

# Unixによるテキストファイル処理

2025/02/05

# 作業場所

- 以降の作業は、以下のディレクトリで行います。

```
~/gitc/data/7_text/
```

cd コマンドを用いてディレクトリを移動し、

pwd コマンドを利用して、カレントディレクトリが上記になっていることを確認してください。

# コマンド復習

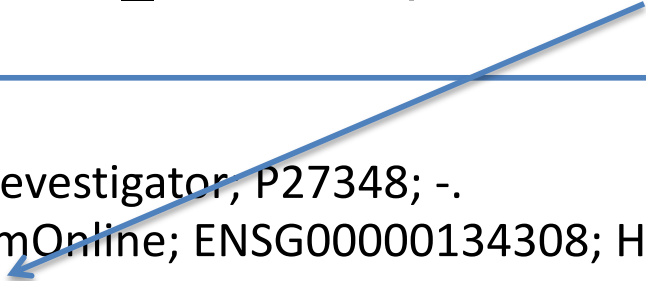
- `wc` [ファイル名]
  - ファイルの行数、単語数、文字数を出力する
- `head` [-行数] [ファイル名]
  - ファイルの先頭から指定した行数(指定しないと10行)を出力する
- `tail` [-行数] [ファイル名]
  - ファイルの最後から指定した行数(指定しないと10行)を出力する
- `less` ファイル名
  - ファイルの内容を閲覧する

# 本講で扱うテキスト処理コマンド


- `grep` 正規表現パターンの検索
- `sed` 文字列置換等によるファイルの変換
- `sort` ファイルのソート
- `awk` 様々なテキストファイル処理

# 正規表現による文字列検索 (grep)

- `grep 'パターン' [ファイル名 ...]`
    - ファイル中でパターンを含む行を出力する
- 例) `grep 'GO' 1433T_HUMAN.sprot`
- 1433B\_HUMAN.sprot から GO を含む行を検索する。



```
...  
DR Genevestigator, P27348; -.  
DR GermOnline; ENSG00000134308; Homo sapiens.  
DR GO; GO:0005813; C:centrosome; IDA:HPA.  
DR GO; GO:0005634; C:nucleus; IDA:HPA.  
DR Bgee; P27348; -.  
....
```



```
DR GO; GO:0005813; C:centrosome; IDA:HPA.  
DR GO; GO:0005634; C:nucleus; IDA:HPA.
```

# 正規表現による文字列検索 (grep)

- `grep 'パターン' [ファイル名 ...]`
  - ファイル中でパターンを含む行を出力する
  - 例) `grep 'GO' 1433T_HUMAN.sprot`
    - 1433B\_HUMAN.sprot から GO を含む行を検索する。
  - 例) `grep '^FT' 1433T_HUMAN.sprot`
    - 1433B\_HUMAN.sprot から FT で始まる行を検索する。
  - `grep -v` パターンを含まない行を出力する。
  - `grep -i` 大文字小文字を区別しない。
  - `grep -w` パターンを単語としてマッチ
  - ファイル名は複数指定可能
  - ファイル名を省略すると、標準入力から文字列を読み込んでパターンを検索する

# 正規表現

grepは「正規表現」によってパターンを指定し、照合したい文字列集合を規定する

- 通常の文字列はそのまま表現される
  - 例) `File1` (`File1`にマッチ)
- 特殊な意味を持つ文字(メタキャラクタ)によって規則を表現
  - 例) `[]`は文字集合を規定する  
`File[1-3]` (`File1`, `File2`, `File3` のいずれにもマッチ)
- `¥`によってメタキャラクタの特殊な意味を打ち消せる
  - 例) `¥[abc¥]` (`[abc]` という文字列にマッチ)

注意) 正規表現にはシェルのメタキャラクタが含まれるので、そのままコマンドラインで指定すると思わぬエラーになることが多い。そこで、パターンは `''` で囲むようにする。

# 正規表現(一部)

- **. (ドット) 任意の1文字**
  - 例) `a.c`  
`abc, adc` など、`a`と`c`の間に任意の1文字を含む文字列にマッチ
- **[ ] (角形括弧) 文字の集合**
  - 例) `[ATGC]`  
`A, T, G, C`のいずれかにマッチ
  - 例) `[a-d]`  
`a, b, c, d`のいずれかにマッチ
  - 例) `[^abd]`  
`a, b, d`以外のいずれかにマッチ
- **^ 行の先頭    \$ 行の終端**
  - 例) `^ID`  
行の先頭が`ID`である行とマッチ
- **\* 0回以上の繰り返し**
  - 例) `a.*m`  
`a`と`m`の間に任意の文字列を含む (`am, arm, alarm, am am`など)



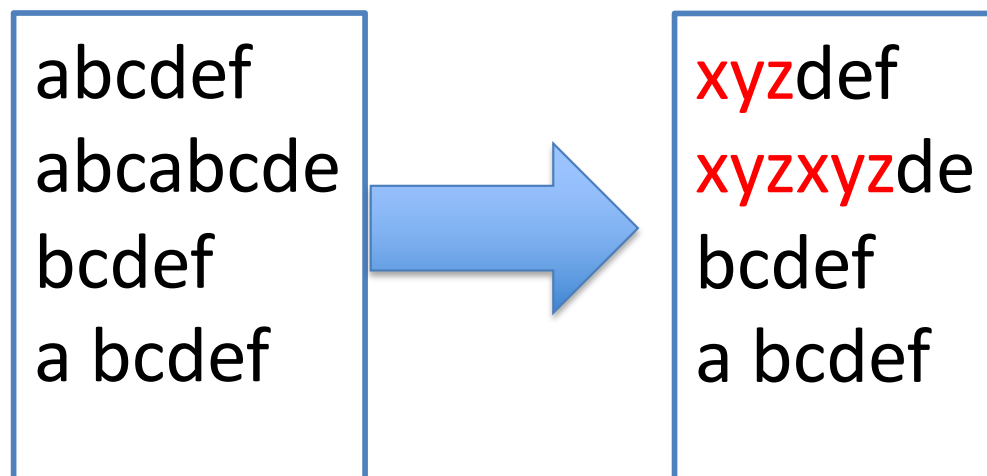
# 演習 (grep)

- `ecoli.sam` ファイルから、`grep` (`egrep`) コマンドを用いてヘッダ行(行頭に@を含む)を表示せよ。

# 文字の置換など (sed)

```
sed 's/置換対象パターン/置換文字列/g' [ファイル名]
```

- ファイル中の、指定した正規表現パターンに合致するすべての文字列を、指定した置換文字列で置き換える。置換文字列が空の場合は文字列の削除になる。
  - 例) `sed 's/abc/xyz/g' abcdef.txt`  
ファイル中の文字列`abc`をすべて`xyz`に置き換える。



# 文字の置換など (sed)

```
sed 's/置換対象パターン/置換文字列/g' [ファイル名]
```

- ファイル中の、指定した正規表現パターンに合致するすべての文字列を、指定した置換文字列で置き換える。置換文字列が空の場合は文字列の削除になる。
  - 例) `sed 's/abc/xyz/g' file`  
`file`中の文字列`abc`をすべて`xyz`に置き換える。
- 最後の`g`をつけない場合は、各行で最初にマッチしたパターンのみが置換される。
  - 例) `sed 's/:/ /' file`  
各行で最初に出現した `:` をスペースに置き換える

sedコマンドにも一般にシェルのメタキャラクタが含まれるので、パターンの指定は常に`'`で囲むようにする。

# 行の並べかえ(sort)

- `sort [オプション] [ファイル名...]`
  - ファイルを行単位で並べかえる。
  - `-k FLD1,FLD2` ソートのキーを、スペース文字で区切られたフィールド単位で指定できる(FLD1開始フィールド、FLD2終了フィールド)。
    - `-k 2,2` --第2フィールドをキーとしてソート

Murton	T	.338	14	84
Kikuchi	C	.325	11	58
Yamada	S	.324	29	89
Ooshima	D	.318	2	28
Luna	D	.317	17	73

Kikuchi	C	.325	11	58
Luna	D	.317	17	73
Ooshima	D	.318	2	28
Yamada	S	.324	29	89
Murton	T	.338	14	84

- `-k 2,2 -k 3,3nr` --第2フィールドを1番目のキーとし、第3フィールドを2番目のキーとして、数値として(n)逆順(大きい順)で(r)ソート

Murton	T	.338	14	84
Kikuchi	C	.325	11	58
Yamada	S	.324	29	89
Ooshima	D	.318	2	28
Luna	D	.317	17	73

Kikuchi	C	.325	11	58
Ooshima	D	.318	2	28
Luna	D	.317	17	73
Yamada	S	.324	29	89
Murton	T	.338	14	84

# 演習 (sort)

- eco depth.csv は、samtools コマンドから得られた深度の統計情報である。  
ファイルは3カラムからなり、順に「染色体名」「位置」「深度」となっている。  
深度が多いものから20個を表示せよ。

染色体名	位置	深度
chr	2753929	1533
chr	2753930	1470
chr	2753931	1446
chr	2753932	1101
chr	2753933	922

# テキストファイルの処理 (awk)

```
awk 'コマンド' ファイル
```

- テキストファイル进行处理する多機能なコマンド
- コマンドの一般形式は  
パターン {アクション}  
パターンに指定した条件に合致した行について、アクションで指定した操作を行う。パターンを省略するとすべての行が対象になる。
- タブ区切りテキストなどテーブル形式のファイルでは、\$1, \$2, ... によって各フィールド(カラム)の値を参照できる。

# テーブルデータの処理 (awk)

- テーブルカラムの抽出

```
awk '{print $3,$4,$5}' datafile
```

- 3, 4, 5カラム目を出力
- パターンが指定されていないのですべての行が出力される。

ecoli.gtf

```
...  
chr  eschColi_K12_refSeq  stop_codon  253  255 ...  
chr  eschColi_K12_refSeq  exon        190  255 ...  
chr  eschColi_K12_refSeq  start_codon 337  339 ...  
chr  eschColi_K12_refSeq  CDS         337  2796 ...
```



```
# awk '{print $3,$4,$5}' ecoli.gtf  
...  
stop_codon  253  255  
exon        190  255  
start_codon 337  339  
CDS         337  2796
```

# テーブルデータの処理 (awk)

- 条件を指定したフィルタリング

```
awk '$4<200 {print}' datafile
```

- 4カラム目が200未満の行を出力
- 出力フィールドが指定されていないので行全体を出力

ecoli.gtf

```
...  
chr  eschColi_K12_refSeq  stop_codon  253  255 ...  
chr  eschColi_K12_refSeq  exon        190  255 ...  
chr  eschColi_K12_refSeq  start_codon 337  339 ...  
chr  eschColi_K12_refSeq  CDS         337  2796 ...
```



```
# awk '$4<200 {print}' ecoli.gtf  
chr  eschColi_K12_refSeq  start_codon 190 192 ...  
chr  eschColi_K12_refSeq  CDS         190 252 ...  
chr  eschColi_K12_refSeq  exon        190 255 ...
```



# テーブルデータの処理 (awk)

- テーブルカラムの抽出

```
awk ' {print $1,$2,$5} ' datafile
```

- 1,2,5カラム目を出力
- パターンが指定されていないのですべての行が出力される。

- 条件を指定したフィルタリング

```
awk '$3<200 {print} ' datafile
```

- 3カラム目が200未満の行を出力
- 出力フィールドが指定されていないので行全体を出力

- 複数の条件の指定

```
awk '$2~/target/ && $3<200{print}' datafile
```

- 2カラム目にtargetを含み、3カラム目が200以下の行を出力
- 変数~/パターン/ は正規表現の照合

awk コマンドにも一般にシェルのメタキャラクタが含まれるので、常に ' ' で囲むようにすると良い

# 演習 (awk)

- awkコマンドを用いて、ex7.sam の中で、以下の条件に合う行または要素を出力せよ。

- 1) FLAG値が 16
- 2) FLAG値が 16 かつ マッピングQV が 30以上
- 3) 2)の条件かつ、フラグメント名(第1カラム)のみ。

自分で考えてもよいですし、ChatGPTを用いても構いません。

# ex7.samの中身

(ヘッダー行は省略しています)

```
SRR1515282.13 16 chr 3425618 1 51M * 0 0 TCACTGGCAGTCTCCTTTGAGTTCCCGGCCGGACCGCT
GGCAACATAGGATFFFHHHHHIJJJJIIJJJJJJIFJJJJIIJJJJJJHHHHFB42FFCCC AS:i:-3 XS:i:-3
XN:i:0 XM:i:1 XO:i:0 XG:i:0 NM:i:1 MD:Z:45A5 YT:Z:UU
SRR1515282.38 16 chr 4424175 42 51M * 0 0
GGGTGATCAGGTAAACGTTAAAGCGGGCTATGCTCGTAACTTCCTTGTACC JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
IJJJJIIJJJJIIHGHHFB2+FDDBC@ AS:i:-3 XN:i:0 XM:i:1 XO:i:0 XG:i:0 NM:i:1 MD:Z:45G5 YT:Z:UU
...
```

## ○規則

### ヘッダー部

@HD VN:1.6 SO:coordinate

@SQ SN:ref LN:45

"@"で開始

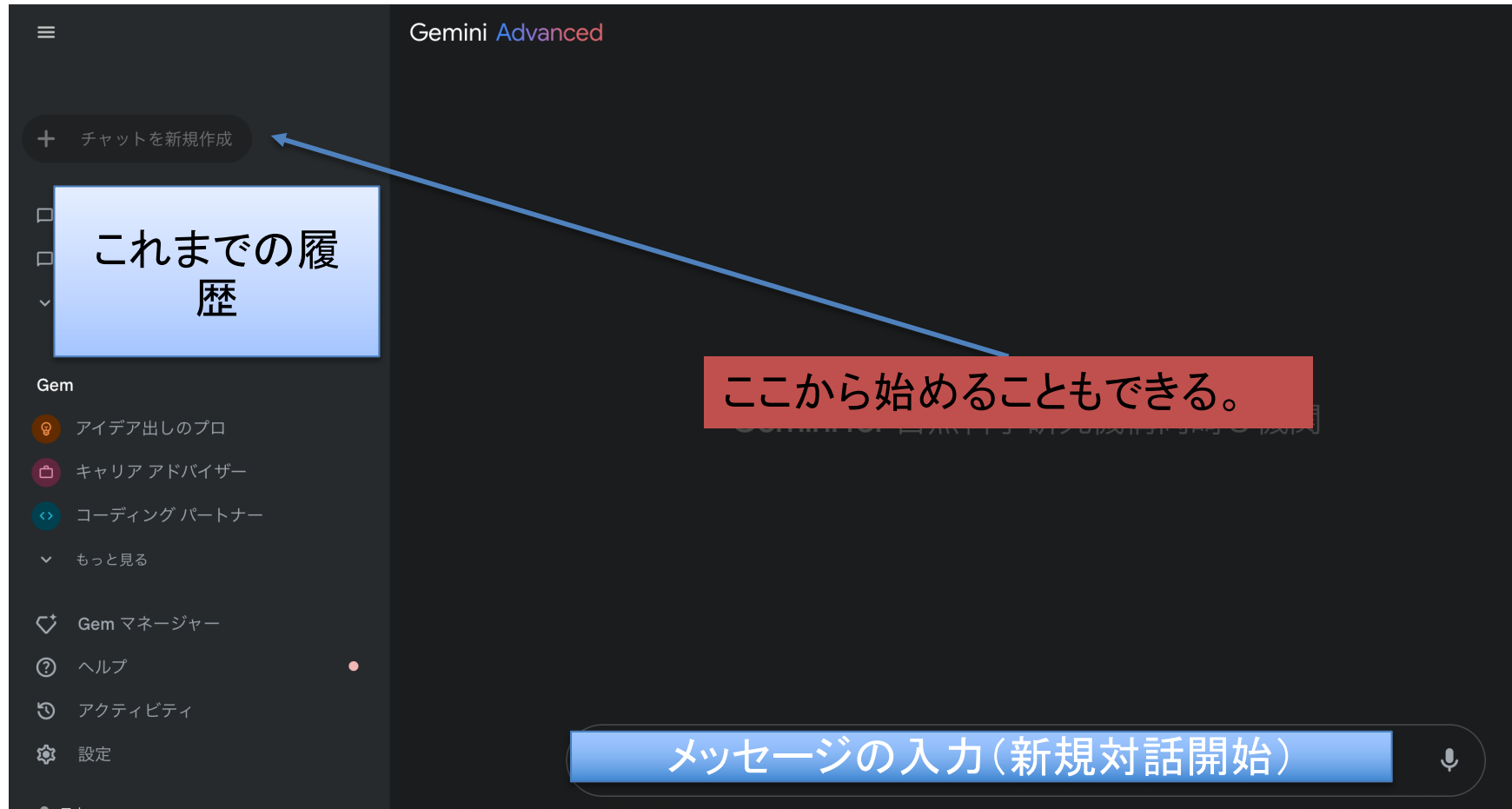
@HD VN: (バージョン) SO: (ソート状況)

@SQ SN: (リファレンス名) LN: (リファレンスの長さ)

### マッピング結果部分 項目間はタブで区切る

クエリ 配列名	FLAG	リファ レンス 配列名	アライ メント 開始位 置	マッ ピン グ QV	CIGAR	ペアフラグメン トの場所			配列	配 列 Q V	オプション
						Ref 名	開 始	長 さ			
r001	99	ref	7	30	8M2I4M1D3M	=	37	39	TTAGATAAAGGATACTG	*	

# 例) Gemini (Google)



# プロンプト

- 生成系AIに与える質問文を「プロンプト /prompt」と呼びます。
  - 生成系AIにプロンプトを与えることで回答が得られます。
  - よりよいプロンプトを与えることで、こちらが目的とする回答を得られる可能性が高くなります。
- プロンプト・エンジニアリング

# 「良い」プロンプトを作るために

- ・「明確かつ具体的である」プロンプトを作るのがまず第一歩です。
- ・ 与えられている課題を分析、分解します。
- ・ 分解されたそれぞれについてプロンプトを作ります。
- ・ 得られた回答が目的通りならその回答をもとにプロンプトを足していきます。

# 例) 演習

- 与えられているのは？ → ex7.samファイル
- 最終目的は？ → awkのコマンドを出すこと
- ファイルの構成はどのようなものか
- どの部分が必要なのか(何番目のカラム？)
- 環境は
- 回答に使用して欲しいのは

以上のことを踏まえたプロンプトを作ってみる

# プロンプトと出力例

NA


スペースを区切り文字とした以下のようなファイル `ex8.sam` があります。第2カラムの値が16であるような行だけを抜き出してください。環境はbashで、awkコマンドを使用してください。

```
aaaa 16 cccc dddd 16 eeee ffff  
aaaa 4 cccc dddd 24 eeee gggg
```



以下のコマンドを使用して、第2カラムが16である行だけを抜き出すことができます。

bash

 Copy code

```
awk '$2 == 16' ex8.sam
```

このコマンドは、``ex8.sam`` ファイル内の各行を処理し、各行の第2カラムが16である場合にその行を表示します。上記のコマンドを実行すると、第2カラムが16である行だけが表示されます。



# プロンプトと出力例

NA

このawkコマンドを改変して、第2カラムが16である、かつ、第5カラムの値が30以上である行を抜き出してください。



第2カラムが16であり、第5カラムの値が30以上である行を抜き出すには、以下のように`awk`コマンドを改変します。

```
bash
```

 Copy code

```
awk '$2 == 16 && $5 >= 30' ex8.sam
```

このコマンドは、`ex8.sam` ファイル内の各行を処理し、第2カラムが16でかつ第5カラムが30以上である場合にその行を表示します。このように改変したコマンドを実行すると、要件に合致する行だけが表示されます。

# プロンプトと出力例

NA

さらにこのawkコマンドを改変します。必要なのは行全体ではなく第1カラムのみです。これまでの条件を満たし、かつ第1カラムのみを表示するようにしてください。



第2カラムが16であり、第5カラムの値が30以上である行から、第1カラムのみを表示するには、次のように`awk`コマンドを改変します。

bash

 Copy code

```
awk '$2 == 16 && $5 >= 30 { print $1 }' ex8.sam
```

このコマンドは、`ex8.sam` ファイル内の各行を処理し、第2カラムが16でかつ第5カラムが30以上である場合に、その行の第1カラムを表示します。したがって、要件に合致する行の第1カラムのみが表示されます。

# プロンプトの注意

- 機密性、秘匿性の高い情報を入力しない

生成系AIは、入力された情報を教育データとして学習を行っています。そのため、入力した情報は外部に漏洩する可能性があります。

利用の際には手元のデータをそのまま使用するのではなく、あたりさわりのない文書や書き換えを適宜行いましょう。

# プロンプトの注意

- 回答結果を検証する

生成系AIの回答は、表現や言い回しが自然であるため、正しいと感じてしまいます。しかし、情報が最新ではないことや、偏った情報が反映されてしまうこともあるなど、必ずしもその内容が「正確」とは限りません。得られた回答が正しいかどうかを検証するようにしましょう。

生成系AIをプログラムコーディングに使用する場合、一番簡単な検証は「実行してみる」ことです。

もしエラーが出たなら、そのエラーをもとにさらに生成系AIにプロンプトを与えると問題解決に近づくことが期待されます。

# プロンプトの注意

- 生成系AIで生成したものであると明示する

生成系AIで資料を作成する場合、  
生成系AIで生成したものであるという明示をしておくといでしょう。

これにより「自分個人の考え」と、「生成系AIでの回答」を別にすることができ、  
論拠をはっきりさせることができます。

# 参考：算術計算(awk)

- 合計値の出力

```
awk ' {sum=sum+$2} END{print sum} '  
ecoli.htseq
```

- 2カラム目の合計値を出力

- プログラムは2つのブロックからなる

`{sum=sum+$2}`      パターン部がないのですべての行が対象となる。  
変数`sum`に各行の2カラム目の値を加える。

`sum+=x` は `sum=sum+x`と同じ。

`END{print sum}`      パターン`END`は最終行のみにマッチ。  
最終行で `sum` の値を出力する。

参考) パターン `BEGIN`は先頭行のみにマッチする。これを用いて変数の初期化などができる。

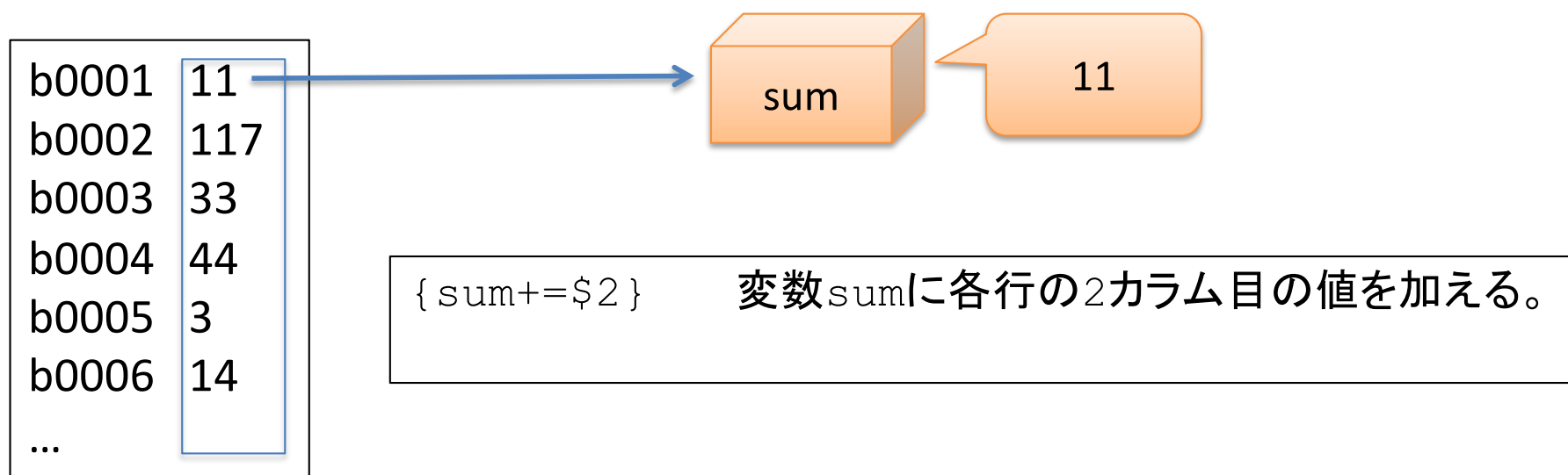
例) `BEGIN{sum=0} {sum+= $2} END{print sum}`

最初に変数 `sum` を0に初期化する。これはデフォルトの動作として省略できるため、上記のプログラムと同じ結果になる。

# 参考：算術計算(awk)

- 合計値の出力

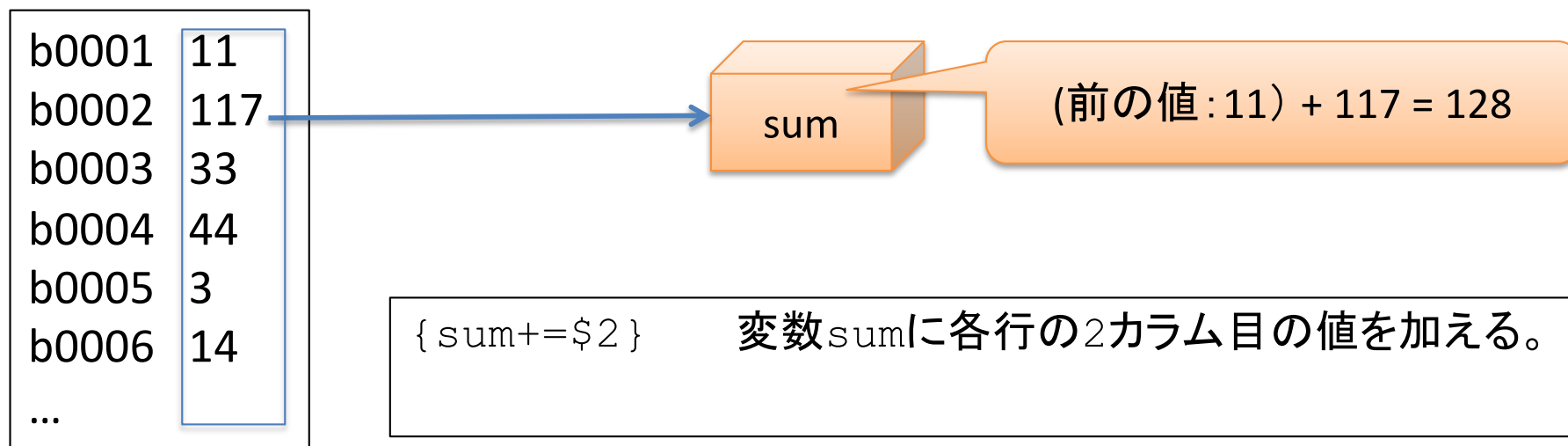
```
awk ' {sum=sum+$2} END{print sum} ' ecoli.htseq
```



# 参考：算術計算(awk)

- 合計値の出力

```
awk ' { sum=sum+$2 } END{print sum} ' ecoli.htseq
```

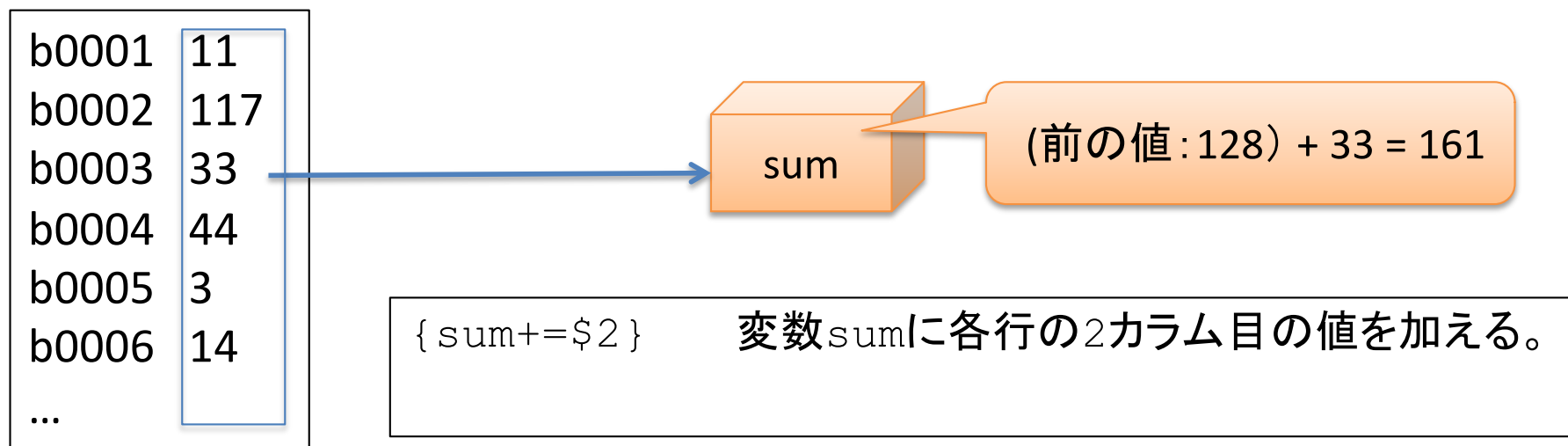




# 参考：算術計算(awk)

- 合計値の出力

```
awk ' { sum=sum+$2 } END{print sum} ' ecoli.htseq
```



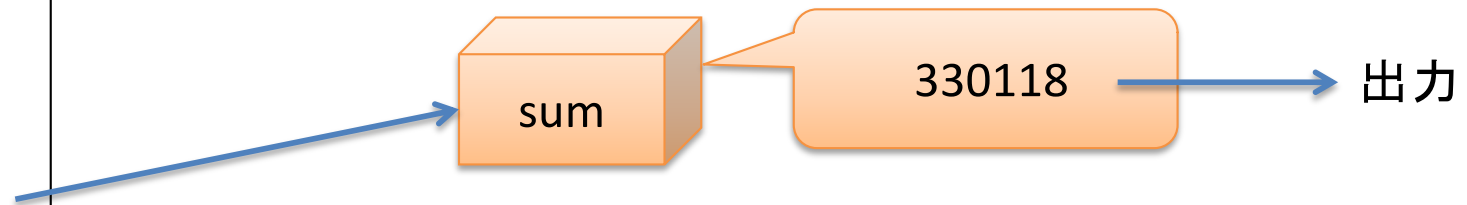
# 参考：算術計算(awk)

- 合計値の出力

```
awk ' {sum=sum+$2} END{print sum} ' ecoli.htseq
```

```
b0001 11  
b0002 117  
b0003 33  
b0004 44  
b0005 3  
b0006 14  
...  
[EOF]
```

```
END{print sum}  
最終行で sum の値を出力する。
```



# 参考：算術計算(awk)

- 合計値の出力

```
awk ' {sum=sum+$2} END{print sum} '  
ecoli.htseq
```

- 2カラム目の合計値を出力

- プログラムは2つのブロックからなる

`{sum=sum+$2}`      パターン部がないのですべての行が対象となる。  
変数`sum`に各行の2カラム目の値を加える。

`sum+=x` は `sum=sum+x`と同じ。

`END{print sum}`      パターン`END`は最終行のみにマッチ。  
最終行で `sum` の値を出力する。

参考) パターン `BEGIN`は先頭行のみにマッチする。これを用いて変数の初期化などができる。

例) `BEGIN{sum=0} {sum+= $2} END{print sum}`

最初に変数 `sum` を0に初期化する。これはデフォルトの動作として省略できるため、上記のプログラムと同じ結果になる。

# 演習 (awk)

- awkコマンドを用いて、ecoli.htseq の2カラム目(カウント数)の平均値を出力せよ。
  - ヒント: 行数を数える必要がある。変数lnを使って行数を数えるにはどうすればよいか？
  - カラムの和を変数sumを用いて表せば、最後にsumを行数で割り算することで平均が出せる。割り算は  $a/b$  で計算できる。

- 1) awkを使用して自分でコマンドを考えてみてください。
- 2) プロンプトを考え、ChatGPTに書かせてみてください。