The Kyoto College of Graduate Studies for Informatics

kcg.edu

コンピュータプログラミング概論

秋期第10回

安　平勲

h _an@kcg.ac.jp

文字列操作と正規表現

一般的な文字列操作

- 文字列を結合・分割できる
- 英字の大文字と小文字をスワップできる
- 部分文字列を探索・入れ替えできる
- 文字列のフォーマットもできる
- ・・・

(公式ドキュメントをクリック)

文字列の操作

第2回資料

```
In [14]: long_string = 'supercalifragilisticexpialidocious'
         len(long_string)

Out[14]: 34

In [15]: long_string1 = 'supercali,fragilisticexpiali,docious'
         long_string1.split(',')

Out[15]: ['supercali', 'fragilisticexpiali', 'docious']

In [16]: long_string_split = long_string1.split(',')
         ','.join(long_string_split)

Out[16]: 'supercali,fragilisticexpiali,docious'
```

- ✓ 変数が**文字列**なら文字数。配列なら要素数
- ✓ 分割（split）で**文字列**→配列への変換
- ✓ 結合（join）で配列→**文字列**への変換

文字列の操作

第3回資料

大文字と小文字

```
In [7]: string = 'we think, therefore we are'

In [8]: string.capitalize()   # 先頭を大文字
Out[8]: 'We think, therefore we are'

In [9]: string.title()    # すべての単語の頭を大文字
Out[9]: 'We Think, Therefore We Are'

In [10]: string.upper()   # すべての文字を大文字
Out[10]: 'WE THINK, THEREFORE WE ARE'

In [11]: string.lower()   # すべての文字を小文字
Out[11]: 'we think, therefore we are'
```

文字列操作の練習問題

Jupyter Notebookで解答してください

1. 三重引用符　”””～”””　の構文を使い，複数行の全角の文字列を書きなさい。そして，それを表示しなさい

2. 下記の文字列を操作しなさい。

   “the Kyoto College of Graduate studies for informatics”

   a. `upper()`を使って，全文字を大文字にしなさい
   b. `lower()`を使って，全文字を小文字にしなさい
   c. `title()`を使って，各単語の最初の文字を大文字にしなさい
   d. `capitalize()`を使って，文章の最初の文字を大文字にしなさい

文字列操作の練習問題

３．　下記の文字列を操作しなさい。

“　　Kyoto College of
Graduate studies for
informatics　　“

a. `.strip()`を使って，上記の文字列から前後のスペースを取りなさい。
b. 同様に`.rstrip()`を使うとどうなるか確認しなさい
c. 練習問題２の全角の文字列で，`.strip()` `.rstrip()`を使うとどうなるか確認しなさい

文字列の操作メソッド

- 文字を数える：count()
- 文字を探す：find()
- 文字を置換する：replace()

```
In [1]: string = 'apple pie'
        string.count('p')
Out[1]: 3

In [2]: string.find('p')
Out[2]: 1

In [3]: string.find('i')
Out[3]: 7

In [4]: string.find('y')
Out[4]: -1

In [5]: string.replace('pie','juice')
Out[5]: 'apple juice'
```

文字列操作の練習問題

４．`find()`, `rfind()` `startswith()`, `endswith()`を使うと，文字列の中の探したい文字や数字など見つけることができる。また，`replace()`は，文字列の一部を変更することができる

下記の文字列について，a~dまでの問題を解きなさい。

”the kyoto college of graduate studies for informatics”

a) `find()`を使って，`kyoto`の文字を探しなさい
b) `find()`を使って，`at`の文字を探しなさい。また、`rfind()`を使って探し，違いを確認しなさい
c) `startswith()`を使って，`kyoto`は一番目の文字であるかどうか探しなさい。次に`the`について調べなさい
d) `replace()`を使って，` at `を`---`に変更しなさい

正規表現とは？

- **正規表現**（Regular Expression）とは，「検索」や「置換」で指定する**文字列をパターン表現する方法**で，**プログラミング言語**やテキストエディタなどで利用できる（つまりコンピュータが理解できる）
- 正規表現には，「パターンを表現するための記号＝**メタ文字**」が多数用意されており，それらを組み合わせることで，「aから始まる半角英字」「3桁の任意の数字」「行頭の2文字」といった柔軟な文字列パターンを指定することができる

※『任意のEXCELファイル名』を示す『*.xlsx』の*がメタ文字

ただし，正規表現にはプログラム言語やテキストエディタなどにより「方言」があり，メタ文字や記述方法が異なるので注意が必要

Pythonでのメタ文字（特殊文字）

- 下記は1例。その他のメタ文字や詳しい使い方は公式ドキュメントを参考 https://docs.python.jp/3/library/re.html

特殊文字	説明	例	matchする例	matchしない例
.	改行以外の任意の一文字	a.c	abc	ac abbc
^	文字列の先頭	^ab	abc	zab
$	文字列の末尾	ab$	zab	abc
*	直前の文字の0回以上の繰り返し	ab*	a ab abb	aa ac
+	直前の文字の1回以上の繰り返し	ab+	ab abb	a
?	直前の文字の0回または1回	ab?	a ab	abb
{m}	直前の文字のm回の繰り返し	a{2}	aa	a aaa
{m,n}	直前の文字のm回からn回の繰り返し	a{1,2}	a aa	aaa
¥	特殊文字のエスケープ	¥.	.	a
[文字の集合]	集合の中の1文字	[a-c]	a b c	d
[^文字の集合]	集合に含まれない1文字	[^a-c]	d	a b c
|	いずれか	a|b	a b	c
()	グループ化	(ab|cd)	ab cd	abc ac

メタ文字を使った検索パターン

探したい文字列	メタ文字によるパターン	当てはまる例	当てはまらない例
「京都」で始まる文字列	^京都.*	京都府 京都タワー 京都好き	滋賀県 今日も京都は晴れ
「ね。」で終わる文字列	.*ね。$	いいね。 ねねね。 だよね。	いいわ。 いいのね！
「〇〇cm」	[0-9]+cm	30cm 1,000,000cm ５０cm	30m 30センチメートル センチcm

正規表現をいつ使うか？

- 文字列の書式チェック
  - 例えば，メールアドレスでは使える文字に制限がある（アットマーク(@)が必須である）。このような場合「メールアドレスとして正しいか」という書式のチェックに正規表現を利用する
- SNSでの記事検索
  - 例えば，ウェブ・スクレイピング技術を使い，自社製品の評価をしているブロガーの記事を特定パターン（WEBのタイトルから抽出など）で収集・分析する
- 自社サイトへのNGワードの投稿の検索

Pythonでの正規表現操作

- **標準ライブラリ**の**モジュール** re の提供メソッドを使い，正規表現されたパターンと一致（マッチ）する文字列を検索する

モジュール re のメソッド	目的
match(*パターン, 文字列*)	文字列の先頭で正規表現とマッチするか判定する
search(*パターン, 文字列*)	文字列を操作して，正規表現がどこにマッチするか調べる
findall(*パターン, 文字列*)	正規表現にマッチする部分文字列を全て探しだしリストとして返す
finditer(*パターン, 文字列*)	正規表現にマッチする部分文字列を全て探しだし iterator として返す

match() と search() はマッチするものが見つかれば，**Match オブジェクト・インスタンス**を返す。**Match オブジェクト**オブジェクトにはマッチした情報が含まれる：マッチの開始と終了位置，マッチした部分文字列，など（次頁参照）
見つからなければ None を返す。

Pythonでの正規表現操作

- **Match オブジェクト・インスタンス**はいくつかのメソッドと属性（インスタンス変数）を持っており，重要なのは以下

メソッド/属性	目的
group()	正規表現にマッチした文字列を返す
start()	マッチの開始位置を返す
end()	マッチの終了位置を返す
span()	マッチの位置 (start, end) を含むタプルを返す

Pythonでの正規表現操作：matchメソッド

```
import re
pattern = "https?://[^/]+/"
URL = "https://www.amazon.co.jp/dd/A09TCPZX"

result = re.match(pattern, URL)
print(result)
print(result.group())
```

```
<re.Match object; span=(0, 25), match='https://www.amazon.co.jp/'>
https://www.amazon.co.jp/
```

‘http://’または’https://’で始まり，’/’までを取り出すプログラム

- **正規表現**のパターン ; https?://[^/]+/
  - https? ; httpまたはhttps。　?は前の文字(s)が0または1つを意味する
  - [^/]+/ ; スラッシュ（/）以外^の文字[]の繰返し+で/が出るまで
- **match**メソッドは文字列の先頭からパターンを検索
  - ・match オブジェクトを変数resultで参照
  - ・groupメソッドは一致したmatch オブジェクトの文字列を返す

Pythonでの正規表現操作：searchメソッド

```
import re
pattern = "https?://[^/]+/"
URL = """アマゾンのサイト：https://www.amazon.co.jp/dd/A09TCPZX
         KCGIのサイト：https://www.kcg.edu/curriculum"""

result = re.search(pattern, URL)
print(result)
print(result.group())
```

```
<re.Match object; span=(9, 34), match='https://www.amazon.co.jp/'>
https://www.amazon.co.jp/
```

‘http://’または’https://’で始まり，’/’までを取り出すプログラム **2**

- 正規表現のパターン ; https?://[^/]+/ は同じ
- 検索対象の文字列が違う（urlは文中）
- **search**メソッドはパターンが文字列のどこにマッチするか検索

（参考）Pythonでの正規表現操作

```
import re
pattern = "https?://[^/]+/"
URL = """アマゾンのサイト：https://www.amazon.co.jp/dd/A09TCPZX
         KCGIのサイト：https://www.kcg.edu/curriculum"""

result = re.match(pattern, URL)
print(result)
```

```
None
```

‘http://’または’https://’で始まり，’/’までを取り出すプログラム**2**’

- 正規表現のパターン；https?://[^/]+/ は同じ
- 検索対象の文字列が違う（urlは文中）
- **match**メソッドでは先頭から検索するため見つからない

Pythonでの正規表現操作：findallメソッド

```
import re
pattern = "https?://[^/]+/"
URL = """アマゾンのサイト：https://www.amazon.co.jp/dd/A09TCPZX
         KCGIのサイト：https://www.kcg.edu/curriculum"""

result = re.findall(pattern, URL)
print(result)
print(result[1],len(result))
```

```
['https://www.amazon.co.jp/', 'https://www.kcg.edu/']
https://www.kcg.edu/ 2
```

‘http://’または’https://’で始まり，’/’までを取り出すプログラム **3**

- 正規表現のパターン；https?://[^/]+/のパターンも検索対象の文字列も同じ
- findallメソッドはマッチする部分文字列を全て探しだしリストとして返す

Pythonでの正規表現操作

```
import re
strings = """家の郵便番号は612-1010です。
           携帯電話は080(1000)2000です"""
#通便番号をだけを抽出したい
pattern1 = "[0-9]{3}-[0-9]{4}"

result = re.findall(pattern1, strings)
print(result)
```

```
['612-1010']
```

```
import re
strings = """家の郵便番号は612-1010です。
           携帯電話は080-1000-2000です"""
#通便番号をだけを抽出したい
pattern1 = "[0-9]{3}-[0-9]{4}"

result = re.findall(pattern1, strings)
print(result)
```

```
['612-1010', '080-1000']
```

郵便番号を取り出すプログラム

- 正規表現のパターン；[0-9]{3}-[0-9]{4}　3桁の数字, - , 4桁の数字
- findallメソッドで検索

左はOKだが・・・

Pythonでの正規表現操作

```
# 全角文字の検索
import re
text="""京都情報大学院大学に通学しています。
京都市に住んでいません、
大阪市に住んでいます。"""

result = re.match('京都', text) # matchメソッド
print(result)
print(result.group())
print(result.start())
print(result.end())
print(result.span())
```

```
<re.Match object; span=(0, 2), match='京都'>
京都
0
2
(0, 2)
```

- 全角(2バイトコード)文字の検索
- matchメソッドで先頭から検索

Pythonでの正規表現操作

```
import re
text="""京都情報大学院大学に通学しています。
京都市に住んでいません、
大阪市に住んでいます。"""

result = re.match('大学', text)
if result:  # 見つかればTrueが返る
    print(result.group())
else:       #見つからないとNoneが返る
    print(result)
```

```
None
```

✓ ‘大学‘では先頭からは見つからない

```
import re
text="""京都情報大学院大学に通学しています。
京都市に住んでいません、
大阪市に住んでいます。"""

# '大学'を後ろに含む文字列パターンで検索
result = re.match('.*大学', text)
print(result)
print(result.group())
```

```
<re.Match object; span=(0, 9), match='京都情報大
京都情報大学院大学
```

➢ .*（ワイルドカード）を使い，matchメソッドで検索

Pythonでの正規表現操作

```
import re
text="""京都情報大学院大学に通学しています。
京都市に住んでいません、
大阪市に住んでいます。"""

# searchメソッド
result = re.search('大学', text)
print(result)
```

```
<re.Match object; span=(4, 6), match='大学'>
```

- searchメソッドなら文中の‘大学‘を見つけられる

```
import re
text="""京都情報大学院大学に通学しています。
京都市に住んでいません、
大阪市に住んでいます。"""

# matchメソッドと等価
result = re.search('.*大学', text)
print(result.group())
print(result.start())
print(result.end())
print(result)
```

```
京都情報大学院大学
0
9
<re.Match object; span=(0, 9), match='京都情報大学院大
```

- ‘.* 大学‘（ワイルドカード）を使った検索ではmatchメソッドとsearchメソッドは等価

Pythonでの正規表現操作

```
import re
text = '''京都情報大学院大学に通学しています。
          京都市に住んでいません、
          大阪市に住んでいます。'''
result = re.findall('京都.', text) # findallメソッド
print(result)                      # listを返す
result = re.findall('.*市', text) # '市'を後ろに持つ文字列
print(result)
```

```
['京都情', '京都市']
['¥u3000¥u3000¥u3000¥u3000¥u3000京都市', '¥u3000¥u3000¥u3000¥u3000¥u3000大阪市']
```

- ➢ findallメソッドによる検索。結果すべてをリストで返す
- ✓ ‘全角空白’も文字列　※改行(¥n)が区切り

```
result = re.split('。|、', text) # splitメソッド　正規表現|
print(result)                    # パターンで分割。リストを返す
```

```
['京都情報大学院大学に通学しています', '¥n¥u3000¥u3000¥u3000¥u3000¥u3000京都市に住んでいません', '¥n¥u
阪市に住んでいます', '']
```

- ➢ reモジュールのsplitメソッドの使用例
- ✓ ‘|’(メタ文字のor)でゆらぎを検索

```
result = re.sub('市', '府', text) # subメソッド
print(result)                     # パターンで置き換え。文字
```

```
京都情報大学院大学に通学しています。
          京都府に住んでいません、
          大阪府に住んでいます。
```

- ✓ reモジュールのsubメソッドの使用例

Pythonでのその他の特殊文字

特殊文字	説明	同義のパターン
¥d	任意の数字	ASCIIの場合，[0-9]
¥D	¥d以外	
¥s	任意の空白文字	ASCIIの場合，[¥t¥n¥r¥f¥v]
¥S	¥s以外	
¥w	任意のUnicode単語文字	ASCIIの場合，[a-xA-Z0-9_]
¥W	¥w以外	
¥A	文字列の先頭	^
¥Z	文字列の末尾	$
¥b	単語の境界（¥wと¥Wの間）	
¥B	単語の境界以外の文字間	

☞ いくつかの特殊文字がエスケープシーケンスとバッティング

・ ’¥b’はエスケープシーケンスでは’バックスペース’

文字列

エスケープ文字（エスケープシーケンス）

エスケープ文字	意味
¥n	改行
¥t	tab
¥r	キャリッジリターン
¥’	シングルクォート
¥”	ダブルクォート

・‘¥’はwindows
・Macは’＼’
(バックスラッシュ)

Pythonでの正規表現操作

```
import re
strings = """家の郵便番号は612-1010です。
           携帯電話は080(1000)2000です"""
#通便番号をだけを抽出したい
pattern1 = r"¥d{3}-¥d{4}"

result = re.findall(pattern1, strings)
print(result)
```

```
['612-1010']
```

- 特殊文字¥dは[0-9]と等価
- 特殊文字を使うパターンには先頭に’ｒ’を付加
  ⇒pythonは’ｒ’で¥を特殊文字と認識

Pythonでの正規表現操作

```
import re
content = 'hellow Python, 123, end'

# 先頭が数字の文字列を探索
pattern1 = '¥b(¥d+)'  # ¥bはエスケープ（バックスペース）
pattern2 = r'¥b(¥d+)' # ¥bは特殊文字（単語以外の境界）
result1 = re.findall(pattern1, content)
result2 = re.findall(pattern2, content)

print(result1)
print(result2)
```

```
[]
['123']
```

- 先頭に’r’を付加しないと、’¥b’をバックスペースと認識
- ’r’を付加すると、 ¥b’を単語以外の境界と認識

Pythonでの正規表現操作

```
import re
content = """hellow Python, 123, end
          goodby Python"""
# ()で探したい任意の数字¥dを指定
pattern = r'.*?(¥d+).*'
result = re.search(pattern, content)
if result:
  # group()で全文字
  print(result.group(0))
  # group(1)で探したい任意の数字
  print(result.group(1))
```

```
hellow Python, 123, end
123
```

特殊文字を使うパターンの検索

- 任意の数字列をsearchメソッドで探す例（数字列の前後にワイルドカード）
- groupメソッドの引数に注目
- ?（直前の文字の0回または1回）の有無で結果が異なる