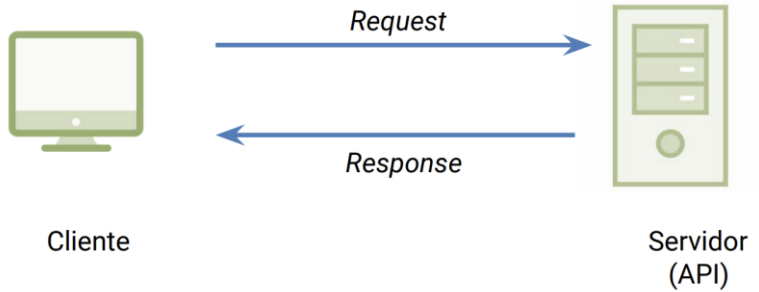
# ¿Qué son las APIs?

***Application Program Interface***

- Es una interfaz de acceso que permite comunicar programas.
- Al darle ciertas instrucciones, permite manipular datos, enviando o recibiendo. Esto es muy útil para integrar información que puede ser de utilidad en distintos sistemas y/o comunicando distintas aplicaciones.
- Al programa que consulta se le denomina cliente, mientras que al programa que entrega la respuesta se le suele llamar servidor.

Existen diversos tipos de APIs, pero particularmente hablaremos de **API REST** que es la cual utilizaremos.

**Funcionamiento de una API**



# Tipos de APIs existentes

Existen miles de APIs para distintos propósitos, por ejemplo:

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


De hecho, cualquier empresa puede construir su propia API y hacerla disponible a través de internet. Dependiendo de su uso, ésta contendrá métodos específicos para tareas específicas, las cuales se detallarán a lo largo de esta unidad.

# ¿Cómo se usa una API?

La manera más habitual en la que se comunica con un API es haciendo:

- **request** (pedido): es información que el cliente envía a una URL (localización del recurso del cual se quiere extraer la información) y se realiza **mediante un cliente**.

- **response** (respuesta): es la información que esa URL devuelve al cliente.

# Probando una API con Postman

- Herramienta muy potente y de gran popularidad que permite probar APIs sin necesidad de utilizar código y que nos permitirá hacernos una idea de lo que estamos haciendo.
- Está disponible para todos los sistemas operativos y se puede descargar desde su página oficial.

Para poder comenzar a entender qué es una API utilizaremos **jsonplaceholder**, una API que no contiene información real y es usada normalmente para el aprendizaje del tema.

# Mi primer request

Para hacer nuestro primer llamado, escribiremos la URL de la API a consultar. En este caso **https://jsonplaceholder.typicode.com/posts** y haremos click en SEND.

- Al hacer esto podremos ver que obtenemos una respuesta.
- El formato de la respuesta que es devuelta es conocido como JSON, y es un formato típico de JavaScript (JSON: **J**ava**S**cript **O**bject **N**otation.
- JSON es un formato para enviar información en texto plano, fácilmente entendible por humanos y por lenguajes de programación. Hoy en día es uno de los formatos más utilizados para enviar información entre sistemas.

# Mi primer request

https://jsonplaceholder.typicode.com/posts

- Esta API representa la información asociada a posteos en un BLOG. Como podemos observar tenemos el **userId** que corresponde al usuario que escribió el post, el **id** que será el **id** asociado a cada post, el **title** que es el título y el **body** que es el contenido de dicho post.
- Lo que acabamos de hacer es extraer información desde la API mediante un **request** de tipo GET. Este **GET** puede verse a la izquierda de la URL e indica que estamos "descargando" datos como JSON.
- El estar haciendo requests en POSTMAN es cómodo y práctico pero solo para explorar. Lo ideal es poder integrar esta información en nuestro código directamente en Python.

# /* Consumiendo una API desde Python */

# Consumiendo una API desde Python

- POSTMAN nos entrega una primera aproximación de cómo implementar un request en Python.
- Si es que hacemos click en Code en el sector derecho, POSTMAN nos entregará una lista de cómo extraer información mediante distintos lenguajes de programación.

Generated code for Python - Requests — Contribute on GitHub

```python
import requests

url = "https://jsonplaceholder.typicode.com/posts"

payload={}
headers = {}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)

```

Para el caso de Python, POSTMAN entrega la posibilidad de hacerlo utilizando la librería `http.client` o `request`.

# Consumiendo una API desde Python

```
import requests

url = "https://jsonplaceholder.typicode.com/posts"

payload={}
headers = {}

response = requests.request("GET", url, headers=headers, data=payload)

print(response.text)
```

# Consumiendo una API desde Python

Como dijimos, esta es sólo una primera aproximación del código. Debido a que este se genera de manera automática, la verdad es que hay varias cosas que se pueden descartar por ahora. Aún así iremos explicando cada parte:

```
import requests
```

Corresponde a la importación de la librería que utilizaremos para conectarnos a las distintas APIs.

```
url =
"https://jsonplaceholder.typicode.com/posts"

payload={}
headers = {}
```

# Consumiendo una API desde Python

**url** corresponde a una variable que contendrá la URL que estamos consultando.
***payload*** *y* ***headers*** *se profundizarán más adelante.*

```
import requests

url = "https://jsonplaceholder.typicode.com/posts"

payload={}
headers = {}
response = requests.request("GET", url,
headers=headers, data=payload)

print(response.text)
```

# Consumiendo una API desde Python

- La variable **response** será la encargada de almacenar el resultado del **request**.
- Para llevar a cabo el **request** se debe indicar el tipo, este caso **GET**, además de la url.
- Los otros parámetros no son necesarios ya que son diccionarios vacíos.
- El atributo **.text** contendrá la respuesta como un JSON, es decir, un string.
- **requests** nos ofrece una interfaz más limpia para poder implementar lo mismo.

```
import requests

response =
requests.get('https://jsonplaceholder.typicode.com/posts')
print(response)
print(response.text)
```

# El código de la respuesta

Según el número con el que se inicia la respuesta, será de distinto tipo:

- 1xx: Información.
- 2xx: Solicitudes exitosas.
- 3xx: Redireccionamiento.
- 4xx: Error del cliente (no hizo la request correctamente).
- 5xx: Error del servidor (no puede dar una respuesta).

Algunos de los códigos de respuesta más relevantes son:

- 200: Ok.
- 401: Unauthorized.
- 403: Forbidden.
- 404: Not Found.
- 500: Server error.

# El contenido de la respuesta

```
import json

results = json.loads(response.text)
print(type(results)) # Veremos que el resultado es una lista
print(results[0]) # Mostramos el primer elemento
```

Debido a que esto es un string (aunque con formato JSON) puede ser sumamente complejo de manipular. Es por eso que Python tiene una librería de manera nativa llamada json que nos permitirá transformar dicho string en objetos manipulables por Python, normalmente listas y diccionarios.

# El contenido de la respuesta

Para poder disponer de estos resultados será necesario poner en práctica lo aprendido en las unidades anteriores con respecto al manejo de estructuras de datos.

**Es importante destacar que cada API estará estructurada de manera distinta dependiendo de cómo haya sido creada.**

Ejemplo: tenemos una lista donde cada elemento se compone de un diccionario y cada clave de este diccionario es un string. De hecho podríamos explorar las claves que hay en cada post:

```
print(results[0].keys())
```

```
print(results[0]["userId"])
print(results[0]["title"])
```

# El contenido de la respuesta

Adicionalmente, en el caso que quisiéramos por ejemplo mostrar todos los títulos de nuestros posteos podemos iterar de la siguiente manera:

```
print([post['title'] for post in results])
```

Es importante destacar que **no hay reglas predefinidas** cuando nos enfrentamos a los resultados. Siempre va a ser necesario explorar la estructura de cada API para poder organizarlo de una manera que nos acomode.

# Transformar el request en una función

Debido a que es muy probable de que los request se hagan de manera muy seguida, puede ser una buena idea encapsular todo esto dentro de una función e incluso un módulo:

```
import requests
import json
def request_get(url):
    return json.loads(requests.get(url).text)
```

Luego para hacer un request bastará con ejecutar lo siguiente:

```
request_get('http://jsonplaceholder.typicode.com/posts')
```

# ¿Qué son las API y para qué sirven?

{desafío} latam_

# Próxima sesión...

- *Aplica los conceptos básicos del lenguaje Python a través de la utilización de la información extraída mediante API para agregar versatilidad y robustez a sus aplicaciones.*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales