

Programación Avanzada

IIC2233 2024-1

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Dante Pinto - Francisca Cattán



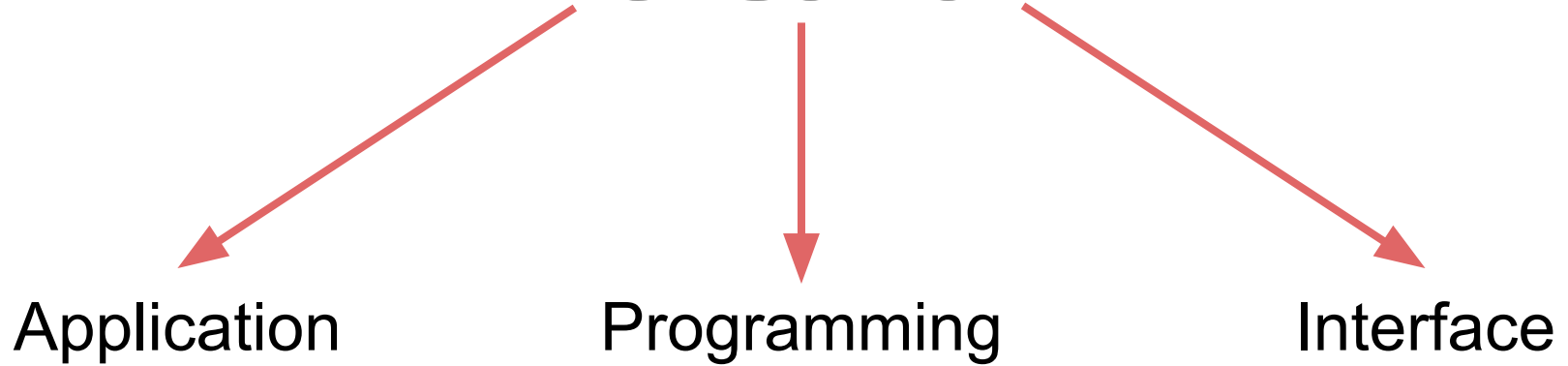
Anuncios

- Hoy tendremos la quinta y **última actividad**.
 - El otro jueves tendremos programación/sala de ayuda para la T4.
 - Jueves 27, cierre de curso, [respondan el formulario](#) por favor.
-

Aplicaciones en Python

- Abordaremos 2 casos muy interesantes de aplicaciones:
 - *Webservices* (Consumo y creación de API)
 - Expresiones regulares
- Si bien los veremos en el contexto de Python, estas aplicaciones son utilizables en cualquier lenguaje de programación.
- ¿Cuál es la relación entre estos dos contenidos?
 - Ninguna.

API



API

En general, API es un conjunto de funciones que son expuestas por un servicio para ser utilizadas por otros programas.

Nosotros nos enfocaremos específicamente en los servicios web o *web services*.

Por lo tanto, primero entendamos un poco cómo nos comunicamos en la web y luego retomaremos más el tema de la API.

**Comunicándose a
través de internet**

¿Cómo me conecto a un servidor?

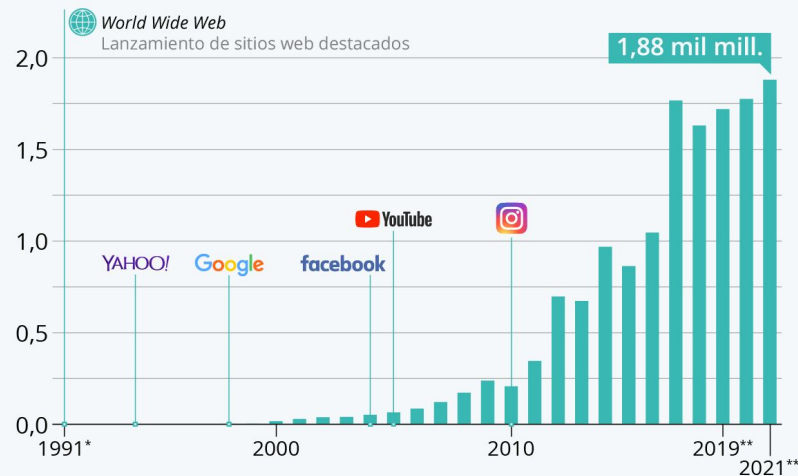
- Para empezar, en internet hay millones de páginas, y cada página puede tener cientos de directorios o recursos.

Fuente:

[Gráfico: ¿Cuántos sitios web hay en el mundo? | Statista](#)

¿Cuántas páginas web existen?

Número de sitios web existentes en Internet (en miles de mill.)



Un sitio web se entiende aquí como un "hostname" único.

* Datos del 1 de agosto de 1991

** Últimos datos disponibles 2019: 28 de octubre, 2020: 2 de junio, 2021: 6 de agosto.

Fuente: Internet Live Stats



statista

¿Cómo me conecto a un servidor?

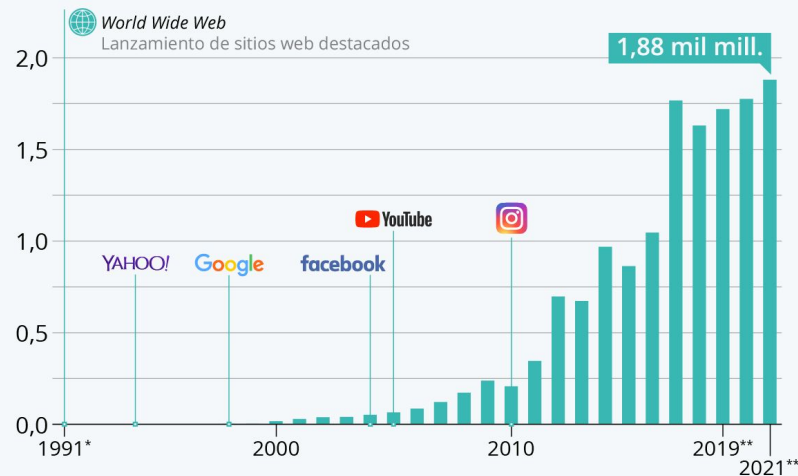
- Para empezar, en internet hay millones de páginas, y cada página puede tener cientos de directorios o recursos.
- Para identificar cada página, directorio y recurso, existe el concepto de **URL**.

Fuente:

[Gráfico: ¿Cuántos sitios web hay en el mundo? | Statista](#)

¿Cuántas páginas web existen?

Número de sitios web existentes en Internet (en miles de mill.)



statista

URL (*Uniform Resource Locator*)

- Más de una vez hemos visto textos así:

https://es.aliexpress.com/item/1005006792016183.html?spm=a2g0o.productlist.main.1.3d1aeGRMeGRMwJ&algo_pvid=4a7a04ee-41fe-457b-8b5a-8e46d7554e61&algo_exp_id=4a7a04ee-41fe-457b-8b5a-8e46d7554e61-0&pdp_npi=4%40dis%21CLP%212457%211899%21%21%212.60%212.01%21%40210318c317174331085181675e3076%2112000038316839442%21se a%21CL%21171744362%21&curPageLogUid=ZC08JbSCPYC s&utparam-url=scene%3Asearch%7Cquery_from%3A



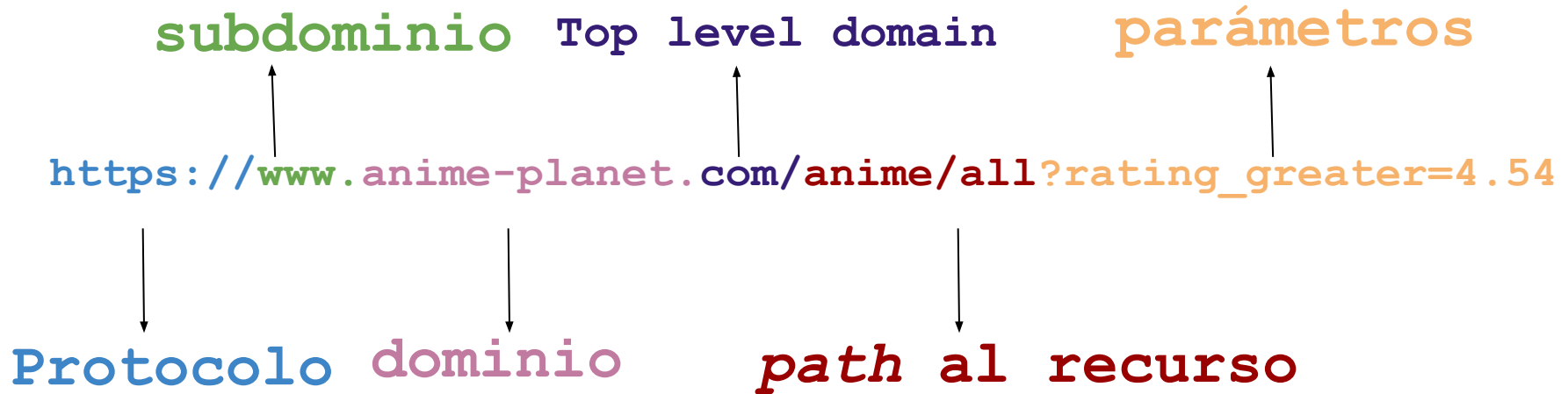
URL (*Uniform Resource Locator*)

Se definió un **formato que deben cumplir las URLs** para encontrar a lo que sea que deseen buscar en internet.

`https://www.anime-planet.com/anime/all?rating_greater=4.54`

URL (*Uniform Resource Locator*)

Se definió un **formato que deben cumplir las URLs** para encontrar a lo que sea que deseen buscar en internet.



URL (*Uniform Resource Locator*)

Todo enlace, por intimidante que se vea, lo cumple.

https://es.aliexpress.com/item/1005006792016183?spm=a2g0o.productlist.main.1.3d1aeGRMeGRMwJ&algo_pvid=4a7a04ee-41fe-457b-8b5a-8e46d7554e61&algo_exp_id=4a7a04ee-41fe-457b-8b5a-8e46d7554e61-0&pdp_npi=4%40dis%21CLP%212457%211899%21%21%212.60%212.01%21%40210318c317174331085181675e3076%2112000038316839442%21sea%21CL%21171744362%21&curPageLogUid=ZC08JbSCPYCs&utparam-url=scene%3Asearch%7Cquery_from%3A

HTTP (*Hypertext Transfer Protocol*)

Así como en *networking* se define un protocolo para comunicarse entre cliente y servidor, en el mundo real existe un protocolo altamente utilizado para comunicarse a través de internet: **HTTP**

- Cliente envía una petición, y el servidor le envía una respuesta
- Una vez enviada la respuesta, el servidor “olvida” dicha petición.

Entonces ¿Cómo nos comunicamos con el servidor que queremos? 🤔

**¿Cómo funciona la
comunicación una
vez que accedo a
una URL?**

Servidor
HTTP

Cliente
HTTP



Solicitud



Respuesta

Servidor
HTTP

Cliente
HTTP



Solicitud

Headers

Body



Respuesta

Headers

Body

Tenemos el cómo comunicarnos.

Ahora...

**¿Qué es lo que
comunicamos?**

Tipos de solicitudes: Informando al servidor lo que queremos hacer.

- **GET:** Pedimos la representación de un recurso **sin cambiar nada** en el servidor.
- **POST:** **Creamos** un recurso.
- **PATCH:** Aplica **modificaciones parciales** a un recurso.
- **PUT:** **Reemplaza completamente** un recurso existente.
- **DELETE:** **Elimina** un recurso.

Códigos de respuesta: Informando al cliente lo que ocurrió.

- **200:** Ok, solicitud exitosa.
- **403:** La solicitud es correcta, pero se rechaza dar una respuesta.
- **404:** El recurso solicitado no se encuentra en el servidor.
- **500:** Error interno del servidor

Y muchos más...

**Una URL puede
(o no) tener
varias acciones
asociadas**

Ejemplo





Supongamos que tenemos la página web <http://www.paginaic2233.com/>, que tiene un recurso estudiantes. Para poder interactuar con los datos de la página, se habilitaron las siguientes URLs

<http://www.paginaic2233.com/estudiantes>

<http://www.paginaic2233.com/estudiante/<id>>

Donde el id es un número. Considerando esto, podríamos pensar que dicha página tiene las siguientes acciones:

Ejemplo

URL	GET	POST	PUT	PATCH	DELETE
<u>/estudiantes</u>	Obtiene una lista de todos los estudiantes	Crea un nuevo estudiante			
<u>/estudiante/1/</u>	Obtiene los datos del estudiante con id 1		Reemplaza todos los datos del estudiante con id 1	Reemplaza algunos atributos del estudiante con id 1	Elimina al estudiante con id 1

*Esto es solo un ejemplo, las funcionalidades dependerán de lo que decida quien implementó el *webservice*

**¿Cómo sabe una
API quién soy?**

Autenticación

Problema

- Hay acciones, como editar la base de datos, que no todo usuario puede hacer.
- Generalmente las APIs se componen de funciones que “no memorizan entre una solicitud y otra”. Por lo tanto, no podemos hacer una solicitud de “login” y luego una de “modificar base de datos” en donde recuerde el login previo.
- Surge la necesidad de un mecanismo que permita, en cada solicitud, poder identificar al usuario.

Autenticación

Solución

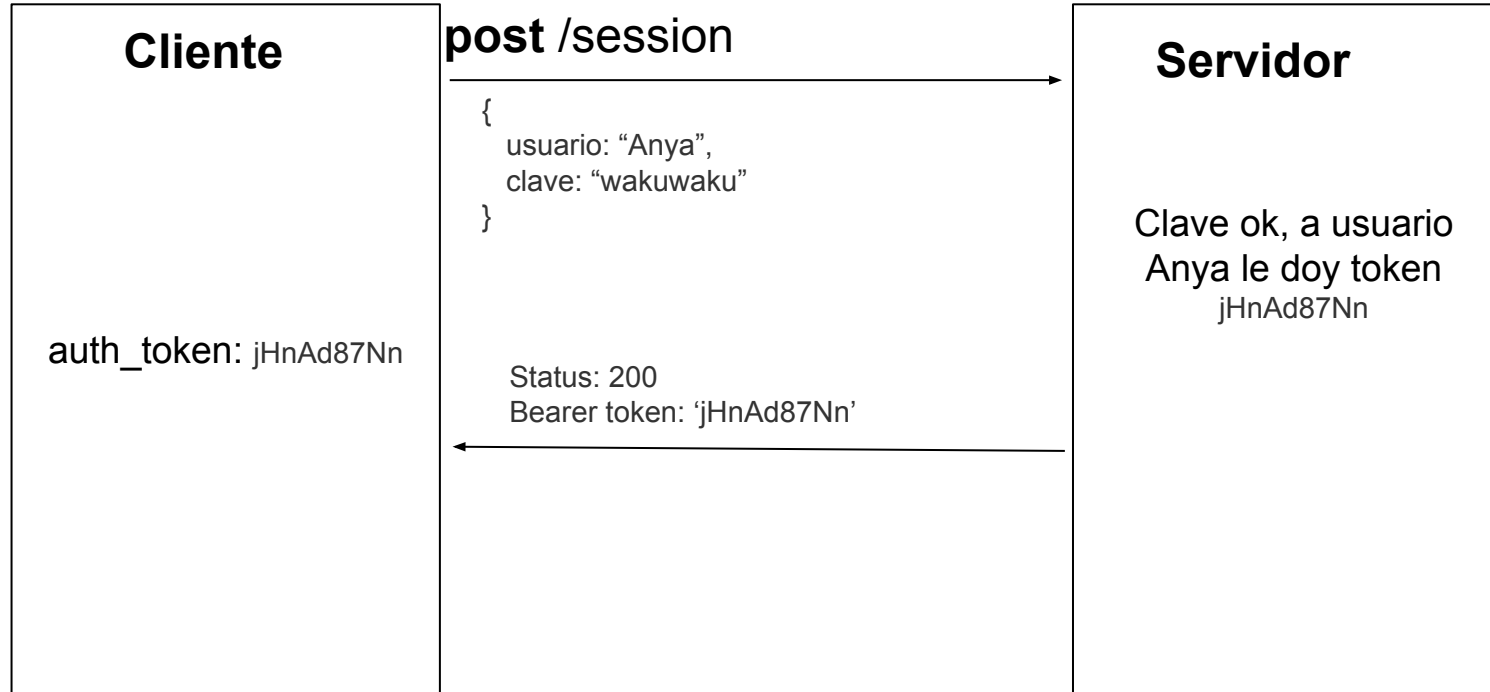
- En el *header* podemos incluir un **token especial** de acceso que a la API le sirve para identificar y verificar si permisos.



Autenticación - Ejemplo (obtener el token)



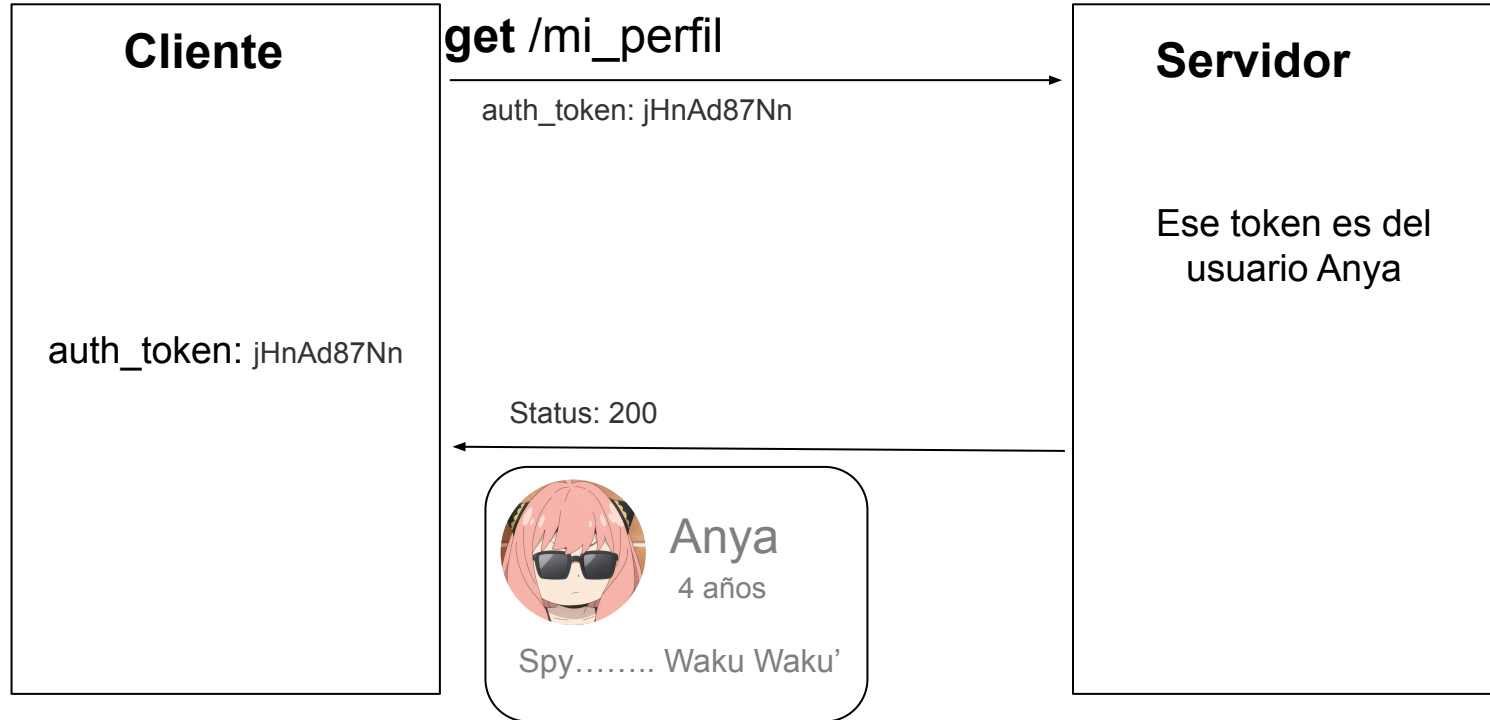
Autenticación - Ejemplo (obtener el token)



Autenticación - Ejemplo (usar el token)



Autenticación - Ejemplo (usar el token)



**¿Y esto en python
cómo se ve?**

requests

```
import requests
```

```
url = "https://fakestoreapi.com/products/1"  
response = requests.get(url) # Método GET  
print(response.status_code)  # Status 200  
print(response.json())       # Respuesta del servidor
```

requests

```
import requests
import json
```

```
url = "https://fakestoreapi.com/products"
# Información adicional (datos + headers) para crear un producto
datos_a_subir = json.dumps({ "producto": "Manga", "precio": 7000 })
headers = { "Authorization": "token GjstdxYkdLSDandsGH" }

# Hacer la request
request.post(url, data=datos_a_subir, headers=headers) # Método POST
```


Levantamiento de una API

Flask

```
from flask import Flask
```

```
app = Flask(__name__) # Creamos la API
```

```
# Definimos endpoint "/" que acepta solo GET
```

```
@app.route("/", methods=["GET"])
```

```
def hello_world():
```

```
    return {"texto": "Holi"}
```

```
if __name__ == "__main__":
```

```
    app.run(host="localhost", port=4444) # Levantamos la API
```

Flask

```
from flask import Flask, request

app = Flask(__name__) # Creamos la API

# Definimos endpoint "/dado" que acepta GET y POST
@app.route("/dado", methods=["GET", "POST"])
def dado():
    if request.method == "POST":
        numero = random.randint(0, 6)
        return {"resultado": numero, "método": "POST"}

    numero = random.randint(-4444, -11)
    return {"resultado": numero, "método": "GET"}
```

Temas adicionales de interés

Temas adicionales de interés

- Hay tokens que incluso pueden almacenar información: [JWT.io](https://jwt.io)
- Existen muchos más códigos HTTP
 - [Códigos de estado de respuesta HTTP](#)
 - Existe su explicación gatuna: [HTTP Cats](#)
 - El [código 418](#) es cuando el servidor se rehúsa a preparar café porque es una tetera.
- Existen otras librerías para programar rápidamente una API como [FastApi](#).

Expresiones Regulares (Regex)



A veces necesitamos encontrar patrones en nuestros strings bastante específicos.

Para estos casos, podríamos tener funciones de varias líneas haciendo uso de métodos de string y procesando el string de forma progresiva...

...O usar una Expresión Regular, también llamadas Regex 😎

Qué es una RegEx

Patrones de búsqueda, compuestos de secuencias de caracteres especializados, aplicables a *strings*. El objetivo es revisar si el patrón se encuentra una o más veces dentro del *string* a analizar.

Esto permite que la validación de un *string* para que siga un formato definido sea más concisa.

En **Python**, usaremos la librería **re** para trabajar con RegEx.

Caracteres básicos de una RegEx

Sintaxis	Descripción
[]	Clases de caracteres
+	Puede estar 1 o más veces
*	Puede estar 0 o más veces
?	Puede estar a lo más 1 vez
{m, n}	Puede estar entre m y n veces
.	Comodín (cualquier carácter)

Sintaxis	Descripción
^	Inicio del <i>string</i>
\$	Final del <i>string</i>
()	Agrupar
	OR
\	Escapar caracteres especiales para usarlos

Caracteres básicos de una RegEx

Sintaxis	Descripción
<code>\s</code>	Cualquier espacio en blanco
<code>[a-z]</code>	Cualquier letra minúscula entre a y z
<code>[a-zA-Z]</code>	Cualquier letra entre A y Z, sin importar mayúscula o minúscula
<code>[0-9]</code>	Cualquier dígito entre 0 y 9
<code>[^arn]</code>	Cualquiera EXCEPTO a, r, n

Y muchos más...

Algunos ejemplos

Validar un rut:

[0-9]{1,2}\.[0-9]{3}\.[0-9]{3}-([0-9kK])

- **[0-9]{1,2}** Un número de 1 o 2 dígitos, que pueden ir entre 0 y 9 cada dígito.
- **\.** Seguido de un punto ‘.’.
- **[0-9]{3}** Número de 3 digitos, que pueden ir entre 0 y 9 cada dígito
- **\.** Seguido de otro punto.
- **[0-9]{3}** Número de 3 digitos, que pueden ir entre 0 y 9.
- **-** Seguido de un guión ‘-’.
- **([0-9kK])** Un dígito entre 0 y 9, o una K (mayúscula o minúscula).

Algunos ejemplos

Validar un correo con formato específico:

[a-zA-Z0-9_.] + @((seccion1|seccion2)\.)?(mimail|mail)\.cl

- **[a-zA-Z0-9_.] +** Tiene que haber una letra, dígito, guion bajo o punto por lo menos una vez.
- **@** Luego un arroba
- **((seccion1|seccion2)\.)?** Luego puede haber o no haber un “seccion1.” o “seccion2.” (no ambos).
- **(mimail|mail):** Luego debe haber un “mimail” o un “mail” (no ambos)
- **\.cl:** Finalizar con .cl

Algunos ejemplos

Validar un correo con formato específico:

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.|+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)|(?:(?:2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

Fuente: [How can I validate an email address using a regular expression? - Stack Overflow](#)

Algunos e

Validar un correo

```
(?:[a-z0-9!#$%&'*~.-\x08\x0b\x0c\x0e-\x7f])*"(@(?:(?:[a-z0-9!#$%&'*~.-\x08\x0b\x0c\x0e-\x7f]|1[0-9][0-9][0-9])|1[0-9][0-9][0-9])|21
```

```
~.-]+)*|"(?:[\x00-\x7f]|[\x0c\x0e-\x7f]|[\x21-\x7f])"(\.([a-z0-9-]*[a-z0-9])?|[\x21-\x7f])|([a-z0-9-]*[a-z0-9])?|([a-z0-9-]*[a-z0-9])?|([a-z0-9-]*[a-z0-9])?)
```

Fuente: [How can I validate an email address?](#)



RegEx no es trivial

Por eso hay muchas páginas que ayudan a entender qué diablos significa una expresión o cómo diablos escribo una:

- <https://regexr.com/>
- <https://regex101.com/>
- <https://www.regextester.com/>

Usar RegEx en Python

El módulo **re** nos permite aplicar RegEx para analizar *strings* en nuestro código:

```
re.match(patron, string) # Encontrar la primera ocurrencia desde el inicio
re.fullmatch(patron, string) # Todo el string cumple el patrón
re.search(patron, string) # Encontrar en cualquier lado el patrón
re.sub(patron, reemplazar_por, string) # Reemplazar un patrón
re.split(patron, string) # Separar según el patrón
```

Programación Avanzada

IIC2233 2024-1

Hernán Valdivieso - Daniela Concha -

- Dante Pinto - Francisca Cattán

