Python for bioinformatics

Contents in this document are licensed under CC BY 4.0.

コンテンツ

Introduction	Basic	Packages	Advanced
1. プログラミング言語概要 2. Python 環境構築	3. データ型 4. 基本文法	5. テキスト処理 6. 数値計算 NumPy 7. データ処理 Pandas 8. データ可視化 matplotlib 9. バイオインフォマティクス	10.オブジェクト指向

テキスト処理

- テキスト処理
- ファイル処理
- 正規表現

テキスト処理

- テキスト処理
- ファイル処理
- 正規表現

テキストデータ

テキストデータはアルファベットや仮名などの文字からなるデータを指す。Python のオブジェクトには、数値の他にアルファベット・漢字・仮名などの文字を代入することもできる。文字をオブジェクトに代入するとき、その文字が、オブジェクトの名前ではなく、文字のデータであることを明示するために、文字データの両側を引用符で囲む。

```
s = 'a'


t = 'B'


u = 't'
u
# 't'


v = t
v
# 'B'
```

t が引用符で囲まれているため、文字データとしての t がオブジェクト u に代入される。

t が引用符で囲まれていないため、t はオブジェクトである。オブジェクト t の内容がオブジェクト v に代入される。

テキストデータ

単語や文章などのような文字列も、1 つのオブジェクトに代入できる。この場合、1 つのオブジェクトに、複数個の文字が保存されている、と捉えることができるため、このオブジェクトを文字からなるリストのように扱うことができる。添字を指定して、特定の位置の文字を取り出したり、スライスして部分文字列を取り出したりすることができる。また、for 構文を使用して、文字列中の文字を 1 つずつ順に取り出すこともできる。

```
s = 'Smile. Tomorrow will be worse.'

s[0]
# 'S'



s[7:15]
# 'Tommorow'



for t in s:
    print(t)
# 'S'
# 'm'
# 'i'
# 'l'
# 'e'
# '.'
```

テキストデータ

文字列を保持しているオブジェクトに対して、+ 演算子が定義されている。+ 演算子を使用することで、複数の文字列を連結することができる。

```
s1 = 'Smile.'
s2 = 'Tomorrow will be worse.'
s3 = ' '

s = s1 + s2
s
# 'Smile.Tomorrow will be worse.'


s = s1 + s3 + s2
s
# 'Smile. Tomorrow will be worse.'


s = s1 + ' ' + s2
s
# 'Smile. Tomorrow will be worse.'
```

問題 T1-1

5 min

赤色で書かれたオブジェクトが保持している値を答えよ。

```
s1 = 'anything that can go wrong'
s2 = 'it will go wrong'

s1[:8]

s2[3:]

s3 = s1 + ',' + s2[3:]
s3
```

```
s1 = 'left to themselves'
s2 = 'things tend to go'
s3 = 'from bad to better'

s = s1 + ', ' + s2
s = s + ' ' + s3[:12] + 'worse.'
s
```

問題 T1-2

10 min

与えられた塩基配列の中から ATG を検索し、ATG が見つかればその位置を、そうでなければ -1 を出力するプログラムを for/while 構文や if 構文で作成せよ。

```
s = 'CCACAGTCATGTGTCAGTCGTAAGT'



# 8
```

```
s = 'CATTGTGTCACAGCCAGTCGTAAGT'



# -1
```

問題 T1-3

⏰ 20 min

与えられた塩基配列中の A、C、G、T の出現回数および出現確率を求めよ。

```
s = 'AAAGGTCTTT'
```

与えられた塩基配列に含まれる AA、AC、・・・、TT のような 2 文字パターンの出現回数を求めよ。

```
s = 'AAAGGTCTTT'
```

問題 T1-4

15 min

for/while 構文や if 構文を使用して、与えられた塩基配列の相補鎖を求めよ。

```
s = 'AAAGGTC'




# 'TTTCCAG'
```

for/while 構文や if 構文を使用して、与えられた塩基配列の逆相補鎖を求めよ。

```
s = 'AAAGGTC'




# 'GACCTTT'
```

文字列操作関数

文字列の操作には、連結、分割、検索、置換などがある。これらの操作は、次の関数（文字列メソッド）で行える。

a + b	文字列 a の後ろに文字列 b を連結する。
a.split(',')	コンマ , を区切り文字として、文字列を分割して、部分文字列のリストに変換する。
a.find('ATG')	文字列 a の先頭から ATG を探す。見つかればその位置の添字を返し、そうでなければ -1 を返す。
a.replace('TAG', '*')	文字列 a 中ににある部分文字列 TAG を * に置き換える。

```
s = '21,31,41,51,61'

s.split(',')
# ['21', '31', '41', '51', '61']

s.find('41')
# 6

s.find('71')
# -1

s.replace(',', ';')
# '21;31;41;51;61'
```

文字と数値の交互変換

Python の世界で、文字と数値の扱い方が異なる。文字と文字の足し算では文字同士が連結され、数値と数値の足し算では数学的に和が計算される。文字と数値の足し算は定義されておらず、エラーが出る。

```
a = '12'
b = 21


a + b
# TypeError: can only concatenate str
(not "int") to str
```

文字列 a に、整数 b を連結できないことを示すエラー。

```
b + a
# TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

整数 b に文字列 a を足せないことを示すエラー。

文字から数値への変換

文字列を整数または小数に変換することができる。整数への変換は int 関数を利用し、小数への変換は float 関数を利用する。これらの関数は、すべての文字列を整数・小数に変換できるわけではない。変換できない場合は、エラーが起こる。

```
a1 = '12'
a2 = '12.345'
a3 = '1.23e6'

int(a1)
# 12

int(a2)
# ValueError: invalid literal for int()
with base 10: '12.345'

float(a1)
# 12.0

float(a2)
# 12.345

float(a3)
# 1230000.0
```

数値から文字への変換

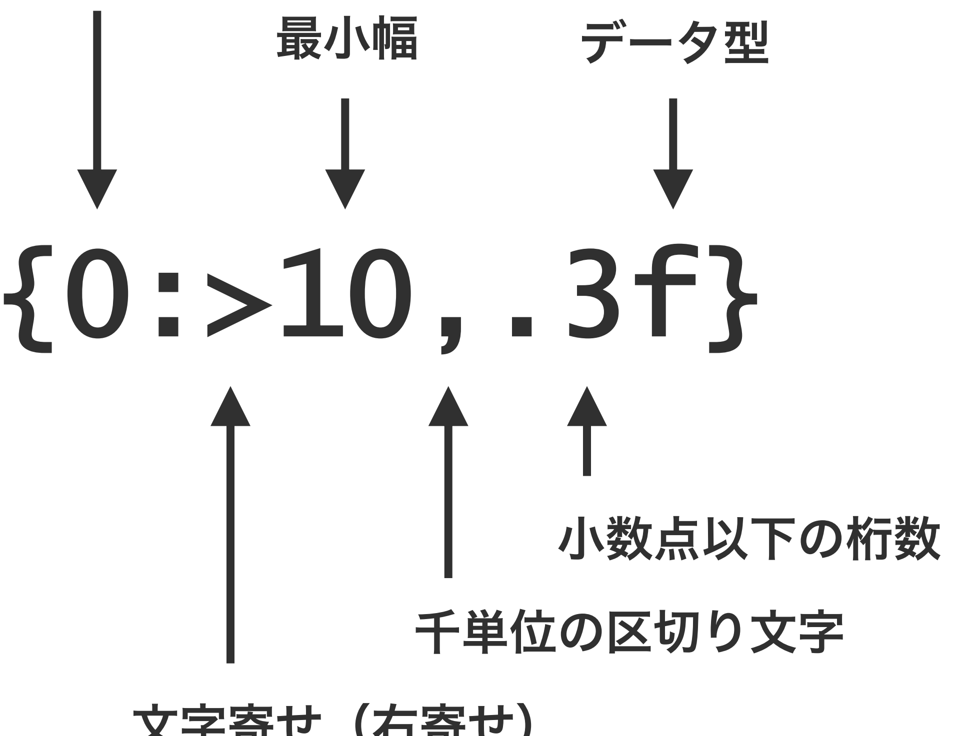
数値から文字への変換は str 関数を使用する。基本的に、すべての数値を文字に変換できる。桁数の多い小数を文字に変換するとき、桁落ちが発生する場合がある。

```
b1 = 12
b2 = 12.345
b3 = 12.3456789012345678901
b4 = 1.23e6


str(b1)
# '12'


str(b2)
# '12.345'


str(b3)
# '12.345678901234567'


str(b4)
# '1230000.0'
```

数値から文字への変換

参考

数値から文字への変換は、str 関数のほかに format 関数も用意されている。format 関数を使うことで、有効桁数を指定したり、ゼロ埋めしたりすることができる。

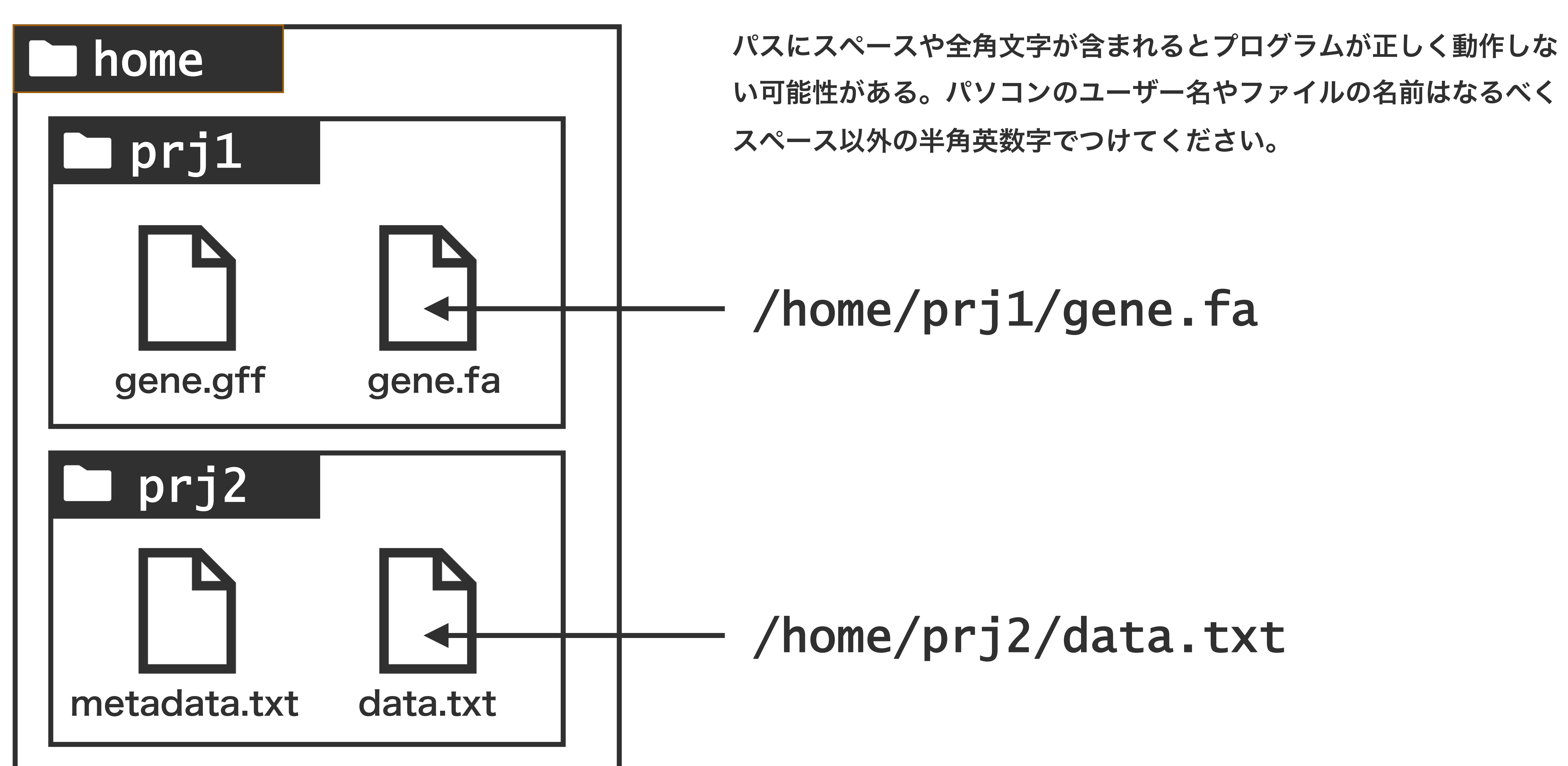



```
b1 = 12
b2 = 12.3456789


'{:4d}'.format(b1)
# '  12'

'{:04d}'.format(b1)
# '0012'

'{:.3f}'.format(b1)
# '12.000'

'{:.3f}'.format(b2)
# '12.345'

'{:.2e}'.format(b2)
# '1.23e+01'
```

テキスト処理

- テキスト処理
- ファイル処理
- 正規表現

パス



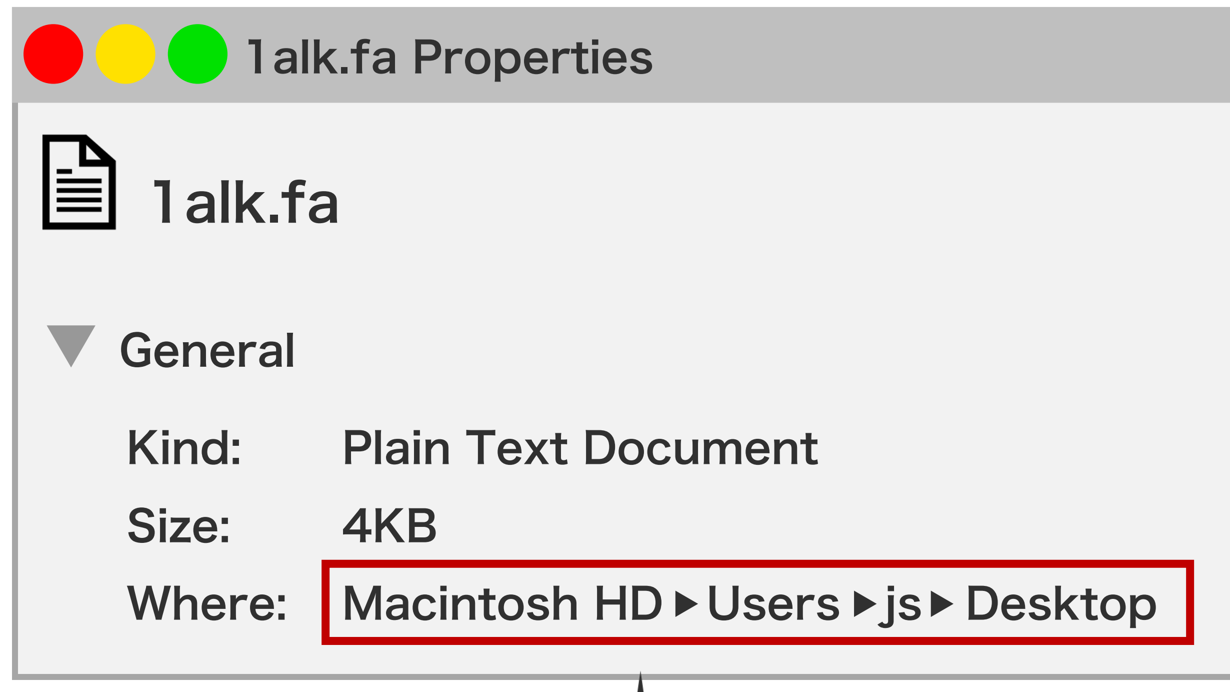
パスにスペースや全角文字が含まれるとプログラムが正しく動作しない可能性がある。パソコンのユーザー名やファイルの名前はなるべくスペース以外の半角英数字でつけてください。

ファイルパス（Macintosh）

1. パスを調べたいファイルを右クリックし、「Get Info」を選ぶ。

2. 「General」タブの「Where」項目にファイルへのパスが記載されている。

   - ファイルパスの「Macintosh HD」は最上層を表し、パスを書くとき「/」と書く。
   - 小さい三角形はフォルダの包含関係を表すので、パスを書くときは「/」と書く。



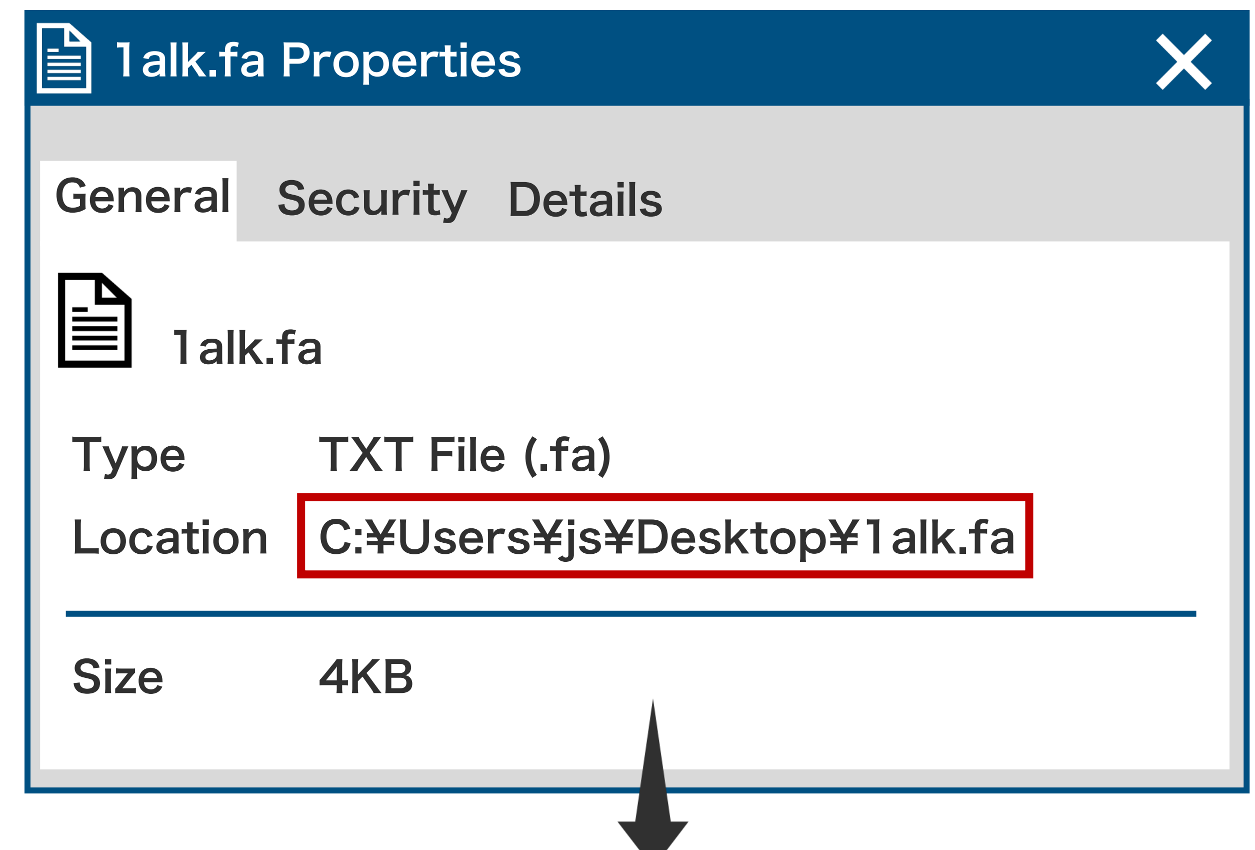
ファイルパス（Windows）

1. パスを調べたいファイルを右クリックし、「Properties」を選ぶ。

2. 「General」タブの「Location」項目にファイルへのパスが記載されている。

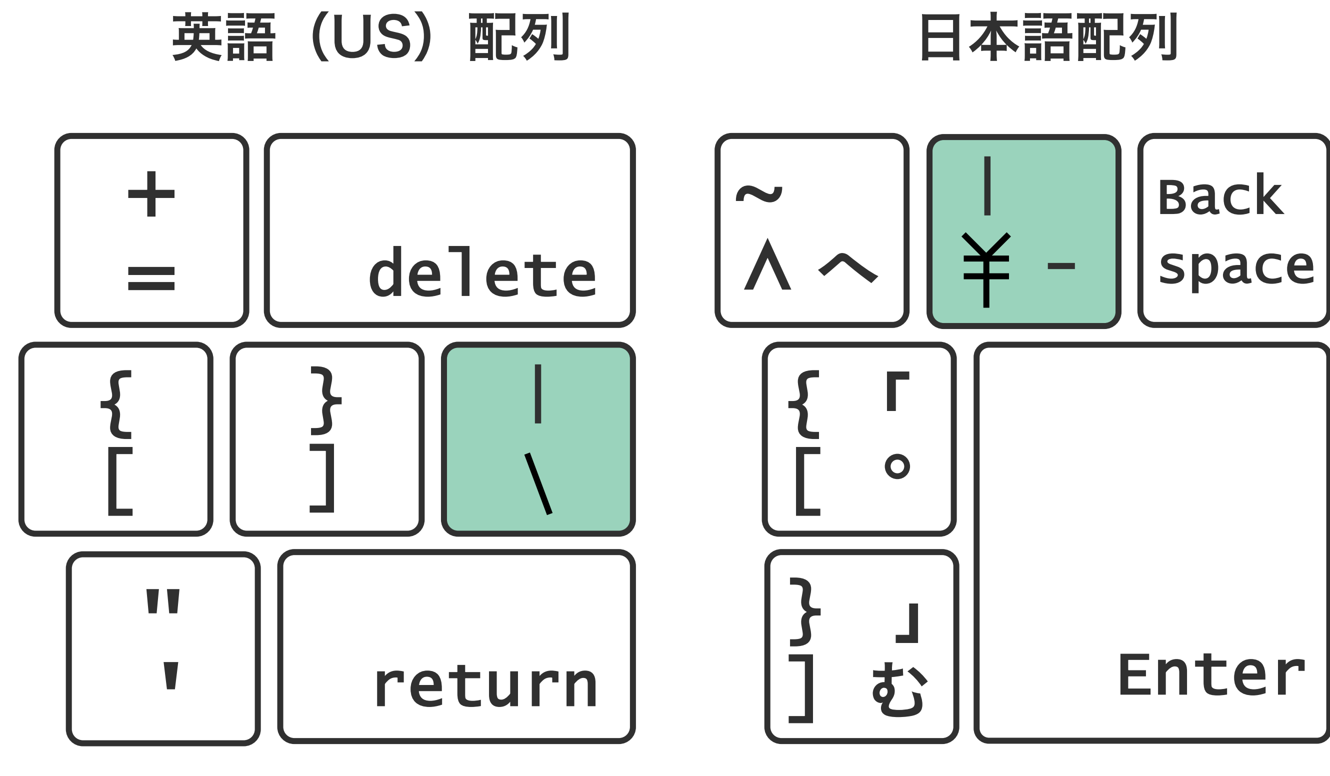
   - **日本語環境の場合、「＼」が「¥」として表示される。**どちらもコンピューター内部では 0x5C として認識されている。

   - パスとして使用するとき、Location 欄に書かれているパス中の**「¥」を「/」に置き換える。**



バックスラッシュ（＼）・円マーク（¥）

バックスラッシュは、OS の種類、言語環境や文字コードによって、ディスプレイでの表示が異なる。日本語環境のシステムであれば円マーク「¥」として表示され、それ以外の言語環境では「＼」として表示される。バックスラッシュと円マークは、表示が異なるものの、コンピューター上では同一のコード 0x5C として扱われる。また、バックスラッシュは特別な制御を行うための文字であり、一般的な文字として使用されることは少ない。



macOS 日本語環境を使用している場合は、＼ と ¥ の両方を入力できる。Option を押しながら＼または¥キーを押すことで、 ＼と¥の入力の切り替えができる。

ファイル

Python でファイルを読み込むには、open 関数を使用する。open 関数に、ファイルへのパスとともにオープン・モードを与えて使う。

モード	意味
r	読み込みモード。ファイルが存在しない場合はエラーになる。
w	書き込みモード。ファイルが存在しない場合は新規作成される。ファイルが存在する場合は、既存のファイルが削除されたうえで新規作成される。
a	追記モード。既存のファイルの最後に追記する。ファイルが存在しない場合は新規作成される。

https://aabbdd.jp/data/1alk.fa

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)

# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

一部のブラウザでは、ダウンロードしたファイルに .txt が付けられることがあるため、エラー出た場合は、'1alk.fa.txt' で試してください。

ファイル読み込み

パス f に保存されているファイルを、r モードで開き、ファイルハンドルにセットアップする。 ▶

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)

# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

https://aabbdd.jp/data/1alk.fa

ファイル読み込み

ファイルの内容がリストに変換されてファイルハンドルに代入される※。ファイルの i 行目の文字列情報が、リストの i 番目の要素に代入されている。そのため、for 構文を利用して、リストの先頭から要素を順に取り出せば、ファイルの内容を 1 行目から順に取り出せるようになる。

このように見えるが、・・・

```
>1ALK
TPEMPV
TGQYTH
AHVTSR
```

```
fh = [
   '>1ALK',
   'TPEMPV',
   'TGQYTH',
   'AHVTSR'
]
```

▶

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)

# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

※ 正確には、ファイルの何行の何文字目まで読み込んだかの位置情報（ファイルポインター）がファイルハンドルに保存される。ファイルハンドルに対して for 構文を適用すると、ポインターを自動的に 1 行ずつ進めさせることができる。また、read メソッドと for 構文を組み合わせることで、ポインターを 1 文字ずつ進めさせることもできる。

ファイル読み込み

ファイルの内容がリストに変換されてファイルハンドルに代入される※。ファイルの i 行目の文字列情報が、リストの i 番目の要素に代入されている。そのため、for 構文を利用して、リストの先頭から要素を順に取り出せば、ファイルの内容を 1 行目から順に取り出せるようになる。

このように見えるが、実際は特殊文字が存在している！

特殊文字

```
>1ALK\n
TPEMPV\n
TGQYTH\n
AHVTSR\n
```

特殊文字

```
fh = [
    '>1ALK\n',
    'TPEMPV\n',
    'TGQYTH\n',
    'AHVTSR\n'
]
```

▶

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)


# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

※ 正確には、ファイルの何行の何文字目まで読み込んだかの位置情報（ファイルポインター）がファイルハンドルに保存される。ファイルハンドルに対して for 構文を適用すると、ポインターを自動的に 1 行ずつ進めさせることができる。また、read メソッドと for 構文を組み合わせることで、ポインターを 1 文字ずつ進めさせることもできる。

ファイル読み込み

すべての行の読み込みが終わり、ファイルが閉じられる。 ▶

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)

# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

エスケープシーケンス

Python では、特殊な用途向けに、いくつかの特殊文字が定義されている。例えば、改行とタブがその特殊文字にあたる。改行文字は、人には見えないが、コンピュータが改行として認識するために必要な特殊文字である。限られたアルファベットと数字の中で、このような特殊文字を表すためには、既存文字を組み合わせて表現する。一般に、バックスラッシュに 1 文字を足して、特殊文字を表現していることが多い。例えば、改行ならば \n、タブならば \t のように表現する。

※ ＼ と ￥ は同じ文字コードであることに注意！

```
s = 'Smile. nTomorrow will be worse.'
s
# 'Smile. nTomorrow will be worse.'
```

```
s = 'Smile. \nTomorrow will be worse.'
s
# 'Smile.'
# 'Tomorrow will be worse.'
```

```
s = 'Smile. \\nTomorrow will be worse.'
s
# 'Smile. \nTomorrow will be worse.'
```

問題 F1-1

5 min

リスト fh の要素のうち、「>」から始まる要素の個数を求めよ。

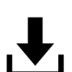
```
fh = ['>1ALK:A',
      'TPEMPVL',
      'TGQYTHA',
      '>1ALK:B',
      'TPEMPVL',
      'TGQYTHA']
```

問題 F1-2

⏰ 10 min

1alk.fa ファイルを読み込んで、「>」から始まる行の行数を求めよ。

```
f = '1alk.fa'
n = 0
```


https://aabbdd.jp/data/1alk.fa

問題 F1-3

⏰ 15 min

ft.fa ファイルは FASTA 形式のテキストファイルである。このファイルは「>」から始まる行に遺伝子名が記載され、それ以降の行に遺伝子の塩基配列が記載されている。ft.fa に記載されいてる遺伝子の塩基配列の長さを求めよ。

```
f = 'ft.fa'
l = 0
```

https://aabbdd.jp/data/ft.fa

問題 F1-4

⏰ 15 min

ft.fa に記載された塩基配列について、A の出現確率を求めよ。

```
f = 'ft.fa'
prob_a = 0
```

https://aabbdd.jp/data/ft.fa

問題 F1-5

⏰ 20 min

diversity_galapagos.txt は、タブ区切りのテキストファイルであり、ガラパゴス島における種の多様性データが記載されている。このファイルを読み込み、面積（Area 列）の最も大きい島の名前（Island 列）を答えよ。ただし、このテキストファイルには、「#」から始まるコメント行とデータの属性を示すヘッダー行が含まれていることに注意。

```
f = 'diversity_galapagos.fa'
max_area_island_name = 0
```

https://aabbdd.jp/data/diversity_galapagos.txt

問題 F1-6

20 min

diversity_galapagos.txt は、タブ区切りのテキストファイルであり、ガラパゴス島における種の多様性データが記載されている。面積（Area）あたりの種数（Species）が最も大きい大きい島の名前とそのときの面積当たりの種数を求めよ。

```
f = 'diversity_galapagos.fa'
max_area_island_name = 0
```

https://aabbdd.jp/data/diversity_galapagos.txt

ファイル書き込み

ファイルの書き込みには w および a モードが定義されている。このほかにも w+ や a+ などのモードも定義されているが、初めは w モードのみ使えればよい。

モード	意味
r	読み込みモード。ファイルが存在しない場合はエラーになる。
w	書き込みモード。ファイルが存在しない場合は新規作成される。ファイルが存在する場合は、既存のファイルが削除されたうえで新規作成される。
a	追記モード。既存のファイルの最後に追記する。ファイルが存在しない場合は新規作成される。

```
f = 'output.txt'

with open(f, 'w') as outfh:
    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

パス f を書き込みモードで開き、ファイルハンドル ▶ にセットアップする。

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

abcdefg をファイルに書き込む ▶

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

1234567 をファイルに書き込む ▶

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

```
f = 'output.txt'


with open(f, 'w') as outfh:

     outfh.write('abcdefg')
     outfh.write('1234567')
```

書き込みが終了し、ファイルが閉じられる。 ▶

output.txt

```
abcdefg1234567
```

ファイル書き込み

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg\n')
    outfh.write('1234567\n')
```

書き込みが終了し、ファイルが閉じられる。 ▶

output.txt

```
abcdefg
1234567
```

問題 F1-7

20 min

ft.fa ファイルは FASTA 形式のテキストファイルである。このファイルは「>」から始まる行に、塩基配列の名前が記載され、それ以降の行は塩基配列が大文字で記載されている。配列の名前を変更しないで、塩基配列をすべて小文字に変換し、これらの情報を新しいファイル new_ft.fasta に保存するプログラムを作成せよ。

https://aabbdd.jp/data/ft.fa

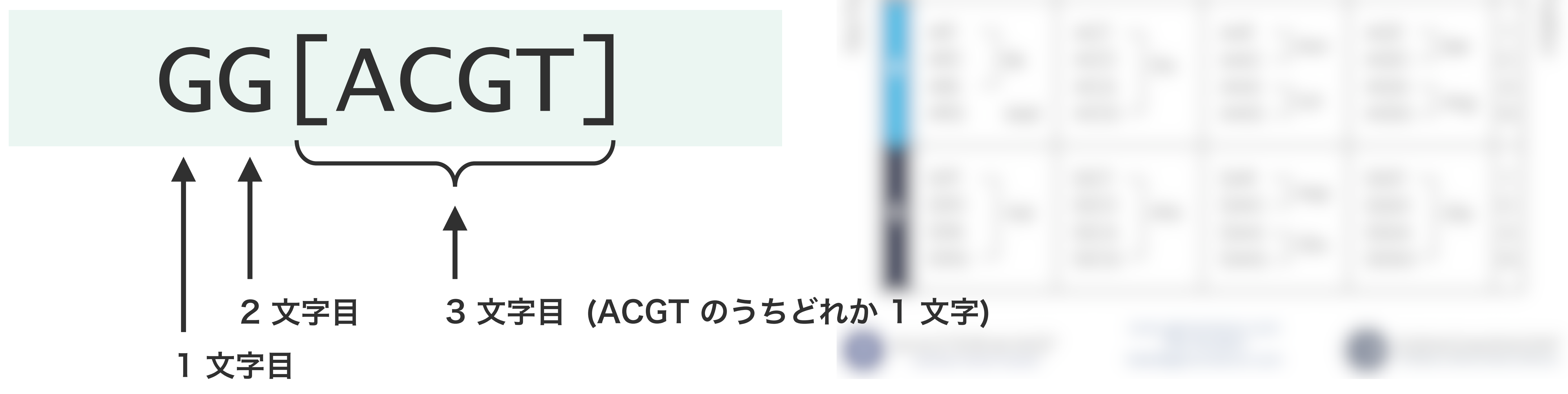
```
f = 'ft.fa'
```

テキスト処理

- テキスト処理
- ファイル処理
- 正規表現

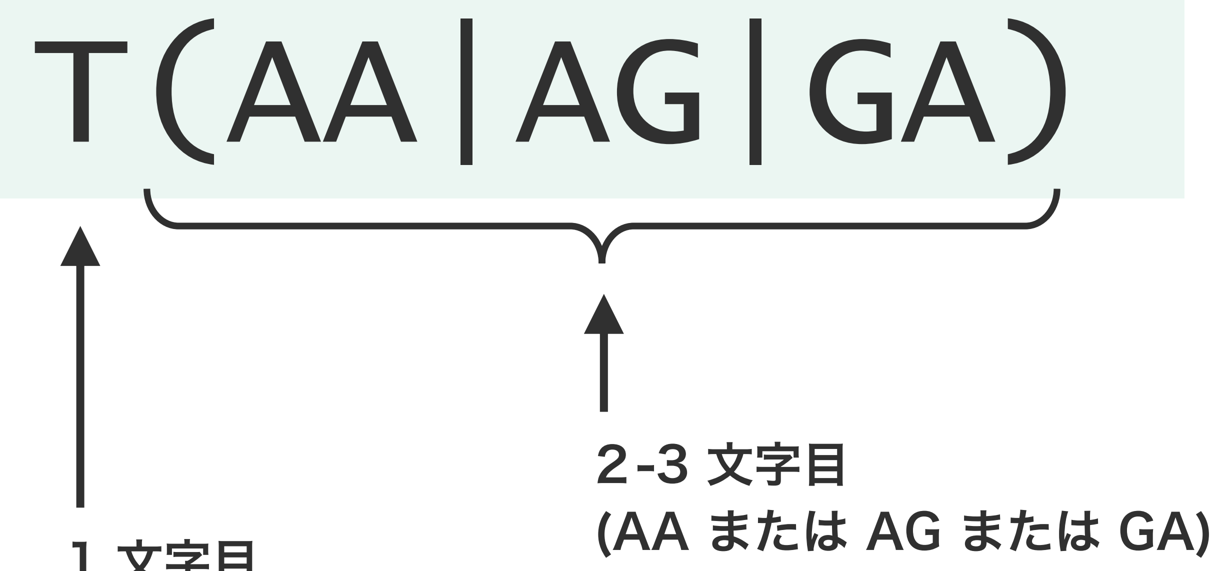
正規表現

正規表現は、文字列のパターンを表現するための表記法である。例えば、グリシンをコードするコドンは GGU、GGC、GGA、GGG がある。このパターンを記述するためには、文字列を使用すると 4 つの文字列を必要とするが、正規表現を使用すると GG[ACGT] のように 1 つのパターンとして記述できる。



正規表現

終止コドンは TAA、TAG、TGA である。1 文字目以外にはパターンが存在しないので、これを正規表現で表すと少し複雑になる。



メタ文字

前出の [| (などの特別な意味を持つ文字をメタ文字と呼ぶ。Python の正規表現でよく使われるメタ文字は、以下のようなものがある。

<table>
<tr><th>メタ文字</th><th>説明</th><th>使用例</th><th>合致する文字列</th></tr>
<tr><td>.</td><td>任意の 1 文字</td><td>GG.</td><td>GGA, GGGBT, ACGGCG, CCAGGDGT, ...</td></tr>
<tr><td>^</td><td>行の先頭</td><td>^TATA</td><td>TATAT, TATAAG, TATAGCC, ...</td></tr>
<tr><td>$</td><td>行の末尾</td><td>TGA$</td><td>TTGA, AACTGA, ACTGA, GTGATGA, ...</td></tr>
<tr><td>*</td><td>0 回以上の繰り返し</td><td>AAC*</td><td>AACCT, TAAG, GCAAT , CGAAACCT, ...</td></tr>
<tr><td>+</td><td>1 回以上の繰り返し</td><td>AAC+</td><td>AACT, TACC, TTACCCT, GCACCCTG, ...</td></tr>
<tr><td>?</td><td>0 回または 1 回の繰り返し</td><td>AAC?</td><td>TAACCG, ATAACC, ACAAA, TAAATAG, ...</td></tr>
<tr><td>{m}</td><td>m 回の繰り返し</td><td>AC{3}</td><td>ACCCCA, TAGACCCAG, TAGACCCCCG, ...</td></tr>
<tr><td>{m,n}</td><td>m 回以上 n 回以下の繰り返し</td><td>AC{2, 4}</td><td>ACCCCA, TACGACCCAG, TAGACCCCCG, ...</td></tr>
<tr><td>[]</td><td>文字クラス</td><td>GG[ACGT]</td><td>GGA, CAGGGC, ACGGC, GGTAC, ...</td></tr>
<tr><td>()</td><td>グループ</td><td>(GG)</td><td></td></tr>
<tr><td>|</td><td>いずれか</td><td>GG(A|CA)</td><td>GGAT, GGGACA, GGGCATG, GGACGT, ...</td></tr>
</table>

文字列検索 search

正規表現で表されている文字列のパターンが、入力文字列に含まれるかどうかを調べることができる。Python で正規表現を利用するには re パッケージの search 関数を利用する。

GTA を含むかどうかを検索 →

GTA が見つかった場合 →

```
import re

dna = 'CAGTCGATAGTACCGAC'



m = re.search(r'GTA', dna)


if m:
    m.group()
    m.start()
    m.end()
    m.span()

# 'GTA'
# 9
# 12
# (9, 12)
```

文字列検索 search

正規表現で表されている文字列のパターンが、入力文字列に含まれるかどうかを調べることができる。Python で正規表現を利用するには re パッケージの search 関数を利用する。

```
import re

dna = 'CAGTCGATAGTACCGAC'
```

GTA, GTC, GTG, GTT を含むかどうかを検索 →

```
m = re.search(r'GT[ACGT]', dna)
```

GTA が見つかった場合 →

```
if m:
    m.group()
    m.start()
    m.end()
    m.span()

# 'GTC'
# 2
# 5
# (2, 5)
```

入力文字列の中に、検索対象パターンが複数含まれている場合、search 関数は最初にマッチした部分の情報のみを出力する。

文字列検索 finditer

finditer 関数は、入力文字列の中に、検索対象パターンにマッチするすべての部分の情報をイテレータ※として取得する。

※ イテレータはリストのようなもの。for 構文で使用することで、各マッチ部分の情報をひとつずつ取得できる。

```
import re

dna = 'CAGTCGATAGTACCGAC'

m = re.finditer(r'GT[ACGT]', dna)
if m:
    for k in m:
        k.group()
        k.start()
        k.end()
        k.span()

# 'GTC'
# 2
# 5
# (2, 5)
# 'GTA'
# 9
# 12
# (9, 12)
```

文字列置換 sub

sub 関数を使用することで、すべてのマッチ部分を他の文字列に置き換えることができる。

```
import re

dna = 'CAGACTAACACAGTGAAGTGTTCA'



dna_2 = re.sub(r'(TAA|TAG|TGA)', '*',
               dna)
dna_2
# 'CAGAC*CACAG*AGTGTTCA'




dna_3 = re.sub(r'(TAA|TAG|TGA)', '*',
               dna, count=1)
dna_3
# 'CAGAC*CACAGTGAAGTGTTCA'
```

特殊シーケンス

正規表現でよく使うパターンは、特殊シーケンスとして定義されている。例えば、任意の英数字の正規表現のパターンでは [a-zA-Z0-9] のように記述しなければならないが、特殊シーケンスを使用すると \w と記述するだけでよい。このような特殊シーケンスには次のようなものがある。

メタ文字	説明	別形式
\d	任意の数字	[0-9]
\D	任意の数字以外	[^0-9]
\s	任意の空白文字	[\t\n\r\f\v]
\S	任意の空白文字以外	[^\t\n\r\f\v]
\w	任意の英数字	[a-zA-Z0-9]
\W	任意の英数字以外	[^a-zA-Z0-9]
\n	改行	
\t	タブ	
(?!ptn)	ptn を含まない条件下で	
(?=ptn)	ptn を含む条件下で	

問題 R1-1

5 min

次の正規表現は、何を表現しているのかを選択肢から選べ。

```
^\d{3}-\d{4}$
^\d{4}-\d{1,2}-\d{1,2}$
^0\d{2,3}-\d{1,4}-\d{4}$
[^\s]+(?=\.(jpg|gif|png)$)'
^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$
^(http|https)://([\w-]+\.)+[\w-]+(/[\w-./?%&=]*)?$
```

- メールアドレス
- ファイル名
- 郵便番号
- URL
- 日付
- 電話番号

後方参照

置換を行うとき、マッチ部分に含まれている部分文字列を一次変数に記録し、置換する際に一次変数に保存された値を参照することができる。

```
+ . ID=g00001; Name=etr2
```

検索対象

```
'ID=([^;=]+); Name=([^;=]+)'
```

置換処理

```
'gene_id \\1; Name \\2'
```

```
+ . gene_id g00001; Name etr2
```

```
import re

gff = '. + ID=g00001; Name=etr2'


p = r'ID=([^;=]+); Name=([^;=]+)'
gtf = re.sub(p,
             'gene_id \\1; Name \\2',
             gff)


gtf
# '. + gene_id g00001; Name etr2'
```