2023年8月25-27日
遺伝統計学・夏の学校　講義実習資料

プログラミング入門

大阪大学大学院医学系研究科 遺伝統計学
東京大学大学院医学系研究科 遺伝情報学
理化学研究所生命医科学研究センター　システム遺伝学チーム

http://www.sg.med.osaka-u.ac.jp/index.html

講義の概要

プログラミング入門

① プログラミングについて

② プログラミング言語の比較

③ Python入門実習

④ AWK入門実習

本講義資料は、Windows PC上で
C:¥SummerSchool にフォルダを配置することを想定しています。

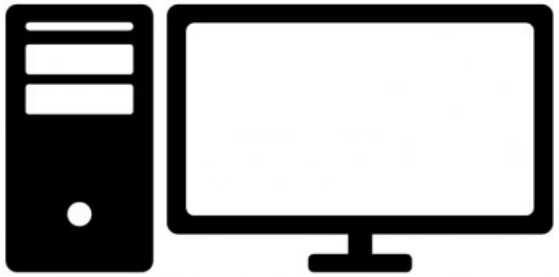
① プログラミングについて

人間の言語

・処理1を実行
・処理2を実行
・条件1なら処理3を実行
・条件2なら処理4を実行
⋮

プログラミング

変換

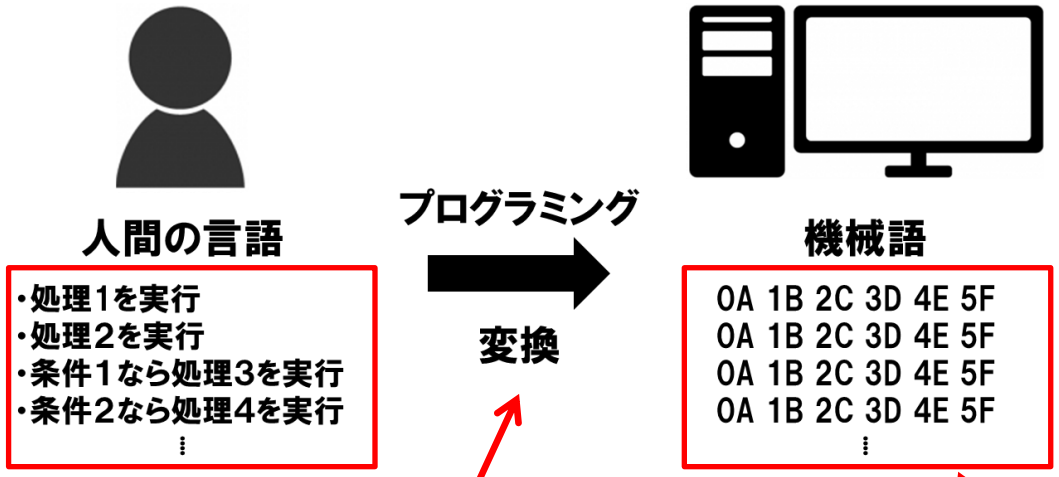


機械語

0A 1B 2C 3D 4E 5F
0A 1B 2C 3D 4E 5F
0A 1B 2C 3D 4E 5F
0A 1B 2C 3D 4E 5F
⋮

- プログラミングとは、意図した処理を実施させる目的で、コンピューターに指示(＝プログラム)を与えることです。
- コンピューターは人間の言語を理解できないため、コンピューターが理解できる言語(＝機械語)に翻訳して、命令する必要があります。
- 人間の言語を機械語に変換する手段が、プログラミング言語です。

① プログラミングについて



コンパイラ、インタプリタ、等の種類がある

プログラミング言語で書かれたソースコード

CPUの種類に応じて書き分ける必要がある

- プログラミング言語で書かれた人間が理解できる命令文(＝ソースコード)を、コンパイラやインタプリタといった手段で機械語に翻訳します。
- 機械語は、コンピューターのCPUの種類によっても異なるため、翻訳手段をどのように行うか、はプログラム言語の性質や性能に影響します。

① プログラミングについて

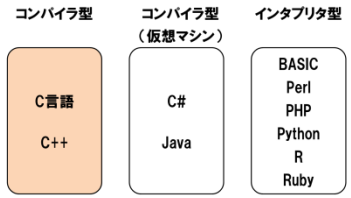
コンパイラ型	コンパイラ型（仮想マシン）	インタプリタ型
C言語	C#	BASIC
C++	Java	Perl
		PHP
		Python
		R
		Ruby

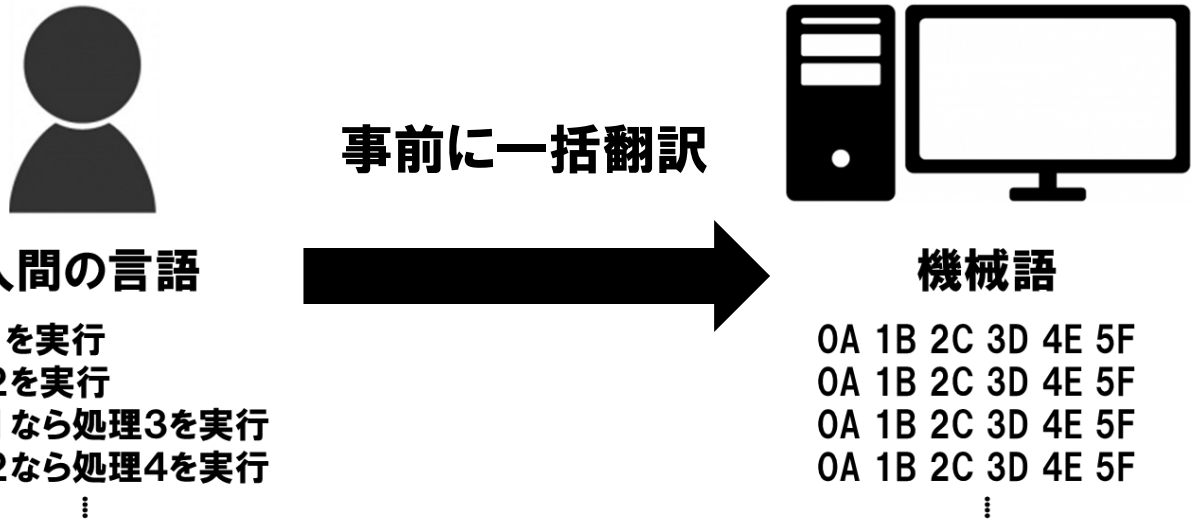
・プログラミング言語には、多くの種類があります。

・翻訳手段の違いによって、大きく3つに分類することができます。

コンパイラ型、コンパイラ型（仮想マシン）、インタプリタ型

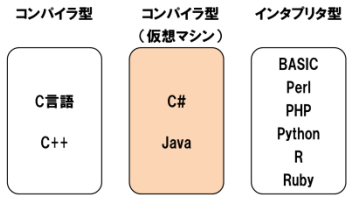
① プログラミングについて

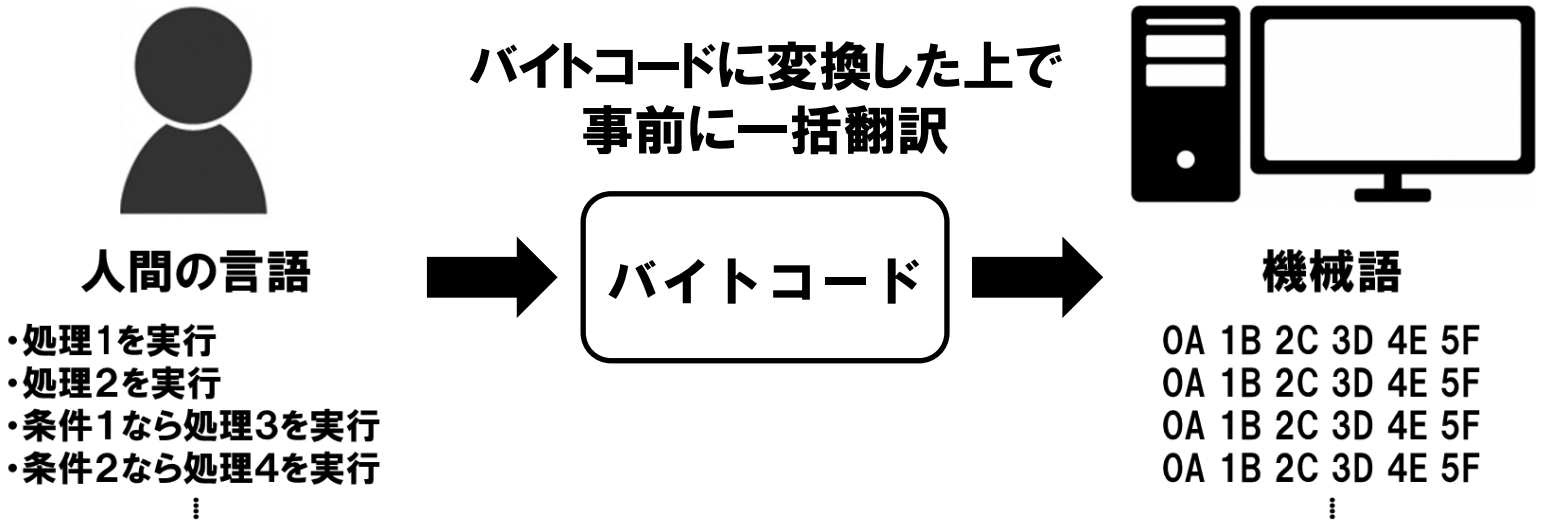




- **コンパイラ型のプログラミング言語**は、ソースコードから機械語へ、**「プログラム実施前に一括して翻訳（＝コンパイル）」**します。
- 予め翻訳済みなので、**高速な処理の実施**が可能です。
- コンパイルは専門性が高く、CPU環境毎に実施する必要があるため、実行環境の共有や初心者の利用が難しい場合があります。

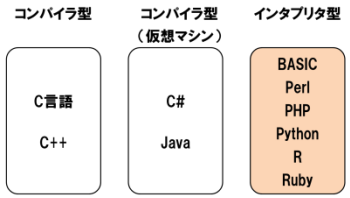
① プログラミングについて

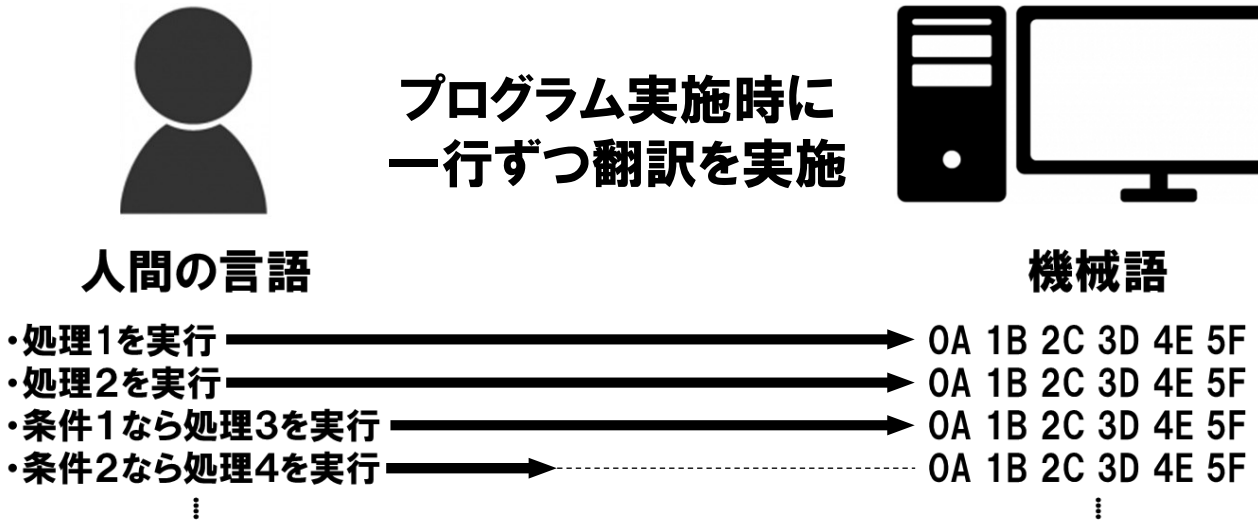




- **仮想マシンを用いたコンパイラ型のプログラミング言語は、ソースコードを一旦、バイトコードという中間的なコードに変換します。**
- **コンピューター上の仮想マシンがバイトコードを機械語に翻訳します。**
- **仮想マシンを使用することで、どのプラットフォーム（CPUやOSの種類）でも共通して実行可能なります。**

① プログラミングについて



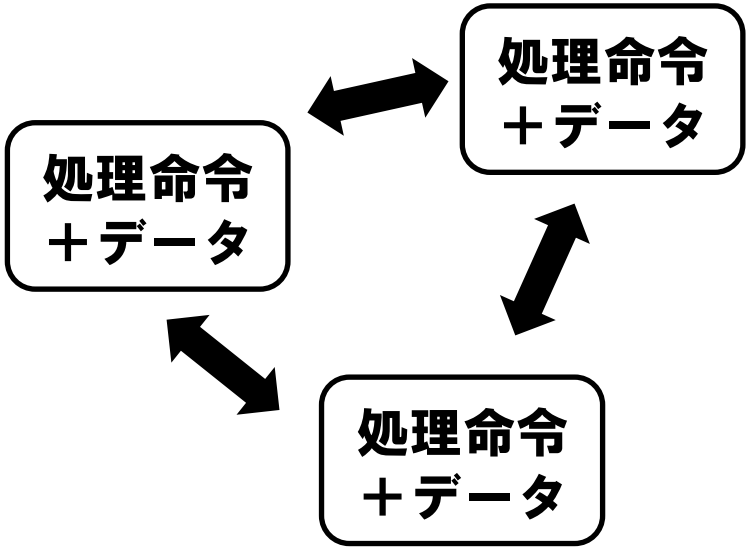


- **インタプリタ型**のプログラミング言語は、**プログラムの実施時に、機械語への翻訳を逐次実施**します。
- コンパイル作業が不要なのと、ソースコードのエラー箇所でプログラムが停止するので、バグ（間違ったソースコードによるエラー）の発見も簡単です。
- 初心者にはお勧めですが、**処理速度は遅くなります**。

① プログラミングについて

手続き型

オブジェクト指向



- プログラミング言語を分類する別の指標として、**手続き型**と、**オブジェクト指向**があります。
- 手続き型とは、一方向性の処理命令で構成されたプログラミングです。
- オブジェクト指向とは、複数のオブジェクト（＝処理命令とデータのセット）で構成されたプログラミングです。
- オブジェクト指向を心掛けると、ソースコードを再利用しやすくなります。

① プログラミングについて

・色々説明してちょっとわかりにくいかもしれませんが、まとめると・・・

コンパイラ型言語：上級者向けで文法が厳密だが処理速度が速い
仮想マシン：コンパイラ型言語の扱いの汎用性が増す
インタプリタ型言語：初心者向けでわかりやすいが処理速度が遅い
オブジェクト指向：ソースコードの効率的な再利用を可能にする書き方

といったところです。

講義の概要

プログラミング入門

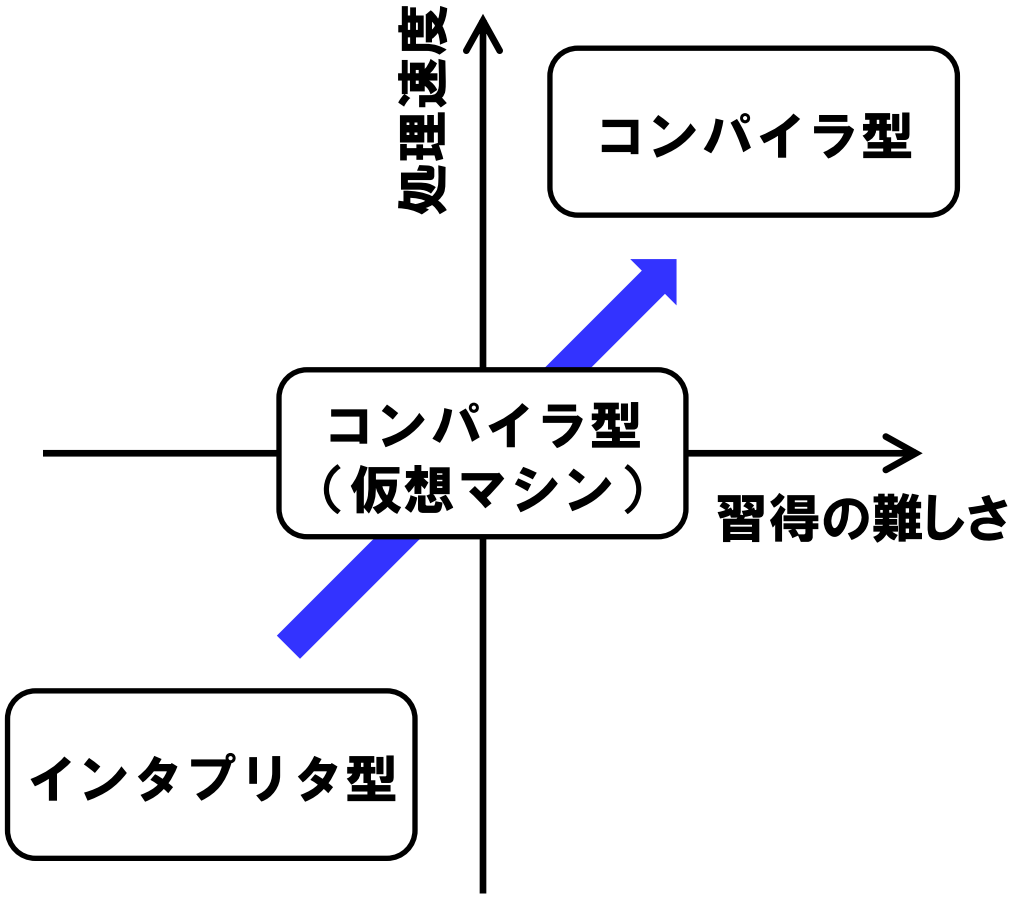
① プログラミングについて

② プログラミング言語の比較

③ Python入門実習

④ AWK入門実習

② プログラミング言語の比較



・どのプログラム言語が一番か、という論争は結論を出しにくいです。
・1つの言語に固執する必要もなく、状況に応じた使い分けも大事です。
・初心者には、**最初に簡単なインタプリタ型言語を1つ習得**した上で、コンパイラ型言語に挑戦する、という手順がオススメです。

② プログラミング言語の比較

C言語

```
#include <stdio.h>

int main(void)
{
    printf("Hello, world!");
    return 0;
}
```

※比較目的で、各プログラミング言語における、”Hello World”を記載してみました。

※”Hello World”とは、そのプログラム言語を使って画面上に”Hello World”と出力するのに必要なソースコードの事です。

- C言語は、コンパイラ型言語です。
- 何十年も使われている歴史ある言語で、処理速度が速いです。
- 他のプログラム言語やOSも、元はC言語で作られている例が多いです。
- メモリ管理やファイルポインタ操作などを、他のプログラミング言語ではあまり意識しなくていい点を、自分で管理する必要があります。

② プログラミング言語の比較

Java

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

- Javaは、仮想マシンを用いたコンパイラ型言語です。
- C言語程ではないですが、処理速度は速い方に分類されます。
- オブジェクト指向を目指した言語です。
- 仮想マシンを使用するため、プラットフォームに依存せず実行可能です。
- 文法が厳密で、ソースコードの量や複雑性が増す傾向にあります。

② プログラミング言語の比較

```
10 PRINT "Hello, world!"
20 END
```

BASIC

- **BASICは、インタプリタ型の言語です。**
- **記述が簡単で、プログラミング入門として使われてきました。**
- **Windowsの普及前、N88-BASIC(PC-9801シリーズ)、F-BASIC(FM TOWNSシリーズ)など、複数のバリエーションで使用されていました。**
- **ソースコード内の行数に基づき実行されるなど、最近の言語にはない特徴が残っています。**

② プログラミング言語の比較

```
print "Hello, world!¥n";
```

Perl

- **Perlは、インタプリタ型の言語です。**
- **文法が厳密でないため、初心者でもソースコードを書きやすいです。**
- **一方で、他人の書いたソースコードを理解しにくい、という面もあります。**
- **テキストファイル処理に優れている点が、ゲノムデータ解析では便利です。**

② プログラミング言語の比較

Python

```
print "Hello, world!"
```

（バージョン2まで）

```
print("Hello, world!")
```

（バージョン3以降）

- Pythonはインタプリタ型の言語です。
- 位置づけはPerlに似ているが、ソースコードが簡潔で理解しやすいです。
- 機械学習や人工知能の研究分野において、重宝されています。
- ゲノムデータ解析分野においても、近年利用者が増えています。

② プログラミング言語の比較

```
print("Hello, world!")
```

R

- Rはインタプリタ型の言語です。
- 統計解析に特化した言語として、データ解析分野で広く使われています。
- その性質上、ベクトル形式の変数の扱いと相性が良いです。
- データ描画機能が充実していて、綺麗なグラフや図が作れます。

② プログラミング言語の比較

A#, A-0 System, A+, A++, ABAP, ABC, ABC ALGOL, ABSET, ABSYS, ACC, Accent, Ace DASL, ACL2, ACT-III, Action!, ActionScript, Ada, Adenine, Agda, Agilent VEE, Agora, AIMMS, Alef, ALF, ALGOL, Alice, Alma-0, AmbientTalk, Amiga E, AMOS, AMPL, Apex, APL, AppleScript, Arc, ARexx, Argus, AspectJ, Assembly language, ATS, Ateji PX, AutoHotkey, Autocoder, AutoIt, AutoLISP, Averest, AWK, Axum, B, Babbage, Bash, **BASIC**, bc, BCPL, BeanShell, Batch, Bertrand, BETA, Bigwig, Bistro, BitC, BLISS, Blockly, BlooP, Blue, Boo, Boomerang, Bourne shell, BREW, BPEL, **C**, C--, C++, C#, C/AL, Caché ObjectScript, C Shell, Caml, Cayenne, CDuce, Cecil, Cel, Cesil, Ceylon, CFEngine, CFML, Cg, Ch, Chapel, CHAIN, Charity, Charm, Chef, CHILL, CHIP-8, chomski, ChucK, CICS, Cilk, Citrine, CL , Claire, Clarion, Clean, Clipper, CLIST, Clojure, CLU, CMS-2, COBOL, Cobra, CODE, CoffeeScript, ColdFusion, COMAL, COMIT, COMPASS, Component Pascal, Converge, Cool, Coq, Coral 66, Corn, CorVision, COWSEL, CPL, Cryptol, csh, Csound, CSP, CUDA, Curl, Curry, Cyclone, Cython, D, DAS, Dart, DataFlex, Datalog, DATATRIEVE, dBase, dc, DCL, Deesel, Delphi, DinkC, DIBOL, Dog, Draco, DRAKON, Dylan, DYNAMO, E, E#, Ease, Easy PL/I, Easy Programming Language, EASYTRIEVE PLUS, ECMAScript, Edinburgh IMP, EGL, Eiffel, ELAN, Elixir, Elm, Emacs Lisp, Emerald, Epigram, EPL, Erlang, es, Escher, ESPOL, Esterel, Etoys, Euclid, Euler, Euphoria, EusLisp, CMS EXEC, EXEC 2, Executable UML, F, F#, Factor, Falcon, Fantom, FAUST, FFP, Fjölnir, FL, Flavors, Flex, FlooP, FLOW-MATIC, FOCAL, FOCUS, FOIL, FORMAC, @Formula, Forth, Fortran, Fortress, FoxBase, FoxPro, FP, FPr, Franz Lisp, Frege, F-Script, Game Maker Language, GameMonkey Script, GAMS, GAP, G-code, Genie, GDL, GJ, GEORGE, GLSL, GNU E, GM, Go, Go!, GOAL, Gödel, Godiva, Golo, GOM, Google Apps Script, Gosu, GOTRAN, GPSS, GraphTalk, GRASS, Groovy, Hack, HAL/S, Hamilton C shell, Harbour, Hartmann pipelines, Haskell, Haxe, High Level Assembly, HLSL, Hop, Hopscotch, Hope, Hugo, Hume, HyperTalk, IBM Basic assembly language, IBM HAScript, IBM Informix-4GL, IBM RPG, ICI, Icon, Id, IDL, Idris, IMP, Inform, Io, Ioke, IPL, IPTSCRAE, ISLISP, ISPF, ISWIM, J, J#, J++, JADE, Jako, JAL, Janus, JASS, **Java**, JavaScript, JCL, JEAN, Join Java, JOSS, Joule, JOVIAL, Joy, JScript, JScript .NET, JavaFX Script, Julia, Jython, K, Kaleidoscope, Karel, Karel++, KEE, Kixtart, Klerer-May System, KIF, Kojo, Kotlin, KRC, KRL, KRL, KRYPTON, ksh, L, L# .NET, LabVIEW, Ladder, Lagoona, LANSA, Lasso, LaTeX, Lava, LC-3, Leda, Legoscript, LIL, LilyPond, Limbo, Limnor, LINC, Lingo, LIS, LISA, Lisaac, Lisp, Lite-C, Lithe, Little b, Logo, Logtalk, LotusScript, LPC, LSE, LSL, LiveScript, Lua, Lucid, Lustre, LYaPAS, Lynx, M2001, MarsCode, M4, M#, Machine code, MAD, MAD/I, Magik, Magma, make, Maple, MAPPER, MARK-IV, Mary, MASM, MATH-MATIC, Mathematica, MATLAB, Maxima, Max, MaxScript, Maya, MDL, Mercury, Mesa, Metafont, Microcode, MicroScript, MIIS, MillScript, MIMIC, Mirah, Miranda, MIVA Script, ML, Moby, Model 204, Modelica, Modula, Modula-2, Modula-3, Mohol, MOO, Mortran, Mouse, MPD, MSIL, MSL, MUMPS, MPL, NASM, Napier88, Neko, Nemerle, nesC, NESL, Net.Data, NetLogo, NetRexx, NewLISP, NEWP, Newspeak, NewtonScript, NGL, Nial, Nice, Nickle, Nim, NPL, NSIS, Nu, NWScript, NXT-G, o:XML, Oak, Oberon, OBJ2, Object Lisp, ObjectLOGO, Object REXX, Object Pascal, Objective-C, Objective-J, Obliq, OCaml, occam, occam-π, Octave, OmniMark, Onyx, Opa, Opal, OpenCL, OpenEdge ABL, OPL, OPS5, OptimJ, Orc, ORCA/Modula-2, Oriel, Orwell, Oxygene, Oz, P", P#, ParaSail, PARI/GP, Pascal, PCASTL, PCF, PEARL, PeopleCode, **Perl**, PDL, Perl6, Pharo, PHP, Phrogram, Pico, Picolisp, Pict, Pike, PIKT, PILOT, Pipelines, Pizza, PL-11, PL/0, PL/B, PL/C, PL/I, PL/M, PL/P, PL/SQL, PL360, PLANC, Plankalkül, Planner, PLEX, PLEXIL, Plus, POP-11, PostScript, PortablE, Powerhouse, PowerBuilder, PowerShell, PPL, Processing, Processing.js, Prograph, Progress, PROIV, Prolog, PROMAL, Promela, PROSE modeling language, PROTEL, ProvideX, Pro*C, Pure, **Python**, Q, Qalb, QtScript, QuakeC, QPL, **R**, R++, Racket, RAPID, Rapira, Ratfiv, Ratfor, rc, REBOL, Red, Redcode, REFAL, Reia, REXX, Rlab, ROOP, RPG, RPL, RSL, RTL/2, Ruby, RuneScript, Rust, S, S2, S3, S-Lang, S-PLUS, SA-C, SabreTalk, SAIL, SALSA, SAM76, SAS, SASL, Sather, Sawzall, SBL, Scala, Scheme, Scilab, Scratch, Script.NET, Sed, Seed7, Self, SenseTalk, SequenceL, SETL, SIMPOL, SIGNAL, SiMPLE, SIMSCRIPT, Simula, Simulink, SISAL, SLIP, SMALL, Smalltalk, Small Basic, SML, Snap!, SNOBOL(SPITBOL), Snowball, SOL, Span, SPARK, Speedcode, SPIN, SP/k, SPS, SQR, Squeak, Squirrel, SR, S/SL, Stackless Python, Starlogo, Strand, Stata, Stateflow, Subtext, SuperCollider, SuperTalk, Swift, SYMPL, SyncCharts, SystemVerilog, T, TACL, TACPOL, TADS, TAL, Tcl, Tea, TECO, TELCOMP, TeX, TEX, TIE, Timber, TMG, Tom, TOM, TouchDevelop, Topspeed, TPU, Trac, TTM, T-SQL, Transcript, TTCN, Turing, TUTOR, TXL, TypeScript, Turbo C++, Ubercode, UCSD Pascal, Umple, Unicon, Uniface, UNITY, Unix shell, Vala, Visual DataFlex, Visual DialogScript, Visual Fortran, Visual FoxPro, Visual J++, Visual J#, Visual Objects, Visual Prolog, VSXu, vvvv, WATFIV, WATFOR, WebDNA, WebQL, Whiley, Windows PowerShell, Winbatch, Wolfram Language, Wyvern, X++, X#, X10, XBL, XC, xHarbour, XL, Xojo, XOTcl, XPL, XPL0, XQuery, XSB, XSLT, Xtend, Yorick, YQL, Z notation, Zeno, ZOPL, Zsh.

- **これまでに、多数のプログラム言語が開発されてきました。**
- **複数のプログラム言語を知ると、プログラミングの理解が深まります。**
- **（余裕があれば）色々な言語を試してみるといいかもしれません。**

講義の概要

プログラミング入門

① プログラミングについて

② プログラミング言語の比較

③ Python入門実習

④ AWK入門実習

③ Python入門実習

どうしたらプログラムを書けるようになりますか？

プログラムの文法を学ぶことと、既に書かれたソースコードを読むことの、両方を行うと効果的です。

・全くのゼロからソースコードを書き始めると、ちょっとしんどいです。
・プログラムの**基礎的な文法を覚える**ことと、**既に書かれたソースコードを読んで試してみる**ことは、両方が重要です（経験則ですが・・・）。

③ Python入門実習

※課金制クラウド環境を使用する場合は、使用料金がかかります。

変なプログラムを書いてPCが壊れないか不安です・・・

大丈夫!!プログラムが間違っていても、PCが壊れることはありません。お金もかかりません。どんどん間違ったプログラムを書いて、経験値を稼いでいきましょう。

- **Wetの実験と事なり、（一般論として）dryの研究では間違ったプログラムでPCが壊れることは(あまり)ありません。研究費もかかりません※。**
- **どんどん間違ったプログラムを書いて、経験を積んで下さい。**
- **サーバーの他ユーザーや管理者に迷惑をかけることは、しばしばあります。素直に謝って原因を教えて貰い、同じ過ちを繰り返さないように。**

③ Python入門実習

Python

② プログラミング言語の比較

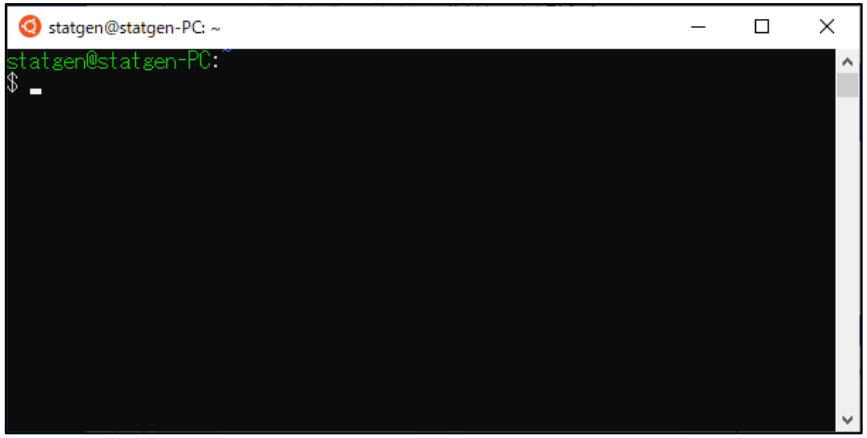
Python

print "Hello, world!"
（バージョン2まで）

print("Hello, world!")
（バージョン3以降）

・Pythonはインタプリタ型の言語です。
・位置づけはPerlに似ているが、ソースコードが簡潔で理解しやすいです。
・機械学習や人工知能の研究分野において、重宝されています。
・ゲノムデータ解析分野においても、近年利用者が増えています。

Ubuntu 18.04-LTS
（WSL2）



・今回の講義では、インタプリタ型言語であるPythonに触れてみます。
・Pythonは現在Python3というバージョンで開発されています。今回の講義でもPython3を使用します（ゲノムデータ解析のツールにはPython 2でなければ動かないものもあるので、注意です）。
・文法が厳密でないため、初心者でもソースコードを書きやすいです。

③ Python入門実習

GeneID.txt

1	A1BG
2	A2M
3	A2MP1
9	NAT1
10	NAT2
11	AACP
12	SERPINA3
13	AADAC
14	AAMP
15	AANAT
16	AARS
17	AAVS1
⋮	

SelectGeneID.py

遺伝子番号が特定の数の倍数になっている遺伝子を抽出(例:3の倍数)

3	A2MP1
9	NAT1
12	SERPINA3
15	AANAT
18	ABAT
21	ABCA3
24	ABCA4
27	ABL2
30	ACAA1
33	ACADL
36	ACADSB
39	ACAT2
⋮	

・遺伝子番号(Gene ID)と遺伝子名が書かれたファイル"GeneID.txt"から、遺伝子番号が特定の数の倍数になっている遺伝子を書き出すPythonソースコード:"SelectGeneID.py"を実行してみましょう。

③ Python入門実習

statgen@statgen-PC: ~

$ cd /mnt/c/SummerSchool/Programming/　※Cygwinの場合 /mnt/を/cygdrive/に変えてください

statgen@statgen-PC: /mnt/c/SummerSchool/Programming

$ python3 SelectGeneID.py GeneID.txt 3 ← **コマンドを打ち込むと・・・**

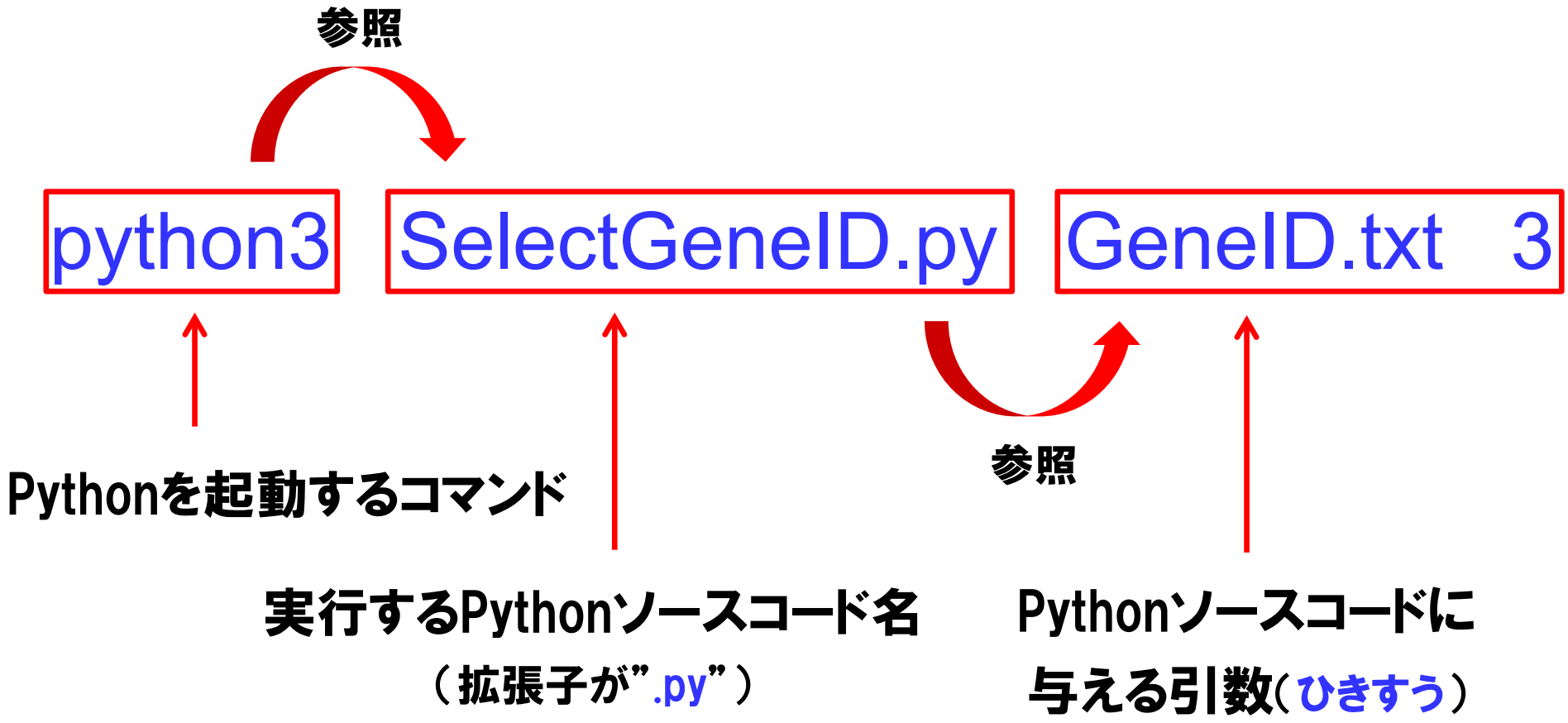
```
3       A2MP1
9       NAT1
12      SERPINA3
15      AANAT
18      ABAT
21      ABCA3
24      ABCA4
27      ABL2
  ⋮
```

結果が出力されます

← **途中で止めるには、”Ctrl+C”を押します**

- **Shellを起動して、作業ディレクトリに移動し、”python3 SelectGeneID.py GeneID.txt 3”と書き込むと、実行できます。**

③ Python入門実習



・Pythonソースコードの実行コマンドは、"python3（ソースコード名）（引数）"という形をとります。

・複数の引数を与える場合は、スペースで区切って入力します。

③ Python入門実習

※Pythonソースコード”SelectGeneID.py”を
開いて、中身を覗いてみましょう。

```
#!/usr/bin/python3

import sys

input_file = sys.argv[1]
number   = int(sys.argv[2])

with open(input_file) as fh:
    for line in fh:
        strings = line.strip().split()
        if(int(strings[0]) % number == 0):
            print(strings[0]+"¥t"+strings[1])
```

③ Python入門実習

※文頭は、Pythonソースコード実行のために必要な呪文のようなもの、と考えておいて下さい。
（毎回同じ呪文を書き込めば大丈夫です。）

赤枠内が、今回新たに書いたソースコードで、実際の処理を担当している部分に相当します。

```
#!/usr/bin/python3

import sys

input_file = sys.argv[1]
number   = int(sys.argv[2])

with open(input_file) as fh:
    for line in fh:
        strings = line.strip().split()
        if(int(strings[0]) % number == 0):
            print(strings[0]+"¥t"+strings[1])
```

③ Python入門実習

```
import sys
```

sysモジュールをインポート

```
input_file = sys.argv[1]
number   = int(sys.argv[2])
```

入力された引数を変数として代入

```
with open(input_file) as fh:
    for line in fh:
        strings = line.strip().split()

        if(int(strings[0]) % number ==0):
            print(strings[0]+"¥t"+strings[1])
```

入力ファイルをファイルハンドル”fh”として開く

fhを一行ずつ読み込む

読み込み行の末尾の改行コードを削除し、タブ区切りで分割

「第一列を引数で割った余りが0かどうか」を確認

標準出力画面へ出力

③ Python入門実習

```
import sys
```

sysモジュールをインポート

- Pythonではすでに誰かが(もしくは自分自身で)別の場所に書いたプログラムを利用(=インポート)できる機能があります。
- "import モジュール名"でモジュールをインポートします。
- コマンドラインの引数を取得するためにはsysモジュールを利用します

③ Python入門実習

```
input_file = sys.argv[1]
```

入力された引数を変数として代入

```
python3  SelectGeneID.py  GeneID.txt  3
```

sys.argv[0]　sys.argv[1]　sys.argv[2]

- Pythonにおけるコマンドライン引数はsys.argvという配列に入ります。
（sys.argvという配列名は、sysモジュールを使用していることに由来します。）
- sys.argv [0] はスクリプト名（SelectGeneID.py）、sys.argv [1] 以降にユーザーが指定した引数が入ります。
- ”=”により左辺の「変数」に右辺の値を代入することができます。
- 数字（整数・小数）や文字列など、幅広く変数として代入できます。

③ Python入門実習

```
number   = int(sys.argv[2])
```

入力された引数を変数として代入

```
python3  SelectGeneID.py   GeneID.txt   3
```

sys.argv[0]　sys.argv[1]　sys.argv[2]

- Pythonにおける引数は「文字列」としてsys.argvに入ります。
- 数字（整数）として変数に代入したい場合は、int（）により「文字列」から「整数」（=integer）に変換します。
- 同じ”3”でも「文字列」、「整数」（int）、「小数」（float）によりプログラムの動き方が変わるため、どの「型」であるかを区別することが大切です。

③ Python入門実習

```
with open(input_file) as fh:
    ....(命令文)
```

入力ファイルをファイルハンドル”fh”として開く

GeneID.txt

1	A1BG
2	A2M
3	A2MP1
9	NAT1
10	NAT2
11	AACP
12	SERPINA3
13	AADAC
14	AAMP
15	AANAT
16	AARS
17	AAVS1
	⋮

ファイルハンドル
“fh”

- Pythonでは、データを読み込む目的や、データを書き出す目的で開いたファイルを、ファイルハンドルという形で取り扱うことができます。
- ファイルハンドル名は自分で決められます。（file handleからfhとした）
- 命令文”with open () ”を使って、入力ファイル”input_file”（=“GeneID.txt”）をファイルハンドル”fh”として開いています。
- 行末に”:”をつけ、命令文はスペース4個でインデント（字下げ）します。

③ Python入門実習

```
for line in fh:
    ▫▫▫▫（命令文）
```

fhを一行ずつ読み込む

GeneID.txt

1	A1BG
2	A2M
3	A2MP1
9	NAT1
10	NAT2
11	AACP
12	SERPINA3
13	AADAC
14	AAMP
15	AANAT
16	AARS
17	AAVS1
	⋮

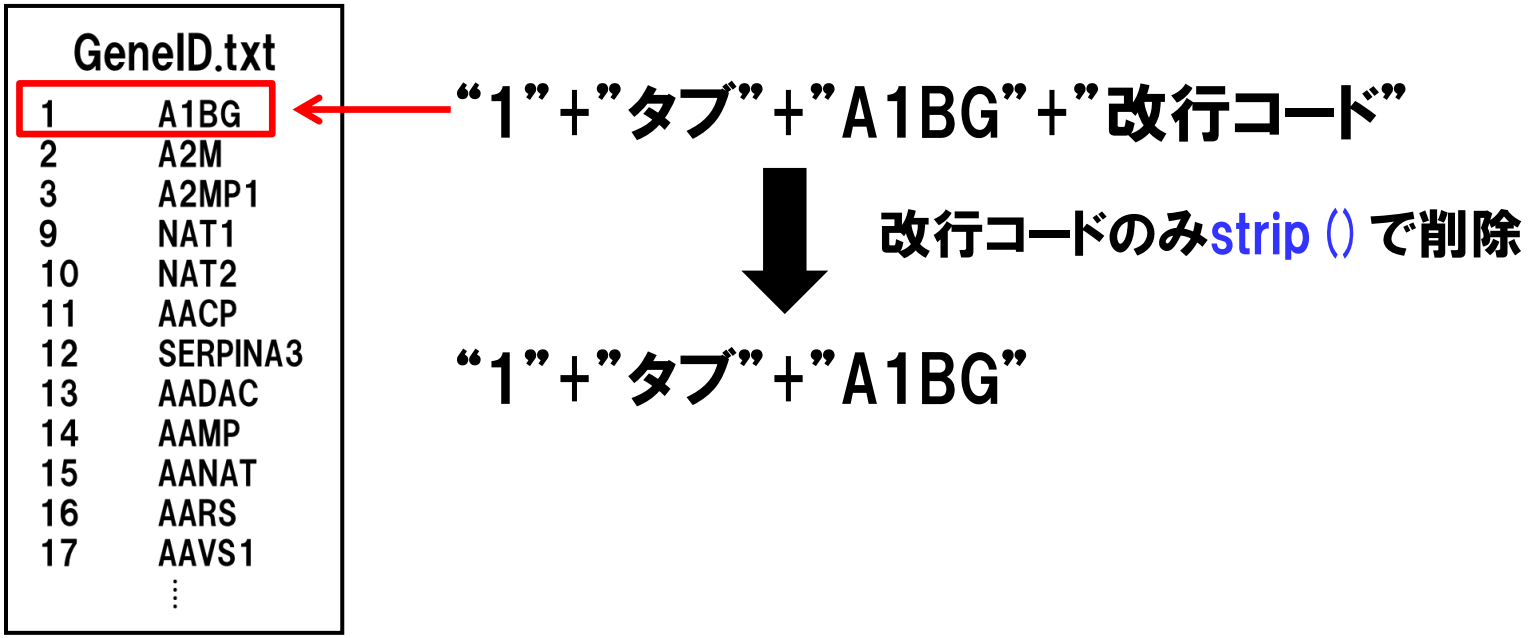
“1 A1BG”を読みこんで命令文を実行
↓
“2 A2M”を読みこんで命令文を実行
↓
“3 A2MP1”を読みこんで命令文を実行
↓
“9 NAT1”を読みこんで命令文を実行
↓
⋮

- 命令文”for line in fh”は、ファイルハンドルを、第一行目から最終行まで、一行ずつ読み込んで、続く命令文で指定された処理を行います。
- for文の行末は”:”で終わります。
- 続く命令文はスペース4個でインデントします。

③ Python入門実習

```
strings = line.strip().split()
```

読み込み行の末尾の改行コードを削除し、タブ区切りで分割



- テキストファイルの各行の最後には、改行を示す文字である改行コードが書き込まれています。
  （その性質上、通常のテキストエディタでは改行コードそのものは表示されません）。
- 読み込んだ行に改行コードが残っていると、後々の処理の邪魔になるので、命令文”strip () ”を使って、改行コードのみ削除します。

③ Python入門実習

```
strings = line.strip().split()
```

読み込み行の末尾の改行コードを削除し、タブ区切りで分割

GeneID.txt

1	A1BG
2	A2M
3	A2MP1
9	NAT1
10	NAT2
11	AACP
12	SERPINA3
13	AADAC
14	AAMP
15	AANAT
16	AARS
17	AAVS1
⋮	

“1”+”タブ”+”A1BG”+”改行コード”

↓ 改行コードのみstrip () で削除

“1”+”タブ”+”A1BG”

↓ split () で区切り文字(=タブ=¥t)で分割し、新しい配列stringsへ代入

“1” ”A1BG” → 配列strings

- 命令文”split () ”を使うと、文字列が代入された変数を、指定した区切り文字で分割して、配列に変換することができます。
- タブは”¥t”、半角スペースは”¥s”、改行コードは”¥n”など、 ”¥”を付けることにより特殊文字であることを表記する、ことになっています。

③ Python入門実習

```
if(int(strings[0]) % number == 0):
```

条件「第一列を引数で割った余りが0かどうか」を確認

□□□□（命令文）

算術演算子	意味
和	+
差	-
積	*
商	/
余	%
累乗	**

比較演算子	意味
==	等しい
!=	等しくない
<	～より小さい
>	～より大きい
<=	～以上
>=	～以下

論理演算子	意味
and	AND
or	OR

- 命令文”if ()”を使うと、”()”内に入力された条件式が真の場合、続く命令文で指定された処理を実施します。
- if文の行末には”:”をつけ、続く命令文はスペース4個でインデントします。

③ Python入門実習

```
print(strings[0]+"¥t"+strings[1])
```

標準出力画面へ出力

・命令文”print () ”を使うと、後に続く文字列を標準出力に表示します。
・配列”XXX”の要素を取り出す際は、”XXX [] ”として、” [] ”内に要素の順番を書きます（要素の順番は0から開始されます）。
・文字を””でかこむと、文字列として扱うことが出来ます。
・文字列を連結する際には”+”を間に挟みます。

※Rの時と異なります。

③ Python入門実習

```
import sys
```

sysモジュールをインポート

```
input_file = sys.argv[1]
number   = int(sys.argv[2])
```

入力された引数を変数として代入

```
with open(input_file) as fh:
    for line in fh:
        strings = line.strip().split()
```

入力ファイルをファイルハンドル”fh”として開く

fhを一行ずつ読み込む

読み込み行の末尾の改行コードを削除し、タブ区切りで分割

```
        if(int(strings[0]) % number ==0):
            print(strings[0]+"¥t"+strings[1])
```

「第一列を引数で割った余りが0かどうか」を確認

標準出力画面へ出力

※ソースコードは一見複雑そうに見えますが、順を追って読み込んでいくことで理解できます。

③ Python入門実習

```
statgen@statgen-PC: /mnt/c/SummerSchool/Programming
$  python SelectGeneID.py GeneID.txt 5000
5000     ORC4
10000    AKT3
30000    TNPO2
55000    TUG1
80000    GREB1L
150000 ABCC13
```

・引数には、色々な値を入力することができます。

③ Python入門実習

```
statgen@statgen-PC: /mnt/c/SummerSchool/Programming
$ python3 SelectGene.py GeneID.txt 0
Traceback (most recent call last):
  File "SelectGene.py", line 10, in <module>
    if(int(strings[0]) % number ==0):
ZeroDivisionError: integer division or modulo by zero
```

数字を0で割ることはできない、というエラー

```
statgen@statgen-PC: /mnt/c/SummerSchool/Programming
$ python3 SelectGene.py GeneID_XXX.txt 3
Traceback (most recent call last):
  File "SelectGene.py", line 7, in <module>
    with open(input_file) as fh:
IOError: [Errno 2] No such file or directory: 'GeneID_XXX.txt'
```

指定された入力ファイルが見つからなかった、というエラー

- **ソースコードを正しく実行できない場合、エラーメッセージとエラー箇所が通知されます。**

講義の概要

プログラミング入門

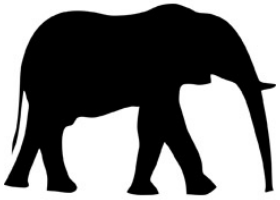
① プログラミングについて

② プログラミング言語の比較

③ Python入門実習

④ AWK入門実習

④ AWK入門実習



Linux
コマンド ⟷ AWK ⟷ プログラミング言語

- AWKはLinuxコマンドの１つですが、プログラミング言語としての側面も持ちます。
- スペースやタブで区切られたテキストファイルの行単位の処理に優れているため、ゲノムデータ解析の分野では重宝されます。

④ AWK入門実習

python3 SelectGeneID.py　GeneID.txt　3

で実施した処理内容は、AWKを使うと、

awk '$1%3==0 {print $1"¥t"$2}' GeneID.txt

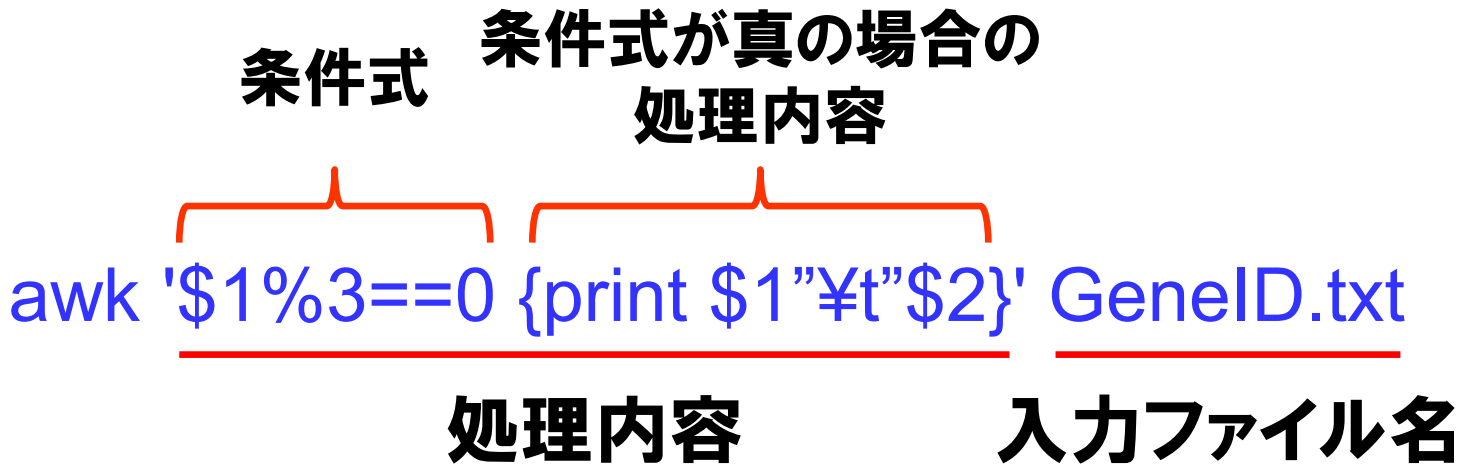
もしくは

awk '$1%3==0 {print $0}' GeneID.txt

と書き込むだけで実行可能です。

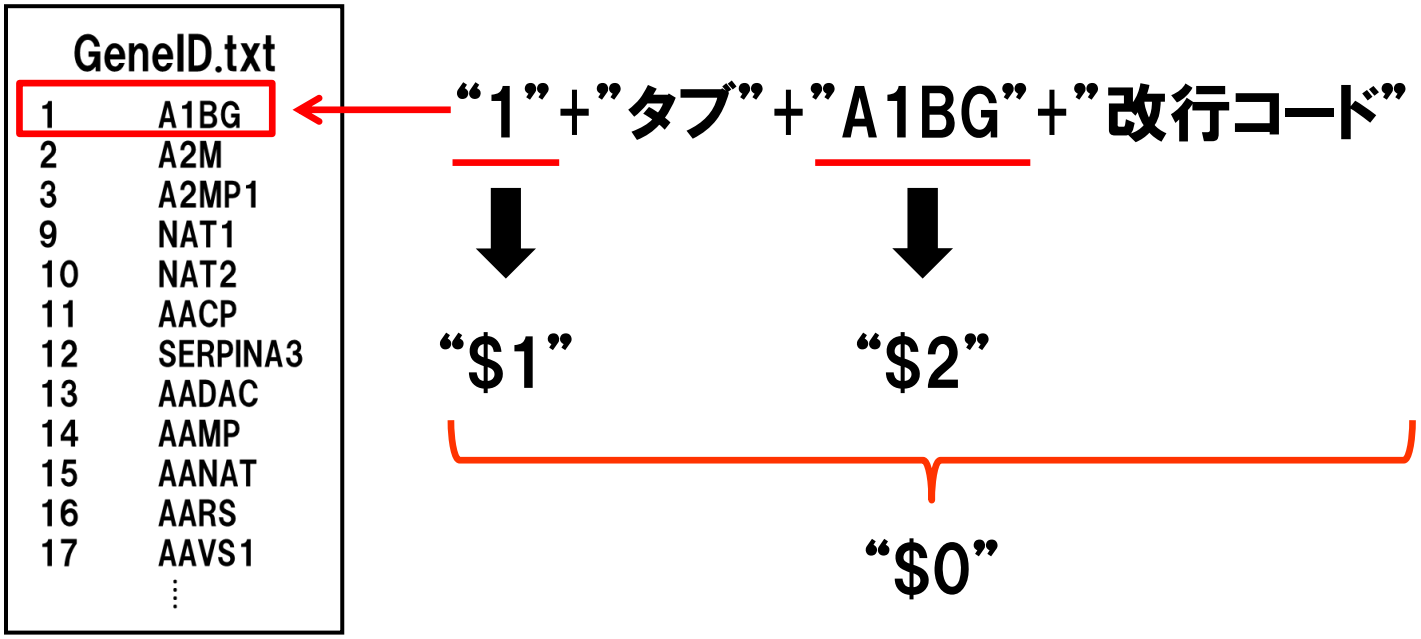
- **AWKを使うと、いちいちソースコードファイルを作成・実行しなくても、Linux上にAWKコマンドを直接書き込むだけで処理を実行可能です。**
- **前項の"SelectGeneID.py"で実施した処理内容は、AWKコマンド一行で代替可能です。**

④ AWK入門実習



- AWKコマンドは、"awk '(処理内容)' 入力ファイル名"という構成です。
- AWKは入力ファイルを一行ずつ読み込んで処理します。
- 上記の場合、「処理内容の前半で条件式」、「後半で真の場合の処理内容」を示しています。

④ AWK入門実習




```
awk '$1%3==0 {print $1”¥t”$2}' GeneID.txt
```

- 行の内容は区切り文字(通常はタブ)で分割され、”$1”、”$2”、のように、順番で指定された変数として扱われます。
- 行内容全部を示すときは、”$0”を使います。
- Pythonと異なり、変数と文字列を連結するとき、間の文字は不要です。

④ AWK入門実習

```
statgen@statgen-PC: /mnt/c/SummerSchool/Programming
$ awk '$1>1000 && /ABCD/ {print $2}' GeneID.txt
```

ABCD3
ABCD4
ABCD1P1
ABCD1P4
ABCD1P3
ABCD1P2
ABCD1P5

遺伝子番号が1000より大きく、遺伝子名に文字列”ABCD”が含まれる場合に、

遺伝子名のみ表示する。

・AWKコマンドを使うことにより、様々な処理を行う事ができます。

④ AWK入門実習

```
statgen@statgen-PC: /mnt/c/SummerSchool/Programming
$ awk '{if ($1<1000 && /ABCD/) print $2; else if ($1<2000 && /DEF/) print $2;
else if (/GHI/) print $2}' GeneID.txt
```

ABCD1
ABCD2
DEFA1
DEFA3
DEFA4
DEFA5
DEFA6
DEFB1
DEFB4A
GHITM

条件文1

条件文2（条件文1が真でないときに検討）

条件文3（条件文1および2が真でないときに検討）

- **AWKコマンドでは、if-else文の繰り返し使用が可能です。**

④ AWK入門実習

長いコマンドを正確に入力するのが大変です・・・

コマンドの先頭から一気に入力する必要はありません。
コマンドを部品に分割し、部品毎に入力するといいです。

- awkコマンドや、パイプ(”|”)で複数のlinuxコマンドを連結した場合、コマンドがとても長くなることがあります。
- 長いコマンドを先頭から一気に入力する必要はありません。コマンドを複数の部品に分割し、部品毎に逐次入力していくのがおすすめです。

④ AWK入門実習

```
$ awk '{}'
        ↓
$ awk ‘{}’ GeneID.txt
        ↓
$ awk '{print $2}' GeneID.txt
        ↓
$ awk '$1>1000 {print $2}' GeneID.txt
        ↓
$ awk '$1>1000 && /ABCD/ {print $2}' GeneID.txt
        ↓
        ⋮
        ↓
$ awk '{if ($1>1000 && /ABCD/) print $2; else if ($1<2000 && /DEF/) print $2;
else if (/GHI/) print $2}' GeneID.txt
```

- **長いコマンドを先頭から一気に入力する必要はありません。コマンドを複数の部品に分割し、部品毎に逐次入力していくのがおすすめです。**

終わりに

・プログラミング言語の簡単な説明と使い方を、なぞってみました。

・最初は、多数のプログラム言語を使いこなせなくても問題ありません。

・何か１つ、覚えやすい言語を一通り習得してみるといいでしょう。入門書に加え、web上の初心者用学習リソースも充実しています。

・既に作成されたソースコードを、お手本として読み込むのも有効です。

・プログラミング自体は決して特別な作業ではなく、誰もが使いこなせるツールの１つにすぎないことがわかってもらえると幸いです。