# Regex

# Inleiding met 06 nummers

## WTF is een reguliere expressie?

Een reguliere expressie (regex) is een reeks symbolen en speciale karakters die wordt gebruikt om patronen in tekst te vinden en/of aan te passen. Het is als een krachtige zoekopdracht die je kunt gebruiken om te zoeken naar specifieke stukjes tekst in een grotere tekst.

Stel je bijvoorbeeld voor dat je een telefoonboek hebt en je wilt alle telefoonnummers vinden die beginnen met "06". Met behulp van een reguliere expressie kun je een patroon definiëren dat overeenkomt met deze telefoonnummers, ongeacht wat er daarna komt. Je zou kunnen zoeken naar het patroon "06\d{8}", waarbij "\d{8}" overeenkomt met acht cijfers na "06". Dit patroon zou alle telefoonnummers vinden die beginnen met "06" gevolgd door acht willekeurige cijfers.

Reguliere expressies kunnen ook worden gebruikt om gegevens te valideren, zoals het controleren of een e-mailadres een geldig formaat heeft of dat een wachtwoord aan bepaalde criteria voldoet.

Reguliere expressies kunnen ingewikkeld lijken, omdat ze verschillende speciale karakters en symbolen bevatten. Elk symbool heeft een speciale betekenis en wordt gebruikt om bepaalde patronen te beschrijven. Bijvoorbeeld, "\d" komt overeen met een cijfer en "\w" komt overeen met een letter of cijfer.

Het begrijpen van reguliere expressies kan nuttig zijn omdat het je in staat stelt om geavanceerde zoek- en vervangbewerkingen uit te voeren, tekstpatronen te analyseren en gegevens te valideren. Er zijn verschillende programmeertalen en tools die reguliere expressies ondersteunen, waaronder Python, JavaScript en vele andere.

Hoewel reguliere expressies krachtig zijn, kan het even duren om ze volledig onder de knie te krijgen. Het is belangrijk om te oefenen en te experimenteren met verschillende patronen om vertrouwd te raken met hun gebruik en functionaliteit.

# Opdracht

Gebruik de volgende Python code als template.

```
import re


def validate_phone_number(phone_number):
    pattern = r"......"
```

```
    if re.match(pattern, phone_number):
        return True
    return False

# Voorbeeldgebruik:
phone_number = input("Voer een 06-nummer in: ")
if validate_phone_number(phone_number):
    print("Het ingevoerde nummer is geldig.")
else:
    print("Het ingevoerde nummer is ongeldig.")
```

Op de plaats van de puntjes maak jij een reguliere expressie.

Lees de tekst goed dan kom je een eind.

Zorg er ook voor dat je nummer begint met 06, dus het eerst wat je moet vinden in een valide nummer is 06. Omdat aan te geven in een regex kan je een ^ gebruiken. ^06 betekent dus "*begint met*"

Het einde van het nummer moet eindigen met 10 cijfers, dus na de 10 cijfers mag niets meer komen. Een eind teken is de $.

Dus ^06\d{2}$ betekent; *begin met 06 en eindig met een decimaal nummer van precies 2 posities* .

Maak de code af en plaats een regex die een 06-nummer valideert.

Plaats daarvoor op de regel met

`pattern = r"....."`

op de plaats van de puntjes de reguliere expressie en test je code!

# Inleveren

- Schermafdruk waarin je laat zien dat je code werkt; geef een juist 06-nummer en een paar foutieve nummers die bijna goed lijken.
- Aangepaste code met jouw naam in commentaar.

--

# Hexadecimale getallen herkennen

Stel we willen valideren of iets een hexadecimaal getal is.

Een hexadecimaal getal is een getal dat bestaat uit de cijfers 0 t/m 9 en A t/m F.

9A is dus een valide hexadecimaal getal, maar 1G niet.

Met de volgende code wordt een hexadecimaal getal herkend:

```
import re

def is_valid_hexadecimaal(hexadecimaal):
    # Reguliere expressie om een geldig hexadecimaal getal te valideren
    patroon = r'^[0-9A-F]+$'

    # Controleer of de opgegeven string overeenkomt met het patroon
    if re.match(patroon, hexadecimaal):
        return True
    else:
        return False

# Voorbeeldgebruik:
hex_nummer1 = "1A2F"
hex_nummer2 = "G3B8"
print(is_valid_hexadecimaal(hex_nummer1))  # Output: True
print(is_valid_hexadecimaal(hex_nummer2))  # Output: False
```

De reguliere expressie werkt als volgt:

De reguliere expressie werkt als volgt:

| Teken | Uitleg |
| --- | --- |
| ^ | Dit betekent dat we een match willen maken vanaf het begin van de string. |

| | |
|---|---|
| [ ] | De rechte haakjes geven aan dat we een groep gaan maken, een groep is een groep karakters waarmee de tekst moet worden gevalideerd. |
| [0-9A-F] | Dit betekent dat we een 0 t/m.9 of een A t/m F willen matchen. |
| + | Het plus teken betekent dat we hetgeen dat tussen [] staat 1 of meer keer willen matchen. |
| $ | Betekent dat we tot aan het einde van de string willen matchen. |

# Begin en eind ( ^en $ )

Stel dat we het begin en eind teken weglaten, dus dat we de reguliere expressie veranderen in:

```
[0-9A-F]+
```

Dan werkt die nog steeds, maar dan wordt bijvoorbeeld QQQ1FXXX ook gevalideerd. Dit is zo omdat er een match is namelijk 1F matched.

Stel we veranderen de expressie in:

```
^[0-9A-F]+
```

Dan valideert QQQ1FXXX niet meer, maar 1FXXX wel.

Met:

```
^[0-9A-F]+$
```

Valideert alleen 1F nog maar.

# Opdracht

Pas de code aan zodat je een derde hexadecimaal getal test.

hex_nummer3 = "13b8"

print(is_valid_hexadecimaal(hex_nummer3))

Met de code die je hebt, wordt dit hexadecimale getal ***niet*** gevalideerd. Er staat namelijk en b (kleine letter) in en die heeft geen match met de reguliere expressie.

Pas de reguliere expressie aan zodat ook hexadecimale getallen met een kleine letter worden gevalideerd. Dus 12b8, of aa of Cb moeten allemaal worden gevalideerd.

# Inleveren

1. Aangepaste code.

--

# Nog meer voorbeelden

Even terugkijken.....wat hebben we tot nu toe gedaan.

Tot nu toe hebben we teksten gevalideerd. We hebben gekeken of een string voldoet aan een bepaald patroon.

We hebben bijvoorbeeld bepaald of een getal een valide 06-nummer is, hiervoor gebruikte we de expressie:

| | |
|---|---|
| `^06\d{8}$` | begin met 06 en dan precies 8 decimalen de d staat voor decimalen en {8} staat voor 8 posities |

We hebben verder bepaald of een string een hex getal is met de reguliere expressie:

| | |
|---|---|
| `^[0-9A-F]+$` | begin met een 0,1,2,3,4,5,6,7,8,9,0,A,B,C,D,E,F en herhaal dat 1 of meer keren tot het eind. |

Als we willen weten of er ergens in de tekst een 06-nummer staat dan kunnen we de ^ en $ weglaten. Er wordt dan niet meer van het begin naar het einde gezocht, maar gekeken of er ergens een rij karakters is dat voldoet aan de expressie.

In de laatste reguliere expressie zie je een plus. In reguliere expressies hebben de symbolen `?`, `+` en `*` verschillende betekenissen en worden ze gebruikt om het aantal herhalingen van een patroon in de tekst te bepalen.

| **Teken** | **Betekenis** | **Voorbeeld** | |
|---|---|---|---|
| | als je niets plaats dan moet het element 1x voorkomen. | A | Alleen A voldoet |
| + | een plus betekent 1 of meer keren | AB+ | AB en ABBB voldoen |
| * | een * betekent 0, 1 of meer keren | AB*C | AC en ABC of ABBBC voldoen |
| ? | een ? betekent 0 of 1 keer | AB?C | AC en ABC voldoen |

Voor het teken kan des een teken staan of een groep tekens, je kunt ook een . Gebruiken.

| | |
|---|---|
| ABC+ | A komt 1 keer voor dan een B en dan één of meer keren een C. *ABCCC* is dus goed, *AABC* niet. |
| [A-C]+ | A,B of C komen 1 of meer keren voor. *ABCAABBAA* is dus goed en A ook. |

| | |
|---|---|
| [A-C]{5,6} | A,B, of C en dat minimaal 5 en maximaal 6 keer. *CBBCBA* is dus goed, maar *CBBCBAC* niet omdat dat een karakter te veel is. |
| .* | Elk karakter (any) komt 0 of meer keren voor (alles matched dus). |

En dan kun je nog ^ of $ gebruiken.

| Expressie | Uitleg | Voorbeeld van een match |
|---|---|---|
| ^ABC | De tekst begint met ABC, daarna mag alles | ABC zijn de beginletters van het alfabet. |
| ^AB*C | De tekst begint met een A dan 0,1, of meer B's en dan een C, daarna mag alles | AC wordt gebruikt als afkorting voor airconditioner |
| ^A*B* | De tekst mag alleen 0,1 of meer A;s bevatten gevolgd door 0,1 of meer B's. | BB staat for bed & breakfast |
| ^[A-Z]+$ | De tekst bestaat alleen uit hoofdletters en er moet er minimaal één voorkomen. | KIJK UIT |

De validatie in python doe je met het commando:

```
re.match(expressie, text)
```

# Opdracht

Jij krijgt nu een tabel met een expressie. De kolom uitleg en match is leeg. Jij moet die kolommen vullen. Leg uit wat de expressie doet en bepaal of de tekst een match is. Probeer de expressie uit met de code van de vorige opgave.

| Expressie | Tekst | Match? | Uitleg in eigen woorden (waarom wel of geen match) |
|---|---|---|---|
| expressie= r'[0-9]+^' | Mijn nummer is: 06 - 1234 1234 | | |
| expressie= r'[0-9]{8,10}^' | Mijn nummer is: 06 - 1234 1234 | | |
| expressie= r'$[A-Z]+' | Mijn nummer is: 06 - 1234 1234 | | |
| expressie= r'max' | Maximale temperatuur is 12 graden | | |
| expressie= r'[max]' | x-men | | |

| | | | |
|---|---|---|---|
| expressie= r'[k-mK-M]' | Dit is een voorbeeld | | |
| expressie= r'$[k-mK-M]' | Matig je alcohol gebruik! | | |
| expressie= r'$[k-mK-M]' | Gaat de KLM failliet? | | |
| expressie= r'^[a-kA-K]+$' | De kakkerlak kakt | | |
| expressie= r'$[k-mK-M]$' | De kakkerlak kakt erg veel | | |

Test alle reguliere expressies en zet in de kolom match of het voorbeeld een matched met de reguliere expressie.

Gebruik de code van de vorige opdracht om je antwoorden te controleren.

# Inleveren

1. PDF met jouw naam en de tabel waarin je in de laatste kolom met *true* of *false* (wel of geen match) aangeeft en daarachter een uitleg geeft.
2. De code waarmee je de voorbeelden hebt gecontroleerd.

--

# Twee spaties

Als we een tekst hebben en we willen controleren of we ergens een spatie te veel hebben ingetypt dan zoeken we naar twee of meer spaties

Dat gaat met het volgende programma.

```
import re


def heeft_dubbele_of_meer_spaties(tekst):
    # Definieer een reguliere expressie om te zoeken naar twee of meer spaties
    patroon = r"\s{2,}"
    return not re.search(patroon, tekst)



def main():
    invoer = input("Voer een tekst in: ")
    if heeft_dubbele_of_meer_spaties(invoer):
        print("De tekst bevat geen twee of meer spaties na elkaar.")
    else:
        print("De tekst bevat twee of meer spaties na elkaar.")



if __name__ == "__main__":
    main()
```

Je ziet dat \s in een reguliere expressie staat voor een spatie. We hadden ook gezien dat \d staat voor een nummer.

{2,} betekent 2 of meer.

Je zou de reguliere expressie ook a anders kunnen schrijven, bijvoorbeeld:

\s\s+

Dit is een spatie gevolgd door 1 of meer spaties. Dus

\s{2,} is hetzelfde als \s\s+

In plaats van controleren of een tekst twee spaties of meer achter elkaar bevat, kan je de tekst ook aanpassen, zodat alle meer dan 1 spaties worden vervangen door één spatie.

Dus soms gebruik je een \ als teken dat het volgende karakter iets speciaals betekent. We kennen de volgende speciale karakters.

| Teken | Betekenis | Alternatief |
|---|---|---|
| \s | een spatie of eigenlijk een 'white space' (= spatie of tab) | |
| \d | een plus betekent 1 of meer keren | [0-9]+ |
| \w | een woord (a-z, A-Z en underscore) | [A-Za-z_]+ |
| \\ | een backslash | |
| \. | een punt | |

# Zoek en vervang

Met het python commando

```
re.sub(r'expressie', 'string', tekst)
```

Met dit commando wordt alles in de variabel tekst dat voldoet aan de reguliere expressie 'expressie' vervangen door string.

# Opdracht

Maak een python programma dat in een tekst alle plaatsen waar mer dan 1 spatie achter elkaar staan vervangt in precies één spatie.

Gebruik daarvoor het commando re.sub voor zoek en vervang.

Test je programma uit en maar daarvoor 3 teksten;

1. één correcte tekst (geen dubbele spaties)
2. één teksts met op drie plaatsten een dubbele spatie en

3. één tekst met op één plaats 6 spaties achter elkaar.

# Inleveren

1. Jouw Python programma met jouw drie testen.
2. Jouw output in een screenshot.

# Twee of meer hoofdletters

## Opdracht

Maak een python programma dat in een tekst zoekt of er ergens meer dan één hoofdletter achter elkaar staat.

Gebruik hiervoor een regex.

Dus:

"*Dit is een Voorbeeld*" zou moeten worden goedgekeurd, maar

"*Het gehaalde cijfer is NVT*", zou moeten worden afgekeurd.

Test je programma uit op tenminste twee juiste teksten en twee onjuiste teksten.

Maak je eigen voorbeelden!

## Inleveren

1. De Python code waarin je dus minimaal 4 teksten test en het resultaat laat zien.
2. Een screen dump waarin je laat zien dat je de code uitvoer en waarin je de resultaten laat zien.

# Hoofdletter aan begin van de string

## Opdracht

Maak een python programma een tekst valideert (goedkeurt) als de tekst met precies één hoofdletter begint.

Gebruik hiervoor een regex.

"*Dit is een Voorbeeld*" zou moeten worden goedgekeurd, en

"*Het gehaalde cijfer is NVT*", zou moeten worden goedgekeurd, maar

"dit is een ander Voorbeeld", zou moeten worden afgekeurd.

Test je programma uit op tenminste twee juiste teksten en twee onjuiste teksten.

Maak je eigen voorbeelden!

## Inleveren

1. De Python code waarin je dus minimaal 4 teksten test en het resultaat laat zien.
2. Een screen dump waarin je laat zien dat je de code uitvoer en waarin je de resultaten laat zien.

# HTML parsing

Met een hele eenvoudige regex kan je controleren of elke html <table> wordt afgesloten met een </table>.

Dit kan met het volgende Python programma.

```
import re


def validate_tables_in_file(filename):
    with open(filename, 'r') as file:
        html_text = file.read()

    # Zoek naar alle geopende tabel tags
    opening_tags = re.findall('<table>', html_text)
    # Zoek naar alle gesloten tabel tags
    closing_tags = re.findall('</table>', html_text)

    # Controleer of het aantal geopende en gesloten tags hetzelfde is
    if len(opening_tags) == len(closing_tags):
        return True
    else:
        return False


print(validate_tables_in_file('your_file.html'))  # Vervang 'your_file.html' met de naam van je bestand
```

Maak een HTML test bestand en test de code uit.

# Opdracht

Je maakt twee aanpassingen aan de code.

## Aanpassing 1

Pas de code zodat je kan controleren of je alle  elementen hebt gesloten. Maak twee voorbeeld bestanden; één juiste en één onjuiste waarmee je laat zien dat je code werkt. Gebruik tenminste 6 div elementen in deze voorbeelden.

## Aanpassing 2

Zorg ervoor dat de code de naam van het bestand afdrukt en daarachter of dit bestand goed is.

# Inleveren

1. De aangepaste code.
2. Jouw eigen voorbeeld bestanden.
3. Een screendump waarin je laat zien dat je de code hebt uitgevoerd met de resultaten.

# Headers uit HTML 1

Neem de code van het volgende voorbeeld over.

```
import re

string = "De kat in de hoed wist dat de vleermuis daar was."
pattern = "de"

matches = re.findall(pattern, string, flags=re.I)  # De vlag re.I maakt de zoekopdracht case-insensitive

print(matches)
```

De output zijn alle delen van de tekst die de regex matchen. Probeer maar!

Stel je hebt een webpagina. Dus de string is een webpagina, bijvoorbeeld als volgt:

```
string="<!DOCTYPE html><html><head><title>Mijn webpagina</title></head><body><h1>Mijn eerste kop</h1><p>Dit is een paragraaf.</p><h1>Mijn tweede kop</h1><p>Dit is nog een paragraaf.</p></body></html>"
```

Zet deze string nu in de code en vervang daarmee regel 3 (de regel van "De kat....").

Pas nu de reguliere expressie zodat alle <h1> stukken uit de tekst worden gehaald.

# Opdracht

Lees de tekst hierboven!

Pas de voorbeeldcode aan.

## Aanpassing 1

Zet op regel 3 de html-code zoals hierboven is gegeven.

## Aanpassing 2

Verander de reguliere expressie (regex) zodat je alle h1 elementen uit de HTML-code laat afdrukken.

# Inleveren

1. Aangepaste code
2. Een screendump waarin je laat zien dat je code werkt.
   Let op, in de screendump moet je ook de **datum en tijd van Windows** zichtbaar maken!

# Headers uit HTML 2 (haakjes)

In de vorige opdracht heb je geleerd hoe je headers uit een HTML tekst kan halen.

Verander de code nu zodat je alleen de header afdrukt.

In de vorige opdracht kreeg je als resultaat bijvoorbeeld:

> *<h1>Dit is een header</h1>*

Pas de code nu aan zodat je alleen

> *Dit is een header*

afdrukt.

Deze keer moet je zelf op zoek gaan hoe dat moet. Gebruik het internet en eventueel ChatGPT. Er zijn meerdere oplossingen mogelijk, maar je moet de door jouw gekozen oplossing wel begrijpen en kunnen uitleggen.

# Tip

Met haakjes kan je aangeven welk deel van de reguliere expressie (regex) je in het resiltaat wilt zien.

Als je in het pattern van

```
re.findall(pattern, html_text)
```

haakjes gebruikt om aan te geven welk deel je als resultaat wilt, dan krijg je in het resultaat alleen dat deel dat tussen haakjes staat.

## voorbeeld

```
text = "Vandaag is het 20-07-2023. Mijn verjaardag is op 01-05-1985 en mijn jubileum is op 12-10-2010."
pattern = r"\d{2}-\d{2}-(\d{4})"
matches = re.findall(pattern, text)
```

In dit voorbeeld wordt gezocht naar een patroon dd-dd-dddd en alleen de laatste dddd wordt als match gevonden.

Er worden in dit voorbeeld dus drie matches gevonden; 2023, 1985 en 2010.

# Opdracht

Pas de code van de vorige opdracht aan zodat je de <h1> en </h1> niet meer afdrukt als onderdeel van het resultaat. Zie hierboven een voorbeeld.

Gebruik internet, ChatGPT ,.... maar zorg dat je de oplossing begrijpt en kan uitleggen.

# Inleveren

1. Aangepaste code en zet in commentaar in je code een uitleg in je eigen woorden van hoe het werkt.
2. Laat zien in een screen dump dat je code werkt.

# Challenge

Bij webontwikkeling en web scraping kan het vaak handig zijn om links uit een HTML-pagina te extraheren. Dit kan gedaan worden met een HTML-parser zoals BeautifulSoup, maar als oefening willen we dat je dit probleem oplost met behulp van reguliere expressies.

# Opdracht

Schrijf een Python-script dat een HTML-bestand leest en alle URL's uit de `<a>`-tags (hyperlinks) extraheert en afdrukt. De URL's zijn de waarden van het `href` attribuut in `<a>`-tags. Hier is een voorbeeld van een `<a>`-tag:

```
<a href="https://www.voorbeeld.nl">Een voorbeeld link</a>
```

In dit voorbeeld zou je script "https://www.voorbeeld.nl" moeten extraheren en afdrukken.

# Tips

1. Je kunt de `re.findall()` functie gebruiken om alle overeenkomsten van een patroon in een string te vinden.
2. Vergeet niet om rekening te houden met de verschillende manieren waarop een attribuut kan worden geschreven in HTML. Bijvoorbeeld, het kan worden omringd door dubbele aanhalingstekens (`href="url"`), enkele aanhalingstekens (`href='url'`), of het kan geen aanhalingstekens hebben (`href=url`).

# Inleveren

1. Jouw eigen test bestand waarin meerdere links staan. De links staan tussen "" en '' (dubbele quote en enkele quotes).
2. Jouw code.
3. Een screendump waarin je laat zien dat jouw code werkt.
   Laat in de screendump ook de datum en tijd van Windows zien (dit is wat je rechtsonder in beeld ziet).

Succes!

# Samenvatting Theorie

## Basis

| Teken | Uitleg |
|---|---|
| ^ | Dit betekent dat we een match willen maken vanaf het begin van de string. |
| [ ] | De rechte haakjes geven aan dat we een groep gaan maken, een groep is een groep karakters waarmee de tekst moet worden gevalideerd. |
| [0-9A-F] | Dit betekent dat we een 0 t/m.9 of een A t/m F willen matchen. |
| + | Het plus teken betekent dat we hetgeen dat tussen [] staat 1 of meer keer willen matchen. |
| $ | Betekent dat we tot aan het einde van de string willen matchen. |

## Place holders

| Teken | Betekenis | Voorbeeld | |
|---|---|---|---|
| | als je niets plaats dan moet het element 1x voorkomen. | A | Alleen A voldoet |
| + | een plus betekent 1 of meer keren | AB+ | AB en ABBB voldoen |
| * | een * betekent 0, 1 of meer keren | AB*C | AC en ABC of ABBBC voldoen |
| ? | een ? betekent 0 of 1 keer | AB?C | AC en ABC voldoen |
| . | een . betekent elk (any) karakter | .* | Elke regel tekst voldoet hieraan. |

## Groeperen

| Expressie | Uitleg |
|---|---|
| ABC+ | A komt 1 keer voor dan een B en dan één of meer keren een C. *ABCCC* is dus goed, *AABC* niet. |
| [A-C]+ | A,B of C komen 1 of meer keren voor. *ABCAABBAA* is dus goed en A ook. |

| | |
|---|---|
| [A-C]{5,6} | A,B, of C en dat minimaal 5 en maximaal 6 keer. *CBBCBA* is dus goed, maar *CBBCBAC* niet omdat dat een karakter te veel is. |
| .* | Elk karakter (any) komt 0 of meer keren voor (alles matched dus). |

# Begin en eind

| Expressie | Uitleg | Voorbeeld van een match |
|---|---|---|
| ^ABC | De tekst begint met ABC, daarna mag alles | ABC zijn de beginletters van het alfabet. |
| ^AB*C | De tekst begint met een A dan 0,1, of meer B's en dan een C, daarna mag alles | AC wordt gebruikt als afkorting voor airconditioner |
| ^A*B* | De tekst mag alleen 0,1 of meer A;s bevatten gevolgd door 0,1 of meer B's. | BB staat for bed & breakfast |
| ^[A-Z]+$ | De tekst bestaat alleen uit hoofdletters en er moet er minimaal één voorkomen. | KIJK UIT |

# Escape karakters (\)

| **Teken** | **Betekenis** | **Alternatief** |
|---|---|---|
| \s | een spatie of eigenlijk een 'white space' (= spatie of tab) | |
| \d | een plus betekent 1 of meer keren | [0-9]+ |
| \w | een woord (a-z, A-Z en underscore) | [A-Za-z_]+ |
| \\ | een backslash | |
| \. | een punt | |

# Haakjes

Met haakjes geef je aan wat je in het resultaat wil.

| Expressie | Uitleg |
| --- | --- |
| \d{2}-\d{2}-(\d{4}) | matched een datum (dd-dd-dddd) en geeft alleen het jaar (dddd) terug. |
| \.\s(.?) | matched het einde van de zin (punt gevolgd door spatie) en geeft het eerste karakter van de zin weer. |

--