# Regular Expressions

## Problem

Write a Python code to check if the given mobile number is valid or not. The conditions to be satisfied for a mobile number are:

a) Number of characters must be 10

b) All characters must be digits and must not begin with a ‘0’

# Validity of Mobile Number

| Input | Processing | Output |
|---|---|---|
| A string representing a mobile number | Take character by character and check if it valid | Print valid or invalid |

## Test Case 1

- abc8967891
- Invalid
- Alphabets are not allowed

# Test Case 2

- 440446845
- Invalid
- Only 9 digits

## Test Case 3

- 0440446845
- Invalid
- Should not begin with a zero

## Test Case 4

- 8440446845
- Valid
- All conditions statisfied

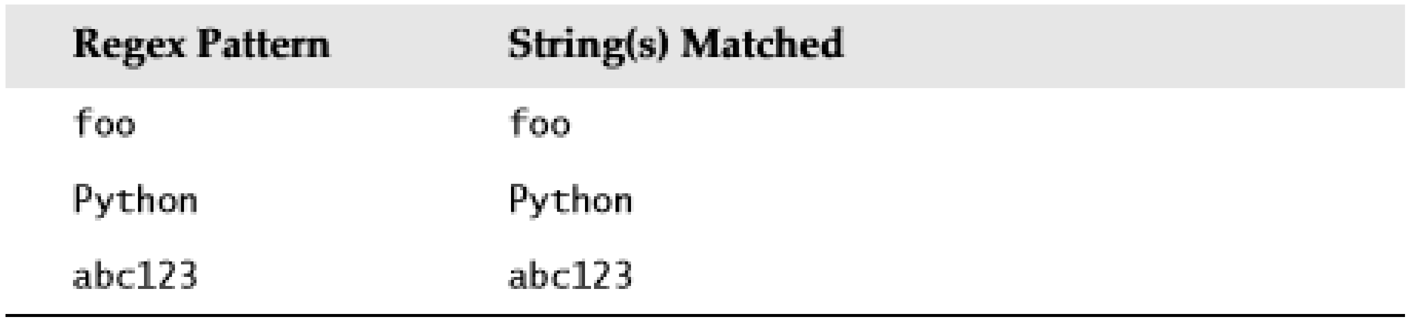
## Python code to check validity of mobile number (Long Code)

```
import sys
number = input()
if len(number)!=10:
    print ('invalid')
    sys.exit(0)
if number[0]==‘0’:
     print ('invalid')
    sys.exit(0)
for chr in number:
    if chr.isalpha():
        print ('invalid')
        break
else:
    print('Valid‘)
```

- Manipulating text or data is a big thing
- If I were running an e-mail archiving company, and you, as one of my customers, requested all of the e-mail that you sent and received last February, for example, it would be nice if I could set a computer program to collate and forward that information to you, rather than having a human being read through your e-mail and process your request manually.

- Another example request might be to look for a subject line like “ILOVEYOU,” indicating a virus-infected message, and remove those e-mail messages from your personal archive.
- So this demands the question of how we can program machines with the ability to look for patterns in text.
- Regular expressions provide such an infrastructure for advanced text pattern matching, extraction, and/or search-and-replace functionality.
- Python supports regexes through the standard library re module

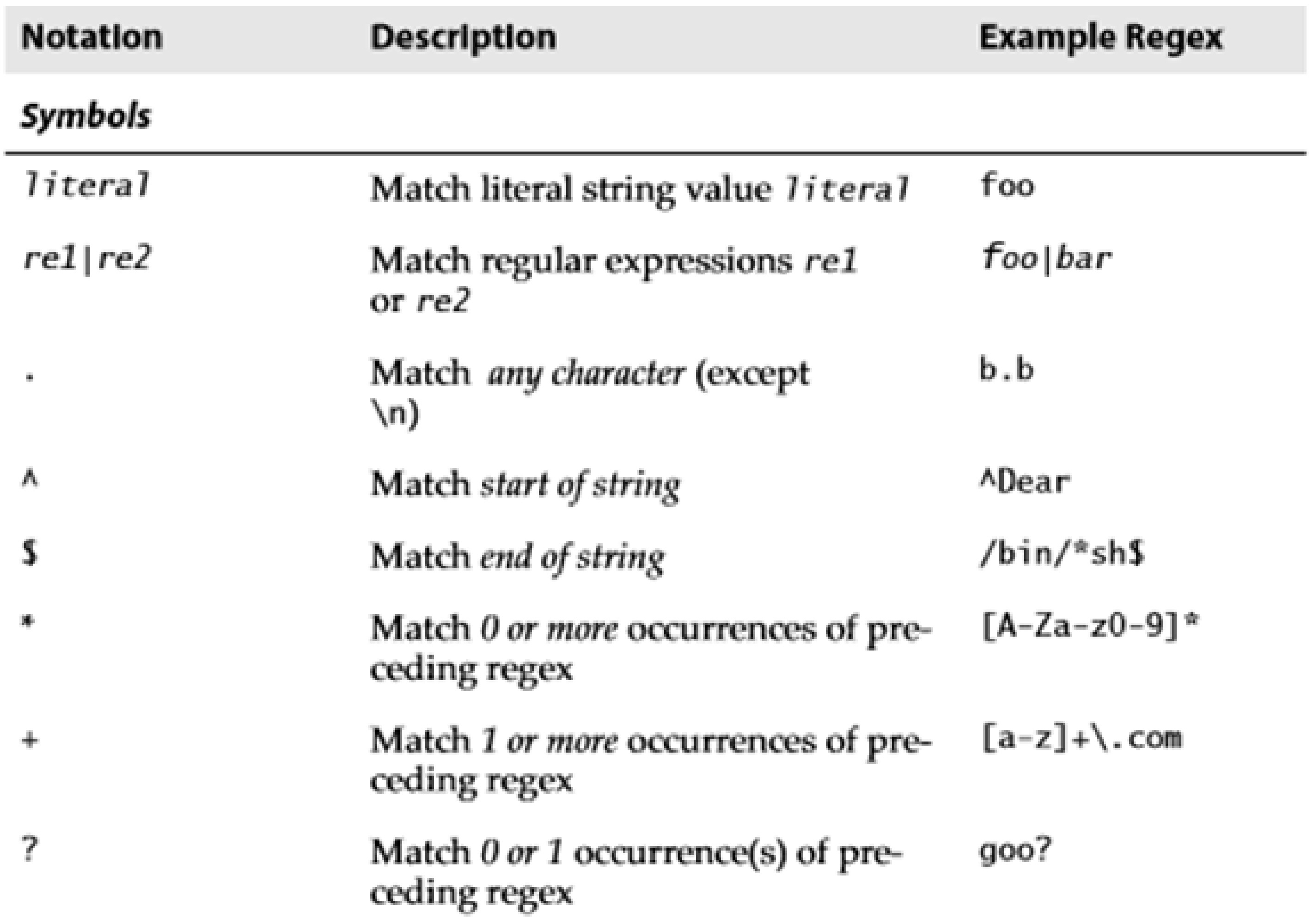
- regexes are strings containing text and special characters that describe a pattern with which to recognize multiple strings.
- Regexs without special characters



| Regex Pattern | String(s) Matched |
|---|---|
| foo | foo |
| Python | Python |
| abc123 | abc123 |

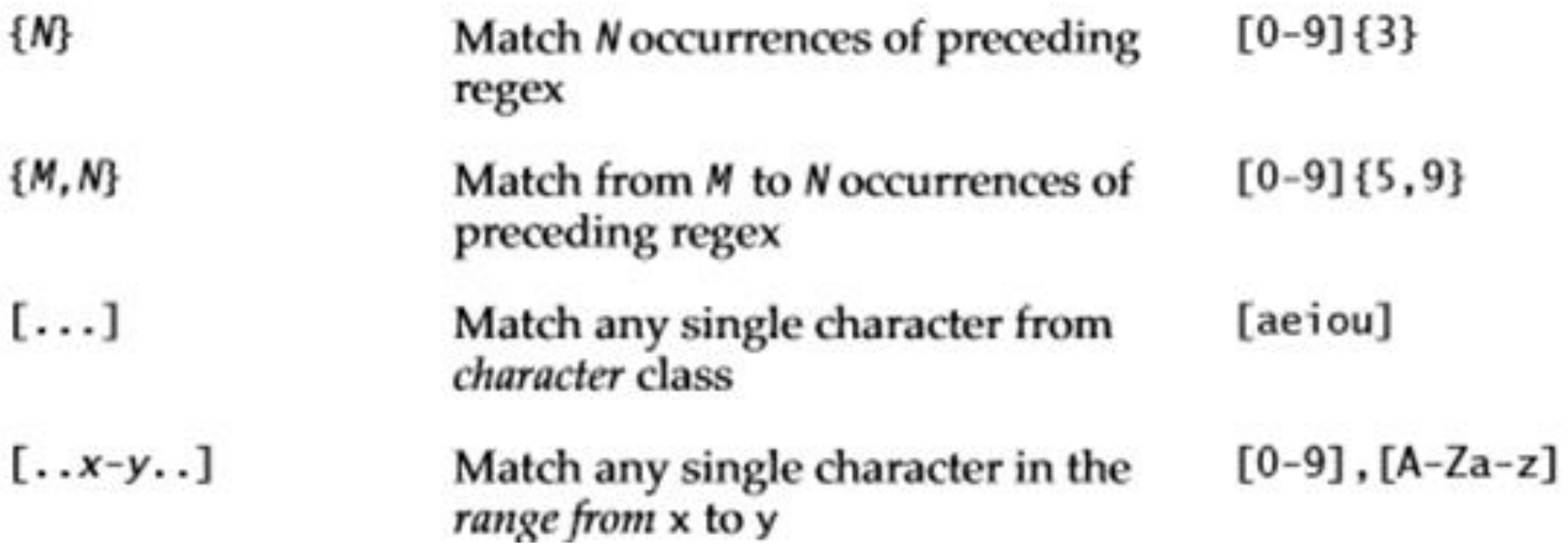
- These are simple expressions that match a single string
- Power of regular expressions comes in when special characters are used to define character sets, subgroup matching, and pattern repetition

# Special Symbols and Characters



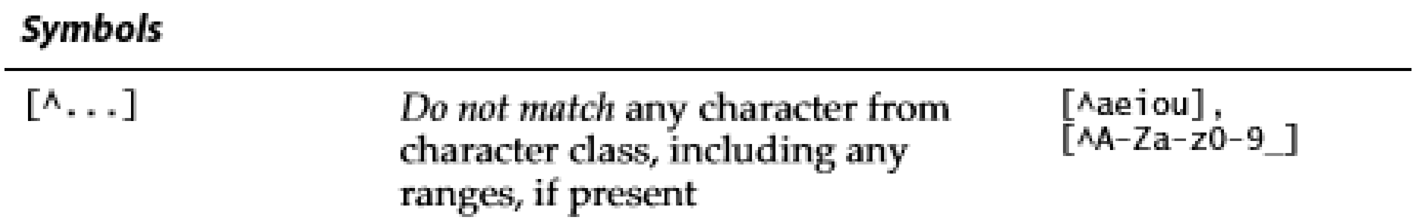
| Notation | Description | Example Regex |
|---|---|---|
| ***Symbols*** | | |
| *literal* | Match literal string value *literal* | foo |
| *re1\|re2* | Match regular expressions *re1* or *re2* | *foo\|bar* |
| . | Match *any character* (except \n) | b.b |
| ^ | Match *start of string* | ^Dear |
| $ | Match *end of string* | /bin/*sh$ |
| * | Match *0 or more* occurrences of preceding regex | [A-Za-z0-9]* |
| + | Match *1 or more* occurrences of preceding regex | [a-z]+\.com |
| ? | Match *0 or 1* occurrence(s) of preceding regex | goo? |

## Special Symbols and Characters



| | | |
|---|---|---|
| {*N*} | Match *N* occurrences of preceding regex | [0-9]{3} |
| {*M*,*N*} | Match from *M* to *N* occurrences of preceding regex | [0-9]{5,9} |
| [...] | Match any single character from *character* class | [aeiou] |
| [..*x*-*y*..] | Match any single character in the *range from* x to y | [0-9],[A-Za-z] |

## Special Symbols and Characters



| Symbols | | |
|---|---|---|
| `[^...]` | *Do not match* any character from character class, including any ranges, if present | `[^aeiou]`, `[^A-Za-z0-9_]` |

## Matching Any Single Character (.)

- dot or period (.) symbol (letter, number, whitespace (not including “\n”), printable, non-printable, or a symbol) matches any single character except for \n
- To specify a dot character explicitly, you must escape its functionality with a backslash, as in “\.”

| Regex Pattern | Strings Matched |
|---|---|
| `f.o` | Any character between "f" and "o"; for example, `fao`, `f9o`, `f#o`, etc. |
| `..` | Any pair of characters |
| `.end` | Any character before the string `end` |

```
import re
if re.match("f.o","fooo"):
    print("Matched")
else:
    print("Not matched")
```

Output:

Prints matched

Since it searches only for the pattern ‘f.o’ in the string

```
import re
if re.match("f.o$","fooo"):
    print("Matched")
else:
    print("Not matched")
```

Check that the entire string starts with ‘f’, ends with ‘o’ and contain one letter in between

```
import re
if re.match("..","fooo"):
    print("Matched")
else:
    print("Not matched")
```

```
Matched
```

Two dots matches any pair of characters.

```
import re
if re.match("..$","fooo"):
    print("Matched")
else:
    print("Not matched")
```

```
Not matched
```

Including a ‘$’ at the end will match only strings of length 2

```
import re
if re.match(".end","bend"):
    print("Matched")
else:
    print("Not matched")
```

```
Matched
```

The expression used in the example, matches any character for ‘.’

```
import re
if re.match(".end","bends"):
    print("Matched")
else:
    print("Not matched")
```

Prints Matched

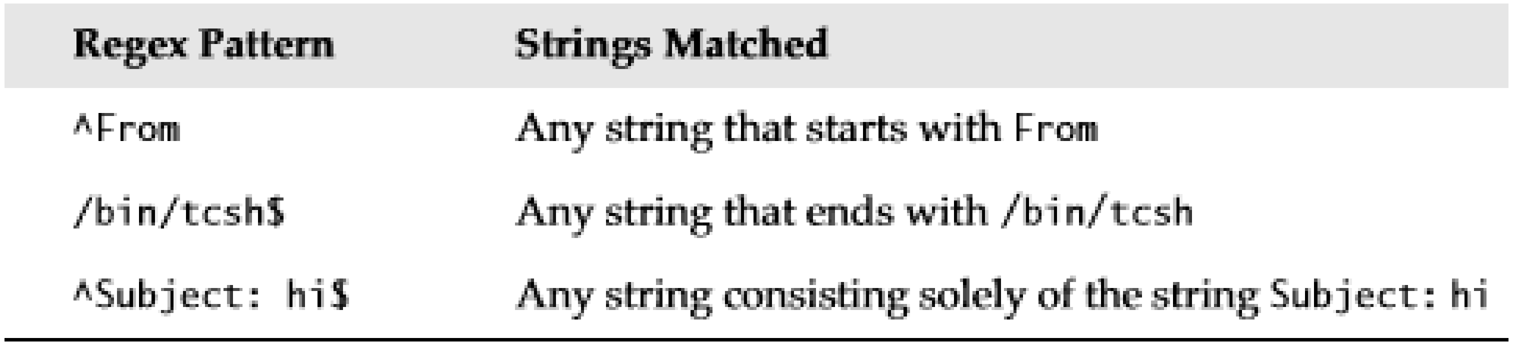
```
import re
if re.match(".end$","bends"):
    print("Matched")
else:
    print("Not matched“)
```

Prints Not matched - $ check for end of string

## Matching from the Beginning or End of Strings or Word Boundaries (^, $)

^ - Match beginning of string

$ - Match End of string



| Regex Pattern | Strings Matched |
|---|---|
| `^From` | Any string that starts with `From` |
| `/bin/tcsh$` | Any string that ends with `/bin/tcsh` |
| `^Subject: hi$` | Any string consisting solely of the string `Subject: hi` |

if you wanted to match any string that ended with a dollar sign, one possible regex solution would be the pattern .*\$$

## But not sufficient

Check whether the given register number of a VIT student is valid or not.

Example register number – 15bec1032

Register number is valid if it has two digits

Followed by three letters

Followed by four digits

## Denoting Ranges (-) and Negation (^)

- brackets also support ranges of characters
- A hyphen between a pair of symbols enclosed in brackets is used to indicate a range of characters;
- For example A–Z, a–z, or 0–9 for uppercase letters, lowercase letters, and numeric digits, respectively

| Regex Pattern | Strings Matched |
|---|---|
| `z.[0-9]` | “z” followed by any character then followed by a single digit |
| `[r-u][env-y][us]` | “r,” “s,” “t,” or “u” followed by “e,” “n,” “v,” “w,” “x,” or “y” followed by “u” or “s” |
| `[^aeiou]` | A non-vowel character (Exercise: why do we say “non-vowels” rather than “consonants”?) |
| `[^\t\n]` | Not a TAB or \n |
| `["-a]` | In an ASCII system, all characters that fall between ‘"’ and “a,” that is, between ordinals 34 and 97 |

## Multiple Occurrence/Repetition Using Closure Operators (*, +, ?, {})

- special symbols *, +, and ?, all of which can be used to match single, multiple, or no occurrences of string patterns
- Asterisk or star operator (*) - match zero or more occurrences of the regex immediately to its left
- Plus operator (+) - Match one or more occurrences of a regex

- Question mark operator (?) - match exactly 0 or 1 occurrences of a regex.
- There are also brace operators ({}) with either a single value or a comma-separated pair of values. These indicate a match of exactly N occurrences (for {N}) or a range of occurrences; for example, {M, N} will match from M to N occurrences

## Code to check the validity of register number

```
import re
register= input()
if re.match("^[1-9][0-9][a-zA-Z][a-zA-Z][a-zA-Z][0-
   9][0-9][0-9][0-9]$",register):
   print("Matched")
else:
   print("Not matched“)
```

^ - denote begin (Meaning is different when we put this symbol inside the square bracket)

$ - denote end

| Regex Pattern | Strings Matched |
|---|---|
| `[dn]ot?` | "d" or "n," followed by an "o" and, at most, one "t" after that; thus, `do`, `no`, `dot`, `not`. |
| `0?[1-9]` | Any numeric digit, possibly prepended with a "0." For example, the set of numeric representations of the months January to September, whether single or double-digits. |
| `[0-9]{15,16}` | Fifteen or sixteen digits (for example, credit card numbers. |

## Refined Code to check the validity of register number

{n} – indicate that the pattern before the braces should occur n times

```
import re
register= input()
```

```
if re.match("^[1-9][0-9][a-zA-Z]{3}[0-
    9]{4}$",register):
    print("Matched")
else:
    print("Not matched")
```

## Check validity of Mobile Number (Shorter Code)

```
import re
number = input()
if re.match(‘[^0][0-9]{9}',number):
    print('valid')
else:
    print('invalid')
```

Bug: Will also accept a843338320

## Check validity of Mobile Number (Shorter Code)

```
import re
number = input()
if re.match(‘[1-9][0-9]{9}',number):
    print('valid')
else:
    print('invalid')
```

## Check validity of PAN card number with RE

```
import re
pan=input()
if len(pan) < 10 and len(pan) > 10 :
    print ("PAN Number should be 10 characters")
    exit
```

```
elif re.search("[^a-zA-Z0-9]",pan):
    print ("No symbols allowed, only
   alphanumerics")
    exit
elif re.search("[0-9]",pan[0:5]):
    print ("Invalid - 1")
    exit
```

```
elif re.search("[A-Za-z]",pan[5:9]):
    print ("Invalid - 2")
    exit
elif re.search("[0-9]",pan[-1]):
    print ("Invalid - 3")
    exit
else:
    print ("Your card "+ pan + " is valid“)
```

Python read all input as string

In some cases it is necessary to check if the value entered is an integer

We can check it using regular expressions

Rules for an integer

optionally begin with a negative sign include ^ symbol

first digit must be a number other than zero

may be followed zero to any number of digits

string must end with it so add $ symbol

```
import re
register= input()
#optionally begin with a negative sign include ^ symol
#first digit must be a number other than zero
# may be followed zero to any number of digits
# string must end with it so add $ symbol
```

```
if re.match("^\-?[1-9][0-9]*$",register):
    #'\' is added in front of '-' to overcome its default
    meaning in REs
    print("Matched")
else:
    print("Not matched")
```

## Rules for an integer or a floating point value

optionally begin with a negative sign include ^ symbol

first digit must be a number other than zero

may be followed zero to any number of digits

string must end with it so add $ symbol

Optionally followed by a ‘.’

Followed by zero or more digits

String ends here

```
import re
register= input()
if re.match("^\-?[1-9][0-9]*\.?[0-9]*$",register):
    # '.' can occur zero or one time followed by a
    digit occurred zero to infinite number of times
    print("Matched")
else:
    print("Not matched")
```