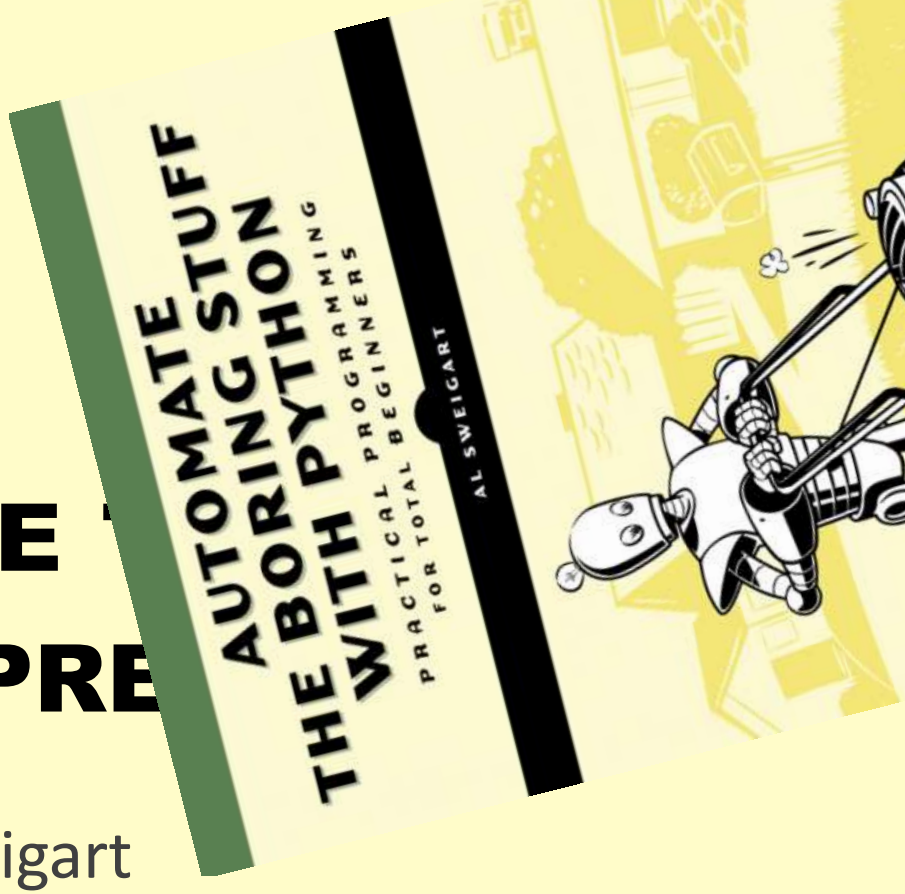
YES, IT'S TIME T
EGULAR EXPRE



@AlSweigart
(last name rhymes with "why dirt")

bit.ly/yesregex

Yes, it's time to learn regular expressions.

aka "regex"

415-555-0000

4,155,550,000

The Three Lines of Code You Need

```
import re
myRegex = re.compile('regex pattern')
mo = myRegex.search('haystack string')
print(mo.group())
```

The Four Lines of Code You Need

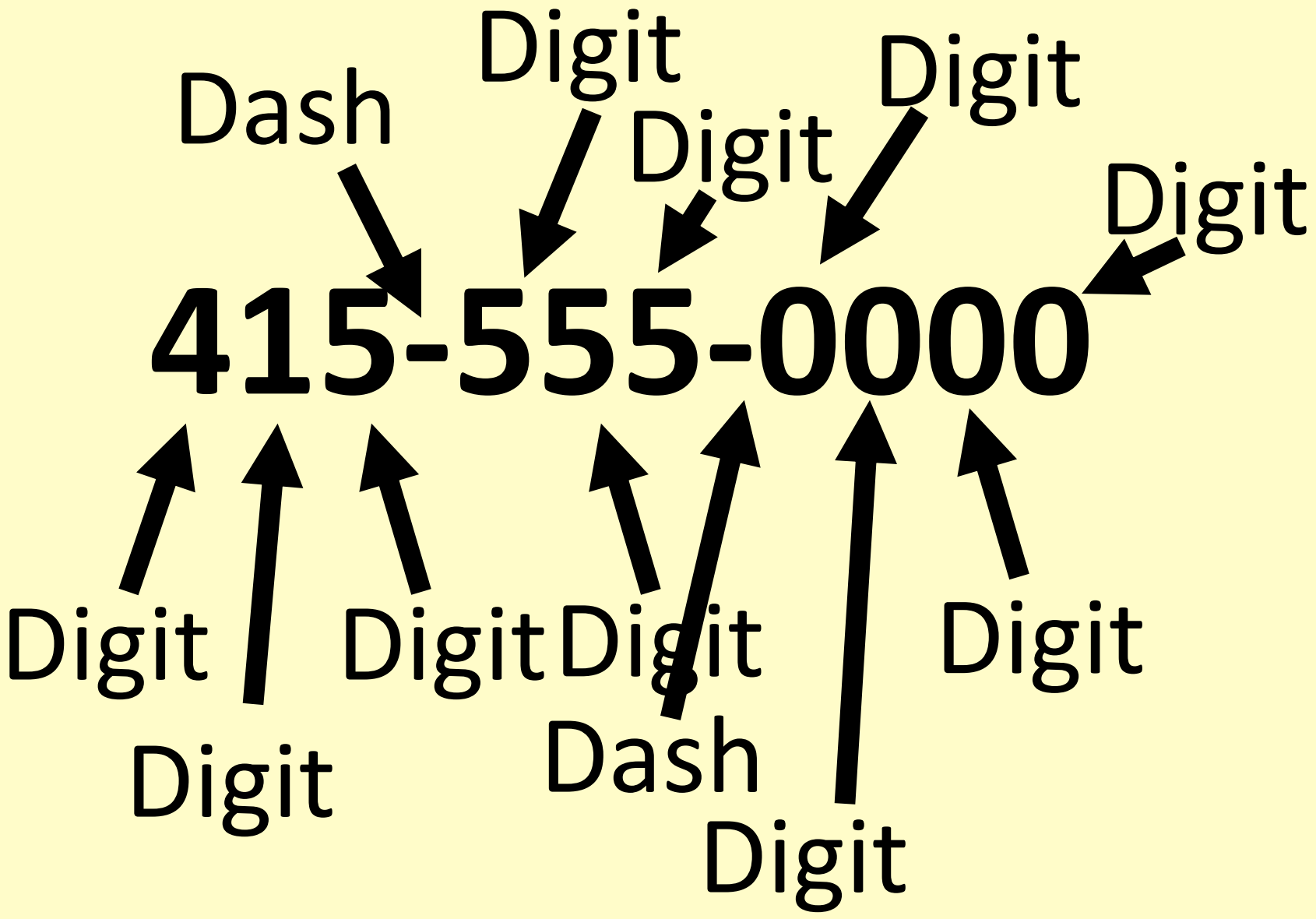
```
import re
myRegex = re.compile('regex pattern')
mo = myRegex.search('haystack string')
print(mo.group())
```

The Four Lines of Code You Need

```
import re
myRegex = re.compile('regex pattern')
mo = myRegex.search('haystack string')
print(mo.group())
```

bit.ly/yesregex

```
myRegex = re.compile('regex pattern')
```


Digit
Dash
Digit
Digit
Digit
Digit
415-555-0000
Digit
Digit
Digit
Digit
Digit
Digit
Dash
Digit

Making the Regex String

```
phoneRegex = re.compile(

                ''

                        )
```

Making the Regex String

```
phoneRegex = re.compile(

                '\d'

                        )
```

Making the Regex String

```
phoneRegex = re.compile(

                r'\d'

                        )
```

Making the Regex String

```
phoneRegex = re.compile(

                '\\d'

                        )
```

Making the Regex String

```
phoneRegex = re.compile(

                r'\d'

                        )
```

Making the Regex String

```
phoneRegex = re.compile(

                r'\d\d\d'

                        )
```

Making the Regex String

```
phoneRegex = re.compile(

                r'\d\d\d-'

                          )
```

Making the Regex String

```
phoneRegex = re.compile(

    r'\d\d\d-\d\d\d-\d\d\d\d'

                        )
```

Making the Regex String

```
>>> mo = phoneRegex.search("""Alice,
        My number is 415-730-0000.
        Call me when it's convenient.
                              -Bob""")
```

Making the Regex String

```
>>> mo = phoneRegex.search("""Alice,
      My number is 415-730-0000.
      Call me when it's convenient.
                          -Bob""")
>>> if mo is not None:
...     print(mo.group())
```

Making the Regex String

```
>>> mo = phoneRegex.search("""Alice,
      My number is 415-730-0000.
      Call me when it's convenient.
                          -Bob""")
>>> if mo is not None:
...     print(mo.group())

415-730-0000
```

Making the Regex String

```
>>> mo = phoneRegex.search("""Alice,
        My number is 415-730-0000.
        Call me when it's convenient.
                              -Bob""")
>>> if mo is not None:
...       print(mo.group())

415-730-0000
```

```
def isPhoneNumber(text):
  if len(text) != 12:
    return False
  for i in range(0, 3): # check area code
    if not text[i].isdecimal():
      return False
  if text[3] != '-':
    return False
  for i in range(4, 7): # check first 3 digits
    if not text[i].isdecimal():
      return False
  if text[7] != '-':
    return False
  for i in range(8, 12): # check last 4 digits
    if not text[i].isdecimal():
      return False
    return True

text = """Alice,
      My number is 415-730-0000.
      Call me when it's convenient.
                      -Bob"""
for i, _ in enumerate(text):
  if isPhoneNumber(text[i:i+12]):
    print(text[i:i+12])
```

Character Class

\d	Digit characters (numbers)
\w	Word characters (letters & numbers)
\s	Space characters (space, tab, \n)
\D	Non-digit
\W	Non-word
\S	Non-space

Create Character Classes

- Put characters inside []
- [aeiouAEIOU] Matches vowels
- [^aeiouAEIOU] Matches non-vowels
- [0-9a-zA-Z] Same as \w

Punctuation = Escape

. * ( ) ^

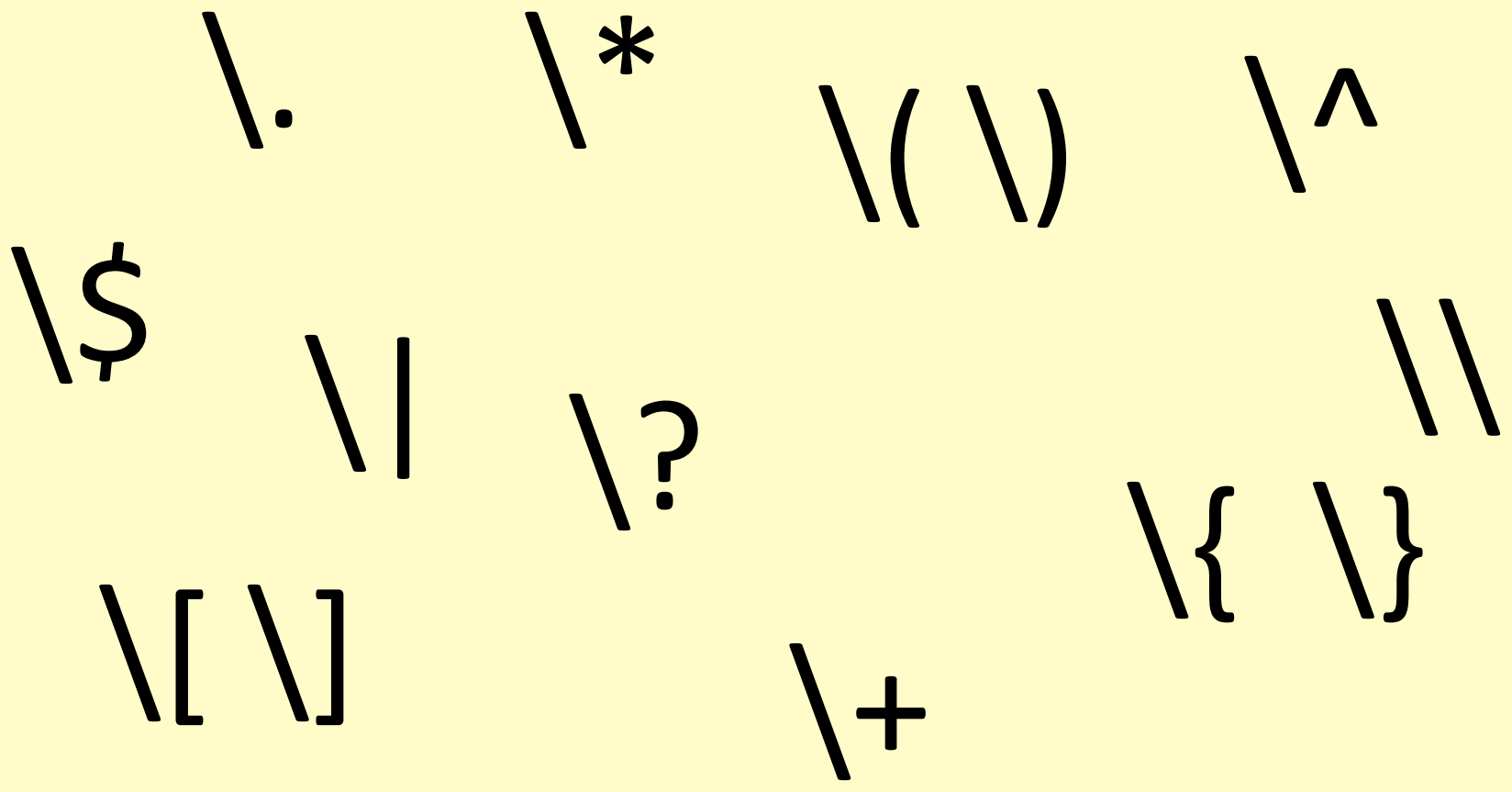
$ | ? \

{ }

[ ]

+

Punctuation = Escape



Create Character Classes

- Put characters inside []
- [aeiouAEIOU] Matches vowels
- [^ aeiouAEIOU] Matches non-vowels
- [0-9a-zA-Z] Same as \w
- [\(\)] Matches (or)

Specifying Quantity

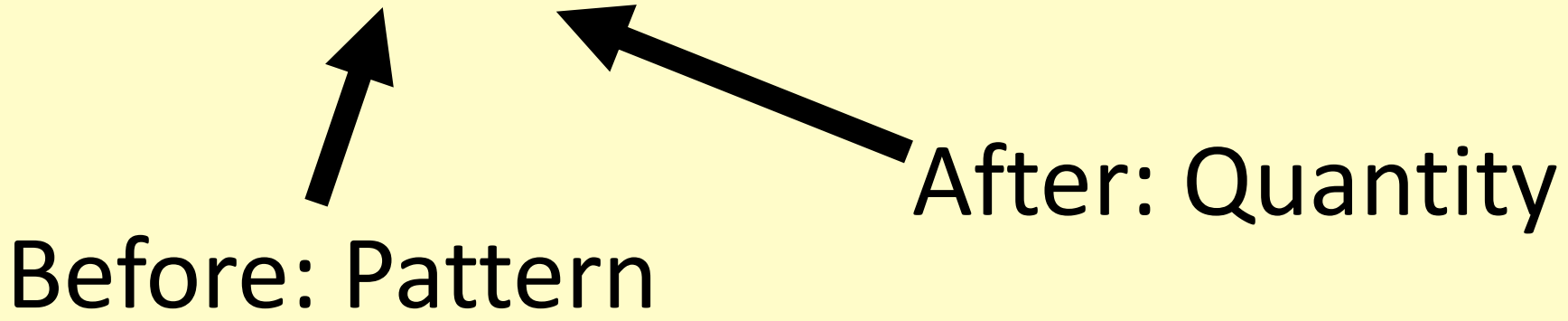
415-555-0000

\d\d\d-\d\d\d-\d\d\d\d

Specifying Quantity

415-555-0000

\d{3}-\d{3}-\d{4}



Specifying Quantity

- \d One digit
- \d**?** Zero or one digits
- \d***** Zero or more digits
- \d**+** One or more digits
- \d**{3}** Exactly 3 digits
- \d**{3,5}** Btwn 3 and 5 digits
- \d**{3,}** 3 or more digits

Specifying Quantity

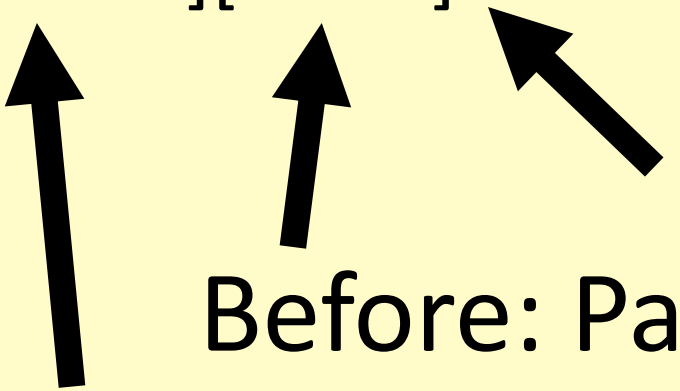
- \s One space
- \s**?** Zero or one space
- \s***** Zero or more space
- \s**+** One or more space
- \s**{3}** Exactly 3 space
- \s**{3,5}** Btwn 3 and 5 space
- \s**{3,}** 3 or more space

Specifying Quantity

- [aeiou] One vowel
- [aeiou]**?** Zero or one vowels
- [aeiou]***** Zero or more vowels
- [aeiou]**+** One or more vowels
- [aeiou]**{3}** Exactly 3 vowels
- [aeiou]**{3,5}** Btwn 3 and 5 vowels
- [aeiou]**{3,}** 3 or more vowels

Grouping

- Japanese letters are usually consonant-vowel combinations.
- **'sayonara' = sa ● yo ● na ● ra**
- [^aeiou][aeiou]+



After: Quantity

Before: Pattern

Just one of this pattern

Grouping

- [^aeiou][aeiou]+
- saaaaaaaaaaaaaa
- saoiaeueaoieuaio
- ([^aeiou][aeiou])+
- sasasasasasasasa
- sayonara

1,234,567,890 Example

- A regex for comma-formatted numbers:
- e.g. 1,234,567,890
- “One to three digits, followed by zero or more groups of comma-digit-digit-digit.”
- **Regex Buddy / Regex Tester**
- **http://pyregex.com/**

Alternatives with Pipe

- eggandspam
- eggbaconandspam
- eggbaconsausageandspam
- spameggspamspambaconandspam
- re.compile(r'(egg)+(bacon)+(sausage)+(and)+(spam)+')

Alternatives with Pipe

- Like [aeiou] but for words.
- egg OR bacon OR sausage OR and OR spam
- Use the | pipe to have alternative groups:
- re.compile(r'((egg)|(bacon)|(sausage)|(and)|(spam))+')

- spamspamspamspamspamspamspamspamspamspamspa
  mspamspamspamspamspamspamspamspamspamspams
  pamspamspamspamspamspamspamspamspamspamspa
  mspamspamspamspamspamspamspamspamspamspam

Match Anything

- The . means “any character except newline”
- The * means “zero or more”
- .* means “match whatever”
- .*? means “match the least of whatever”

Match Anything

- 'Looking for text <in between angle brackets>'
- re.compile('<.*?>')
- **'<TO SERVE HUMANS>'**

- re.compile('<.*>')
- **'<TO SERVE HUMANS> FOR DINNER>'**

- DUN DUN DUUUHN!!!

What Regexes Can't / Shouldn't Do

- **DON'T PARSE HTML WITH REGEX.**
- A regex for strong passwords.
  - Includes lowercase, uppercase, numbers, special character, at least 12 characters.
  - (Just use multiple regexes.)
- (Match(ing) ((nested) (parentheses.)))
  - (Regexes don't have variables or flow control!)
  - "A regex to match regex strings."

YES, IT'S TIME TO LEARN REGULAR EXPRESSIONS.

@AlSweigart

ymes with "why dirt")

ly/yesregex