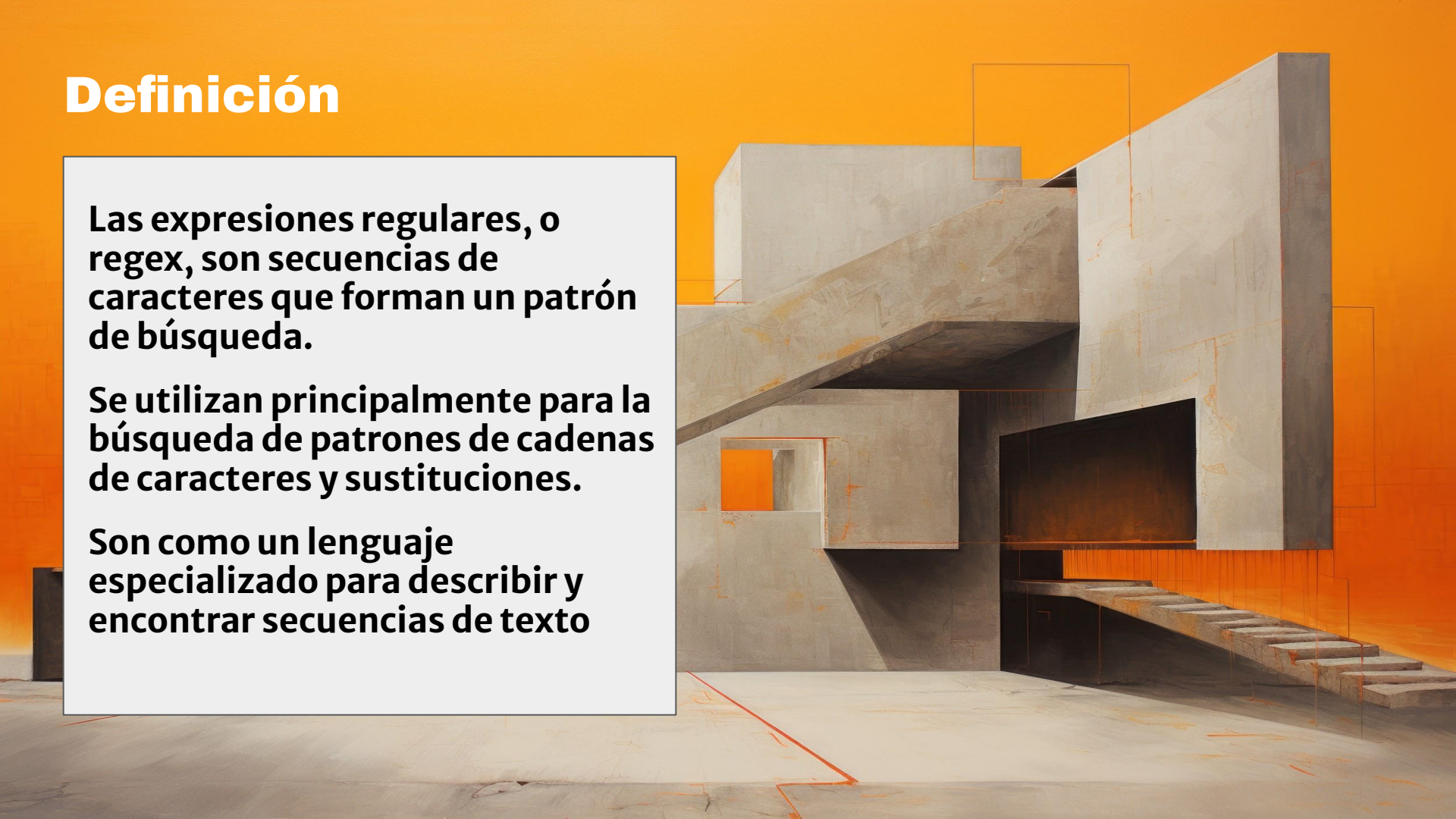

Regex

Definición

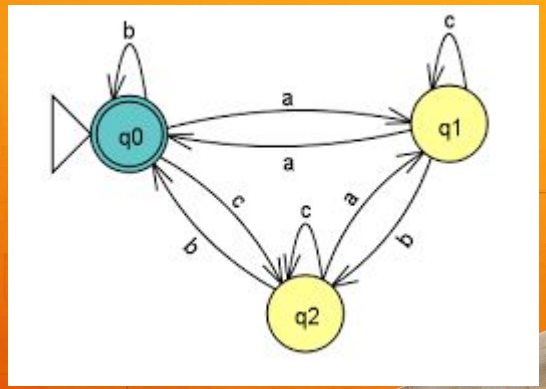
Las expresiones regulares, o regex, son secuencias de caracteres que forman un patrón de búsqueda.

Se utilizan principalmente para la búsqueda de patrones de cadenas de caracteres y sustituciones.

Son como un lenguaje especializado para describir y encontrar secuencias de texto



Historia



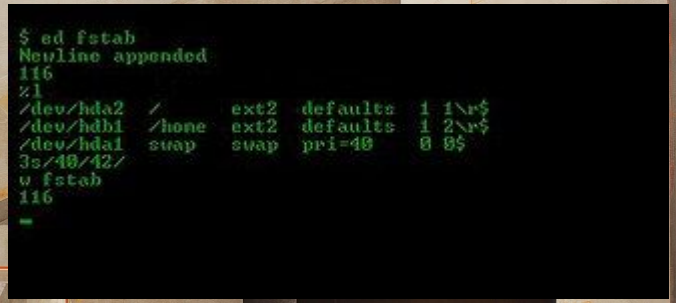
1. 1940s:

Stephen Cole Kleene desarrolla los autómatas regulares, que forman la base teórica de las expresiones regulares.

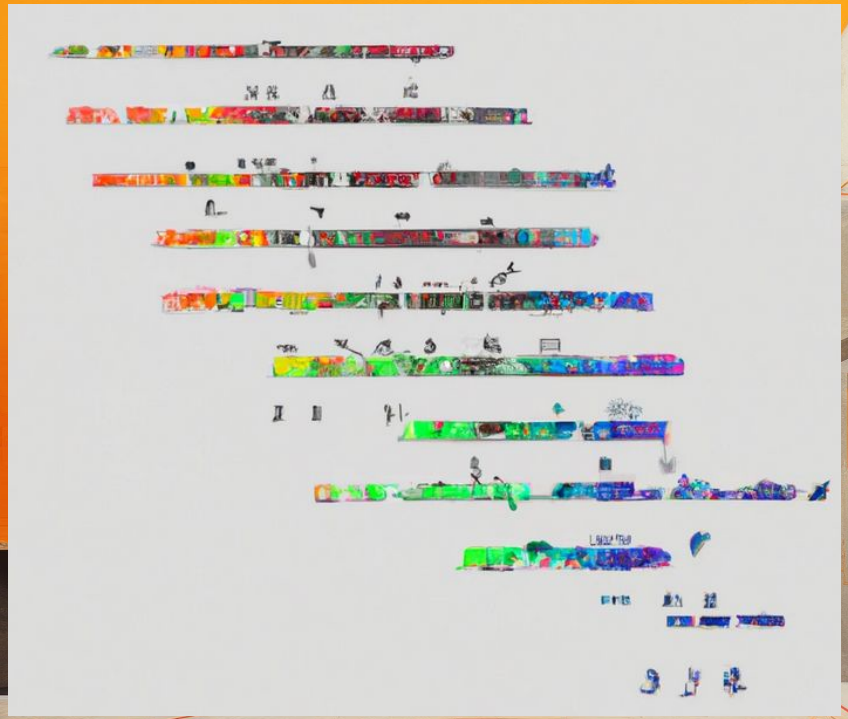
Historia

2. 1960s:

Ken Thompson implementa el primer comando de búsqueda basado en regex en el editor QED, y posteriormente en el editor "ed" en UNIX.



Historia



3. 1970s y 1980s:

Las expresiones regulares se popularizan con herramientas UNIX como "grep" y "vi".

El lenguaje de programación Perl también adopta regex como una de sus características clave.

Historia

4. 1990s – Presente:

Las expresiones regulares se incorporan en la mayoría de los lenguajes de programación modernos y en herramientas de software, convirtiéndose en una competencia esencial para los profesionales de TI.

Importancia

1. Búsqueda avanzada:

Encuentra patrones específicos en grandes volúmenes de texto, como direcciones de correo electrónico, números de teléfono, URLs, etc.

Importancia

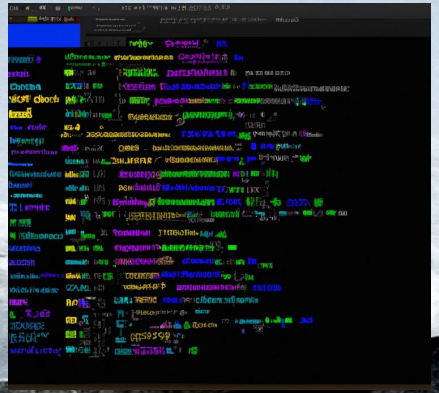
2. Validación de datos:

Asegúrate de que el texto cumpla con ciertos formatos o reglas, como contraseñas seguras o formatos de fecha válidos.

Importancia

3. Sustitución y edición:

Reemplaza o elimina ciertos patrones de texto de forma masiva y precisa.

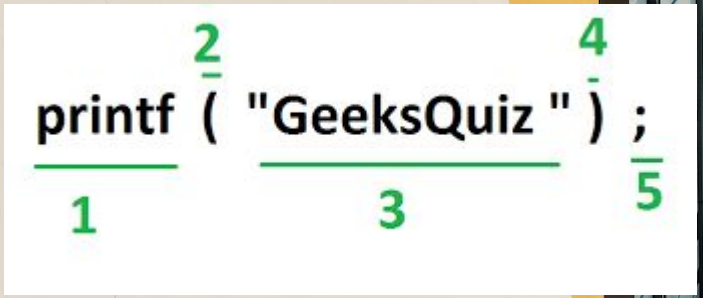


Importancia

4. Desglose de datos:

Extrae información específica de textos, como todos los enlaces en una página web o datos de un archivo de log.

Compiladores y Regex



Tokenización:

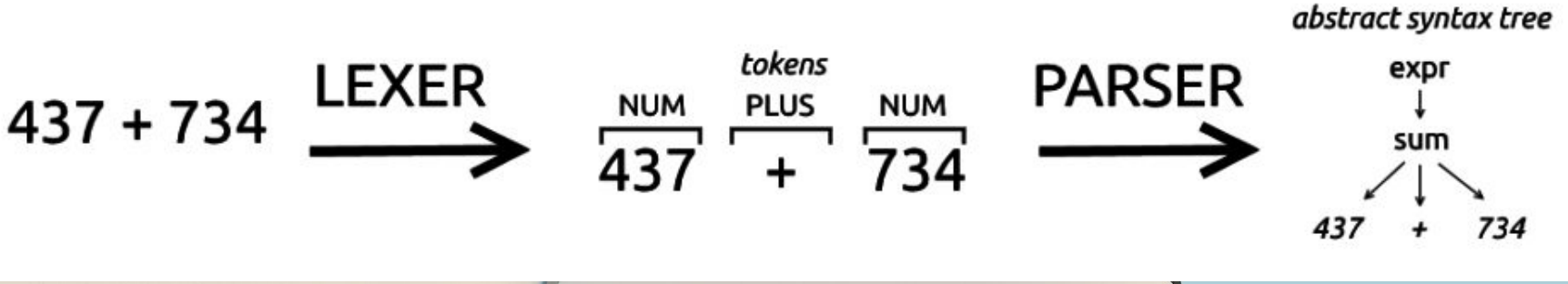
Una etapa clave en la compilación es la tokenización, donde el código fuente se descompone en tokens. Regex es una herramienta valiosa en este proceso para identificar patrones como identificadores, operadores y literales.

Compiladores y Regex

Lexers y Parsers:

Muchos "lexers" (analizadores léxicos) utilizan expresiones regulares para definir los tokens que pueden aparecer en un lenguaje.

Los "parsers" luego toman estos tokens para construir una representación estructurada del código, como un árbol sintáctico abstracto.



Compiladores y Regex

```
impo cvxpy as cp
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
sns.set(rc={'figure.figsize':(16,6)})
```

Errores

Los errores que su IDE llega a detectar en su código antes de ser ejecutado se debe a que rompen alguna regla que es especificada por Regex.

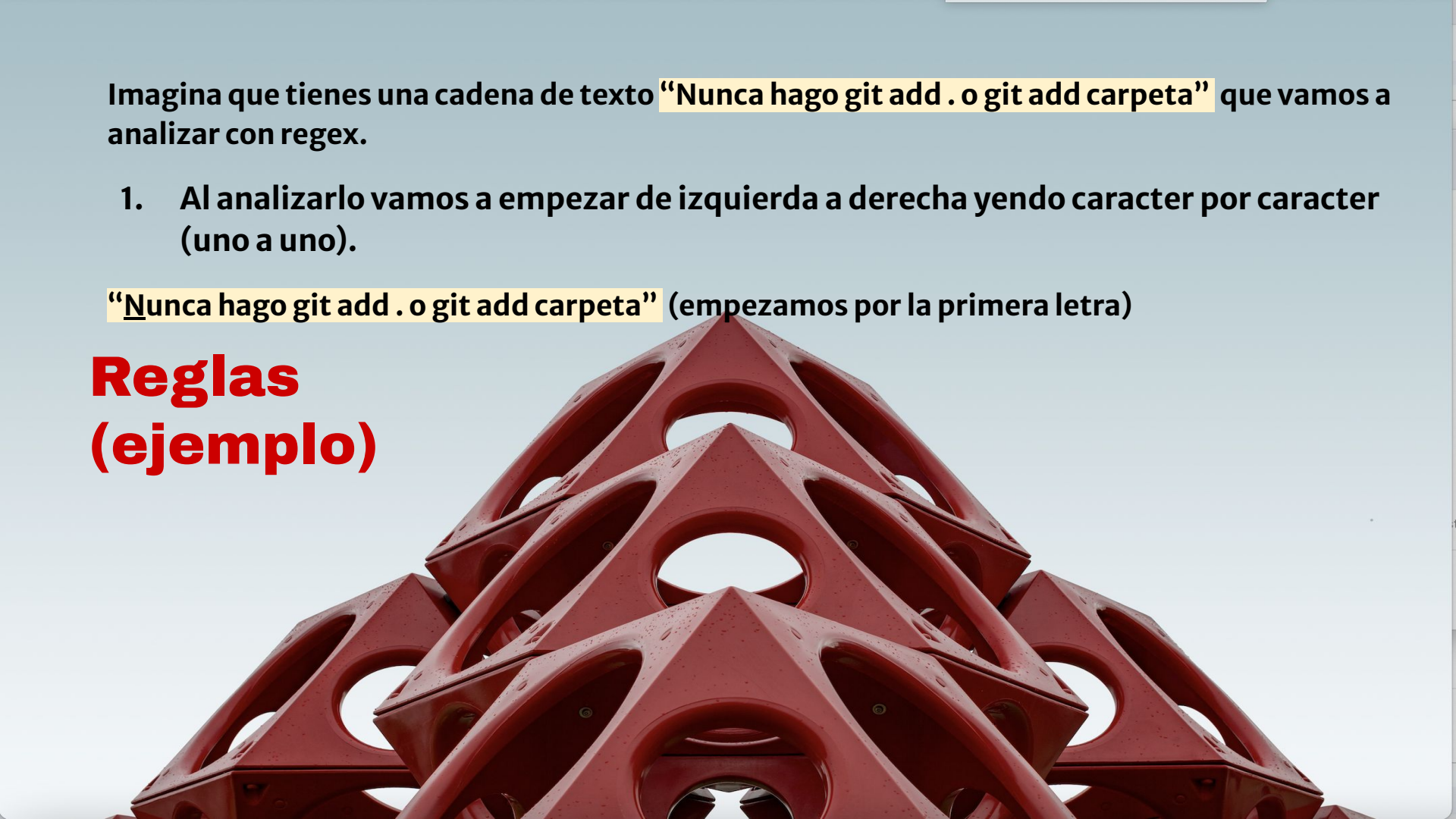
Video de Intro

Imagina que tienes una cadena de texto “Nunca hago git add . o git add carpeta” que vamos a analizar con regex.

1. Al analizarlo vamos a empezar de izquierda a derecha yendo caracter por caracter (uno a uno).

“Nunca hago git add . o git add carpeta” (empezamos por la primera letra)

Reglas (ejemplo)

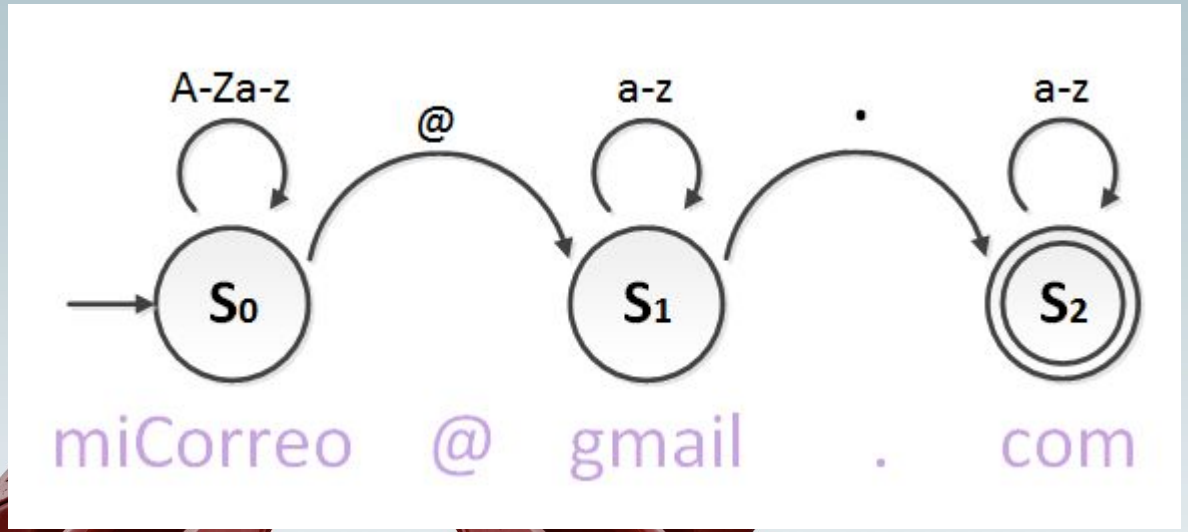


Para especificar las reglas que queremos evaluar debemos especificar el Regex usando las letras y caracteres básicos.

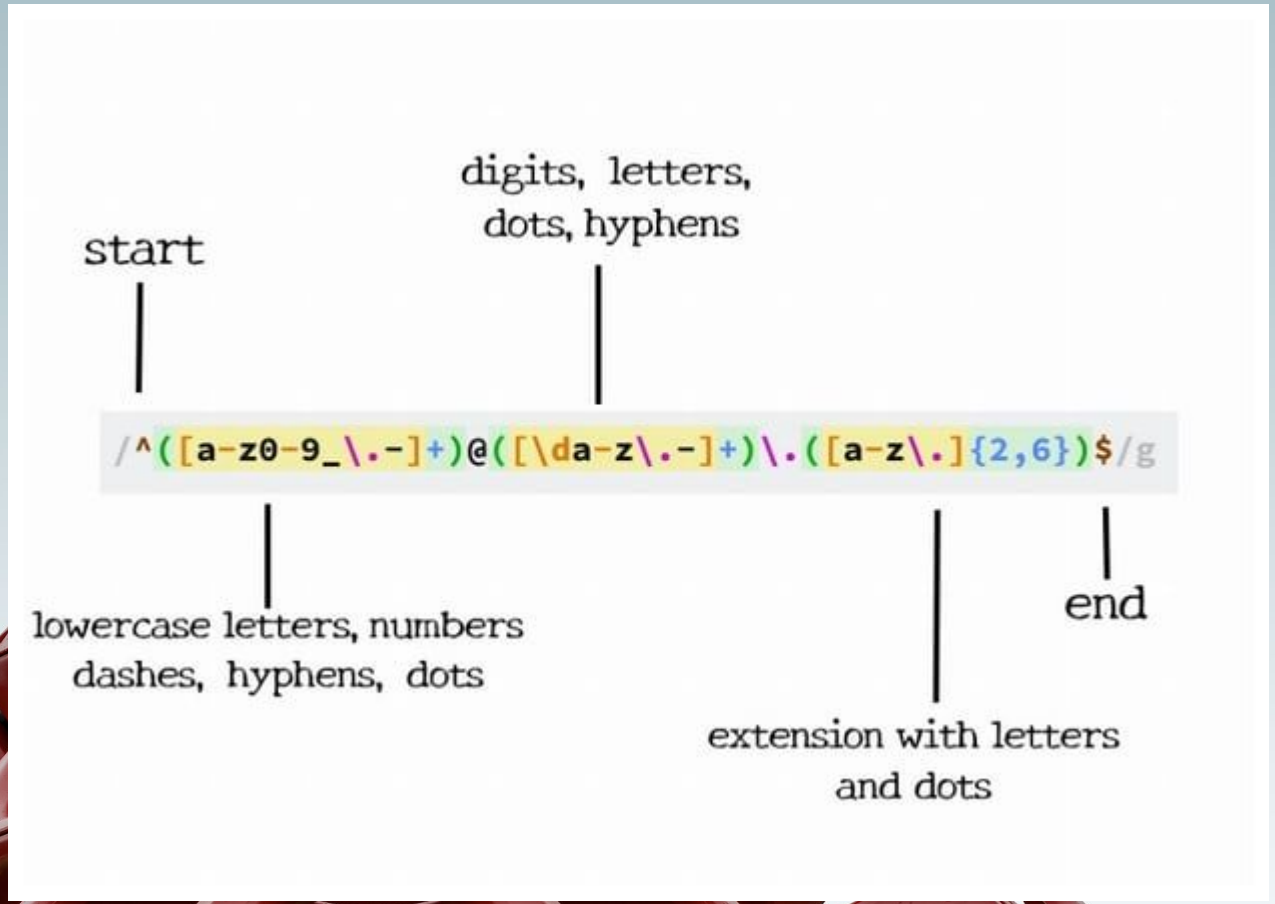
Reglas

- `.` : Cualquier carácter (excepto un salto de línea).
- `^` : Inicio de una cadena.
- `$` : Final de una cadena.
- `*` : Cero o más repeticiones del carácter anterior.
- `+` : Una o más repeticiones del carácter anterior.
- `{n}` : Exactamente n repeticiones.
- `[abc]` : Cualquiera de los caracteres en los corchetes.
- `[^abc]` : Ninguno de los caracteres en los corchetes.

Reglas (Ejemplo visual)



Reglas (Ejemplos)



grep

En la terminal podemos usar el comando grep para buscar expresiones regulares. Este comando regresa las líneas (una línea es separada por un salto de línea '\n')

En python podemos usar la paquetería re

grep - Literales

1. Caracteres literales: Un carácter literal simplemente coincide con un carácter específico en la cadena de texto.
    a. Por ejemplo:
        i. El patrón "a" coincide con el carácter "a" en la cadena "sdaf". Por lo tanto regresaría “sdaf”

```
~ echo 'sdaf' | grep a
sdaf
```

        ii. El patrón “ax” no coincide con ningún carácter en la cadena "sdaf", por lo tanto no regresa nada.

```
~ echo 'sdaf' | grep ax
~
```

grep - Conjuntos

2. Conjunto de caracteres: Un conjunto de caracteres permite especificar una variedad de caracteres que se aceptarán. Denotados por “[]”

   a. Por ejemplo, [aeiou] coincide con cualquier vocal en una cadena de texto, como "a" en "hola".

```
~ echo 'hola' | grep "[qwera]"
hola
```

grep - Clases

3. Clases de caracteres predefinidos: Algunos caracteres tienen un significado especial en las expresiones regulares, como \d para cualquier dígito y \w para cualquier carácter alfanumérico.

   a. Por ejemplo, el patrón \d\d\d coincide con tres dígitos consecutivos en una cadena de texto, como "123".

```
~ echo 'asdf123asdf' | grep -P '\d'
asdf123asdf
~ echo 'asdf123asdf' | grep -P '\d\d\d\d'
~
```

Nota: El -P se agrega para que grep pueda detectar la expresión regular correctamente. El comando “/d” no viene por default en el “GNU grep” se agregó posteriormente como parte del regex de Pearl. Fuente: Post

grep - Repetición

4. Repetición: Las expresiones regulares pueden especificar la cantidad de veces que se repetirá una expresión.

a. **Por ejemplo, \d{3} coincide con tres dígitos consecutivos en una cadena de texto, como "123".**

```
~ echo 'asdf123asdf' | grep -P '\d{1}'
asdf123asdf
~ echo 'asdf123asdf' | grep -P '\d{2}'
asdf123asdf
~ echo 'asdf123asdf' | grep -P '\d{3}'
asdf123asdf
~ echo 'asdf123asdf' | grep -P '\d{4}'
~
```

grep - Grupos

5. Grupos: Los grupos permiten agrupar una secuencia de caracteres y tratarla como un solo elemento. Denotado por “()”

   a. Por ejemplo, (\d\d) coincide con dos dígitos consecutivos y permite capturar el número completo para su posterior uso.

```
~ echo 'asdf123asdf' | grep -Po '\d{1}'
1
2
3
~ echo 'asdf123asdf' | grep -Po '\d{2}'
12
~ echo 'asdf123asdf' | grep -Po '\d{3}'
123
~ echo 'asdf123asdf' | grep -Po '\d{4}'
~
```

grep - Grupos

5. Grupos: Los grupos permiten agrupar una secuencia de caracteres y tratarla como un solo elemento. Denotado por “()”

Nota: La bandera “-Po” convierte el regex en grupos, sin necesidad de usar “()”. Sin embargo, en python se utiliza “()”.

```
~ echo 'asdf123asdf' | grep -Po '\d{1}'
1
2
3
~ echo 'asdf123asdf' | grep -Po '\d{2}'
12
~ echo 'asdf123asdf' | grep -Po '\d{3}'
123
~ echo 'asdf123asdf' | grep -Po '\d{4}'
~
```

grep - Alternativas

6. Alternativas: Las alternativas permiten especificar varias opciones para una coincidencia.

   a. Por ejemplo, (cat|dog) coincide con "cat" o "dog" en una cadena de texto.

```
~ echo 'high cat' | grep -P 'cat|dog'
high cat
~ echo 'high cat' | grep -P 'ca|dog'
high cat
~ echo 'high ca' | grep -P 'cat|dog'
~ echo 'high ca' | grep -P '[cat|dog]'
high ca
~
```

grep - Caracteres Especiales

. (punto): Coincide con cualquier carácter excepto un salto de línea.

* (asterisco): Coincide con cero o más repeticiones del carácter o expresión previa. (También conocida como la cerradura de Kleene)

+ (más): Coincide con una o más repeticiones del carácter o expresión previa.

? (interrogación): Coincide con cero o una sola repetición del carácter o expresión previa.

grep - Caracteres Especiales

{n} (llaves con número): Coincide con exactamente n repeticiones del carácter o expresión previa.

{n,} (llaves con número y coma): Coincide con al menos n repeticiones del carácter o expresión previa.

{n,m} (llaves con dos números y coma): Coincide con al menos n y como máximo m repeticiones del carácter o expresión previa.

[...] (corchetes): Especifica un conjunto de caracteres que se aceptarán en la coincidencia.

grep - Caracteres Especiales

Regex:

```
^Hola
```

Esto coincidirá con:

```
Hola, ¿cómo estás?
Hola Mundo
```

Pero no coincidirá con:

```vbnet
Me dijeron: Hola
Adiós, Hola
```

^ (circunflejo): Especifica el comienzo de la línea o una negación en un conjunto de caracteres.

"^A": Esta expresión regular busca una coincidencia al principio de cada línea con la letra "A".

Por ejemplo, si buscamos esta expresión en el texto "Aquí hay una línea con la letra A", la coincidencia sería "A".

grep - Caracteres Especiales

^ (circunflejo): Especifica el comienzo de la línea o una negación en un conjunto de caracteres.

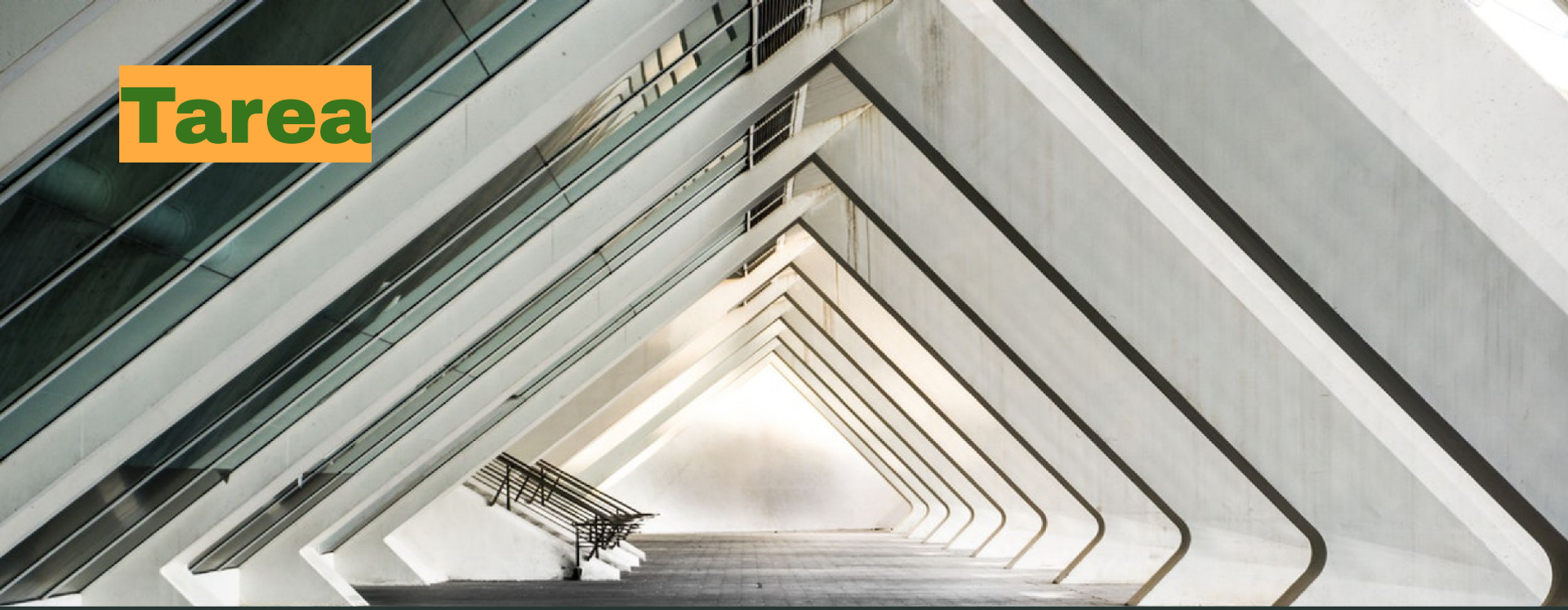
"[^0-9]": Esta expresión regular busca una coincidencia con cualquier carácter que no sea un número.

Por ejemplo, si buscamos esta expresión en el texto "Aquí hay un número 123", la coincidencia sería "Aquí hay un número " (sin incluir los números).

grep - Caracteres Especiales

$ (dólar): Especifica el final de la línea.

?! : significa negative look ahead, osea que no sea el siguiente pattern.



Tarea

1. Si aún te queda duda de regex, ver los siguientes dos videos de apoyo o investigar más
2. Ver el calendario para las siguientes tareas: regex, bandit y docker

Video de Apoyo

Videos/Tutorial de Apoyo