

# ゲノムデータベースとプログラミング


法政大学 生命科学部 大島研郎

## 本日の講義資料

kiso2 ← 本日の講義で使用するWebページへのリンクが載せてあります。  
parse.py  
BLAST.txt

本日の講義では, Pythonを使います.

◆ コマンドプロンプトを立ち上げてください

 スタート → Windowsシステムツール → コマンドプロンプト

```
> python -help
```

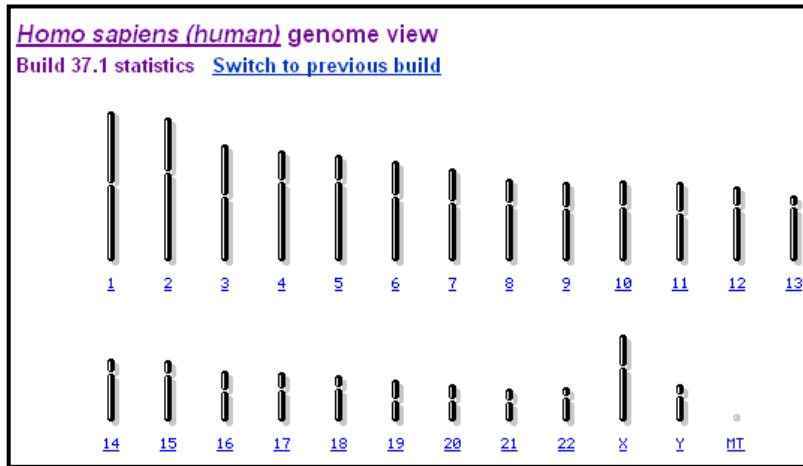
と入力して, エラーが出ないことを確認してください

# ゲノムとは

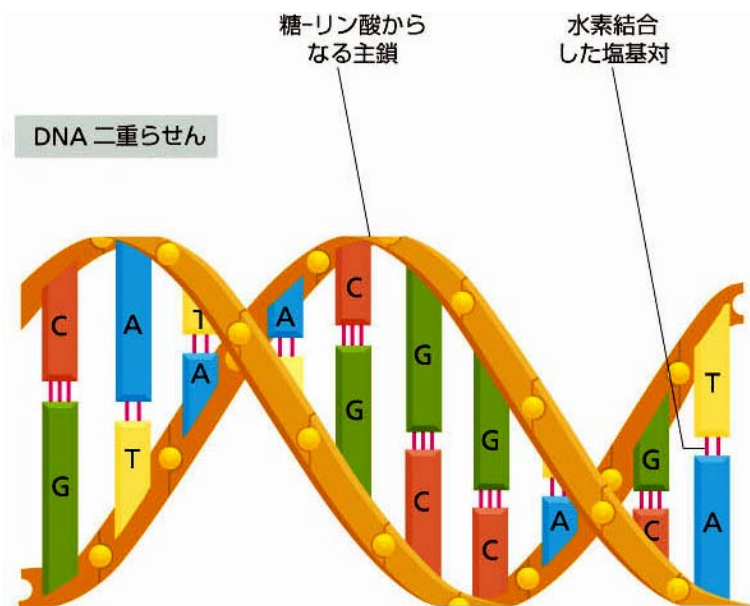
gene (遺伝子) + -ome (総体)

ゲノム = ある生物のもつ全ての遺伝情報

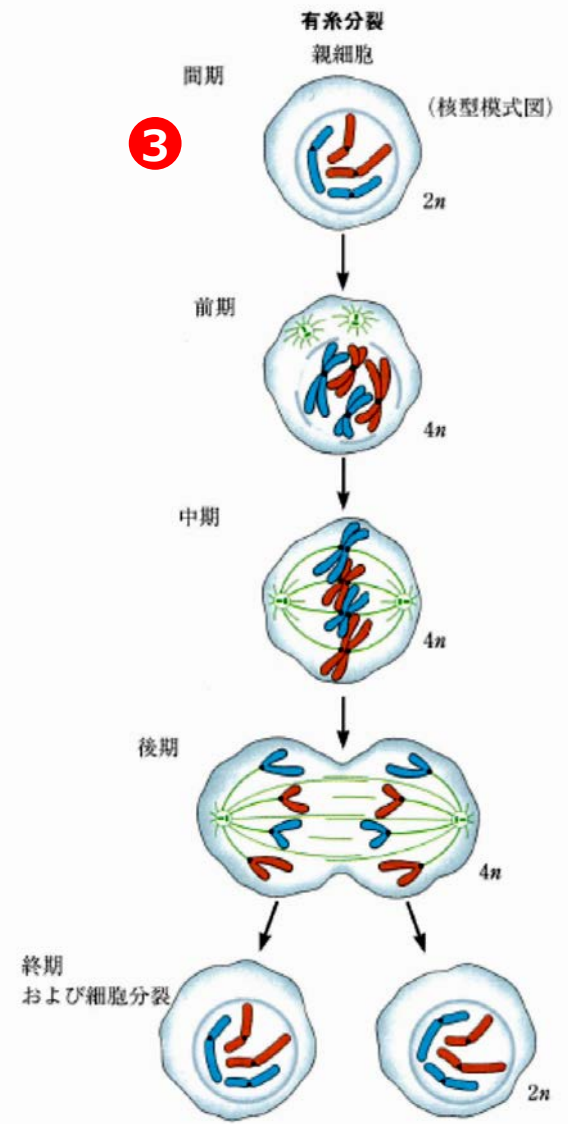
1



2



3



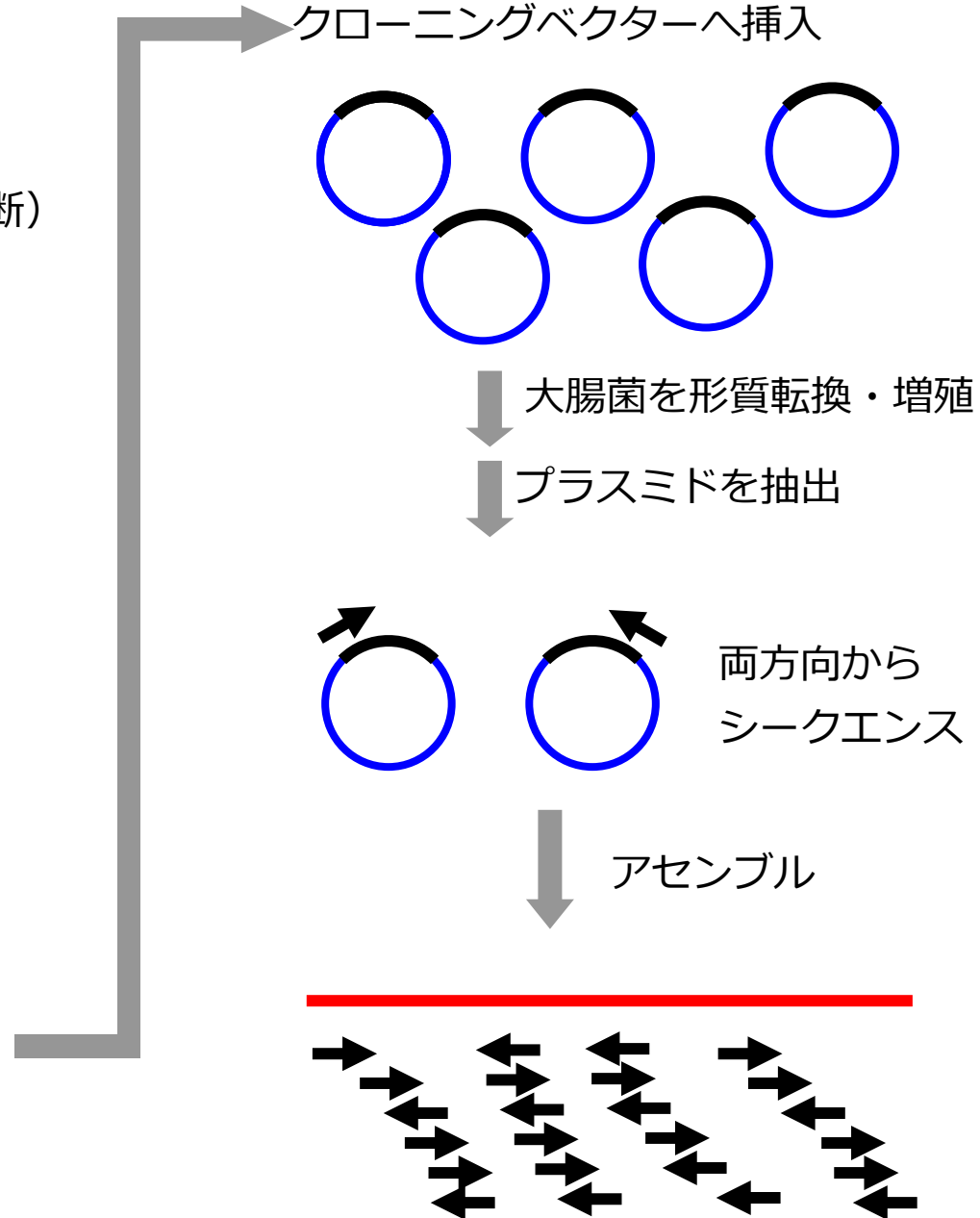
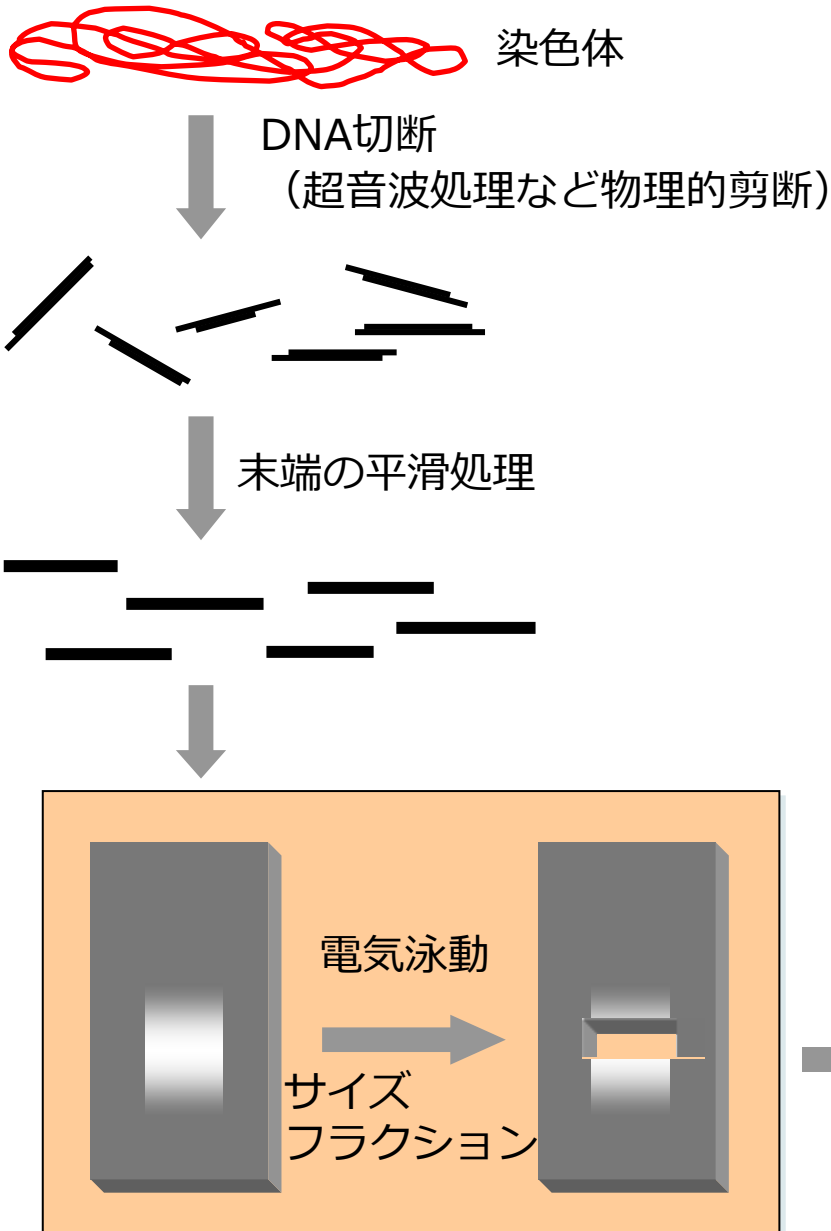
# ゲノム解読の歴史

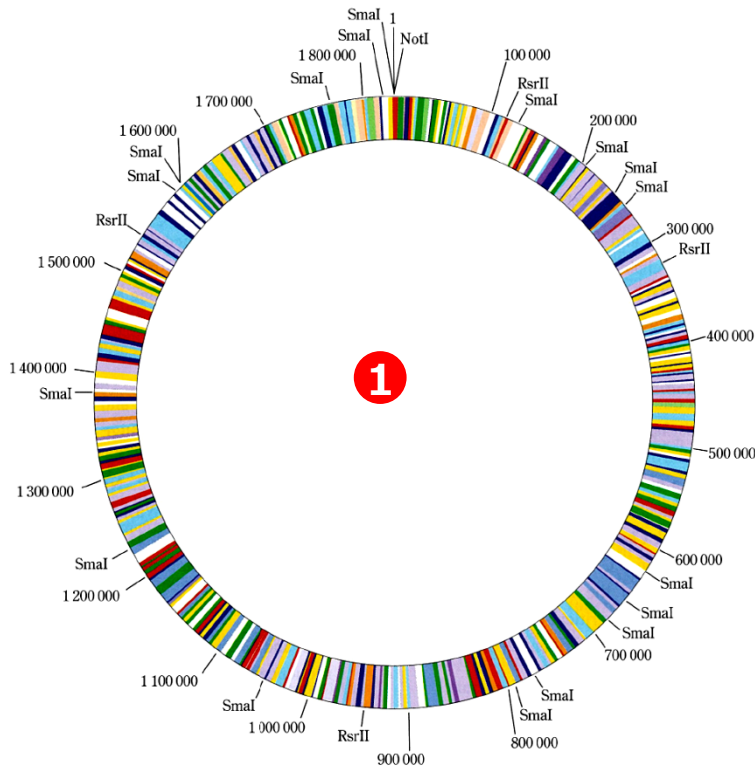
表 1-1 配列が完全に決定されたゲノムの例

生物種	特 徴	生息場所	ゲノムサイズ (一倍体ゲノム あたりの塩基対 数, ×1000)	タンパク質指令遺 伝子の数(推定)
<b>細菌</b>				
マイコプラズマの一種 <i>Mycoplasma genitalium</i>	既知の細胞ゲノムのうちで最小のゲノムをもつ	ヒトの生殖道	580	468
<i>Synechocystis</i> sp.	光合成を行い, 酸素を作り出す (シアノバクテリアの一種)	湖や小川	3573	3168
大腸菌 <i>Escherichia coli</i>	実験室でよく使われる	ヒトの腸	4639	4289
ヘリコバクター・ピロリ <i>Helicobacter pylori</i>	胃潰瘍を起こし, 胃がんの原因となる	ヒトの胃	1667	1590
<b>真核生物</b>				
出芽酵母 <i>Saccharomyces cerevisiae</i>	最小のモデル真核生物	ブドウ果皮, ビール	12,069	約 6300
シロイヌナズナ <i>Arabidopsis thaliana</i>	顕花植物のモデル生物	土壌と大気	約 142,000	約 26,000
線虫 <i>Caenorhabditis elegans</i>	発生を完全に記載できる単純な動物	土 壌	約 97,000	約 20,000
キイロショウジョウバエ <i>Drosophila melanogaster</i>	動物発生の遺伝学に貢献	腐りかけの果物	約 137,000	約 14,000
ヒト <i>Homo sapiens</i>	最も精力的に研究されている哺乳類	家	約 3,200,000	約 24,000

ゲノムサイズや遺伝子数は、特に細菌と古細菌の場合、同じ種でも系統によって異なる。表のデータは配列決定された特定の系統のもの。遺伝子には何通りものタンパク質を生じるものが多いので、ゲノムによって規定されるタンパク質の総数は遺伝子数よりかなり多い。

# ショットガン シーケンス法





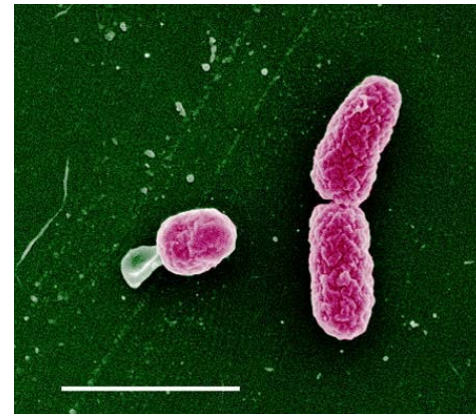
DNA は二本鎖なので、塩基対(base pair; bp)の数で分子の長さを表す。キロ塩基対(kilobase pair; kb)は  $10^3$  bp, メガ塩基対(megabase pair; Mb)は  $10^6$  bp, ギガ塩基対(gigabase pair; Gb)は  $10^9$  bp。まとめると,

$$1 \text{ kb} = 1000 \text{ bp}$$

②  $1 \text{ Mb} = 1000 \text{ kb} = 1,000,000 \text{ bp}$

$$1 \text{ Gb} = 1000 \text{ Mb} = 1,000,000 \text{ kb} = 1,000,000,000 \text{ bp}$$

RNA 分子はたいてい一本鎖なので長さの単位に bp は使えず、ヌクレオチドの数で表す。




- 1995年, 生物として初めて *Haemophilus influenzae* の全ゲノムが解読された
- その後, 多くの生物の全ゲノムが解読され, 現在では1万種以上の生物のゲノム情報がデータベースに登録されている

▶ ヒトゲノムマップ を開く

<http://www.lif.kyoto-u.ac.jp/genomemap/html/pdf.html>

Number: **09**  
1億4000万bp  
1076個

遺伝子名	通称名
<b>ABO</b>	ABO血液型遺伝子
◎赤血球に目印をつける酵素。 ◎目印にはA型、B型の2種類があり、この組み合わせで血液型が決まる。 ◎目印がつかない場合はO型になる。	
同等の遺伝子(オノログ)を持つ生物	
	
※2006年2月時点で公開されているデータベースに基づいて作成したものです。	

Number: **12**  
1億4200万bp  
1268個

遺伝子名	通称名
<b>ALDH2</b>	アルデヒド分解酵素2
	
◎アルコールから生成される有毒なアセトアルデヒドを無毒な酢酸に変える酵素。 ◎お酒に強い人は、この酵素のはたらきが強い。	
同等の遺伝子(オノログ)を持つ生物	
	


Number: **07**  
1億6300万bp  
1378個

遺伝子名	通称名
<b>FOXP2</b>	発音と言語に関わる遺伝子
◎発音や言語に関する脳の神経回路網の重要な役割を果すタンパク質。 ◎同：遺伝子動物とチンパンジーを比較すると、この遺伝子の方が異なることがわかっており、この変化は「進化的な言語能力を有した」可能性がある。	
同等の遺伝子(オノログ)を持つ生物	
	
※2006年2月時点で公開されているデータベースに基づいて作成したものです。	

Number: **X**  
1億6300万bp  
1141個

遺伝子名	通称名
<b>OPN1LW</b>	赤色識別遺伝子
<b>OPN1MW</b>	緑色識別遺伝子
◎OPN1LWは赤色を識別する際に、OPN1MWは緑色を識別する際に機能するタンパク質。 ◎そのいずれかのタンパク質が変異すると、赤と緑が判別しにくい色覚を持つことになる。	

Number: **Y**  
5100万bp  
255個

遺伝子名	通称名
<b>SRY</b>	性決定遺伝子
	
◎男性化に関わるタンパク質。 ◎ヒトの体は元々女性型になっているが、このタンパク質が作用すると精巣ができる。	

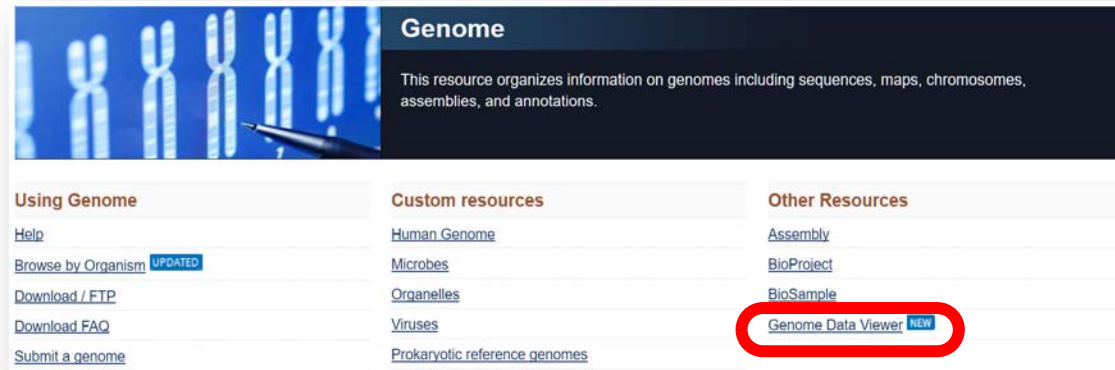
Number: **Y**  
5100万bp  
255個

遺伝子名	通称名
<b>SRY</b>	性決定遺伝子
	
◎非遺伝子領域が延々と続く不毛な地帯。 ◎このような領域はゲノム上の様々な場所に存在している。	

**TTCCAの反復配列**

# ゲノムブラウザを使ってヒトゲノムを見てみよう

NCBIトップページ右のリンクから「Genome」→「Genome Data Viewer」



Select organism

Homo sapiens (human)

fruit fly  
Aedes albopictus  
nematode  
chimp/panzee  
zebrafish  
chicken  
rat  
mouse  
Plasmodium falciparum 3D7  
maize  
rice  
pig  
sheep  
Arabidopsis  
grape  
soybean  
horse  
dog  
cattle

**human**

**Homo sapiens (human) genome**

Search in genome

Location, gene or phenotype

Examples: TP53, chr17:7667000-7689000, rs334, DNA repair

Assembly

GRCh38.p12

**Browse genome** BLAST genome

**Assembly details**

Name	GRCh38.p12
RefSeq accession	GCF_000001405.38
GenBank accession	GCA_000001405.27
Download via FTP	RefSeq, GenBank
Submitter	Genome Reference Consortium
Level	Chromosome
Category	Reference genome

**Annotation details**

Annotation Release	109
Release date	2018-03-26

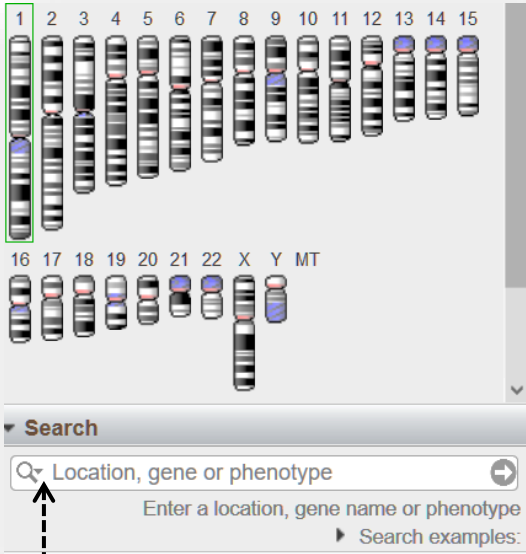
ゲノム解読された真核生物の  
系統図が表示される

「human」



「Browse genome」  
をクリック

クリックした染色体のゲノムマップが表示される



遺伝子マップ

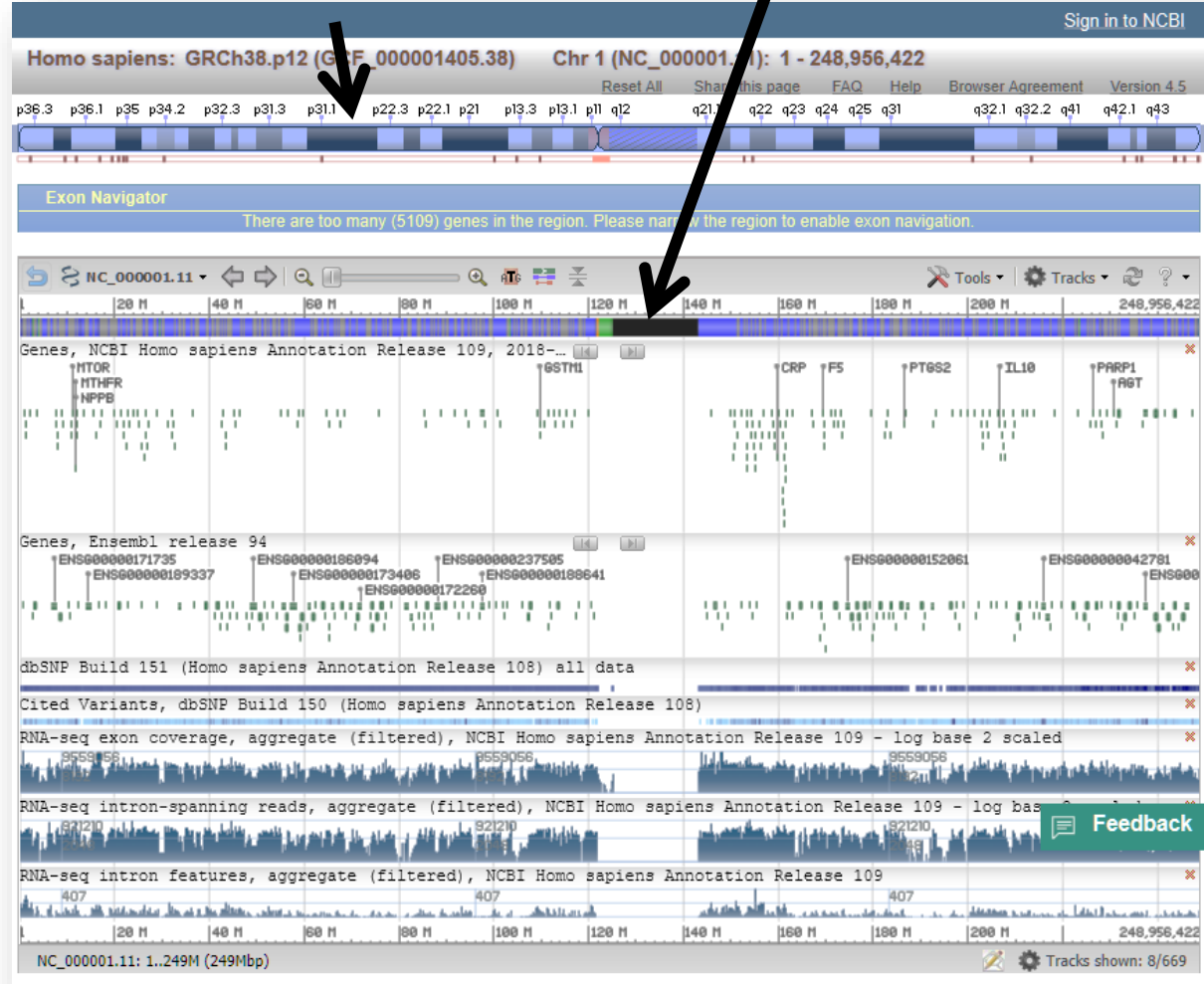
太線：エキソン

細線：イントロン

遺伝子発現量

① 青い領域にコードされる遺伝子が下に表示される

② 黒い領域は塩基配列が決まっていない



③

「ABO」と入力してみる

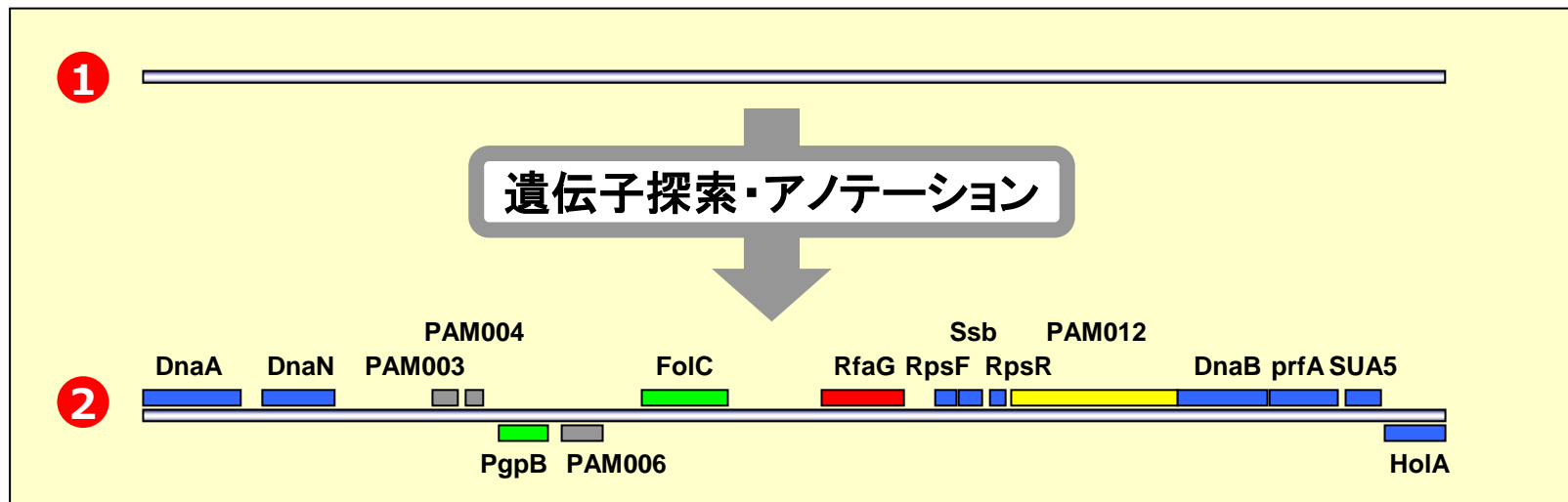
セントロメアには遺伝子がコードされていない



# 遺伝子探索

```

agttatattttttttcattttataattaaaagccaatttaaaaaaggagtattaattatgccat
atattgaaagtatttttagcgcgcgaagtgctagattccagaggaaatcctacagtagaagtaga
agtttatacagaatcaggagcgtttggaagagctattgttccttcaggagcttctaccggacaa
tacgaagcagttgaattaagggatggggatgcccaaagatttttaggtaaaggcgttttgcaag
ctgttaaaaatgttattgaagttattcaaccagaattagaaggttattctgtccttagaacaac
tttaattgataaattattaattaaacttgacggaactcctaacaatctaatttaggagctaac
gctattttaggtgtttcctttggcttgtgctaaagctgcagctaactacttaaactcttgagtttt
atcaatatgtaggaggcgttttacctaacaatgccagttcctatgatgaatgttatcaacgg
tggagc.....
  
```

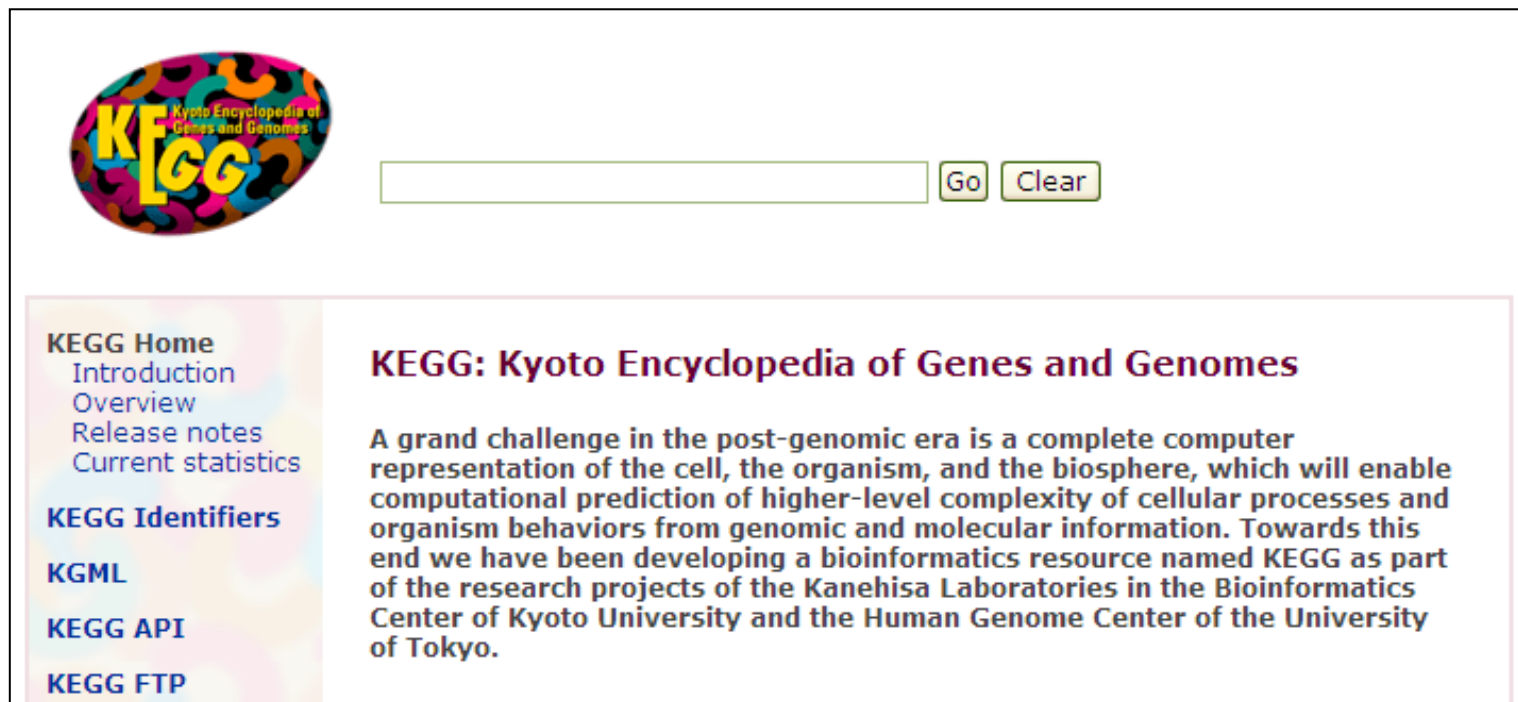


ゲノム配列から生命活動に関わる機能や分子進化に関する考察などを行うためには、タンパク質をコードしている遺伝子領域を同定することが重要となる。

# 代謝パスウェイデータベース

KEGG

[https://www.genome.jp/kegg/kegg\\_ja.html](https://www.genome.jp/kegg/kegg_ja.html)



The screenshot shows the KEGG website homepage. At the top left is the KEGG logo, which is a colorful oval with the text 'KEGG Kyoto Encyclopedia of Genes and Genomes'. To the right of the logo is a search bar with a 'Go' button and a 'Clear' button. Below the search bar is a navigation menu on the left side with the following items: 'KEGG Home', 'Introduction', 'Overview', 'Release notes', 'Current statistics', 'KEGG Identifiers', 'KGML', 'KEGG API', and 'KEGG FTP'. The main content area on the right features the title 'KEGG: Kyoto Encyclopedia of Genes and Genomes' in a large, bold, purple font. Below the title is a paragraph of text: 'A grand challenge in the post-genomic era is a complete computer representation of the cell, the organism, and the biosphere, which will enable computational prediction of higher-level complexity of cellular processes and organism behaviors from genomic and molecular information. Towards this end we have been developing a bioinformatics resource named KEGG as part of the research projects of the Kanehisa Laboratories in the Bioinformatics Center of Kyoto University and the Human Genome Center of the University of Tokyo.'

生命システム情報統合データベース。完全にゲノムが決まった生物種（一部、ドラフト配列も含む）の代謝系や一部の制御系（シグナル伝達や細胞周期など）をまとめ、そこから様々な物質データベースや 酵素データベースを参照することができる。

# 代謝系データベースの参照

データタイプごとのエントリーポイント

- 1** **KEGG PATHWAY** KEGG パスウェイマップ
- KEGG BRITE** BRITE 機能階層・テーブル
- KEGG MODULE** KEGG モジュール
- KEGG ORTHOLOGY** KO 機能オーソログ [Annotation]
- KEGG GENOME** ゲノム [Pathogen | Virus | Plant]



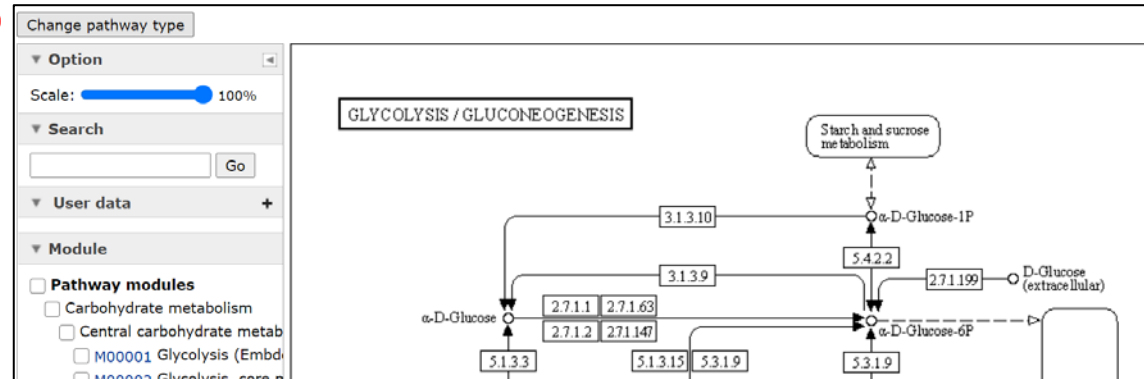
## 1.1 Carbohydrate metabolism

- 00010 M N Glycolysis / Gluconeogenesis **2**
- 00020 M Citrate cycle (TCA cycle)
- 00030 M Pentose phosphate pathway
- 00040 M Pentose and glucuronate interconversions
- 00051 M Fructose and mannose metabolism
- 00052 M N Galactose metabolism
- 00053 M Ascorbate and aldarate metabolism



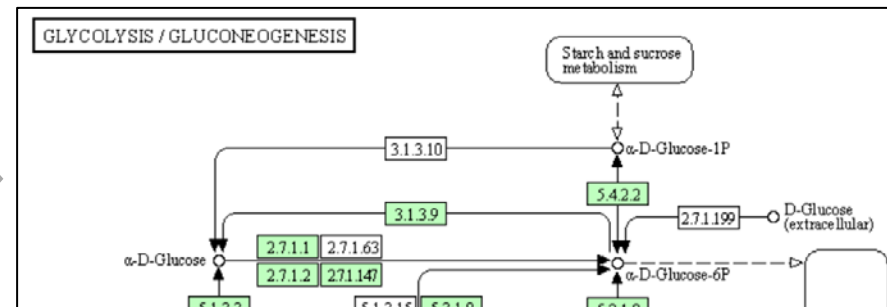
**3**

Change pathway typeを押す



hsaを押す

- ▼ Organism specific
  - ▼ Animals
    - ▼ Mammals
      - 4** hsa Homo sapiens (human)
      - ptr Pan troglodytes (chimpanzee)
      - pps Pan paniscus (bonobo)
      - ggo Gorilla gorilla gorilla (western lowland g)
      - pon Pongo abelii (Sumatran orangutan)
      - nle Nomascus leucogenys (northern white-cheeke



ヒトの持つ酵素が緑色で表示される

# 次世代シーケンサー

Roche Diagnostics社

1

Genome Sequencer FLX System (454)

2005年発売



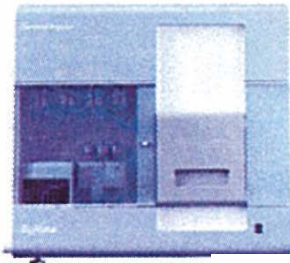
ライフテクノロジー社  
Ion PGM



Applied Biosystems社

SOLiD 3

2007年発売



Solexa / illumina社

Genome Analyzer IIx

2005発売



イルミナ株式会社

MiSeq 2



PacBio RS II

# 次世代シーケンサーの比較

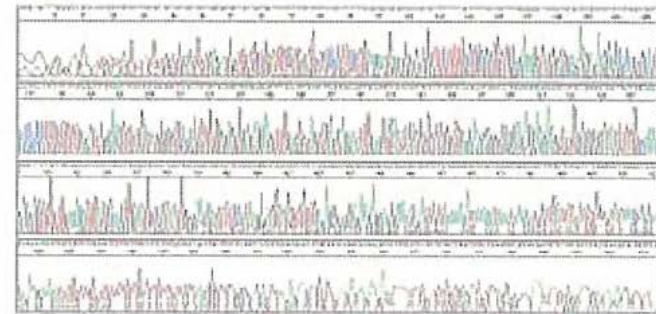
	Ion PGMシステム		MiSeq	HiSeq 2000/2500 (SBS v3試薬使用)	PacBio RS II
	Ion Protonシステム	Ion 318 chip			
<b>1リード長</b>	~200 base		150/250/300 base	100 base	約10,000 base
<b>リード数</b>	約5,000万リード (1ランあたり)	約400万リード (1ランあたり)	約3,000万リード (1ランあたり) ※ペアエンド解析	<b>1</b> 約3億リード (1レーンあたり) ※ペアエンド解析	約5万リード (1セルあたり)
<b>データ量 (リード長 200 base の場合)</b>	約7.5 Gb  (平均150 bpの amplicon、  1チップあたり)	約800 Mb  (1チップあたり)	約3~9 Gb  (1ランあたり)  ※ペアエンド解析	<b>2</b> 約30 Gb  (1レーンあたり)  ※ペアエンド解析	約500 Mb  (1セルあたり)
<b>解析手法</b>	Ion semiconductor sequencing法		Sequencing by Synthesis法	Sequencing by Synthesis法	SMRT(Single Molecule Real-Time) sequencing法
<b>アプリケーション例</b>	・癌遺伝子などの変異 解析  (409遺伝子をター ゲットとしたCancer Panelなど)	・癌遺伝子などの変異 解析  (50遺伝子をターゲッ トとしたCancer Panel など)	・微生物の新規ドラフ ト配列決定  ・癌遺伝子などの変異 解析 ・PCR産物のディープ シーケンス	・ゲノム変異解析  ・ChIP解析 ・small RNA解析 ・mRNA解析 ・cDNA配列解析	・ゲノムドラフト解析  ・cDNA配列解析

## イリミナ株式会社

## 次世代シーケンサー: Genome Analyzer

## ① 従来型キャピラリーシーケンサー

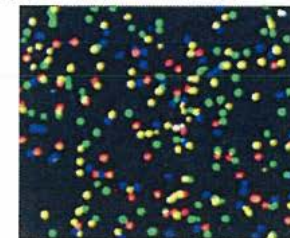
- 酵素反応+電気泳導+塩基読取 (384x600塩基)
- コスト、時間がかかる
- 例)「ヒトゲノムプロジェクト」  
約13年、3000億円かかった



MiSeq

## ② 次世代シーケンサー

- 酵素反応+電気泳導+塩基読取 (100,000,000x50塩基)
- これまでの技術と比べて、「100分の1のコストで100倍のデータ」
- 例)現在ヒトゲノム1人読むのに 数週間、数千万円  
→ 1週間 数百万円 ...



illumina®

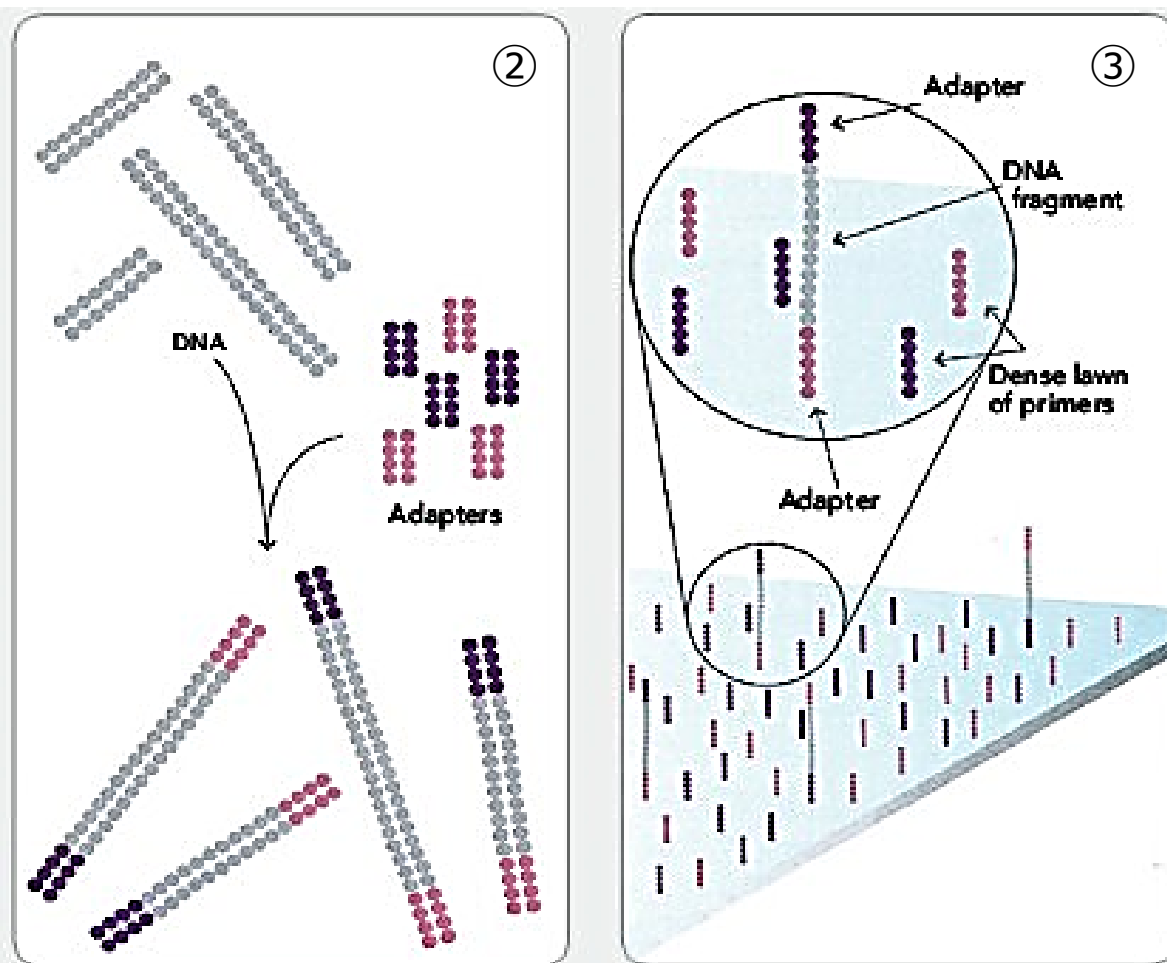
ゲノムDNA



① 断片化



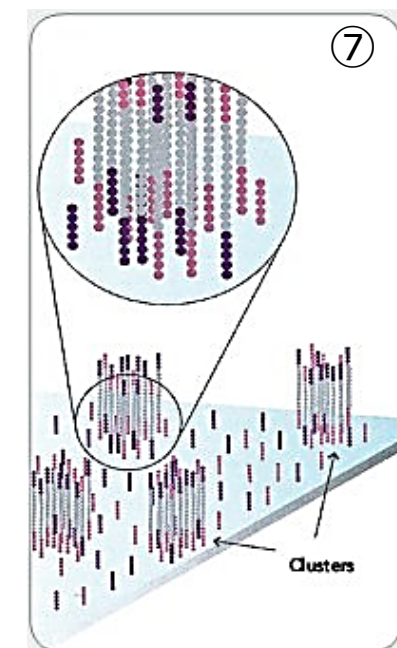
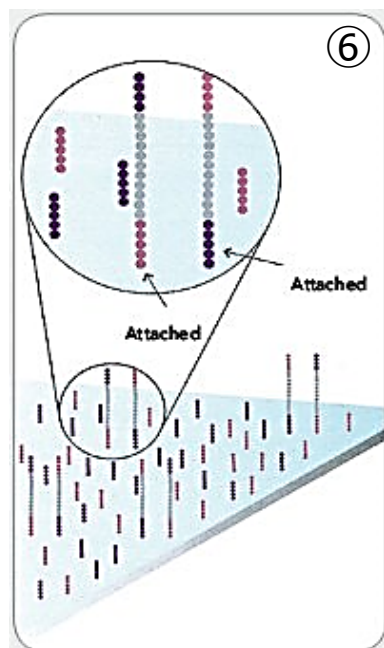
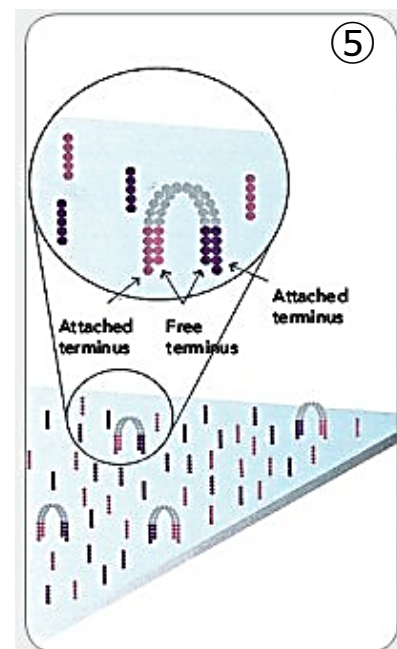
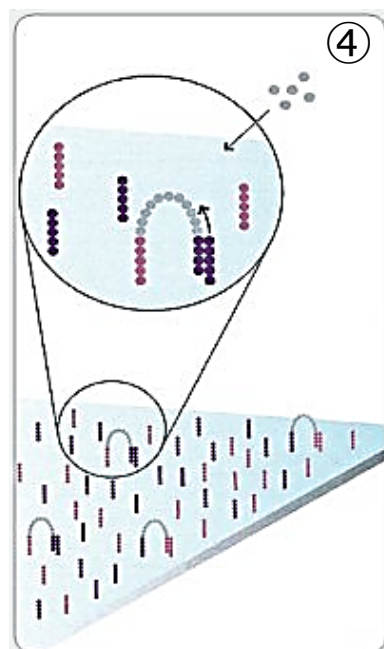
- ゲノムDNAを抽出し、断片化する
- DNA断片の両端に2種類の**アダプター**を連結させる



- 1本鎖にして、5'末端を**フローセル**上に固定する
- フローセル上には、あらかじめアダプターと相補的に結合するプライマーが高密度に配置されている

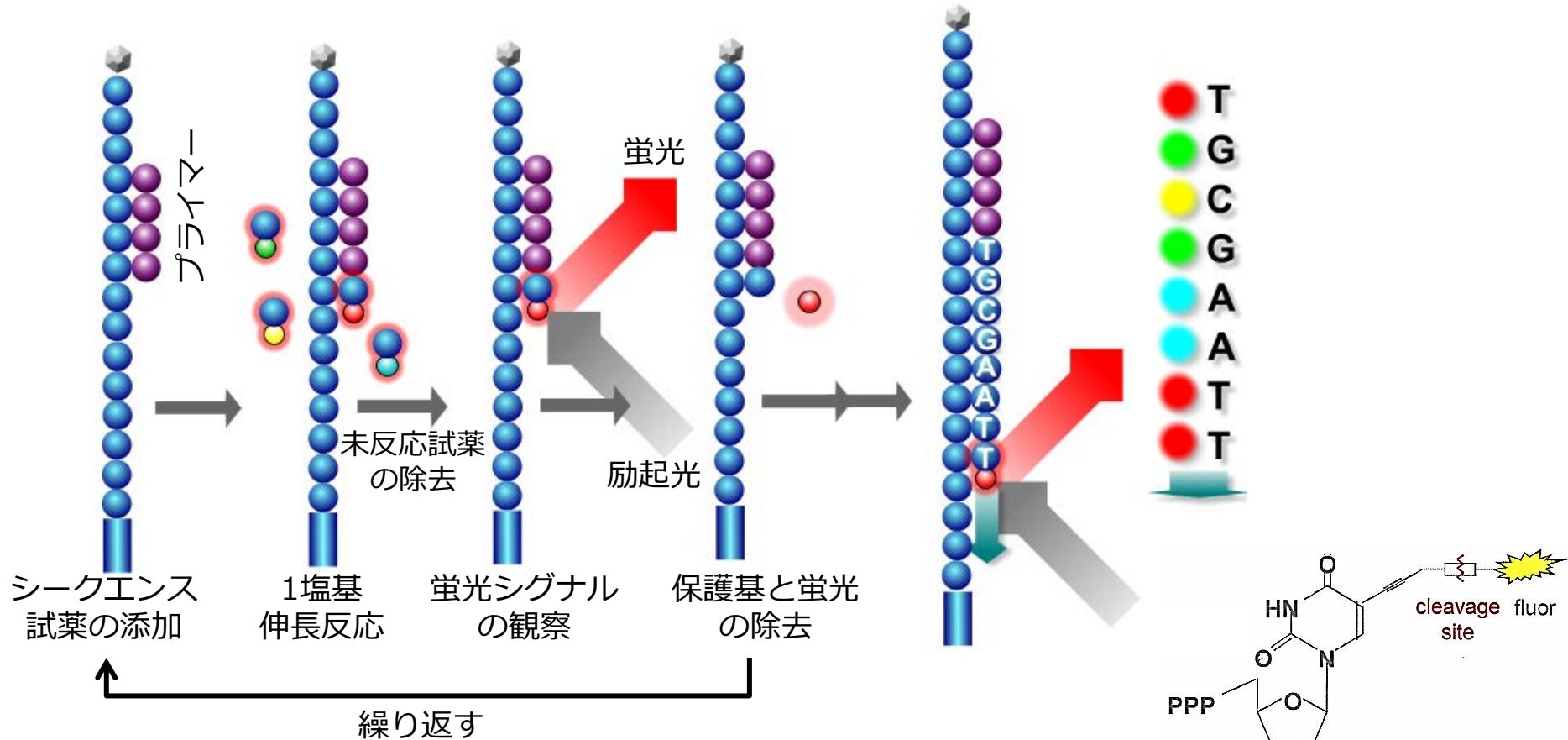
## ブリッジPCR

- 固定された1本鎖DNAは、もう一方のアダプターの側でプライマーと結合する (橋がかかったような構造になる ④)
- DNAポリメラーゼによる伸長反応を行う ⑤
- 変性させると、フローセル上には根元がアダプター配列の1本鎖DNAが2本できあがる ⑥
- この反応を繰り返すことで、狭い面積の中でDNAを増幅することができる  
→ フローセル上に多数のDNAの「束」ができる ⑦
- これらを鋳型として、配列解析を行う





## Sequencing-by-synthesisによる塩基配列決定

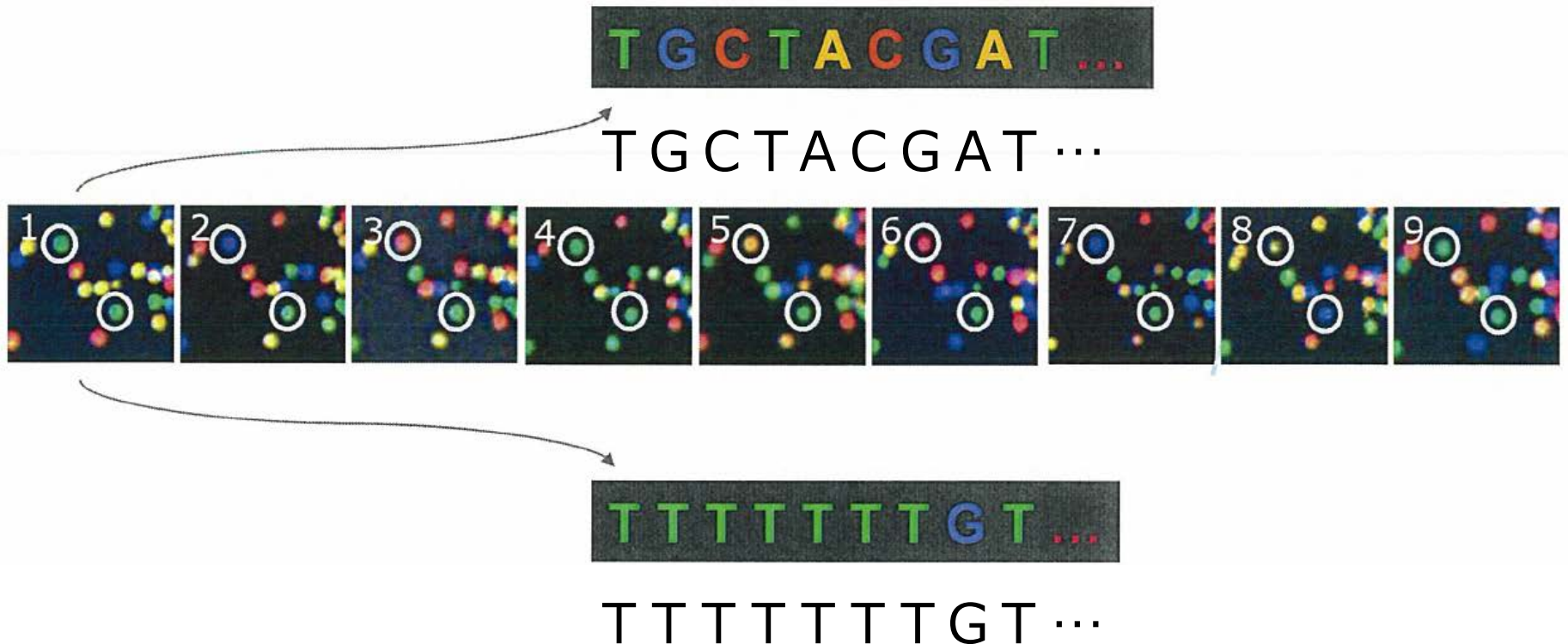
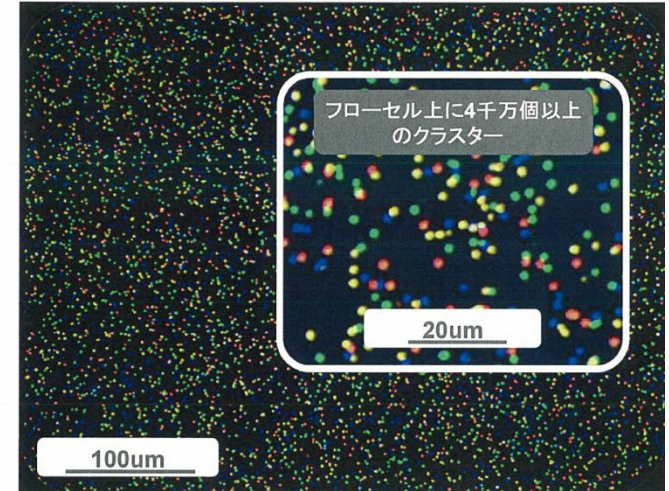


- 蛍光標識したdNTPの取り込みを**蛍光顕微鏡によって解析する**
- このdNTPは3'末端がブロックされており、1回の伸長反応で1塩基しか伸ばせない
- そのため、1塩基ごとにどのdNTPが取り込まれたかを観察し、蛍光物質とブロックを外して次の伸長反応を行うというステップで、解析を進めていく

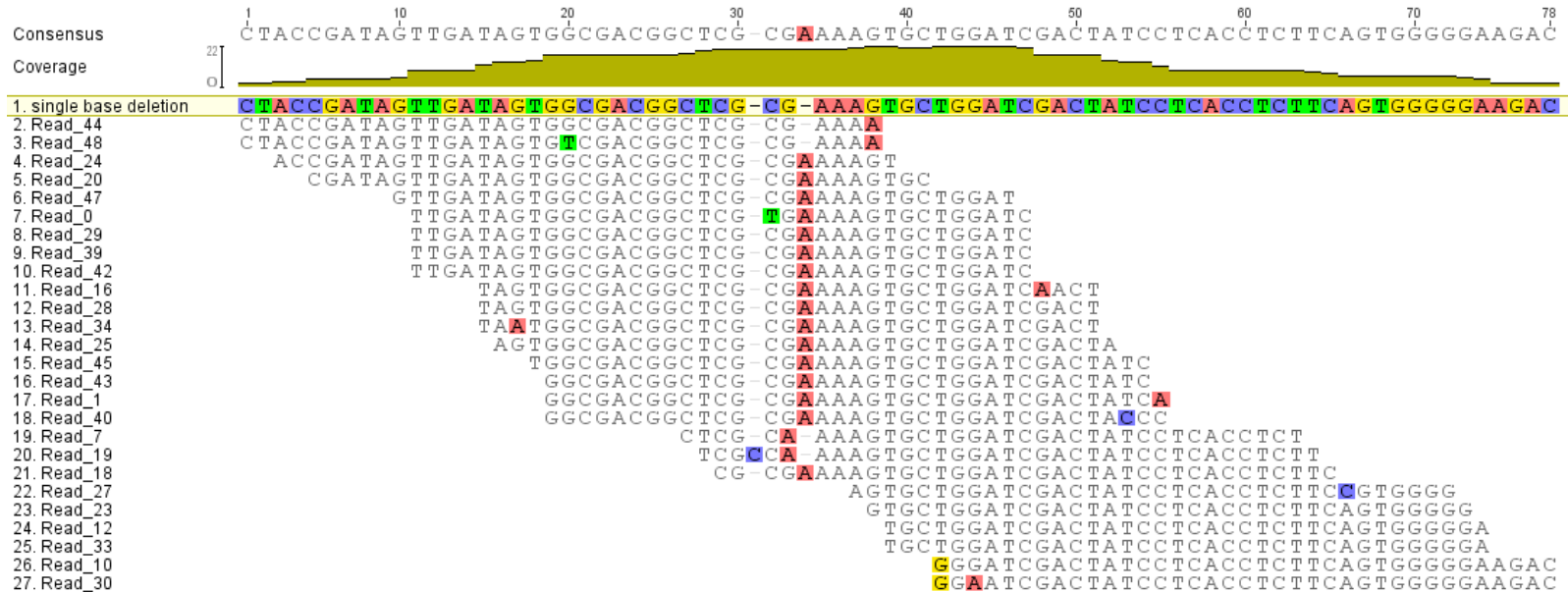
## 画像蛍光シグナルから塩基への返還

- **1塩基 伸長**するごとに蛍光イメージを取得する
- それぞれのDNAの「束」の蛍光色の変化を調べることで、塩基配列を決定する
- 数千万～数億本の塩基配列が得られる

画像イメージの取り込み



- 一つ一つの断片の塩基配列が短いと、アセンブルするのが困難
- 次世代シーケンサーで読み取ることができる塩基配列長は短いので、既に全塩基が解読されているゲノム配列（リファレンス配列）を利用したリシーケンスや、リファレンス配列へのマッピングなどに用いられることが多い



マッピング : Bowtie, Bowtie2, BWA など  
 アセンブル : Velvet, EDENA, Phrap など  
 ビューア : Tablet, IGV など

有償ではあるが、CLC Genomics Workbenchなどの解析ソフトも良く使われる

*Pectobacterium carotovorum*  
ssp. *carotovorum*

1



*Pectobacterium carotovorum*  
PR1 strain (病原性 強い)

2



*P. carotovorum* PR1株  
のゲノムを抽出



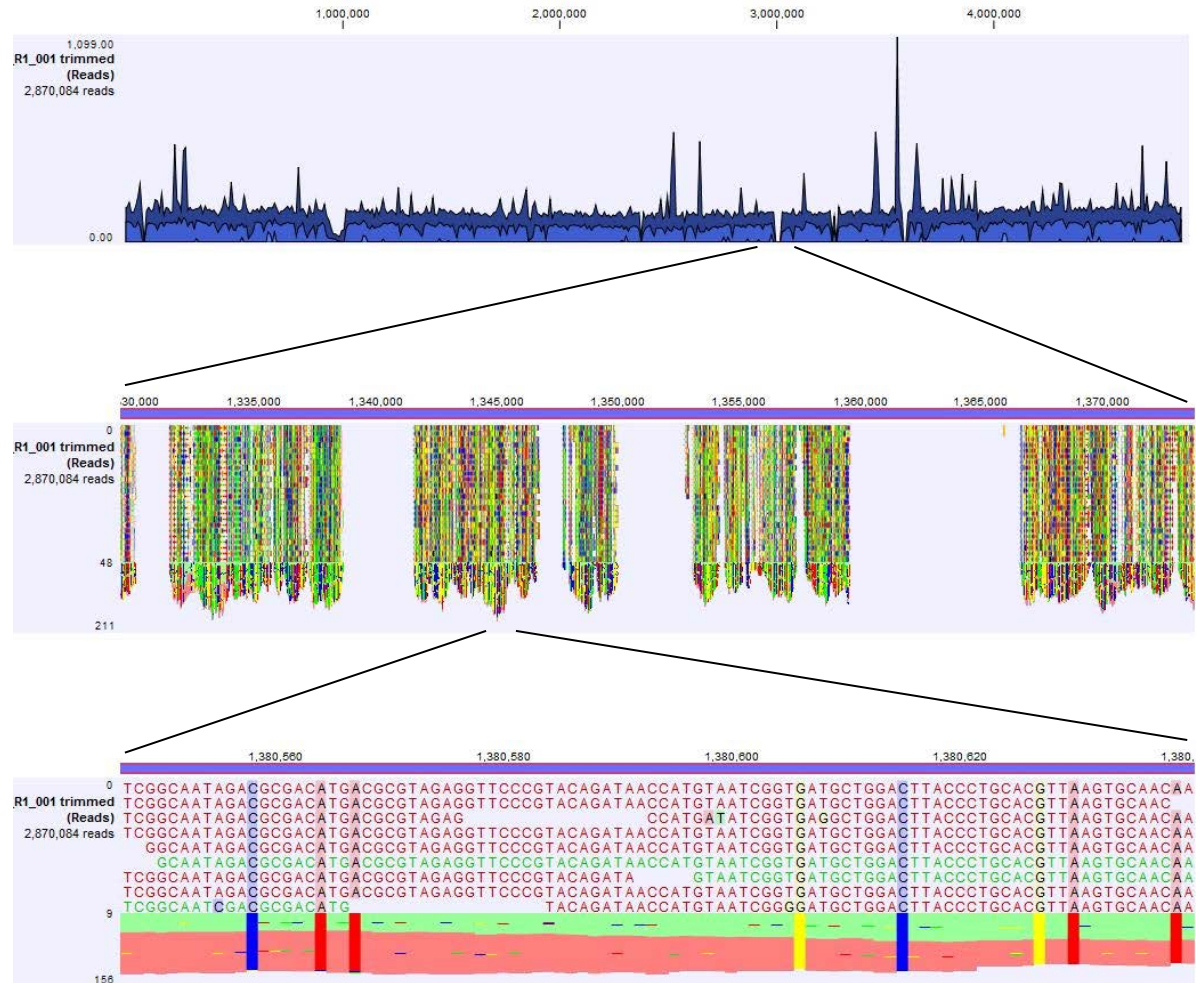
MiSeqを用いてシーケンス  
(約300万リード)



*P. carotovorum* ssp.  
*carotovorum*  
のゲノム (リファレンス配列)  
に対してマッピング

- 遺伝子の有無
- ゲノム構造の比較
- SNPの検出

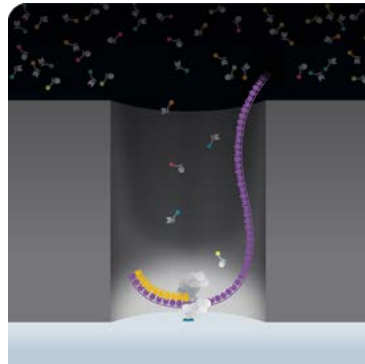
などの比較ゲノム解析ができる



## 第3世代シーケンサー

## Pacbio RS II DNA Sequencing System

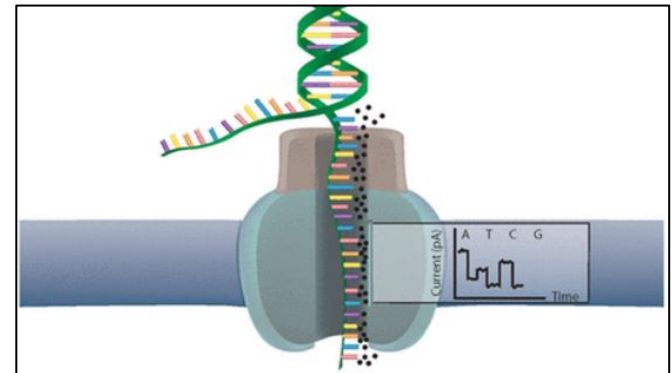
1



- DNA **1分子**を鋳型としてDNAポリメラーゼによるDNA合成を行う
- 1分子レベルでリアルタイムに塩基を読み取る
- 長いリード(平均10,000bp)が出力される

## MinION

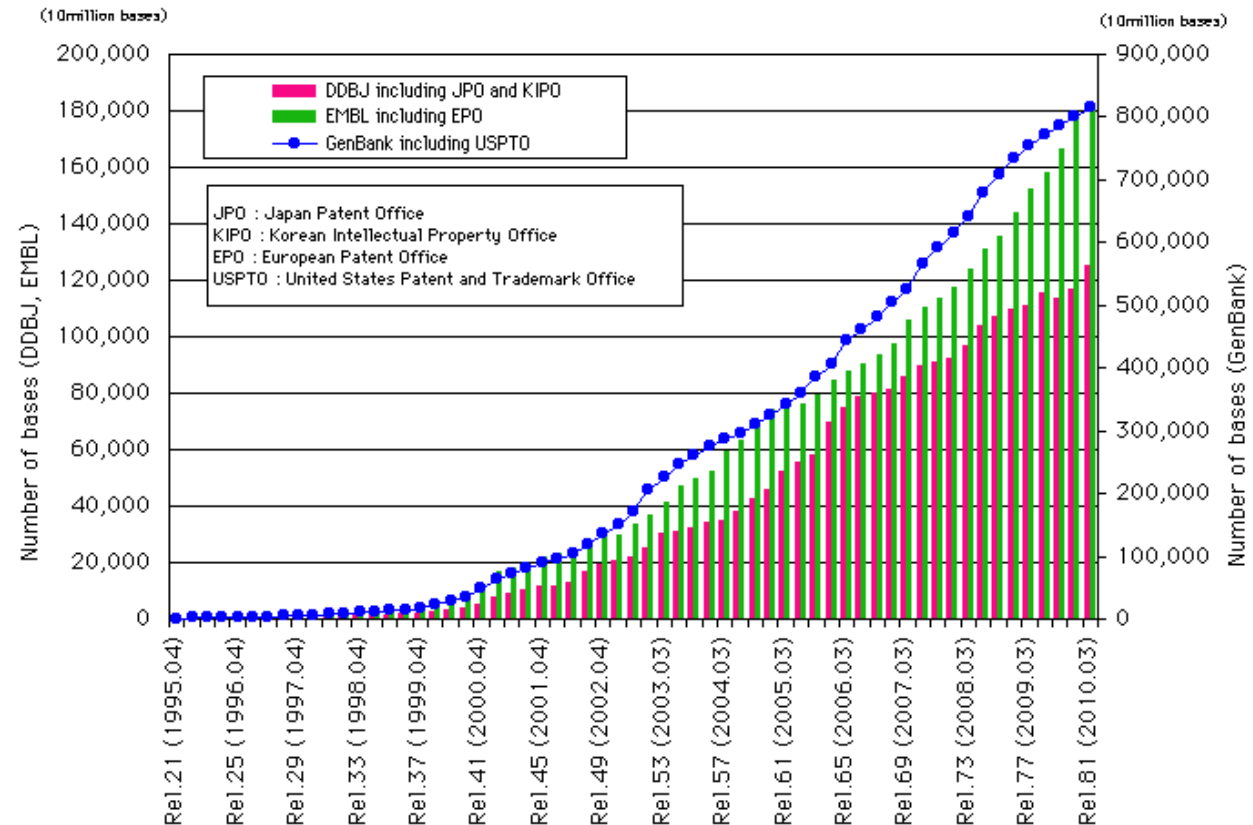
2



- USBメモリー用のシーケンサー
- DNAポリメラーゼを用いて1本鎖DNAに解きほぐす
- **ナノポア**を通過させる → 電流の変化を検知して配列を決定する

# 塩基データ登録数の推移

- シークエンス技術の進歩によって、塩基配列決定の速度はますます加速している



- 遺伝子の検出, アノテーション, 機能予測, 進化系統解析, 比較解析などを効率よく行い, 大量のシーケンスデータを有効に活用することが重要

ゲノムにコードされる遺伝子を網羅的に使用してホモロジー検索を行ったり、比較ゲノム解析を行いたい



大量のデータを処理するためのプログラミング技術が必要

バイオインフォマティクス分野では、Perl, C++, Java, Pythonなどが良く使われていますが、本日はPythonを用いて実習を行います

## Pythonの特徴

- コードの記述がシンプル
- 他人が作ったプログラムでも理解しやすい
- 深層学習（Deep learning）など、AIの分野で良く使われている
- YouTubeやInstagramなど、Webサービスにも使われている

プログラミング言語の  
人気ランキング（2020）

1位	C/C++
2位	Python
3位	JavaScript
4位	SQL
5位	C#
6位	Java
7位	VBA
8位	HTML/CSS
9位	PHP
10位	VB.NET

# プログラミングを用いたデータ処理

- 複数の質問配列で**BLAST検索**を行うと、結果が羅列した形で出力されます
- この中から、**必要な情報だけを取り出す**ためのプログラムを作ってみましょう
- 質問配列のアクセシオン番号と、検索の結果ヒットしたタンパク質のアクセシオン番号のリストを出力するプログラムを作成してみたいと思います

QueryのGene Index

gi|13507740|  
gi|13507741|  
gi|13507742|  
⋮

ヒットしたタンパク質のref番号

NP\_072661.1  
NP\_072662.1  
NP\_072663.1 NP\_072865.1  
⋮



## ■ デスクトップ上に、「kiso」フォルダを作成してください

※ ユーザ名に日本語が使われているときなどに、デスクトップではうまく作動しない場合があります。その場合は、kisoフォルダを C:ドライブの直下に移動させてみてください。



### 1. 生物配列解析基礎

#### 授業の目標・概要

生命科学のためのデータベースの利用と基本的な解析手法について講義します。データベースの基礎、配列データベース、機能データベース、ホモロジー検索、モチーフ解析などの基本的な手法について解説します。

kiso2

parse.py


BLAST.txt

BLAST.txt

parse.py

の2つのファイルをダウンロードして、kisoフォルダに入れてください

- コマンドプロンプトを立ち上げてください

 スタート → Windowsシステムツール → コマンドプロンプト

まず, kisoフォルダに移動します

```
> cd _
```

「cd (スペース)」と入力した後 (まだEnterキーは押さない) , kisoフォルダをコマンドプロンプト上にドラッグ&ドロップしてください

下記のように表示されますので, Enterキーを押してください

```
> cd C:¥Users¥iu¥Desktop¥kiso
```

- `parse.py`をメモ帳やワードパッドなどを使って開いてください

```
# Agribioinformatics
```

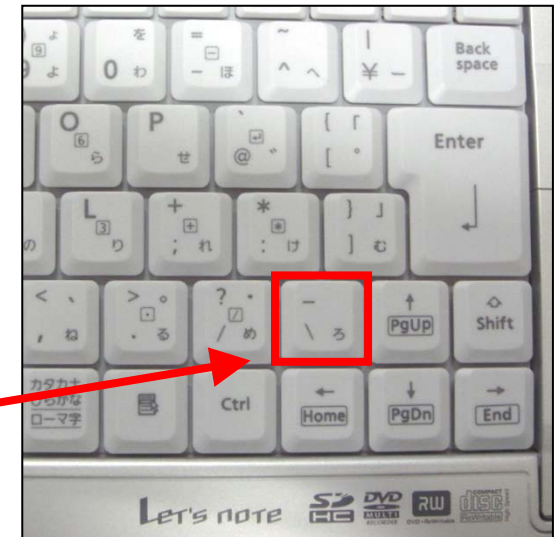
- 以下のように編集して, 上書き保存してください

```
# Agribioinformatics  
print('Hello¥nPython')
```

¥n は改行を表します

「¥」は, バックスラッシュ「\」を押してください

Windows上だと「¥」と表示されます



以下のコマンドを入力して, プログラムを実行してください

```
> python parse.py
```

↑ ※ 先頭の > は入力しないでください

# 変数

- 変数は、「**文字列**」で表します
  - ・ 使用できる文字は アルファベット、数字、漢字など
  - ・ 一文字目に数字(0~9)は使用できない
  - ・ 大文字と小文字は区別される
  - ・ 予約語 ( if など) は使用できない
- 以下のように編集して保存してください (灰色の文字は入力しないでください)

```
# Agribioinformatics  
lines = 'HelloPython' ← lineに文字列を入れる  
print(lines) ← lineを出力する
```

以下のコマンドを入力して、プログラムを実行してください

```
> python parse.py
```

# リダイレクト

- コマンドプロンプトに文字列を表示する代わりに、テキストファイルに**データを出力**することができます

以下のコマンドを入力して、プログラムを実行してください

↓ ↓ スペースを入れてください

```
> python parse.py > output1.txt
```

1

- フォルダの中に output1.txt という名前のファイルができるはず
- メモ帳やワードパッドでoutput1.txtを開いてみてください

```
Hello  
Python
```

# BLAST検索結果のデータを読み込む

- 既存のテキストファイルからデータを読み込ませて、それを構文解析していきます
- 「BLAST.txt」という**BLAST検索結果のファイル**を用意しておきました。メモ帳などを使って中身を見てください。
- *Mycoplasma pneumoniae*の約700個のタンパク質を質問配列に使用して、*Mycoplasma genitalium*のタンパク質に対してProtein BLASTを行った結果になります（700回分の結果が連なっています）

BLASTP 2.2.19 [Nov-02-2008]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Res.* 25:3389-3402.

.  
.

## BLAST検索結果の構造

質問配列 (Query) について書かれた行



BLASTP 2.2.5 [Nov-16-2002]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Res.* 25:3389-3402.

Query= **gi|16131851|ref|NP\_418449.1|** glucosephosphate isomerase [Escherichia coli K12] (549 letters)

Database: yeast.aa  
6298 sequences; 2,974,038 total letters

Sequences producing significant alignments:		Score	E
		(bits)	Value
ref NP_009755.1	Glucose-6-phosphate isomerase; Pgilp	641	0.0
ref NP_011646.1	Ygr130cp	30	0.98
ref NP_013146.1	spindle pole body component; Stu2p	29	1.7
ref NP_013847.1	(putative) involved in cell wall biogenesis; Ec...	28	3.7
ref NP_013523.1	Ylr419wp	28	3.7

BLAST検索の結果、ヒットした配列について書かれた行



>**ref|NP\_009755.1|** Glucose-6-phosphate isomerase; Pgilp  
Length = 554

Score = 641 bits (1654), Expect = 0.0  
Identities = 326/549 (59%), Positives = 401/549 (73%), Gaps = 16/549 (2%)

スコア や E-value



```

Query: 7  TQTAAWQALQKHFDEM-KDVTIADLFAKDGDRFSKFSATFDD----QMLVDYSKNRITEE 61
          T+  AW  LQK ++   K +++   F KD  RF K + TF +   ++L DYSKN + +E
Sbjct: 13  TELPAWSKLQKIYESQGKTLVVKQEFQKDAKRFKLNKFTFTNYDGSKILFDYSKNLVNDE 72

Query: 62  TLAKLQDLAKECDLAGAIKSMFSGEKINRTENRAVLHVALRNRSNTPILVDGKDVMP EVN 121
          +A L +LAKE ++ G   +MF GE IN TE+RAV HVALRNR+N P+ VDG +V PEV+
Sbjct: 73  IIAALIELAKEANVTGLRDAMFKGEHINSTEDRAVYHVALRNRANKPMYVDGVNVAPEVD 132
    
```



一つの検索結果が終わると、次の質問配列の検索結果が始まる

## 読み込み用のファイルを開く

- `open` 関数を使ってファイルを開きます
- 以下のように編集して保存してください

```
# Agribioinformatics
DATA = open('BLAST.txt', 'r') ←ファイルを開いてDATAに入れる
lines = 'Hello¥nPython'      'r' 読み込み用に関く
print(DATA)                  ←DATAを出力する
DATA.close()                 ←ファイルを閉じる
```

プログラムを実行してください

(上矢印キー + Enter で前回と同じコマンドを実行できます)

```
> python parse.py > output1.txt
```

(ただし、上記のプログラムではファイルの中身を見ることはできません)



# ファイルの中身を読み込む

- `readlines`メソッドを使ってファイルの中身を読み込みます
- 以下のように編集して保存してください

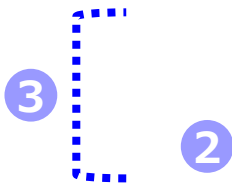
```
# Agribioinformatics
DATA = open('BLAST.txt', 'r') ←ファイルを開いてDATAに入れる
lines = DATA.readlines() ←DATAを読み込んでlinesに入れる
print(lines) ←linesを出力する
DATA.close() ←ファイルを閉じる
```

- プログラムを実行してください（上矢印キー + Enter で前回と同じコマンドを実行できます）
- メモ帳やワードパッドで `output1.txt` を開いてみてください
- `BLAST.txt`と同じ内容が出力されているはずです

# for文を使って1行ずつ解析する

- 1行ずつ構文解析するプログラムにしたいので、**for文**を利用して1行ずつ読み込むようにしてみましょう

```
# Agribioinformatics
DATA = open('BLAST.txt', 'r')
lines = DATA.readlines()
for line in lines:      ← linesを1行ずつlineに入れる
    print(line)        ← lineを出力する
DATA.close()
```



- ① 次の行からfor文の中身になるという意味のコロンを入れます
  - ② for文で繰り返す行（ブロック）には、**先頭にtab**を入れて字下げします
  - ③ linesの中身がなくなるまで、この部分が繰り返されます
- 実行してみましょう。全てのデータが出力されていればOKです。

## Pythonにおけるブロックについて

他のプログラミング言語、例えばJava や Perl では { から } までがブロックとなります

```
if (条件式) {  
    ブロック内の処理1  
    ブロック内の処理2  
}
```

それに対して Python ではインデント（字下げ）されている行をブロックとして扱います

```
if 条件式:  
    ブロック内の処理1  
    ブロック内の処理2
```

# パターンに当てはまるかどうかを調べる「re.search関数」

- 文字列「DNA」を含むかどうかを1行ずつ調べてみましょう

```
# Agribioinformatics
① import re
DATA = open('BLAST.txt', 'r')
lines = DATA.readlines()
for line in lines:
  ②     if re.search('DNA', line):
  ③         print(line)
DATA.close()
```

- ① このあと正規表現（regular expression）を使うので、reモジュールを読み込みます
- ② for文のブロックなので**tabを1回**入れて、1段下げます。変数lineにDNAという文字列が含まれていれば「真」となり、次の行のブロックを実行します。
- ③ for文のブロックであるとともに、if文のブロックでもあるので**先頭にtabを2回**入れて、2段下げてください。

- Query= で始まる行には、質問配列の情報が書かれています
- 文字列「Query=」を含む行を抜き出してみましょう

```
# Agribioinformatics
import re
DATA = open('BLAST.txt', 'r')
lines = DATA.readlines()
for line in lines:
    query = re.search('Query=', line)
    if query:
        print(line)
DATA.close()
```

← 変数queryに何か入っていたら  
← その行を出力する

このあと質問配列の番号 (Gene index) を抜き出していくので、`re.search`関数の結果を変数`query`に入れておきます

```
Query= gi|13507740|ref|NP_109689.1| DNA polymerase III beta subunit
Query= gi|13507741|ref|NP_109690.1| similar to j-domain of DnaJ
Query= gi|13507742|ref|NP_109691.1| DNA gyrase subunit B [Mycoplasma
      ⋮
```

# 正規表現による検索

- 検索する文字列の部分には **パターン**と呼ばれるものを入れることができます
- パターンとは, 「Mで始まる文字列」や「3文字の文字列」など, **文字列の特徴**を記述したものです
- このパターンの記述方法を**正規表現**といいます.

例えば,

DNA

DNNA

DNNNNNA

DNNNNNNNNNNNNNNNNNA

これらすべてを検索するには, **DN+A** と記述します

# 正規表現の例

## ■文字クラス

以下のものは、次のような1文字にマッチします。

[abc]	aかbかcのどれか
[a-z]	任意の小文字
[^abc]	aでもbでもcでもない文字
<b>3</b> ¥d	数字 (digit)
¥D	数字以外
¥w	英数字 (word)
¥W	英数字以外
<b>2</b> ¥s	空白文字 (space)
¥S	空白文字以外
¥b	単語境界 (word boundary)
<b>1</b> .	任意の一文字

## ■位置指定

パターンの位置を指定します。

<b>4</b> ^	先頭
\$	末尾

## ■エスケープ

/、^、\$などの、正規表現的に意味のある特殊記号自体を検索したい局面では、¥でエスケープします。

^¥^	^という字で始まる行にマッチ
<b>5</b> ¥¥	¥自体にマッチ

## ■ 繰り返し

以下の記号を使って、文字または文字クラスの繰り返しとマッチします。ここでは文字または文字クラスをxと書きます。

x*	0回以上の繰り返し
<b>6</b> x+	1回以上の繰り返し。xx*と同じ
x?	0回か1回
x{5}	5回繰り返し。xxxxxと同じ
x{3,}	3回以上繰り返し。xxx+と同じ
x{3,5}	3回以上5回以下繰り返し。xxxx?x?と同じ

## ■ グループと選択

文字列を繰り返すときは()を使ってグループ化します。

su(mo)+      sumo, sumomo, sumomomoなどにマッチする

いくつかのパターンのどれかにマッチさせるときは|を使います。

love|kiss      loveかkissにマッチする

stud(y|ies)      studyかstudiesにマッチする

su(mi|mo){2,3}      sumimi, sumimo, sumomi, sumomo, sumimimi, sumimimo,  
sumimomi, sumomimi, sumomomi, sumomimo, sumimomo,  
sumomomoのいずれかにマッチする



## 正規表現の練習

- 以下のアミノ酸配列はATP結合モチーフ（P-loop）です  
... G X X X X G K S ...  
あるいは  
... G X X X X G K T ... （X はどのアミノ酸でも良い）
- ATP結合モチーフを正規表現で表してみてください

```
⋮  
for line in lines:  
    query = re.search('Query=', line)  
    if query:  
        print(line)  
DATA.close()
```

試しにこの部分を上の正規表現に変えて、プログラムを動かしてみてください

## 正規表現による検索

- Gene indexを含む文字列を抽出してみましょう。

```
Query= gi|13507742|ref|NP_109691.1| DNA gyrase
```

**Query=** と **ref** ではさまれた連続した文字列を含む行を抽出するには

「**.**」 (**任意の文字**) と 「**+**」 (**1文字以上の連続文字**)

を使って以下のようにします

```
    ⋮  
for line in lines: ①  
    query = re.search(r'Query=¥s.+ref', line)  
    if query:  
        print(line) ②  
DATA.close()
```

- ① バックスラッシュを検索するときのために、raw文字にしておきます
- ② ¥s (バックスラッシュ s) はスペースを表します

## カッコを使った記憶

- 1 パターンの中で**括弧** ( ) を使うと、その括弧で囲まれた文字列が変数queryの中に格納されます
- 2 格納された文字列は **groupメソッド** で呼び出すことができます

```
    ⋮  
for line in lines:  
    query = re.search(r'Query=¥s(.+)ref', line)  
    if query:  
        print(query.group(1))  
DATA.close()
```

Diagram illustrating the code execution flow:

- 1 points to the `re.search` function call.
- 2 points to the `query.group(1)` call.

- 以下のようにGene indexが出力されていればOKです

```
gi|13507740|  
gi|13507741|  
gi|13507742|  
    ⋮
```

- 次に、BLAST検索によってヒットしたタンパク質の情報  
(例えば >gi|12044851|ref|NP\_072661.1| DNA polymerase III)  
を含む行を抽出してみましょう
- lineの中に「>gi」が含まれるかどうかを変数hitに入れます

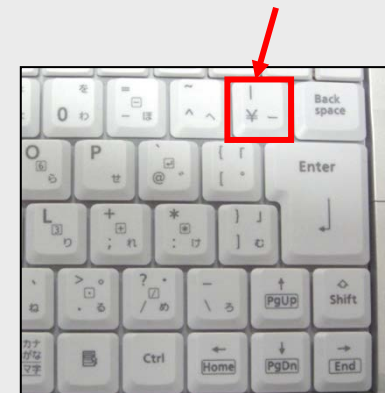
```
        ⋮  
for line in lines:  
    query = re.search(r'Query=¥s(.+)ref', line)  
    ① hit = re.search(r'>gi', line)  
    if query:  
        print(query.group(1))  
    ② if hit:  
        print(line)  
DATA.close()
```

- ① lineの中に「>gi」が含まれていれば、その情報がhitに入る
- ② もしhitに何か入っていれば出力する

- ヒットしたタンパク質情報のref番号だけを抽出してみよう  
>gi|12044851|ref|NP\_072661.1| DNA polymerase III )
- | (バーティカルバー) ではさまれた文字列を取り出したいのですが、  
'>gi.+ref|.+' ( >giで始まり、任意の連続した文字列、ref、  
| で挟まれた連続した文字列) の表現ではうまくいきません

```
    :  
for line in lines:  
    query = re.search(r'Query=¥s(.+)ref', line)  
    hit = re.search(r'>gi.+ref|.+', line)  
    if query:  
        print(query.group(1))  
    if hit:  
        print(line)  
DATA.close()
```

「|」 Shiftを押しながら



- "|"は正規表現で使用する特殊な文字であるため、別の意味になってしまうからです
- ここで使う "|" が正規表現でないことを示すために、 "|" の前に ¥ (バックスラッシュ) をつけます

```
        :  
for line in lines:  
    query = re.search(r'Query=¥s(.+)ref', line)  
    hit = re.search(r'>gi.+ref¥|.+.¥|', line)  
    if query:  
        print(query.group(1))  
    if hit:  
        print(line)  
DATA.close()
```

- 括弧を使って, ref番号だけを抽出してみましょう

```
>ref|NP_072866.1| topoisomerase IV, subunit A
```

```
    :  
for line in lines:  
    query = re.search(r'Query=¥s(.+)ref', line)  
    hit = re.search(r'>gi.+ref¥I(.+)¥I', line)  
    if query:  
        print(query.group(1))  
    if hit:  
        print(hit.group(1))  
DATA.close()
```

- 以下のようにQueryとGene indexが出力されていればOKです

質問配列の番号→  
ヒットした配列の番号→

```
gi|13507740|  
NP_072661.1  
gi|13507741|  
NP_072662.1|  
    :  
    :
```

- 質問配列とヒットした配列が1行で表示されるように、改行とタブ区切りを入れましょう
- 改行は **¥n**、タブ区切りは **¥t** で表します
- print関数では自動的に最後に改行が入ってしまうので、**end=''**と指定して改行しないようにしておき、質問配列が見つかったら改行するようにします

```
        :  
for line in lines:  
    query = re.search(r'Query=¥s(.+)ref', line)  
    hit = re.search(r'>gi.+ref¥l(.+)¥l', line)  
    if query:  
        print('¥n', query.group(1), end='')  
    if hit:  
        print('¥t', hit.group(1), end='')  
DATA.close()
```



質問配列とヒットした配列のアクセション番号を抽出できるようになりました

QueryのGene Index

ヒットしたタンパク質のref番号

gi|13507740|  
gi|13507741|  
gi|13507742|  
gi|13507743|  
gi|13507744|  
gi|13507745|  
gi|13507746|  
gi|13507747|  
gi|13507748|  
.  
.  
.

NP\_072661.1  
NP\_072662.1  
NP\_072663.1 NP\_072865.1  
NP\_072664.1 NP\_072866.1  
NP\_072665.1  
NP\_072666.1  
NP\_072667.1  
NP\_072668.1 NP\_072998.1  
NP\_072669.1

- ヒットした配列のうち**最も相同性の高いもの**だけを表示してみましょう
- 新たに mode という変数を使い、これが **0** か **1** かを指標にします

```
# Agribioinformatics
import re
DATA = open('BLAST.txt', 'r')
lines = DATA.readlines()
mode = 0
for line in lines:
    query = re.search(r'Query=¥s(.+)ref', line)
    hit = re.search(r'>gi.+ref¥l(.+)¥l', line)
    if query:
        print('¥n', query.group(1), end='')
        mode = 1
    if hit and mode == 1:
        print('¥t', hit.group(1), end='')
        mode = 0
DATA.close()
```

- 1 質問配列 (Query= の行) を見つけたら **mode = 1** にします
- 2 その後の最初に出てくるヒット配列 (>ref の行) を見つけたら、番号を抽出して **mode = 0** に戻します (次の質問配列を見つけるまで抽出しません)

2つ以上の条件が揃ったときにif文の中身を動かしたいときは and でつなぎます

```
if hit and mode == 1:
```

if文の中で「もし～なら」を表現するときはイコールを2つ (==) にします

以下のように**質問配列**と**最も相同性の高いヒット配列**のアクセション番号が抽出できていれば大丈夫です

質問配列のGene Index

ヒットしたタンパク質のref番号

gi|13507740|  
gi|13507741|  
gi|13507742|  
gi|13507743|  
gi|13507744|  
gi|13507745|  
gi|13507749|  
gi|13507750|  
⋮  
⋮

NP\_072661.1  
NP\_072662.1  
NP\_072663.1  
NP\_072664.1  
NP\_072665.1  
NP\_072666.1

} 相同性の高い配列がヒットし  
ない場合は空欄になります

## <課題 1>

- 質問配列のGene Indexのうち、**数字の部分だけ**を取り出して、以下のような出力結果になるプログラムを作成してください  
(46ページ参照)

```
13507740      NP_072661.1
13507741      NP_072662.1
13507742      NP_072663.1
13507743      NP_072664.1
13507744      NP_072665.1
13507745      NP_072666.1
13507746      NP_072667.1
13507747      NP_072668.1
13507748      NP_072669.1
.
.
```

## <課題2>

- **E-valueの値**を抽出して、以下のような出力結果になるプログラムを作成してください

質問配列の Gene Index	ヒットしたタンパク質 のref番号	E-value
13507740	NP_072661.1	1e-148
13507741	NP_072662.1	1e-125
13507742	NP_072663.1	0.0
13507743	NP_072664.1	0.0
13507744	NP_072665.1	0.0
13507745	NP_072666.1	1e-077
	・	
	・	

課題1と同じファイル名にならないように、**output2.txt**というファイル名で結果を出力してください

- 出力したテキストファイル (**output1.txt** と **output2.txt**) を、メールに添付して提出してください
- 送付先は kenro[at]hosei.ac.jp です（ [at] を@に変換）
- メールのはじめの件名は「**Python課題**」にしてください
- メール本文に、以下のように「氏名」「所属」「学生証番号」「本日の講義の感想」を記載してください

氏名：○○ ○○

所属：××××専攻 △△△△研究室

学生証番号：□□□□□

講義の感想：