20.JSON データの読み込み

解答：

```
import gzip
import json
data = []
with open("/content/jawiki-country.json") as f:
  for line in f:
    data=json.loads(line)
    if data["title"]=="イギリス":
      text=data["text"]
      print(text)
      break
```

実行結果：

```
{{redirect|UK}}
{{redirect|英国|春秋時代の諸侯国|英（春秋）}}
{{Otheruses|ヨーロッパの国|長崎県・熊本県の郷土料理|いぎりす}}
{{基礎情報 国
|略名  =イギリス
|日本語国名 = グレートブリテン及び北アイルランド連合王国
|公式国名 = {{lang|en|United Kingdom of Great Britain and Northern Ireland}}<ref>英語以外での正式国名:<br />
*{{lang|gd|An Rìoghachd Aonaichte na Breatainn Mhòr agus Eirinn mu Thuath}}（[[スコットランド・ゲール語]]）
*{{lang|cy|Teyrnas Gyfunol Prydain Fawr a Gogledd Iwerddon}}（[[ウェールズ語]]）
*{{lang|ga|Ríocht Aontaithe na Breataine Móire agus Tuaisceart na hÉireann}}（[[アイルランド語]]）
*{{lang|kw|An Rywvaneth Unys a Vreten Veur hag Iwerdhon Glédh}}（[[コーンウォール語]]）
*{{lang|sco|Unitit Kinrick o Great Breetain an Northren Ireland}}（[[スコットランド語]]）
**{{lang|sco|Claught Kängrick o Docht Brätain an Norlin Airlann}}、{{lang|sco|Unitet Kängdom o Great Brittain an Norlin Airlann}}（アルスター・スコットランド語）</ref>
|国旗画像 = Flag of the United Kingdom.svg
|国章画像 = [[ファイル:Royal Coat of Arms of the United Kingdom.svg|85px|イギリスの国章]]
|国章リンク =（[[イギリスの国章|国章]]）
|標語 = {{lang|fr|[[Dieu et mon droit]]}}<br />（[[フランス語]]:[[Dieu et mon droit|神と我が権利]]）
|国歌 = [[女王陛下万歳|{{lang|en|God Save the Queen}}]]{{en icon}}<br />''神よ女王を護り賜え''<br />{{center|[[ファイル:United States Navy Band - God Save the Queen.ogg]]}}
|地図画像 = Europe-UK.svg
|位置画像 = United Kingdom (+overseas territories) in the World (+Antarctica claims).svg
|公用語 = [[英語]]
|首都 = [[ロンドン]]（事実上）
|最大都市 = ロンドン
|元首等肩書 = [[イギリスの君主|女王]]
|元首等氏名 = [[エリザベス2世]]
|首相等肩書 = [[イギリスの首相|首相]]
|首相等氏名 = [[ボリス・ジョンソン]]
|他元首等肩書1 = [[貴族院（イギリス）|貴族院議長]]
|他元首等氏名1 = [[:en:Norman Fowler, Baron Fowler|ノーマン・ファウラー]]
|他元首等肩書2 = [[庶民院（イギリス）|庶民院議長]]
|他元首等氏名2 = {{仮リンク|リンゼイ・ホイル|en|Lindsay Hoyle}}
|他元首等肩書3 = [[連合王国最高裁判所|最高裁判所長官]]
|他元首等氏名3 = [[:en:Brenda Hale, Baroness Hale of Richmond|ブレンダ・ヘイル]]
|面積順位 = 76
|面積大きさ = 1 E11
|面積値 = 244,820
|水面積率 = 1.3%
```

.....

コンテンツが多すぎてすべてを表示できない

まとめ：

考え方はファイルを開いた後、各行を data 変数に読み込むことです

21.カテゴリ名を含む行を抽出

解答：

```
import re
lines = text.splitlines()
for line in lines:
    if re.search(r'\[\[Category:.*\]\]', line):
        print(line)
```

実行結果：

```
[[Category:イギリス|*]]
[[Category:イギリス連邦加盟国]]
[[Category:英連邦王国|*]]
[[Category:G8 加盟国]]
[[Category:欧州連合加盟国|元]]
[[Category:海洋国家]]
[[Category:現存する君主国]]
[[Category:島国]]
[[Category:1801 年に成立した国家・領域]]
```

まとめ：
一致する結果を見つけるには、'[[Category:'で始まり、']]'で終わる文字列に一致することを意味する正規表現が使用されています。
re.search 関数は文字列全体をスキャンし、最初に成功した一致を返します。

22.カテゴリ名の抽出

解答：

```
for line in lines:
    if re.search(r'\[\[Category:.*\]\]', line):
      print(re.search(r'\[\[Category:(.*)\]\]', line).group(1))
```

実行結果：

```
イギリス|*
イギリス連邦加盟国
英連邦王国|*
G8 加盟国
欧州連合加盟国|元
海洋国家
現存する君主国
島国
1801 年に成立した国家・領域
```

まとめ：

全体的な考え方は前の問題と同じだが、中間の内容を単独で分類し、最後にこの内容を出力する必要がある

23. セクション構造

解答：

```
for line in lines:
    if re.search(r'^==.*==$', line):
      print(line)
```

実行結果：

==国名==
==歴史==
==地理==
===主要都市===
===気候===
==政治==
===元首===
===法===
===内政===
===地方行政区分===

===外交・軍事===

==経済==
===鉱業===
　　……省略
====モータースポーツ====
====野球====
==== カーリング ====
==== 自転車競技 ====
==脚注==
==関連項目==
==外部リンク==

まとめ：
ここでの正規表現の意味は、中間の文字が何であれ、戻ることができる'=='で始まり、'=='で終わる行を探すことです。

23. セクション構造

解答：

```
for line in lines:
    if re.search(r'^==.*==$', line):
        level = len(re.match(r'^=*', line).group()) - 1
        title = re.search(r'^=*(\D*?)=*$', line).group(1)
        print(level, title)
```

実行結果：

1 国名
1 歴史
1 地理
2 主要都市
2 気候
1 政治
2 元首
2 法
2 内政
2 地方行政区分

2 外交・軍事

1 経済

……省略

3　自転車競技
1 脚注
1 関連項目
1 外部リンク

まとめ：
前の問題を結果に一致させた上で、re.match 関数を使用して各行に対して連続'='の数を最初から一致させ、この数を 1 つ減らすとタイトルのレベルになります。
再使用 re.search 関数は、タイトルの中間にある非デジタル文字をグループ化し、このグループを取り出すと純文字のタイトルを得ることができます。

24.ファイル参照の抽出

解答：

```
for line in lines:
    if re.search( r'\[\[ファイル:(.+?)\|', line):
      result=re.search( r'\[\[ファイル:(.+?)\|', line)
      print(result.group(1))
```

実行結果：

```
Royal Coat of Arms of the United Kingdom.svg
Descriptio Prime Tabulae Europae.jpg
Lenepveu, Jeanne d'Arc au siège d'Orléans.jpg
London.bankofengland.arp.jpg
Battle of Waterloo 1815.PNG
Uk topo en.jpg
BenNevis2005.jpg
```

……省略

```
UKpop.svg
Anglospeak.svg
Royal Aberdeen Children's Hospital.jpg
CHANDOS3.jpg
The Fabs.JPG
Wembley Stadium, illuminated.jpg
```

まとめ：
'[[ファイル：'で始まり、'｜'で終わる文字列を探して、中間部分をグループ化して、中間部分を取り出します

25.テンポレートの抽出

解答：

```
for i, line in enumerate(lines):
    if line.startswith('{{基礎情報'):
```

```
        start = i
    elif line.startswith('}}'):
        end = i
        break
templete = [
    re.findall(r'\|([^=]*)=(.*)', line)
    for line in lines[start+1 : end]
]
templete = [x[0] for x in templete if x]
dct = {
    key.strip() : value.strip()
    for key, value in templete
}#stored as a dictionary
dct
```

実行結果：

```
{'略名': 'イギリス',
 '日本語国名': 'グレートブリテン及び北アイルランド連合王国',
 '公式国名': '{{lang|en|United Kingdom of Great Britain and Northern Ireland}}<ref>英語以外での正式国名:<br />',
 '国旗画像': 'Flag of the United Kingdom.svg',
 '国章画像': '[[ファイル:Royal Coat of Arms of the United Kingdom.svg|85px|イギリスの国章]]',
 '国章リンク': '（[[イギリスの国章|国章]]）',
```

……省略

```
'時間帯': '±0',
 '夏時間': '+1',
 'ISO 3166-1': 'GB / GBR',
 'ccTLD': '[[.uk]] / [[.gb]]<ref>使用は .uk に比べ圧倒的少数。</ref>',
 '国際電話番号': '44',
 '注記': '<references/>'}
```

まとめ：

列挙により'{{基礎情報'で始まる行を選択し、start 変数に行数を記録し、'}}'で終わる行を選択し、end 変数に行数を記録する。次に、’r’¥|([^=]*)=(.*)’’式を満たす行を取り出し、リスト型に変換し、最後に dict 型で保存します

26. 強調マークアップの除去

解答：

```
dct2 = {
    key : re.sub(r"''+", '', value)
    for key, value in dct.items()
}
dct2
```

実行結果：

{'略名': 'イギリス',
 '日本語国名': 'グレートブリテン及び北アイルランド連合王国',
 '公式国名': '{{lang|en|United Kingdom of Great Britain and Northern Ireland}}<ref>英語以外での正式国名:
',
 '国旗画像': 'Flag of the United Kingdom.svg',
 '国章画像': '[[ファイル:Royal Coat of Arms of the United Kingdom.svg|85px|イギリスの国章]]',
 '国章リンク': '（[[イギリスの国章|国章]]）',

……省略

'時間帯': '±0',
 '夏時間': '+1',
 'ISO 3166-1': 'GB / GBR',
 'ccTLD': '[[.uk]] / [[.gb]]<ref>使用は .uk に比べ圧倒的少数。</ref>',
 '国際電話番号': '44',
 '注記': '<references/>'}

まとめ：

他との区別（斜体）	''他との区別''	他との区別
強調（太字）	'''強調'''	**強調**
斜体と強調	'''''斜体と強調'''''	***斜体と強調***

re.sub 関数を使用して、特殊フォントを空白に置き換えます

27. 内部リンクの除去

解答：

```
def remove_link(x):
    x = re.sub(r'\[\[[^\|\]]+\|[^{}\|\]]+\|([^\]]+)\]\]', r'\1', x)
    x = re.sub(r'\[\[[^\|\]]+\|([^\]]+)\]\]', r'\1', x)
    x = re.sub(r'\[\[([^\]]+)\]\]', r'\1', x)
    return x

dct3 = {
    key : remove_link(value)
    for key, value in dct2.items()
}
dct3
```

実行結果：

例えば：'首都': '[[ロンドン]]（事実上）'→'首都': 'ロンドン（事実上）',

まとめ：

内部リンク	[[記事名]]	記事名
	[[記事名|表示文字]]	表示文字
	[[記事名#節名|表示文字]]	表示文字

[]を空白文字で置換。

28.MediaWiki マークアップの除去

解答：

```
def remove_markups(x):
    x = re.sub(r'{{.*\|.*\|([^}]*)}}', r'\1', x)
    x = re.sub(r'<([^>]*)( .*|)>.*</\1>', '', x)
    x = re.sub(r'<[^>]*?/>', '', x)
    x = re.sub(r'\{\{0\}\}', '', x)
    return x

dct4 = {
    key : remove_markups(value)
    for key, value in dct3.items()
}
dct4
```

実行結果：

例えば：'GDP 値元': '1 兆 5478 億<ref name="imf-statistics-gdp">[http://www.imf.org/external/pubs/ft/weo/2012/02/weodata/weorept.aspx?pr.x=70&pr.y=13&sy=2010&ey=2012&scsm=1&ssd=1&sort=country&ds=.&br=1&c=112&s=NGDP%2CNGDPD%2CPPPGDP%2CPPPPC&grp=0&a=IMF>Data and Statistics>World Economic Outlook Databases>By Countrise>United Kingdom]</ref>',

↓

'GDP 値元': '1 兆 5478 億',

29.国旗画像の URL を取得する

解答：

```
import requests
filename = dct4['国旗画像']
session = requests.Session()
url = 'https://en.wikipedia.org/w/api.php'
params = {
    'action' : 'query',
    'format' : 'json',
    'prop' : 'imageinfo',
    'titles' : 'File:' + filename,
    'iiprop' : 'url',
}
r = session.get(url=url, params=params)
data = r.json()
pages = data['query']['pages']
```

```
flag_url = pages[list(pages)[0]]['imageinfo'][0]['url']
flag_url
```

実行結果：
https://upload.wikimedia.org/wikipedia/en/a/ae/Flag_of_the_United_Kingdom.svg

まとめ：

```
import requests

S = requests.Session()

URL = "https://en.wikipedia.org/w/api.php"

PARAMS = {
    "action": "query",
    "format": "json",
    "prop": "imageinfo",
    "titles": "File:Billy_Tipton.jpg"
}

R = S.get(url=URL, params=PARAMS)
DATA = R.json()

PAGES = DATA["query"]["pages"]

for k, v in PAGES.items():
    print(v["title"] + " is uploaded by User:" + v["imageinfo"][0]["user"])
```

公式に与えられたコードに置き換えてください

（次のページは正規表現に関するまとめです）

正規表現に関するまとめ

正規表現：非常に簡潔なパターンを使用して文字列を取得し置換する操作。

1. 正規検索：

   関数：search/match/fullmatch/findall/finditer

   例えば：word = ‘ab123cd347ef321’、 検索 3 で始まる後に数字

   3 は文字 3、 \d は任意の数字、+は 1 回または複数回出現することを表す.

   Search

```
import re
word = 'ab128cd347820ef3214'
result=re.search(r'3\d+',word)
print(result)

<re.Match object; span=(7, 13), match='347820'>
```

   search という関数を使って一度だけ調べる

```
import re
word = 'ab128cd347820ef3214'
result=re.search(r'3\d',word)
print(result)

<re.Match object; span=(7, 9), match='34'>
```

   +なし、3 の後ろに 1 つの数字しかありません

   Match

```
import re
word = 'ab128cd347820ef3214'
result=re.match(r'3\d+',word)
print(result)

None
```

   match では文字列の最初からマッチングを行い、最初は a ではマッチングに失敗し、1 回だけ検索されます.

```
import re
word = 'ab128cd347820ef3214'
result=re.match(r'ab\d+',word)
print(result)

<re.Match object; span=(0, 5), match='ab128'>
```

   ab の場合は一致しました

fullmatch

正規規則を使用して文字列全体を一致させる

```
import re
word = 'ab128cd347820ef3214'
result=re.fullmatch(r'ab\d+',word)
print(result)

None
```

文字列全体がこの規則に一致していないため、None です

これらの関数は文字列ではなく、re.match タイプというオブジェクトを返します

Finditer

iter は反復を表し、文字列の中のすべての一致した結果をクエリし、得られた結果は反復器であり、for ループを通じて遍歴することができ、得られた反復器の中の各要素はまた 1 つの re.match タイプのオブジェクトである

```
import re
word = 'ab128cd347820ef3214ab34x'
result=re.finditer(r'3\d+',word)
print(result)

<callable_iterator object at 0x7f526a406090>
----------------------------------------------------------
import re
word = 'ab128cd347820ef3214ab34x'
result=re.finditer(r'3\d+',word)
for x in result:
    print(x)

<re.Match object; span=(7, 13), match='347820'>
<re.Match object; span=(15, 19), match='3214'>
<re.Match object; span=(21, 23), match='34'>
```

Findall

結果は、一致する結果文字列を保持するリストです。

```
import re
word = 'ab128cd347820ef3214ab34x'
result=re.findall(r'3\d+',word)
print(result)

['347820', '3214', '34']
```

2. re.match タイプの紹介

re モジュールの match/search/fullmatch/finditer これらの方法で得られた結果には、すべて re.Match タイプがあります

例えば：

```
import re

x1=re.search(r'\d+','ab12cd34')
print(type(x1),x1)

<class 're.Match'> <re.Match object; span=(2, 4), match='12'>
```

re.Match の内部プロパティとメソッドのクエリーと解析：

```
import re

x1=re.search(r'\d+','ab12cd34')
print(dir(x1))#dir 用来列出一个对象所有的属性和方法
[ 'end', 'endpos', 'expand', 'group', 'groupdict', 'groups', 'lastgroup',
'lastindex', 'pos', 're', 'regs', 'span', 'start', 'string']
```

end

```
import re
x1=re.search(r'\d+','ab12cd34')
print(x1.end())

4
```

一致する文字列の終了下付き文字列の取得

endpos

```
import re

x1=re.search(r'\d+','ab12cd34')
print(x1.endpos)

8
```

文字列全体の長さを取得する（最大下付き+1）

span

```
import re

x1=re.search(r'\d+','ab12cd34')
print(x1.span())

(2, 4)
```

結果は、一致する文字列の開始（含む）と終了（含まない）の下付き文字列を表すタプルです

string

```
import re

x1=re.search(r'\d+','ab12cd34')
print(x1.string)
```

```
ab12cd34
```

完全な文字列の検索

group

```
import re

x1=re.search(r'\d+','ab12cd34')
print(x1.group())
```

```
12
```

```
import re

x2=re.search(r'b\d+c\d+d\d+','a10b2453c5896d717e123')
print(x2.group())#print（x1.group（0））と等しい
```

```
b2453c5896d717
```

一致する文字列が得られました

```
import re

x2=re.search(r'(b\d+)(c\d+)(d\d+)','a10b2453c5896d717e123')
print(x2.group())
```

```
b2453c5896d717
```

結果的には違いはありませんが、括弧はグループ化に使用されています

```
import re
x2=re.search(r'(b\d+)(c\d+)(d\d+)','a10b2453c5896d717e123')
#合計 4 つのグループ第 0 グループは文字列全体です
print(x2.group(1))
b2453
```

groups

```
import re

x2=re.search(r'(b\d+)(c\d+)(d\d+)','a10b2453c5896d717e123')
print(x2.groups())
```

```
('b2453', 'c5896', 'd717')
```

すべてのサブグループが一致する文字列からなるタプルを取得する

groupdict

```
import re

x2=re.search(r'(b\d+)(c\d+)(d\d+)','a10b2453c5896d717e123')
print(x2.groupdict())
```

{}は、グループ名が辞書形式で保存されたパケットのデータである（key，value 形式で保存）

```
import re

x2=re.search(r'(?P<test>b\d+)(c\d+)(d\d+)','a10b2453c5896d717e123')
print(x2.groupdict())
```

```
{'test': 'b2453'}
```

(?P<test>b¥d+)グループに test という名前を付けることです

```
import re

x2=re.search(r'(?P<test>b\d+)(c\d+)(d\d+)','a10b2453c5896d717e123')
print(x2.groupdict()['test'])
```

```
b2453
```

3. re.compile の紹介

```
import re

print(re.search(r'm\d+','ab322wm234dasdd'))

pattern=re.compile(r'm\d+')
print(pattern.search('ab322wm234dasdd'))
```

```
<re.Match object; span=(6, 10), match='m234'>
<re.Match object; span=(6, 10), match='m234'>
```

正規表現をコンパイルしてオブジェクトにし、直接呼び出すことができます。結果は以前の方法と変わらない

4. 正規修飾子

正則修飾子は正則規則を修飾し、正則規則に異なる意味を持たせる

```
import re

print(re.search(r'x','goodokXhi'))
```

```
None
```

正則ルールでも大文字と小文字を区別しています

```
import re

print(re.search(r'x','goodokXhi',re.I))
<re.Match object; span=(6, 7), match='X'>
```
青い部分が正則修飾部分であり、re.I は正則規則が大文字と小文字を区別しないようにするために使用される

```
import re

print(re.findall(r'.','goodo\nkXhi'))
['g', 'o', 'o', 'd', 'o', 'k', 'X', 'h', 'i']
```
.改行以外の任意の文字を表す

```
import re

print(re.findall(r'.','good\nokXhi',re.S))
['g', 'o', 'o', 'd', '\n', 'o', 'k', 'X', 'h', 'i']
```
re.S–改行を含む任意の文字を一致させる

5. 正規一致規則

① 数字とアルファベットはそれ自体を表す

```
print（re.search(r'd','ab123')）#None
```

ab 123 でアルファベット d を探して None に戻る

② \ 特殊な意味があり、転じて意味を持つ

```
print('hello\\nworld')#hello\nworld

import re
x='hello\\nworld'
re.search('\\\\',x)
<re.Match object; span=(5, 6), match='\\'>
re.search(r'\\',x)# 加 r 転じて意味解除
<re.Match object; span=(5, 6), match='\\'>
```

③ 大部分のアルファベットの前に\を付けると意味が変わってくる（重点/難点）

例：d はアルファベット d を表し、 \d は数字を表す

④ ほとんどの句読点は特殊な意味を持つ（重点/難点）

```
import re
print(re.search(r'1+2','hello1+2=5'))#None
print(re.search(r'1+2','hello1111112=5'))
#<re.Match object; span=(5, 12), match='1111112'>
```
'+'には、前の文字が 1 回または複数回現れることを示す特別な意味があります

```
print(re.search(r'1+2','hello12=5'))
#<re.Match object; span=(5, 7), match='12'>
```

⑤ 句読点そのものを表すには、'\'文字を使用する必要があります

```
print(re.search(r'1\+2','hello1+2=5'))
#<re.Match object; span=(5, 8), match='1+2'>
```

6. '\+アルファベット'の特別な意味のまとめ

\n 改行 \r \n==\n ; \r Enter（前の行は印刷せず、次の行のみ印刷）
\t タブ、 \s 空白文字

```
print(re.findall(r'\s','hello good\ryes\thi\n'))
#[' ', '\r', '\t', '\n']
```

\S 非空白文字

```
print(re.findall(r'\S','hello good\ryes\thi\n'))
['h', 'e', 'l', 'l', 'o', 'g', 'o', 'o', 'd', 'y', 'e', 's', 'h', 'i']
```

\d は数字を表し、\D は非数字を表し

```
print(re.findall(r'\D+','hello 123'))
['hello ']
```

\w は数字とアルファベットと'__'を表し、\W は非数字とアルファベットと'__'を表し

```
print(re.findall(r'\w','a1_!@#$3'))
['a', '1', '_', '3']
print(re.findall(r'\W','a1_!@#$3'))
['!', '@', '#', '$']
```

7. 句読点の特殊な意味

()グループ化に使用
+：少なくとも 1 回表示==={1,}
*：任意の回数==={0,}
?: 文字が最大 1 回表示されることを示します。貪欲を非貪欲に変換する
{}：文字の出現回数を制限する{n}：文字が n 回出現したことを示す
{,n}：文字が n 回まで出現することを示す（n を含む）
{m, n}：文字が m から n 回現れることを表す

```
print(re.search(r'a\d{3,5}x','a345x'))
#<re.Match object; span=(0, 5), match='a345x'>
```

[ ]区間範囲を表すために使用する

```
print(re.search(r'x[1-5]+p','dskx347pix12321p'))
<re.Match object; span=(9, 16), match='x12321p'>
```

#範囲 1 から 5 が規定されているので、x 347 p に遭遇したとき、7 はその範囲内にないので捨てます。

```
print(re.search(r'x[a-d]+p','dskxxyapixabcp'))
<re.Match object; span=(9, 14), match='xabcp'>
```
#アルファベットの範囲を指定することもできます

｜オプションの値を表すために使用され、[]とは異なり、通常は()に合わせて使用されます

```
import re
print(re.findall(r'u[352a]x','u3xiu5xpu2xouax'))
['u3x', 'u5x', 'u2x', 'uax']

import re
x=re.finditer(r'u(3|5|2|a)x','u3xiu5xpu2xouax')
for i in x:
  print(i.group(0))
u3x  u5x  u2x  uax

import re
x=re.finditer(r'u(23|67|34)x','u23xiu67xlu34x')
for u in x:
  print(u.group())
u23x
u67x
u34x
```
#中央に 2 つの数字がある場合は、小括弧に finditer を付けるだけで検索できます

^は指定した文字で始まり、[]に置くと反をとる
```
print(re.search(r'^st','amstp'))#== re.match（r’st’,’amstp）
None

print(re.findall(r'a[1-5]+x','a123xdklsa789xauidsx'))
['a123x']

print(re.findall(r'a[^1-5]+x','a123xdklsa789xauidsx'))
['a789xauidsx']# 1～5 の数字ではありません
```

$は指定された文字で終了します
```
print(re.search(r'mp','xdkadskjsmmpdlaied'))
<re.Match object; span=(10, 12), match='mp'>

print(re.search(r'mp$','xdkadskjsmmpdlaied'))
```
None mp で終了するため，None を出力

```
print(re.search(r'mp$','xdkadskjsmmp'))
<re.Match object; span=(10, 12), match='mp'>
```

. 改行以外の任意の文字を表す

```
import re

print(re.findall(r'.','goodo\nkXhi'))
['g', 'o', 'o', 'd', 'o', 'k', 'X', 'h', 'i']
```

8. 正規置換

```
import re
re.sub(r'\d','T','a45b7d23x95')
aTTbTdTTxTT

import re
re.sub(r'\d+','T','a45b7d23x95')
aTbTdTxT
```

文字列内の数字を見つけて置換する 2 つのプロセスとみなすことができます。
では、T は関数に置き換えることができます

```
import re
a='a45b7d23x95'
def test(x):
  return 'p'
re.sub(r'\d+',test,a)
#apbpdpxp p で置換

import re
a='a45b7d23x95'
def test(x):
  data=x.group()#取得したのは文字列なので int に変換しますが、戻り値には文字列が必要なので文字列に戻します
  return str(int(data)*2)
re.sub(r'\d+',test,a)
#a90b14d46x190
```

これにより、文字列内の数字を 2 に乗算して置換する操作が可能になります

ラムダ式で操作することもできます

```
import re
a='a45b7d23x95'
re.sub(r'\d+',lambda x:str(int(x.group())*2),a)
#a90b14d46x190
```

9. 貪欲と非貪欲

```
import re

a='jkdx9087p908'
result=re.search(r'x(.+)(\d+)',a)
print(result.group())
#x9087p908
```

```
print(result.group(1))
#9087p90  取り終わっていないのは、後ろに数字があるからです

print(result.group(2))
#8
```
実は p を取るのもこの正規表現の規則に合っていて、このように取るのは python の正規表現の中で、デフォルトのは貪欲なモード（できるだけ多くのマッチング）だからです

？：貪欲を非貪欲に変換する

```
a='jkdx9087'
result=re.search(r'x(.+)(\d*)',a)
print(result.group(1))
print(result.group(2))
#9087  None

result=re.search(r'x(.+?)(\d*)',a)
print(result.group(1))
print(result.group(2))
#9   087

result=re.search(r'x(.{3})(\d*)',a)
print(result.group(1))
print(result.group(2))
#908  7

result=re.search(r'x(.{3}?)(\d*)',a)
print(result.group(1))
print(result.group(2))
#908  7
```
以上の例から分かるように、すでに文字数が規定されているものに対して、貪欲モードであるかどうかは結果に影響しない

```
result=re.search(r'x(.{,3}?)(\d*)',a)
print(result.group(1))
print(result.group(2))
#       9087
```
上限を指定しているので 3 つまで、また貪欲モードなのでできるだけ少ないマッチングになっています