Working with regexes

We should talk about strings

Before we talk about regexes, we should talk about how PHP and Python display and treat strings.

You maybe know these two lines in PHP will print a different result:

```php
$text1 = "Hello!\nHow are you?\n";
$text2 = 'Hello!\nHow are you?\n';

echo $text1;
echo $text2;
```

This will print:

```
Hello!
How are you?
Hello!\nHow are you?\n
```

`$text1` was declared with double quotes (`"`) and `$text2` with single quotes (`'`).
With double quotes, PHP interprets some escape characters as special sequences (`\n` meaning a new line).
With single quotes, it does not take this into account and just displays what you typed.

If you try the same in Python:

```python
text_1 = "Hello!\nHow are you?\n"
text_2 = 'Hello!\nHow are you?\n'

print(text_1)
print(text_2)
```

It prints this instead:

```
Hello!
How are you?

Hello!
How are you?
```

So the single or double quotes do not affect Python strings (except the fact that you will need to either escape `'` or `"` with a `\` inside your string if needed).

Then how can we achieve the same result as the single-quoted text in PHP?

Python has special strings called *raw strings* allowing you to do that. Just prefix your string with `r` (for *raw*):

```python
text_1 = "Hello!\nHow are you?\n"
text_2 = r"Hello!\nHow are you?\n"

print(text_1)
print(text_2)

# Prints:
# Hello!
# How are you?
#
# Hello!\nHow are you?\n
```

You can use `r` with single or double quotes, it does not matter.

They can be compared to "normal" strings:

```python
'Hello!' == r'Hello!' # True

'Hello!\nHi!' == r'Hello!\nHi!' # False
```

These raw strings are particularly interesting when you have to deal with complex strings having special characters in them.

Regexes are the perfect case for them, and that's why you should use raw strings in all your regex by default to avoid any surprises. They're also used a lot in path and route parameters for example.

Finally, let's talk about regexes!

I will not talk about what regexes are or how to use the metacharacters and quantifiers because as far as I know, they work the same way as in PHP.

If you need to learn about them or refresh your memory you can read this **nice article** (and its **second part**).

Everything related to regexes lives in the built-in `re` module (for **R**egular **E**pressions).

Let's see how to search for a match:

```python
import re

# Here I use a raw string with r'...'
# but a usual string ('...') would work too
if re.search(r'^B.+d$', 'Bored'):
    print('It\'s a match!')
else:
    print('Sorry, not found :(')

# "It's a match!"
```

```php
if (preg_match('/^B.+d$/', 'Bored')) {
    echo 'It\'s a match!';
} else {
    echo 'Sorry, not fond :(';
}

// "It's a match!"
```

One big difference here is that we use a delimiter in our PHP string (`/`) but not in Python.

If you need to add flags you can add a third argument. In PHP we add these flags after the end delimiter.

```python
import re

# Ignore case
re.search(r'^h.+o$', 'Hello', re.I)

# Include new lines when using `.`
re.search(
    r'^Hello.+Bye$',
    'Hello\nNice to meet you!\nBye',
    re.DOTALL
)

# If you need to combine flags:
re.search(
    r'^Hello.+Bye$',
    'hello\nnice to meet you!\nbye',
    re.DOTALL|re.I
)
```

```php
preg_match('/^h.+o$/i', 'Hello');
preg_match('/^Hello.+Bye$/s', "Hello\nNice to meet you!\nBye");
preg_match('/^hello.+bye$/si', "Hello\nNice to meet you!\nBye");
```

If you want to know more about all the available flags, you can read **this**.

The `search` method returns a `Match` object if the pattern was found and `None` otherwise.

```python
import re

# <re.Match object; span=(0, 5), match='Bored'>
re.search(r'^B.+d$', 'Bored')

# None
re.search(r'^B.+d$', 'foo')
```

```php
preg_match('/^H.+o$/', 'Hello'); // 1

preg_match('/^H.+o$/', 'foo'); // 0
```

Accessing groups

To fetch groups:

```python
import re

txt = 'Lois is a beautiful name'
m = re.search(r'^.*(L\w+s)\s.*(n\w+e).*$', txt)

# The group 0 is the entire match
m.group(0) # 'Lois is a beautiful name'

m.group(1) # 'Lois'

m.group(2) # 'name'

# If you try to access a non existing group it throws an error
# IndexError: no such group
m.group(3)

# If you give multiple group indexes, it returns a tuple
# ('Lois', 'name')
m.group(1, 2)
```

```php
$txt = 'Lois is a beautiful name';
preg_match('/^.*(L\w+s)\s.*(n\w+e).*$/', $txt, $matches);

$matches[0]; // 'Lois is a beautiful name'
$matches[1]; // 'Lois'
$matches[2]; // 'name'
```

If you use named groups:

```python
import re

txt = 'Well, 34 and 8734 are my favorite numbers'
m = re.search(
    r'(?P<first_number>\d+)\s.*\s(?P<second_number>\d+)',
    txt
)

m.group('first_number') # '34'
m.group('second_number') # '8734'
```

```php
$txt = 'Well, 34 and 8734 are my favorite numbers';
preg_match(
    '/(?P<first_number>\d+)\s.*\s(?P<second_number>\d+)/',
    $txt,
    $matches
);

$matches['first_number']; // '34'
$matches['second_number']; // '8734'
```

To fetch all the groups:

```python
import re

txt = 'Well, 34 and 8734 are my favorite numbers'
m = re.search(
    r'(?P<first_number>\d+)\s.*\s(?P<second_number>\d+)',
    txt
)

m.groups() # ('34', '8734')

m.groupdict() # {'first_number': '34', 'second_number': '8734'}
```

```php
$txt = 'Well, 34 and 8734 are my favorite numbers';
preg_match(
    '/(?P<first_number>\d+)\s.*\s(?P<second_number>\d+)/',
    $txt,
    $matches
);

/*
   [
     0 => "34 and 8734",
     "first_number" => "34",
     1 => "34",
     "second_number" => "8734",
     2 => "8734",
   ]
*/
$matches;
```

The full documentation of the `Match` object can be found **here**.

The `findall` method

The `findall` method returns a simple list of all the matches, and an empty list if nothing was found:

```python
import re
regex = r'\w+@\w+\.\w+'
txt_1 = 'Both emails lorem@gmail.com and foo@hotmail.com are valid'

# ['lorem@gmail.com', 'foo@hotmail.com']
re.findall(regex, txt_1)

txt_2 = 'This string contains no email'

# []
re.findall(regex, txt_2)
```

```php
$regex = '/\w+@\w+\.\w+/';
$txt1 = 'Both emails lorem@gmail.com and foo@hotmail.com are valid';

preg_match_all($regex, $txt1, $matches); // 2

$matches; // [ ['lorem@gmail.com', 'foo@hotmail.com'] ]

$txt2 = 'This string contains no email';

preg_match_all($regex, $txt2, $matches); // 0

$matches; // [ [  ] ]
```

Others

The equivalent of PHP's `preg_split` is `re.split`:

```python
import re

txt = 'Sure, here are some words'

# ['Sure', 'here', 'are', 'some', 'words']
re.split(r'[\s,]+', txt)
```

```php
$txt = 'Sure, here are some words';

// ['Sure', 'here', 'are', 'some', 'words']
preg_split('/[\s,]+/', $txt);
```

The equivalent of PHP's `preg_replace` is `re.sub`:

```python
import re

txt = 'The customer n°34098340498 paid their bill n°979879'

# 'The customer n°**** paid their bill n°****'
re.sub(r'\d+', '****', txt)
```

```php
$txt = 'The customer n°34098340498 paid their bill n°979879';

// 'The customer n°**** paid their bill n°****'
preg_replace('/\d+/', '****', $txt);
```

You can also give a callback as the replacement value. Then it will be called for every replacement with a `Match` object as a parameter:

```python
import re

def anonymize(match):
    length = len(match.group(0))

    return '*' * length

txt = 'The customer n°34098340498 paid their bill n°979879'

# 'The customer n°********** paid their bill n°******'
re.sub(r'\d+', anonymize, txt)
```