Regular Expressions in 3 Languages

Lyell Read, CS321H / F2020

Starting Off Easy

Regular Expression Password Checker in Python 3

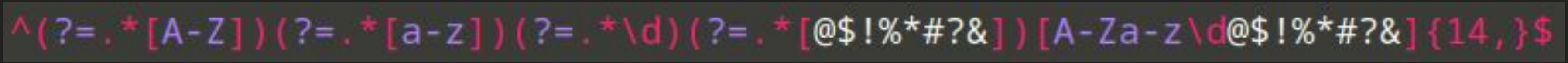
Details

- Uses the re module for Python 3 (import re)
    - Pre-installed with Python 3
- Accepts Perl/PCRE-like regular expressions
- Two usage methods
    - Can re.compile(pattern) and then compiled.match(string)
    - Can run re.fullmatch(pattern, string)
- Returns either None or a Match object.

Password Requirements

- At least 14 characters long
- Contains at least one from each
    - Lowercase
    - Uppercase
    - Number
    - Special
        - Special characters are @$!%*#?&

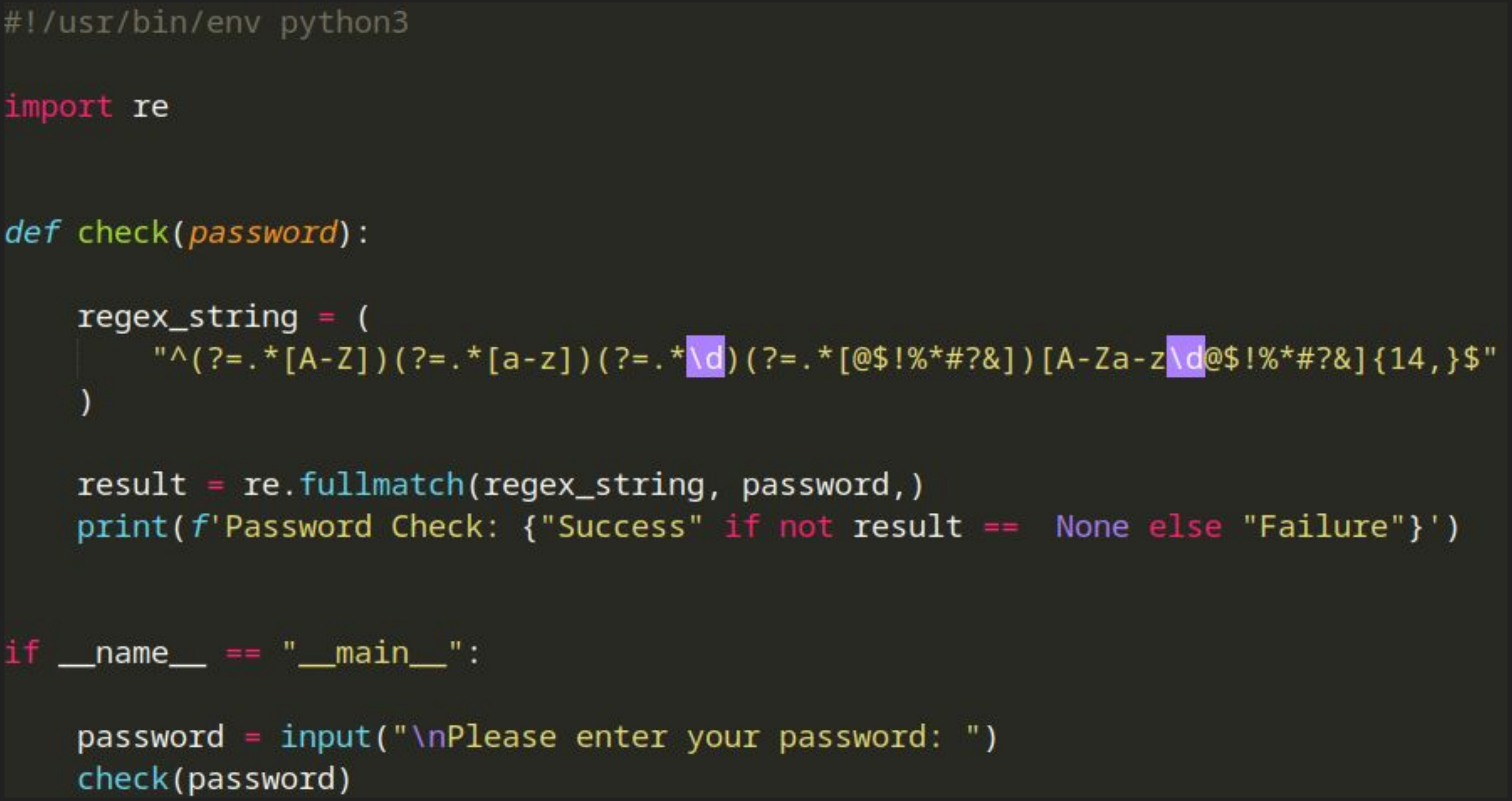
Password Regular Expression



```
^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!%*#?&]{14,}$
```

- The ˆ matches start of string
- The `(?=.*[A-Z])` performs a forward search for A-Z in string
- The `(?=.*[a-z])` performs a forward search for a-z in string
- The `(?=.*\d)` performs a forward search for 0-9 in string
- The `(?=.*[@$!%*#?&])` performs a forward search for special in string
- The `[A-Za-z\d@$!%*#?&]{14,}` matches all valid, and asserts that characters must total 14 or more
- The $ matches end of string.

The Code



```
#!/usr/bin/env python3

import re


def check(password):

    regex_string = (
        "^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!%*#?&]{14,}$"
    )

    result = re.fullmatch(regex_string, password,)
    print(f'Password Check: {"Success" if not result ==  None else "Failure"}')


if __name__ == "__main__":

    password = input("\nPlease enter your password: ")
    check(password)
```

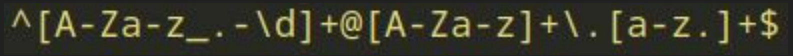
Shell Script?

Regular Expression Email Checker in a Shell Script

Details

- Uses the [[ ]] operator
    - Native to most modern shells, i.e. zsh, ksh, bash
- Accepts Perl/PCRE-like regular expressions
- Returns either 1 or 0.

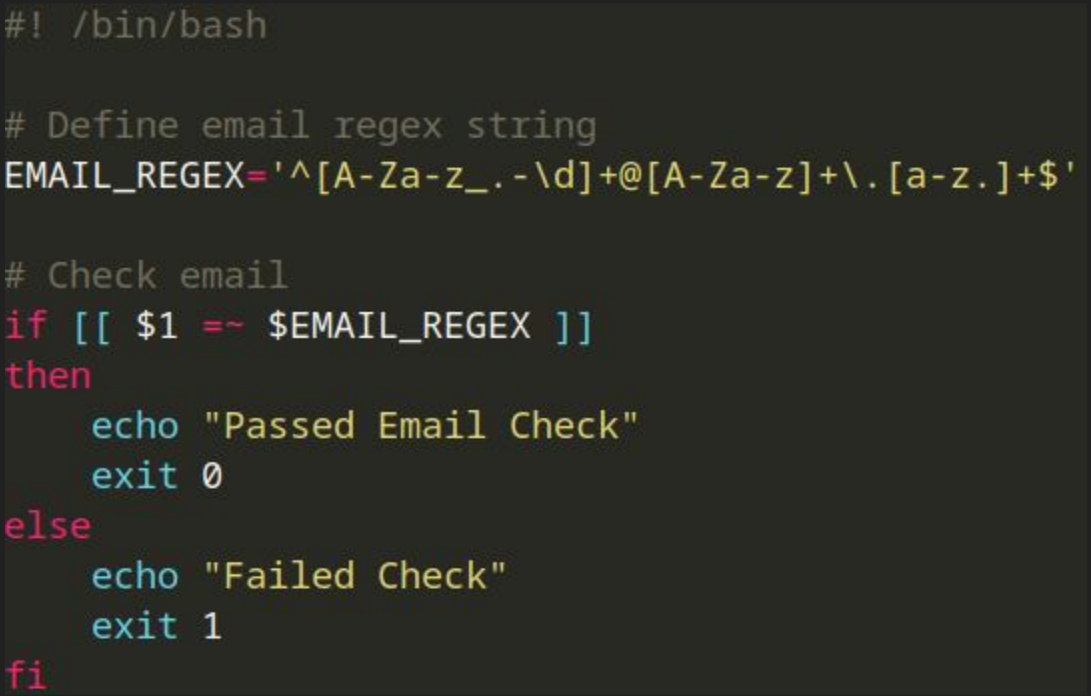
Email Regular Expression



```
^[A-Za-z_.-\d]+@[A-Za-z]+\.[a-z.]+$
```

- The ˆ matches start of string
- The `[A-Za-z_.-\d]+` matches any number of numbers and letters and ._-
- The @ matches a single @
- The `[A-Za-z]+` matches any number of upper or lowercase alphabetic characters
- The `\.` matches a period, must be escaped with \, as the . is a regular expression special character.
- The `[a-z.]+` matches the domain extension, i.e com, co.nz
- The $ matches end of string

The Code



```bash
#! /bin/bash

# Define email regex string
EMAIL_REGEX='^[A-Za-z_.-\d]+@[A-Za-z]+\.[a-z.]+$'

# Check email
if [[ $1 =~ $EMAIL_REGEX ]]
then
    echo "Passed Email Check"
    exit 0
else
    echo "Failed Check"
    exit 1
fi
```

Regex in C?!

Regular Expression to Accept Full Names in C

Details

- Uses the POSIX C <regex.h> library
    - **Not** part of ANSI C, however present on most-all *nix.
- Accepts POSIX [extended] regular expressions
- Usage method
    - regcomp() (compile) pattern
    - regexec() (execute) pattern against string
    - regerror() (get info about match error)
- regexec() returns:
    - Either returns 0 on success
    - REG_NOMATCH on no match
    - Other error codes interpretable with regerror() on error

Full Name Requirements

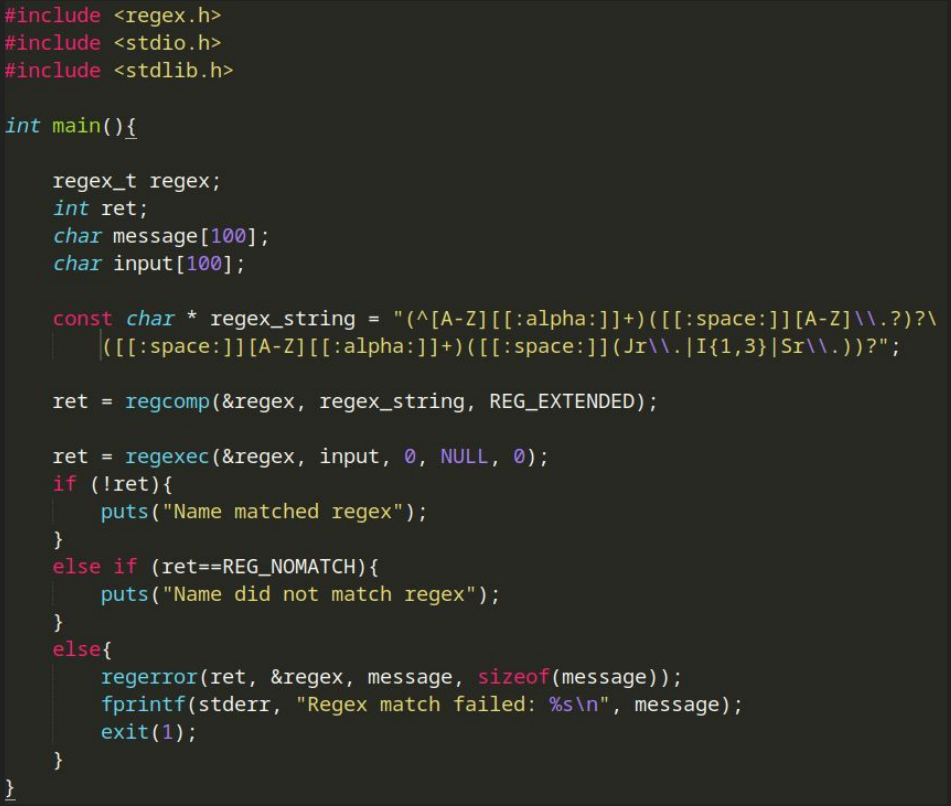
- Full Names such as:
    - John Appleseed
    - John A. Appleseed
    - John A Appleseed
    - John A. Appleseed Sr.
    - John Appleseed II
    - ...

Password Regular Expression

```
(^[A-Z][[:alpha:]]+)([[:space:]][A-Z]\\.?)?([[:space:]][A-Z][[:alpha:]]+)([[:space:]](Jr\\.|I{1,3}|Sr\\.))?'
```

- The `(^[A-Z][[:alpha:]]+)` matches one or more capitalized, alphabetic words at the start of a string
- The `([[:space:]][A-Z]\\.?)?` matches zero or one of a space, followed by a capital letter, optionally followed by a period
- The `([[:space:]][A-Z][[:alpha:]]+)` matches one or more last names with a preceding space, and a capital letter at the start of the name
- The `([[:space:]](Jr\\.|I{1,3}|Sr\\.))?` matches either one or zero of {Jr., Sr., I, II, III}

The Code



```c
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>

int main(){

    regex_t regex;
    int ret;
    char message[100];
    char input[100];

    const char * regex_string = "(^[A-Z][[:alpha:]]+)([[:space:]][A-Z]\\.?)?\
        ([[:space:]][A-Z][[:alpha:]]+)([[:space:]](Jr\\.|I{1,3}|Sr\\.))?";

    ret = regcomp(&regex, regex_string, REG_EXTENDED);

    ret = regexec(&regex, input, 0, NULL, 0);
    if (!ret){
        puts("Name matched regex");
    }
    else if (ret==REG_NOMATCH){
        puts("Name did not match regex");
    }
    else{
        regerror(ret, &regex, message, sizeof(message));
        fprintf(stderr, "Regex match failed: %s\n", message);
        exit(1);
    }
}
```

Works Cited

- https://en.wikipedia.org/wiki/Regular_expression#Perl_and_PCRE
- https://docs.python.org/3/library/re.html#re.compile
- https://docs.python.org/3/library/re.html#re.fullmatch
- https://stackoverflow.com/a/21456918/8704864
- https://serverfault.com/questions/52034/what-is-the-difference-between-double-and-single-squarebrackets-in-bash
- https://www.gnu.org/software/bash/manual/bash.html#Conditional-Constructs
- https://pubs.opengroup.org/onlinepubs/7908799/xsh/regex.h.html
- https://stackoverflow.com/a/1085120/8704864
- https://stackoverflow.com/questions/1085083/regular-expressions-in-c-examples
- https://www.regextester.com/99203
- https://www.regular-expressions.info/posixbrackets.html