

به نام خدا



پردازش زبان‌های طبیعی

گزارش تکلیف اول

ناشناس کردن اطلاعات شخصی متن

سید پوریا لقایی

مجتبی زمانی ایمنی

مهدی کافی

در این تمرین سعی کردیم با استفاده از **regular expression** ها تا حد ممکن اطلاعات شخصی افراد را در متون مخفی کنیم همچنین تلاش کردیم که به صورت بهینه، پیاده سازی را انجام دهیم. در ادامه کارهایی را که برای مخفی کردن هر کدام از اطلاعات انجام داده ایم توضیح خواهیم داد.

شماره تلفن ثابت


برای شماره تلفن، در ابتدا با جستجوهای که انجام دادیم، فرمت های متفاوت شماره تلفن ثابت را استخراج کردیم و همچنین پیش شماره شهرهای متفاوت را در متغیری ذخیره کردیم. سپس برای هر کدام از فرمت ها **regex** ای نوشتیم و همه را با یکدیگر **or** کردیم و از آن به عنوان پترن کلی شماره تلفن ثابت استفاده کردیم.

```
#phone_number_pattern
self.pre_phone_num =
"41|44|31|26|45|84|77|21|38|56|51|58|61|24|23|54|71|28|25|87|34|83|74|17|13|66|11|86|76|81|35"
self.phone_pattern = fr"\b(0({self.pre_phone_num})(\d{{8}}|\d{{5}})|1\d{{2}}|15\d{{2}}|18\d{{2}}|096\d{{2}}|[2-8](\d{{3}}|\d{{4}}))\b"
```

در تمامی پترن ها می خواهیم که رشته ای که با پترن، پیدا می شود پترن در ابتدا و انتهای رشته باشد. برای این منظور از **\b** استفاده کردیم. حال در اولین پترن، می خواهیم که رقم صفر دیده شود سپس یکی از کدهای شهرها آورده شود و سپس ۸ یا ۵ رقم دیده شود. پترن دوم برای شماره تلفن های ۵ رقمی است و همینطور تا انتها، برای حالت های ممکن شماره تلفن که از اینترنت استخراج کرده بودیم، پترن نوشتیم.

کد ملی

برای کد ملی، در ابتدا پترن آن را به صورت یک رشته شامل ارقام 0 تا 9 و به طول 10 تعریف کردیم. در اینجا نیز می خواهیم که شماره ملی با رشته ای یکسان سازی شود که پترن در ابتدا و انتهای آن باشد و از **\b** استفاده کردیم.



```
#national code
self.national_code = r"\b\d{10}\b"
```

حال توسط الگوریتم زیر معتبر بودن یا نبودن آن را بررسی کردیم:

- برای محاسبه رقم کنترل از روی سایر ارقام، هر رقم را در موقعیت آن ضرب کرده و حاصل را با هم جمع می کنیم.
- مجموع بدست آمده از مرحله یک را بر 11 تقسیم می کنیم.
- اگر باقیمانده کمتر از 2 باشد، رقم کنترل باید برابر باقی مانده باشد در غیر اینصورت رقم کنترل باید برابر یازده منهای باقی مانده باشد.

بدین صورت که اگر پترن یافت شد در الگوریتم بررسی می کند که آیا کد ملی معتبر است یا خیر.

```

def hide_national_code(self, text):
    def is_national_code(ncode):
        if len(ncode) != 10:
            return 0
        sum = 0
        ncode = str(ncode)
        for i in range(len(ncode)-1):
            sum += int(ncode[i])*(10-i)
        sum = sum % 11
        flag = 0
        if sum == 1 or sum == 2:
            if int(ncode[9]) == sum:
                flag = 1
        else:
            if 11-int(ncode[9]) == sum:
                flag = 1
        return flag
    margin = 0
    for matched in re.finditer(self.national_code, text):
        start, end = matched.span()
        if is_national_code(matched.group(0).strip()):
            text = text[:start-margin] + \
                "<کد-ملی-معتبر#" + text[end-margin:]
            margin += (end-start) - len("<کد-ملی-معتبر#")
        else:
            text = text[:start-margin] + \
                "<کد-ملی-نامعتبر#" + text[end-margin:]
            margin += (end-start) - len("<کد-ملی-نامعتبر#")
    return text

```

مثال : آیا کد 7731689951 یک کد ملی معتبر است؟

حاصل جمع ضرب ارقام 2 الی 10 را در موقعیت آنها محاسبه می کنیم

$$7 \times 10 + 7 \times 9 + 3 \times 8 + 1 \times 7 + 6 \times 6 + 8 \times 5 + 9 \times 4 + 9 \times 3 + 5 \times 2 = 313$$

$$313 \div 11 = 28, \text{Remainder} = 5$$

چون باقیمانده برابر 5 و بزرگتر مساوی 2 است پس باید رقم کنترل این کد برابر 6 (یازده منهای 5 برابر 6) باشد.

با دقت در کد متوجه می شویم که رقم کنترل ورودی برابر 1 است پس کد مورد نظر به عنوان یک کد معتبر قابل قبول نیست.

شماره تلفن همراه

برای شماره تلفن همراه ابتدا پترن آن را با استفاده از regex به صورت زیر می نویسیم.

```
#mobile_phone_pattern
self.mobile_pattern = r'\b09[0-39][0-9]-?[0-9]{3}-?[0-9]{4}\b'
return text
```

در این پترن مشخص شده است که ابتدا شماره تلفن باید با 09 آغاز شده و پس از آن باید یک عدد از 0 تا 39 قرار گیرد. سپس یک رقم دیگر آمده و بعد از آن می تواند یک علامت «-» قرار بگیرد یا نگیرد و بعد از آن یک سه رقمی دلخواه بار دیگر یک بار ظاهر شدن علامت «-» به دلخواه و بعد از آن یک چهار رقمی دلخواه قرار گیرد. تابع match کردن این پترن به صورت زیر است.

```
def hide_mobile_num(self, text):
    margin = 0
    for matched in re.finditer(self.mobile_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + \
            "<#شماره-تلفن-همراه>" + text[end-margin:]
        margin += (end-start) - len("<#شماره-تلفن-همراه>")
    return text
```

در این تابع ابتدا به دنبال این پترن با استفاده از finditer نوشته شده در text ورودی هستیم و پس از پیدا کردن این بخش از رشته ورودی، با استفاده از span آغاز و پایان این بخش آن را با "<#شماره-تلفن-همراه>" جایگزین خواهیم کرد.

نام شهرها

در این بخش برای یافتن نام شهرها از یک پایگاه داده که قابل به‌روزرسانی است در کنار یک پترن استفاده می‌شود. طبق شکل زیر ابتدا عناصر موجود در پایگاه داده را در یک لیست ذخیره کرده و در ابتدا و انتها **regex** نوشته شده از **\b** استفاده می‌کنیم که ابتدا و انتهای رشته را مشخص کند.

```
# city pattern
self.city_names = '|'.join(list(pd.read_csv('cities.csv')['City']))
self.city_name_pattern = fr"\b({self.city_names})\b"
```

تابع نوشته شده برای این بخش به صورت زیر است. در این تابع با استفاده از **finditer** ابتدا پترن مورد نظر در رشته وردی جستجو شده و پیدا می‌شود و سپس این بخش از رشته با استفاده از **span** آغاز و پایان الگوی پیدا شده با **<#نام-شهر>** جایگزین خواهد شد.

```
def hide_city(self, text):
    margin = 0
    for matched in re.finditer(self.city_name_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<#نام-شهر>" + text[end-
margin:]
        margin += (end-start) - len("<#نام-شهر>")
    return text
```

نام شرکت

در این بخش هم مانند نام شهر، ابتدا از یک پایگاه داده که قابل تغییر نیز است استفاده می‌شود. پترن نوشته شده برای آن به صورت زیر است. در این حالت نیز از **\b** برای مشخص کردن ابتدا و انتهای رشته استفاده می‌کنیم و در نهایت با کمک پایگاه داده مربوط به نام شرکت‌ها - که قابل تغییر و گسترش نیز هست - پترن مورد نظر را کامل می‌کنیم.

```
# company pattern
self.company_names = '|'.join(list(pd.read_csv('company.csv')['company']))
self.company_name_pattern = fr"\b({self.company_names})\b"
```

تابع `match` کردن این پترن به صورت زیر است. در این تابع پس از اینکه این پترن موردنظر با استفاده از تابع `finditer` پیدا شد با کمک `span`های آن، پترن پیدا شده را با `<#نام-شرکت>` جایگزین می‌کنیم.

```
def hide_company(self, text):
    margin = 0
    for matched in re.finditer(self.company_name_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<#نام-شرکت>" + text[end-
margin:]margin += (end-start) - len("<#نام-شرکت>")
    return text
```

ایمیل

برای ایمیل پترن آن را به صورت زیر تعریف می‌کنیم:

```
# email pattern
self.email_pattern = r"\b(\w+([-+([\dot\])'\w+)*([\at\])\w+
([-([\dot\])\w+)*([\dot\])\w+([-([\dot\])\w+)*)\b"
```

پترن مورد نظر در این بخش با رشته‌هایی یکسان سازی می‌شود که اولاً پترن در ابتدا و انتهای آنها باشد سپس در ابتدا تعدادی حرف یا رقم و یا «_»

سپس با استفاده از `finditer` می‌توانیم پترن مربوطه را در متن پیدا کرده و با استفاده از `span` جای آن `<#آدرس-ایمیل>` را قرار می‌دهیم.

```
def hide_email(self, text):
    margin = 0
    for matched in re.finditer(self.email_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<#آدرس-ایمیل>" + text[end-margin:]
        margin += (end-start) - len("<#آدرس-ایمیل>")
    return text
```

نام شخص

برای نام شخص، ابتدا نام‌ها را از دیتاست می‌خوانیم و «ی»های عربی را به فارسی تبدیل می‌کنیم و سپس با استفاده از regex پترن آن را تعریف می‌کنیم.

```
# first name pattern
self.first_names = '|'.join(list(pd.read_csv("names_dataset.csv")
                                     .first_name))
self.first_name_pattern = re.sub("ی", "ي", self.first_names)
self.first_name_pattern = fr"\b({self.first_names})\b"
```

حال با استفاده از یک تابع بررسی می‌کنیم که آیا پترن مربوطه در متن وارد شده است یا خیر. البته با پترن مربوط به complex verbها نیز مقایسه می‌کند تا مشخص کند که آیا اسم جزئی از یک فعل است یا خیر مانند «به دنیا آمده است».


```
# last name pattern
self.last_names = '|'.join(list(pd.read_csv("Dataset/lastname.csv")['lastname']))
self.last_name_pattern = fr"\b({self.last_names})\b"
```

حال با استفاده از یک تابع بررسی می‌کنیم که آیا پترن مربوطه در متن وارد شده است یا خیر.

```
def hide_lname(self, text):
    margin = 0
    for matched in re.finditer(self.last_name_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<نام-خانوادگی#" + text[end-margin:]
        margin += (end-start) - len("<نام-خانوادگی#")
    return text
```

زمان و تاریخ

با استفاده از نصب پکیج `parstdex` و کد گیت‌هاب آن <https://github.com/kargaranamir/parstdex> زمان و تاریخ را بدست می‌آوریم.

```

# date time pattern
self.date_time_model = Parstdex()

def hide_date_time(self, text):
    margin = 0
    for x in self.date_time_model.extract_span(text)['date']:
        start , end = x
        text = text[:start-margin] + \
            " <#تاریخ> " + text[end-margin:]
        margin += (end-start) - len(" <#تاریخ> ")
    margin = 0
    for x in self.date_time_model.extract_span(text)['time']:
        start , end = x
        text = text[:start-margin] + \
            " <#زمان> " + text[end-margin:]
        margin += (end-start) - len(" <#زمان> ")
    return text

```

آدرس

برای بخش آدرس از کار انجام شده در پکیج `parsi_io` بسیار الهام و ایده گرفتیم. در پترن مربوط به آدرس، بخش‌هایی را از پترن `parsi_io` استفاده کردیم اما این پترن مشکلات عدیده‌ای نیز داشت که سعی کردیم برخی از این مشکلات را حل کنیم. به طور مثال در جمله «خیابان انقلاب بسیار تمیز است». این پکیج، هیچ آدرسی پیدا نمی‌کند زیرا که در پترنش نیاز دارد که به طور مثال بعد از دیدن کلمه خیابان که به نوعی شروع کننده آدرس است بار دیگر این کلمه را ببیند و اگر جمله را به «خیابان خیابان انقلاب بسیار تمیز است.» تغییر دهیم بخش «خیابان خیابان انقلاب» را به عنوان آدرس شناسایی می‌کند. با اضافه کردن پترنی به پترن کلی پکیج `parsi_io` سعی کردیم که از این مشکل جلوگیری کنیم و آدرس‌هایی مانند جمله بالا را نیز شناسایی کنیم. همچنین در بسیاری از جملات قبل از آدرس حروف اضافه‌ای مانند از، در، که و ... می‌آیند که در این جملات نیز پکیج `parsi_io` به درستی آدرس را شناسایی نمی‌کند و به طور مثال در جمله «علی در کشور زیмбаوه زندگی می‌کند.» هیچ آدرسی شناسایی نمی‌کند برای رفع این مشکل نیز با تغییراتی در کد پکیج `parsi_io` به طور مثال اضافه کردن این حروف ربط به کلماتی که آدرس می‌تواند با آنها شروع شود، سعی کردیم که عملکرد پکیج را بهبود ببخشیم. پترن نهایی در ادامه آمده است، البته به دلیل بزرگ بودن این پترن فقط بخش آخر آن در تصویر زیر آمده است.

```
self.address_pattern = fr"(\b({self.starter_keywords})(^\\.|{{{spaces_count}}}){{self.middle_address_keywords}}|{self.separators}}){{{keyword_count}}}( *({self.special_place})? *\\w+))|^(({{self.middle_address_keywords}}|{self.separators}}){{{keyword_count}}})({{self.special_place}})? *\\w+)"
```

سپس با استفاده از تابع زیر آدرس‌ها را در متن شناسایی و آنها را با <#آدرس> جایگزین می‌کنیم.

```
def hide_address(self, text):
    proposition = "در|از|که|به"
    for keyword_count in range(10, 0, -1):
        count_pattern = self.address_pattern.format(
            keyword_count=keyword_count, spaces_count="0,20")
        margin = 0
        for matched in re.finditer(count_pattern, text):
            start, end = matched.span()
            prop = re.search(fr"({proposition}) .+", matched.group(0))
            if prop:
                prop_len = len(prop.group(1))
                start+=prop_len
                text = text[:start-margin] + " <#آدرس> " + text[end-margin:]
                margin += (end-start) - len(" <#آدرس> ")
    return text
```

در بخش ابتدایی این تابع از تابع نوشته شده در پکیج `parsi_io` کمک گرفتیم که سعی می‌کند با استفاده از متغیر `keyword_count` بزرگترین آدرس ممکن را شناسایی کند به این صورت که این متغیر را از مقدار ۱۰ شروع می‌کند و اجازه می‌دهد که بخشی از پترن مورد نظرش ۱۰ مرتبه تکرار شود و سپس ۹ مرتبه الی آخر. سپس هنگامیکه آدرسی را شناسایی می‌کند به دلیل اینکه حروف ربط را نیز در پترن شناسایی آدرس آورده بودیم، بررسی می‌کنیم که آیا این حروف در آدرس یافت شده وجود دارند یا خیر و اگر در ابتدای آدرس، این حروف آمده باشند آنها را از آدرس حذف می‌کنیم و بخش آدرس را با تگ <#آدرس> جایگزین می‌کنیم.

شماره شبا

برای پیدا کردن شماره شبا یک پترن به این صورت تعریف شده است که ابتدا IR و سپس 24 رقم قرار می‌گیرد.

```
# Sheba pattern
self.sheba_pattern = r"\bIR\d{24}\b"
```

یک تابع نیز تعریف شده است که به کمک finditer شماره شباهای match شده با پترن را در متن پیدا می‌کند و به جای آن تگ <#شماره-شبا> را قرار می‌دهد.

```
def hide_sheba(self, text):
    margin = 0
    for matched in re.finditer(self.sheba_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<#شماره-شبا>" + text[end-margin:]
        margin += (end-start) - len("<#شماره-شبا>")
    return text
```

شماره IBAN

شماره IBAN در کشورهای مختلف پترن متفاوتی دارد به همین دلیل برای پیدا کردن شماره IBAN از سایت <https://www.regextester.com/115565> استفاده شده است. ابتدا پترن به صورت زیر تعریف شده است:

```
#iban pattern
self.iban_pattern = r"\b(?:IT|SM)\d{2}[A-Z]\d{22}|CY\d{2}[A-Z]\d{23}|NL\d{2}[A-Z]{4}\d{10}|LV\d{2}[A-Z]{4}\d{13}|(?:BG|BH|GB|IE)\d{2}[A-Z]{4}\d{14}|GI\d{2}[A-Z]{4}\d{15}|RO\d{2}[A-Z]{4}\d{16}|KW\d{2}[A-Z]{4}\d{22}|MT\d{2}[A-Z]{4}\d{23}|NO\d{13}|(?:DK|FI|GL|FO)\d{16}|MK\d{17}|(?:AT|EE|KZ|LU|XK)\d{18}|(?:BA|HR|LI|CH|CR)\d{19}|(?:GE|DE|LT|ME|RS)\d{20}|IL\d{21}|(?:AD|CZ|ES|MD|SA)\d{22}|PT\d{23}|(?:BE|IS)\d{24}|(?:FR|MR|MC)\d{25}|(?:AL|DO|LB|PL)\d{26}|(?:AZ|HU)\d{27}|(?:GR|MU)\d{28})\b"
```

یک تابع نیز تعریف شده است که به کمک `finditer` شماره ایبان های `match` شده با پترن را در متن پیدا می کند و به جای آن `>#شماره-` ایبان را قرار می دهد.

```
def hide_iban(self, text):
    margin = 0
    for matched in re.finditer(self.iban_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<#شماره-ایبان" + text[end-margin:]
        margin += (end-start) - len("<#شماره-ایبان")
    return text
```

شماره کارت

در این بخش برای به دست آوردن پترن مربوط به شماره کارت از کد زیر استفاده می کنیم. در این کد حالات مختلف برای یک شماره کارت 16 رقمی در نظر گرفته شده است. این حالات شامل این است که شماره کارت 16 رقم پشت سر هم یا 4 رقم، 4 رقم با خط فاصله یا فاصله قرار گیرد.

```
# bank card
self.ws = "[ -]"
self.bank_card_pattern = fr"\b\d{{16}}|\d{{4}}{self.ws}\d{{4}}{self.ws}\d{{4}}{self.ws}\d{{4}}\b"
```

بررسی معتبر بودن یا معتبر نبودن این شماره کارت در تابع مربوط به این بخش انجام میگیرد. تابع به صورت زیر تعریف شده است.

```

def hide_bank_card(self, text):
    def card_num_validation(card_num):
        sum = 0
        for idx, digit in enumerate(card_num):
            digit = int(digit)
            if not idx % 2:
                sum += digit*2 if digit*2 < 9 else digit*2-9
            else:
                sum += digit
        if not sum % 10:
            return True
    margin = 0
    for matched in re.finditer(self.bank_card_pattern, text):
        start, end = matched.span()
        if card_num_validation(re.sub(self.ws, "", matched.group(0))):
            text = text[:start-margin] + \
                "<#شماره-کارت-معتبر#" + text[end-margin:]
            margin += (end-start) - len("<#شماره-کارت-معتبر#")
        else:
            text = text[:start-margin] + \
                "<#شماره-کارت-نامعتبر#" + text[end-margin:]
            margin += (end-start) - len("<#شماره-کارت-نامعتبر#")
    return text

```

برای بررسی صحت شماره کارت بانکی کافی است که ارقام در جایگاه فرد شماره کارت بانکی را در عدد ۲ و ارقام در جایگاه زوج را در عدد ۱ ضرب کنیم و در هر مرحله اگر حاصل ضرب از ۹ بیشتر شد، ۹ واحد از آن کم می‌کنیم تا عددی تک رقمی تولید شود. سپس تمام اعداد را با یکدیگر جمع می‌کنیم و اگر حاصل جمع بر ۱۰ بخش پذیر بود به این معنا است که شماره کارت بانکی صحیح است در غیر اینصورت شماره کارت نامعتبر قلمداد می‌شود.

آدرس اینترنتی

پترن آدرس اینترنتی مانند پترن نوشته شده در پکیج `parsi.io` به صورت زیر تعریف شده است:

```

#url
patternself.url_pattern = r"\b((https|http|ftp):\/\/)?(www\.)?([-a-zA-Z0-9@:%_\+~#={1,256}\. [a-zA-Z0-9( )]{1,6})\b([-a-zA-Z0-9@:%_\+~#?&\/\=]*)\b"

```

یک تابع نیز تعریف شده است که به کمک `finditer` آدرس‌های اینترنتی `match` شده با پترن را در متن پیدا می‌کند و به جای آن تگ `<#تارنما>` را قرار می‌دهد.

```
def hide_url(self, text):
    margin = 0
    for matched in re.finditer(self.url_pattern, text):
        start, end = matched.span()
        text = text[:start-margin] + "<#تارنما>" + text[end-margin:]
        margin += (end-start) - len("<#تارنما>")
    return text
```

توابع کمکی

در انتها توابعی نوشتیم که برخی حالات خاص را بهبود می‌دادند به طور مثال در برخی مواقع در انتهای آدرسی نام شهر قرار می‌گرفت و ترجیح می‌دادیم که این نام شهر نیز در تگ `<#آدرس>` قرار بگیرد و نه در تگ `<#نام-شهر>` به این منظور و رفع حالات مشابه دیگر، تابع زیر را نوشتیم که اگر چنین حالاتی پیش آمد، نام شهر را نیز در تگ `<#آدرس>` قرار دهد.

```
def merge_addresses(self, text):
    res = text.split()
    for idx, word in enumerate(res):
        if word == "<#آدرس>":
            if res[idx+1] == "<#نام-شهر>":
                res.pop(idx+1)
            elif res[idx+1] == "شهر" and res[idx+2] == "<#نام-شهر>":
                res.pop(idx+1)
                res.pop(idx+1)
    return " ".join(res)
```

تابع run

در این تابع تمامی متدهای نوشته شده در کلاس را اجرا می‌کنیم. در نتیجه در متنی که به عنوان ورودی به این تابع داده می‌شود، اطلاعات شخصی شناسایی و با تگ مناسب جایگزین می‌شوند.

```
def run(self, text):
    text = self.number_normalizer(text)
    text = self.hide_address(text)
    text = self.hide_iban(text)
    text = self.hide_phone(text)
    text = self.hide_mobile_num(text)
    text = self.hide_national_code(text)
    text = self.hide_bank_card(text)
    text = self.hide_sheba(text)
    text = self.hide_email(text)
    text = self.hide_url(text)
    text = self.hide_company(text)
    text = self.hide_date_time(text)
    text = self.hide_city(text)
    text = self.merge_addresses(text)
    text = self.hide_name(text)
    text = self.hide_lname(text)
    return text
```

عملکرد بر روی تست کیس‌های داده شده

در تصویر زیر برای هر تست کیس در ابتدا خود تست کیس و سپس خروجی مازول نوشته شده، آمده‌اند.

```
#####
.علی احمدی در شهرستان اصفهان شهر فولادشهر به دنیا آمد
-----
.نام-شخم<#نام-خانوادگی> در <#آدرس> به دنیا آمد#>
#####
.علیرضا و زهرا در کشور زیمباوه زندگی میکنند
-----
.نام-شخم<#نام-شخم> در <#آدرس> زندگی میکنند#>
#####
.به نظر سخت است france زندگی در کشور
-----
.زندگی در <#آدرس> به نظر سخت است
#####
.کشور به منطقه‌ای گفته میشود که مرز آن با سیاست تعیین شده است
-----
.آدرس<#منطقه‌ای گفته میشود که مرز آن با سیاست تعیین شده است#>
#####
.من در ۱۶ بهمن ۱۳۷۵ به دنیا آمدم
-----
.من در <#تاریخ> به دنیا آمدم
#####
.در تاریخ ۱۶ فروردین به مکه رفتم و از شرکت داده پردازش نیز دیدن کردم
-----
.در <#تاریخ> به <#نام-شهر> رفتم و از شرکت <#نام-شرکت> نیز دیدن کردم
#####
.کدملی من ۱۱۳۰۳۹۴۷۸۹ است
-----
.کدملی من <#کد-ملی-نامعتبر> است
#####
.کد ملی من است ۱۱۳۰۳۹۴۷۸۹
-----
.کد-ملی-نامعتبر<#کد ملی من است#>
#####
سلام نام من جیسون محمدنژادپور است و در میامی زندگی میکنم. شماره کارت اعتباری من 6104337958646987 است که
رو مشاهده کردم و ایمیل کارمند مایکروسافت microsoft.com از آن به حساب شما پول میریزم. من سایت
akabr@micr.com را برداشتم
-----
سلام نام من <#نام-شخم> <#نام-خانوادگی> است و در <#نام-شهر> زندگی میکنم. شماره کارت اعتباری من <#شماره-
کارت-نامعتبر> است که از آن به حساب شما پول میریزم. من سایت <#تارنما> رو مشاهده کردم و ایمیل کارمند
مایکروسافت، <#آدرس-ایمیل> را برداشتم
#####
.است IR06960000001032420000011 شماره شبای من
-----
.شماره شبای من <#شماره-شبا> است
#####
.است IBAN KW81CBKU000000000001234560101 من شماره
-----
.من <#شماره-ایبان> است IBAN شماره
#####
.ساعت ۸ صبح پرواز به روستای احمدآباد را دارم
-----
.زمان پرواز به <#آدرس> را دارم#>
#####
.من در اصفهان، کوشک، خیابان احمدی، کوچه شهید علی علیزاده پلاک ۱۴۳ زندگی میکنم
-----
.من در <#آدرس> زندگی میکنم
#####
.این نشانی خیابان بهشتی است
-----
.این <#آدرس> است
#####
.شماره تلفن من ۰۹۱۲۳۲۵۶۷۸۹ و شماره خانه علیرضا ۰۲۱۳۳۴۵۵۶۶ است
-----
.شماره تلفن من <#شماره-تلفن-همراه> و شماره خانه <#نام-شخم> <#شماره-تلفن-تاب> است
#####
.خیابان اهواز شهر اهواز بسیار تمیز است
-----
.آدرس<#بسیار تمیز است#>
```

نکته پایانی

کد به صورتی نوشته شده است در ابتدا که شی‌ای از کلاس می‌سازیم به دلیل لود کردن تمام دیتاست‌ها، مقداری زمان می‌برد ولی پس از آن برای اجرای تابع `run` و تست کردن عبارات و متون سرعت خوبی دارد و سریعاً خروجی را تولید می‌کند. برای اجرا نیز کافی است که نوت‌بوک `main` را اجرا کنید.