

# Unix/Linuxの仕組み

- OS/CUIとGUI/ターミナル/コマンドとプログラム
- ディレクトリ/主要なディレクトリ/パスの概念（絶対パス/相対パス）
- コマンドと引数
- 基本的なコマンド `pwd/mkdir/cd/ls/less/rm ...`

1

## OS

OS(Operating System)とは、コンピュータ（個人向けに限らずサーバなども含めたコンピュータ全般）を動かすのに必要なソフトウェアのことです。

「Windows」や「Mac OS」、「Linux」というのは、OSの種類を表しています。

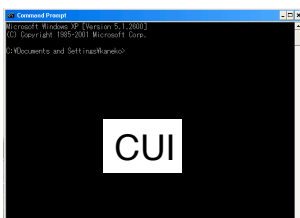
\*Mac OS Xは、UNIX(FreeBSD)がベースになっています。

## CUIとGUI

OSにはたくさんの種類がありますが、それらは大きく2つに分類できます。

UNIXやLinuxなどキーボードを使ったコマンド入力(コンピュータと直接やりとりをするための命令文)によって操作するCUI (Character User Interface) 環境が中心となったものと、

WindowsやMacなどマウスを使ってファイルやフォルダを直接操作するGUI (Graphical User Interface) 環境です。



2

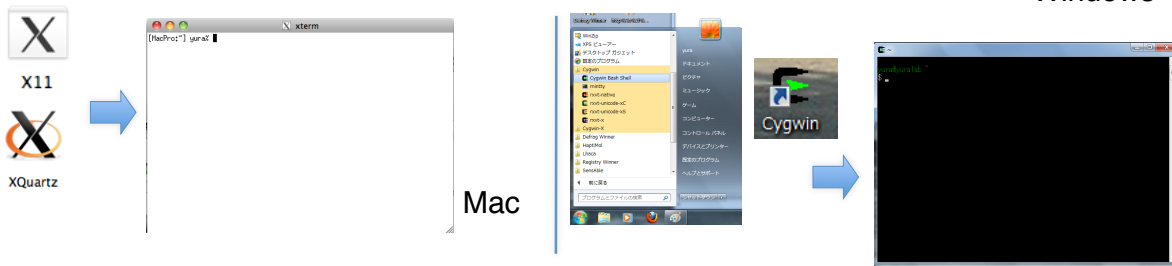
# ターミナル

ターミナルとはコマンドを入れる場所のことです。

Macではアプリケーション「X11」を通じて、OSに命令を送ります。

Windowsではアプリケーション「cygwin」を通じて、OSに命令を送ります。

Windows



\$または%の後の四角(カーソル)に、コマンドを入力して使います。

## コマンドとプログラム

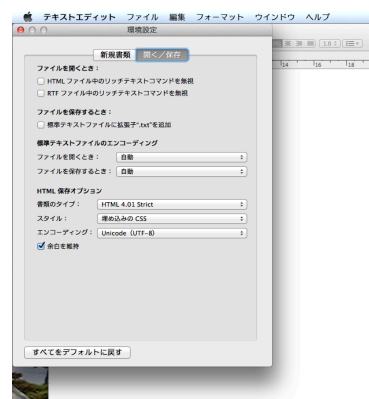
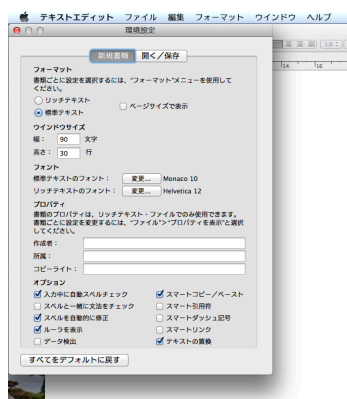
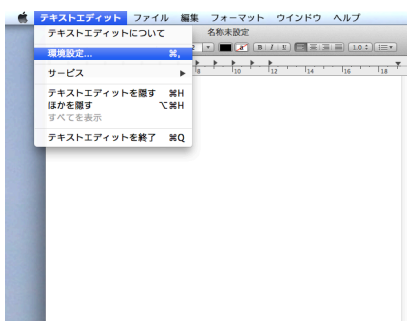
何か処理を実行したい場合、その作業について手順を示したものをプログラムといい、そのファイル名がコマンドです。プログラミングは、プログラム用の言語(Perl, Python, C, Javaなど)を用いて行い、各プログラムにより実行する際の約束事が異なります。

この講義の中では、「コマンド」/「プログラム」を実行するという事は、「Xウインドウ」に指令(例えば、echo \$PATH)を送り、実行するということを指します。

3

# Macのテキストエディットをちゃんと使えるようにする。

- 1) テキストエディットを立ち上げて、環境設定を開く
- 2) 「新規書類」の「フォーマット」を「標準テキスト」にする。
- 3) 「開く/保存」の「ファイルを保存するとき」のチェックをはずす。

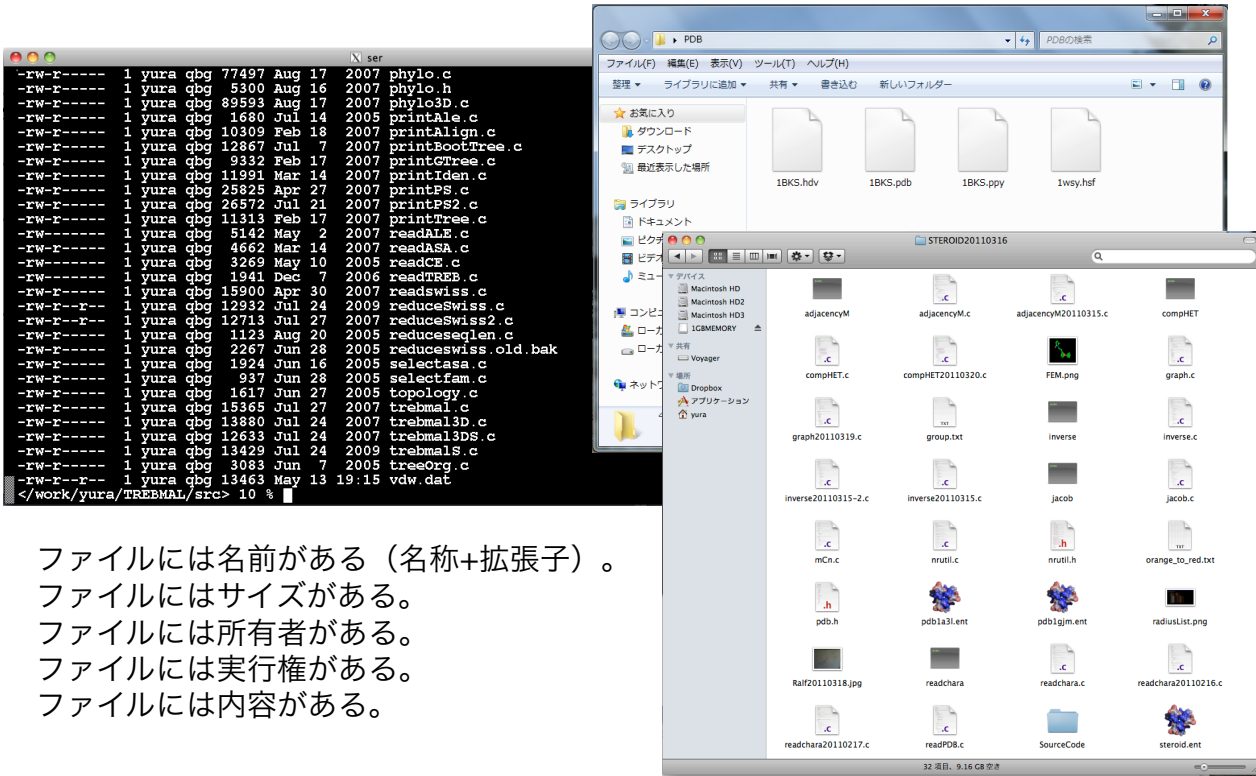


これらを設定したら、テキストエディットを終了し、再び立ち上げ利用開始。

4

# ファイル

コンピュータでは、いかなる情報も「ファイル」に書き、「ファイル」単位で保存します。



The image shows a terminal window on the left with a file listing command and its output. The output lists files with their permissions, owner, group, size, date, and filename. The file explorer window on the right shows a directory structure with various files and folders.

```
ln ser
-rw-r----- 1 yura qbg 77487 Aug 17 2007 phylo.c
-rw-r----- 1 yura qbg 5300 Aug 16 2007 phylo.h
-rw-r----- 1 yura qbg 89593 Aug 17 2007 phylo3D.c
-rw-r----- 1 yura qbg 1680 Jul 14 2005 printA1a.c
-rw-r----- 1 yura qbg 10309 Feb 18 2007 printAlign.c
-rw-r----- 1 yura qbg 12867 Jul 7 2007 printBootTree.c
-rw-r----- 1 yura qbg 9332 Feb 17 2007 printGTree.c
-rw-r----- 1 yura qbg 11991 Mar 14 2007 printIden.c
-rw-r----- 1 yura qbg 25825 Apr 27 2007 printPS.c
-rw-r----- 1 yura qbg 26572 Jul 21 2007 printPS2.c
-rw-r----- 1 yura qbg 11313 Feb 17 2007 printTree.c
-rw-r----- 1 yura qbg 5142 May 2 2007 readA1E.c
-rw-r----- 1 yura qbg 4662 Mar 14 2007 readA1S.c
-rw-r----- 1 yura qbg 3269 May 10 2005 readCF.c
-rw-r----- 1 yura qbg 1941 Dec 7 2006 readTREE.c
-rw-r----- 1 yura qbg 15900 Apr 30 2007 readswiss.c
-rw-r----- 1 yura qbg 12932 Jul 24 2009 reduceSwiss.c
-rw-r----- 1 yura qbg 12713 Jul 27 2007 reduceSwiss2.c
-rw-r----- 1 yura qbg 1123 Aug 20 2005 reducesq1en.c
-rw-r----- 1 yura qbg 2267 Jun 28 2005 reduceswiss.old.bak
-rw-r----- 1 yura qbg 1924 Jun 16 2005 selectasa.c
-rw-r----- 1 yura qbg 937 Jun 28 2005 selectfam.c
-rw-r----- 1 yura qbg 1617 Jun 27 2005 topology.c
-rw-r----- 1 yura qbg 15365 Jul 27 2007 trebmal.c
-rw-r----- 1 yura qbg 13880 Jul 24 2007 trebmal3D.c
-rw-r----- 1 yura qbg 12633 Jul 24 2007 trebmal3DS.c
-rw-r----- 1 yura qbg 13429 Jun 24 2009 trebmalS.c
-rw-r----- 1 yura qbg 3083 Jun 7 2005 treeOrg.c
-rw-r----- 1 yura qbg 13463 May 13 19:15 vdw.dat
</work/yura/TREBMA1/src> 10 %
```

File Explorer (PDB) contents:

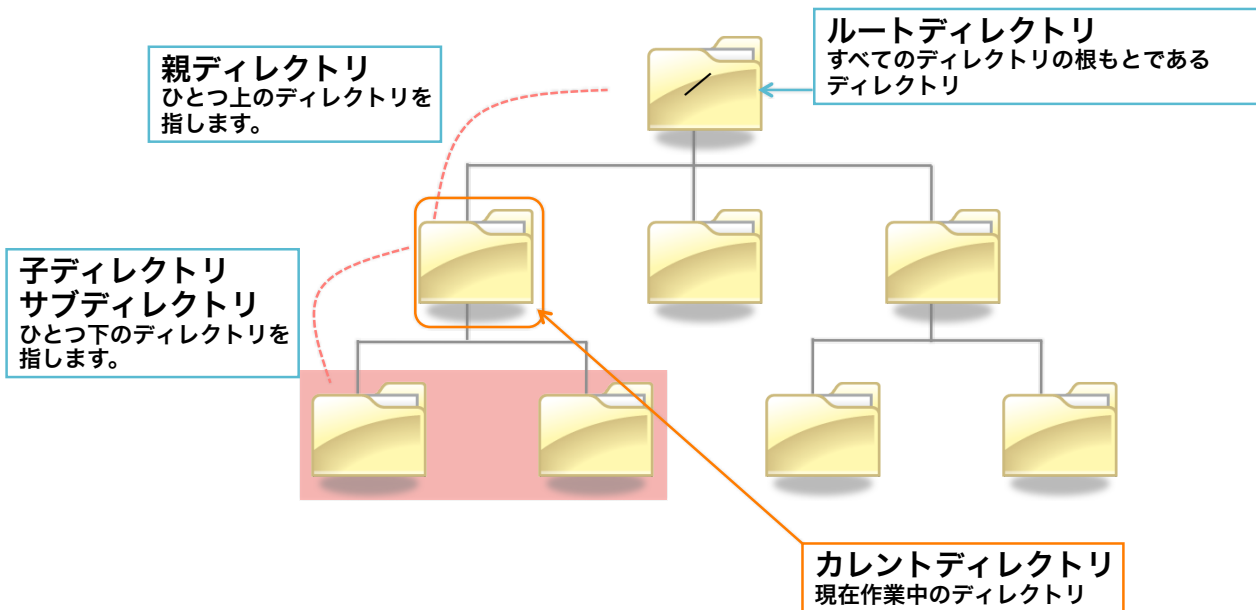
- 1BKS.hdv
- 1BKS.pdb
- 1BKS.ppy
- 1wsy.hsf
- adjacencyM
- adjacencyM.c
- adjacencyM20110315.c
- compHET
- compHET.c
- compHET20110320.c
- FEM.png
- graph.c
- graph20110319.c
- group.txt
- inverse
- inverse.c
- inverse20110315-2.c
- inverse20110315.c
- jacob
- jacob.c
- mCn.c
- nrutil.c
- nrutil.h
- orange\_to\_red.txt
- pdb.h
- pdbla3l.ent
- pdblgjm.ent
- radiusList.png
- RaiF20110318.jpg
- readchara
- readchara.c
- readchara20110216.c
- readchara20110217.c
- readPOB.c
- SourceCode
- steroid.ent

ファイルには名前がある（名称+拡張子）。  
ファイルにはサイズがある。  
ファイルには所有者がある。  
ファイルには実行権がある。  
ファイルには内容がある。

5

# ディレクトリ/フォルダー

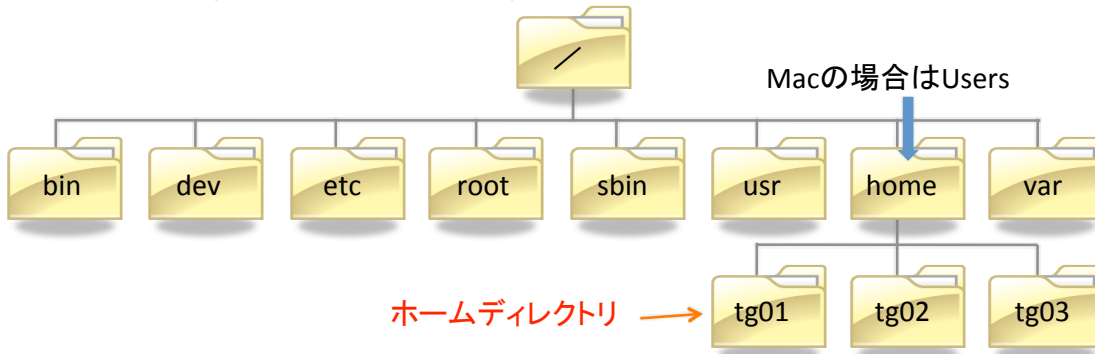
ファイルを保存する場所がディレクトリという単位に分けられています。  
ディレクトリはツリー構造（階層構造）になっています。



6

# 主要なディレクトリ

ホームディレクトリやルートディレクトリ以外にも、よく出てくる代表的なディレクトリをいくつか紹介します。(UNIXを例にしています)



ディレクトリ	役割
bin	バイナリ形式の実行ファイルやコマンドが保管されています。
dev	デバイス関係のファイルが保管されています。
etc	各種設定ファイルなどさまざまな保管されています。
root	ルートディレクトリとは別に用意された、システム管理者用のホームディレクトリです。
sbin	管理者用のシステム標準コマンドが保管されています。
usr	各ユーザーのデータやアプリケーションが保管される場所です。
home (Users)	この下にユーザー毎のディレクトリが作られ、そこが各ユーザーのホームディレクトリになります。
var	アプリケーションの記録 (ログ) ファイルやメールデータなどが保管される場所です。

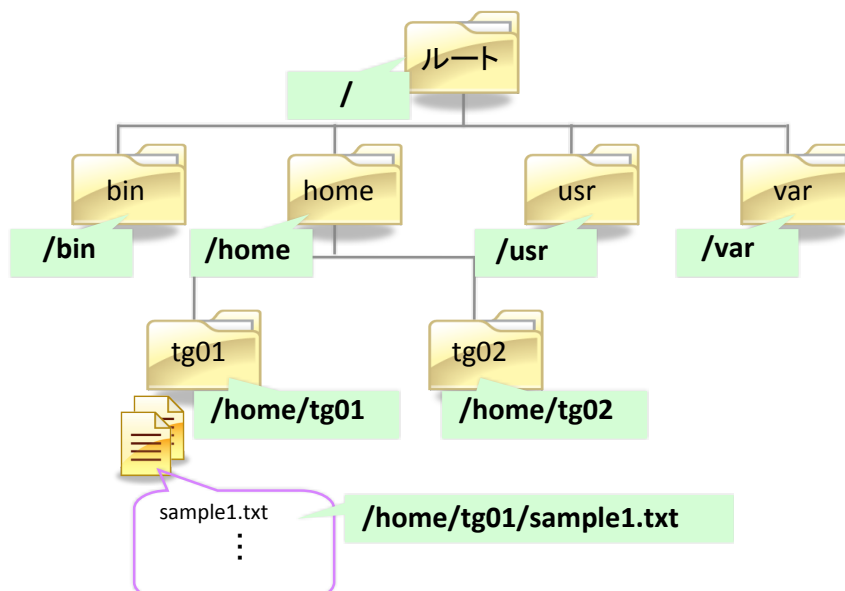
7

# パスの概念

ファイルを開いたり、コマンドを実行したりするには、そのファイルやコマンドの場所を正確に指定する必要があります。この指定方法をパス(PATH)といいます。

## 絶対パス

ルートディレクトリを基点として指定する方法です。この表示方法は、カレントディレクトリが (現在表示しているディレクトリ) どこであっても、間違いなく目的のファイルを指定できます。



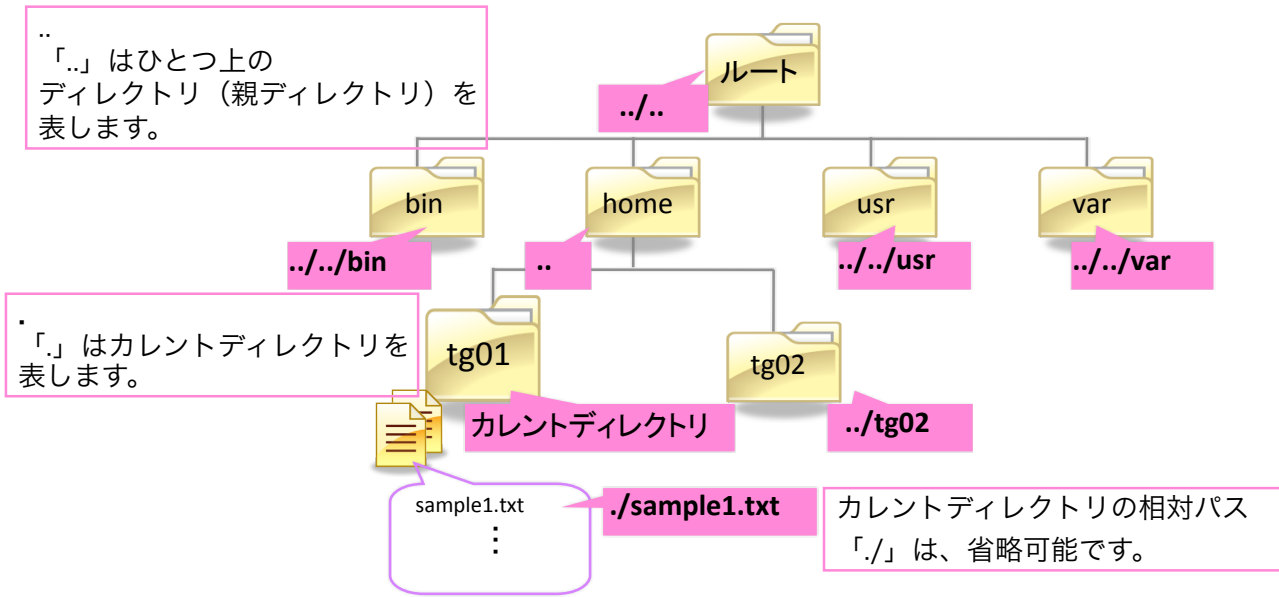
8

# パスの概念 つづき

## 相対パス

カレントディレクトリを基点として指定する方法です。

下の図では、tg01をカレントディレクトリとした場合を例に示します。



9

# コマンド と引数 (Command and Argument)

## コマンドを入力するときの決まり

- ・半角英数字を使用する。
- ・大文字と小文字の違いを正しく入力する。
- ・コマンドと引数の間は半角スペースを空ける。
- ・入力が終わったら[Enter]キーを押す。

## 引数

コマンドの対象になるファイル名やディレクトリ名（パス）などの文字列のことを引数と呼ぶ。

特に、コマンドの挙動を変化させる引数は「オプション」と呼ぶことがある。

多くのオプションは「-」で始まる。

複数のオプションと引数を半角スペースでつなげていくこともできる。

## 例

```
$ man ls          man(manual)はマニュアルを表示するコマンド
                  この場合、lsというコマンドについてのマニュアルを表示。

$ ls -a /home/yura  /home/yuraの中について、「-a」でファイルの先頭に"."が付く
                   ファイルも含めて表示。
```

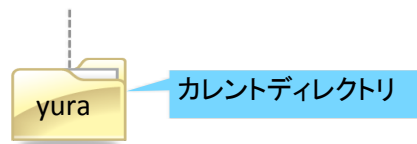
10

# 基本的なコマンド (1)

**pwd** (Print Work Directory)

カレントディレクトリの絶対パスを表示できます。

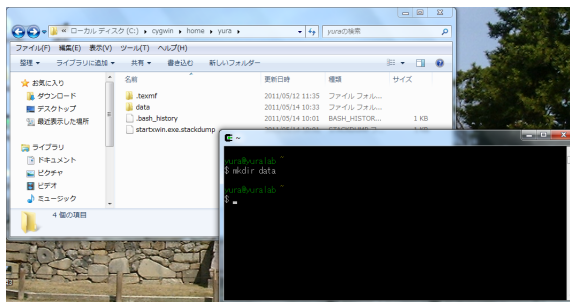
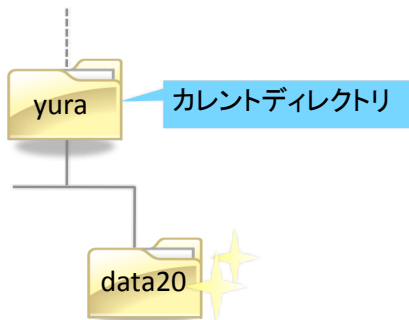
```
$ pwd  
[結果 /home/yura]
```



**mkdir** (MaKe DIRectory)

新しくディレクトリを作ります。

```
$ mkdir data20  
$ ls
```



11

# 基本的なコマンド (2)

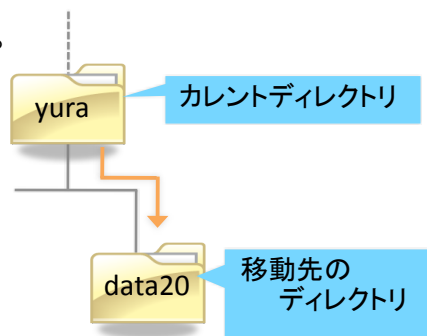
**cd** (Change Directory)

カレントディレクトリを変更するためのコマンドです。

移動したいディレクトリ名をコマンドの後ろに指定します。

```
$ cd ディレクトリ名  
[絶対パスの場合 $cd /home/yura/data20  
相対パスの場合 $cd data]
```

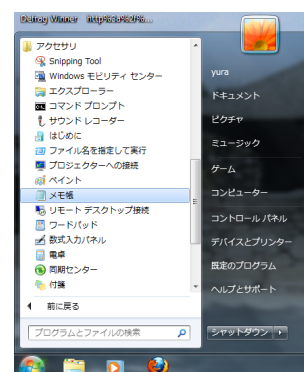
ディレクトリを指定しないで cd と入力すると、どこにいてもホームディレクトリに戻ります。



テキストエディット/メモ帳を起動して、自分の名前を書いて、「name.txt」という名前で保存します。そのファイルを「data20」ディレクトリに移動して下さい。

次に、X 1 1 /Cygwinで自分の位置を確認して、data20ディレクトリに移動して下さい。

そして、新しく「a」という名前のディレクトリを作成して下さい。

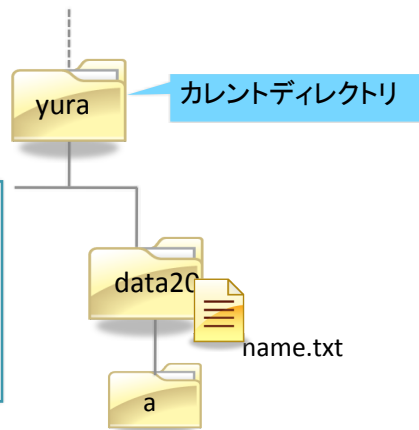


12

## 基本的なコマンド (3)

ls (LiSt directory)  
ディレクトリの情報を調べるコマンドです。

```
$ ls ディレクトリ名  
[$ls data20  
結果  
a name.txt]  
$ls -a 全てのファイルを見ることができます。
```



less / more / cat / head / tail  
テキストファイルを開覧するコマンドです。

```
$ less ファイル名  
[$less name.txt  
結果  
(自分の名前) ]
```

\*qを押すと、元の画面に戻ります。

13

## 基本的なコマンド (4)

rm (ReMove file)  
ファイルを削除するときに使います。

```
$ rm ファイル名  
[$rm name.txt]
```

\$ls で、name.txtが無くなっていることを確認して下さい。

ディレクトリを削除するときは

```
$ rm -rf ディレクトリ名
```

-rfというオプションはディレクトリ内にファイルがあっても強制的に消す。  
使うときは要注意！！

\$ls -a でdataディレクトリに何も無いことを確認して下さい。

\*UNIXには、「ゴミ箱」はないため、  
一度削除したら元には戻せないなので十分注意して下さい。

14

## 基本的なコマンド (5)

### find

ファイルをその名称で検索するときに使います。

```
$ find . -name ファイル名 -print  
[$find . -name name.txt -print]
```

カレントディレクトリー(.)以下のすべてのディレクトリを探します。

### grep

ファイルの中の特定の文字列を検索するときに使います。

```
$ grep '文字列' ファイル名  
[$grep 'abcdef' name.txt]
```

15

## 基本的なコマンドの演習

- 1) カレントディレクトリを確認して
- 2) 「bin」という名前のディレクトリをホームディレクトリに作成
- 3) ディレクトリが新しくできていることを確認
- 4) 「bin」へ移動
- 5) ホームディレクトリへ戻る

「bin」ディレクトリは次回以降使います。

\* 「tab」キーでコマンドやファイル名を補完することができます。

例：カレントディレクトリに「aabbccdd」と「bbccddee」というファイルがある場合  
ls aa[tab]とするとls aabbccdd となります。（aaに一致しているものを呼び出す）

\* キーボードのpage ▲, page ▼を押していくと前に入力したコマンドが表示されます。

16



# プログラミング言語Pythonの基礎

- ・ 規則が一番少ないプログラミング言語
- ・ 生命情報解析用のツール（biopython, bioconductorなど）が充実している言語
- ・ ヨーロッパで広く用いられている言語



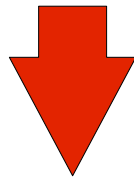
[www.python.org](http://www.python.org)



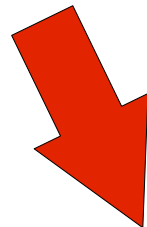
[biopython.org](http://biopython.org)

17

## プログラムを作成する

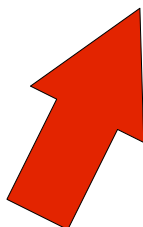


問題の解き方をよく検討する



ソースコードを

ファイルに書く



ファイルを

実行する

18

# Pythonにおけるお約束ごと

## ・ソースコードをファイルに書く

第1行目に

```
#!/usr/bin/env python  
と書く。
```

第2行目は空行にする。

第3行目と第4行目に

```
from sys import *  
from math import *  
と書く
```

第6行目以降にプログラムをかき始める。

ファイル名は、○○○.py

## ・ファイルを実行する

ファイルに実行権を与えること (ファイルを作った際に1回だけ行えばよい)

```
% chmod 700 (ファイル名)
```

19

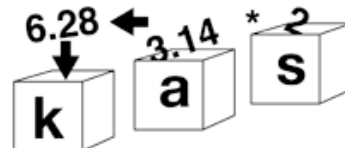
## 変数、四則演算

```
a = 1  
print a
```



```
a = 1  
b = 2  
c = a + b  
print c
```

```
a = [1, 2, 3]  
print a
```



```
MacPro  
[MacPro2: ~/Desktop] yura2%  
[MacPro2: ~/Desktop] yura2%  
[MacPro2: ~/Desktop] yura2%  
[MacPro2: ~/Desktop] yura2% cat practice01.py  
#!/usr/bin/env python  
  
from sys import *  
from math import *  
  
a = 1  
print a  
[MacPro2: ~/Desktop] yura2% ./practice01.py  
1  
[MacPro2: ~/Desktop] yura2%
```

20

## 変数、四則演算

```
#!/usr/bin/env python
```

```
from sys import *  
from math import *
```

```
a = 1
```

```
print a
```

おまじない

変数aに1を代入

変数aを画面に表示する

```
[MacPro2: ~/Desktop] yura2%  
[MacPro2: ~/Desktop] yura2%  
[MacPro2: ~/Desktop] yura2%  
[MacPro2: ~/Desktop] yura2% cat practice01.py  
#!/usr/bin/env python  
from sys import *  
from math import *  
a = 1  
print a  
[MacPro2: ~/Desktop] yura2% ./practice01.py  
1  
[MacPro2: ~/Desktop] yura2%
```

やってみよう!

- 1) 画面に数字の3.14を表示する。
- 2) 画面に塩基配列ATGACAを表示する。

21

## 変数、四則演算

```
#!/usr/bin/env python
```

```
from sys import *  
from math import *
```

```
a = 3.14
```

```
print a * 2.0 # double
```

```
c = a / 2.0 # divide
```

```
print c
```

```
d = a ** 2 # power
```

```
print d
```

```
e = a % 2 # modulo 2
```

```
print e
```

```
a += 3.0 # add three
```

```
print a
```

おまじない

四則演算など

22

## 変数、四則演算

```
#!/usr/bin/env python  
  
from sys import *  
from math import *  
  
a = 'ATGACA'  
print a  
print a[0]  
print a[1]  
print a[2]  
print a[-1]    # last  
print a[1:3]  
print a[0:6:2] # step 2
```



おまじない

何が起こるか？

23

## 変数、四則演算

```
#!/usr/bin/env python  
  
from sys import *  
from math import *  
  
a = 'ATGACA'  
print a  
b = a + a    # duplicate  
print b  
b = a * 3    # triplicate  
print b
```



おまじない

文字の演算？

24

## 変数、四則演算

```
#!/usr/bin/env python

from sys import *
from math import *

a = [9, 4, 8, 3.1, 2, 7.5]
print a
print a[0]
print a[1]
print a[2]
print a[-1]
print a[1:3]
print a[0:6:2]
```



おまじない

数字のセット?

25

## 変数、四則演算

```
#!/usr/bin/env python

from sys import *
from math import *

a = [9, 4, 8, 3.1, 2, 7.5]
print a
b = a + a
print b
b = a * 3
print b
```



おまじない

数字のセット?

aの中に何個の要素があるかをどのようにして調べるか?  
(こたえ) `len(a)`

26

## a[3]に0.6を足したい...

```
#!/usr/bin/env python  
  
from sys import *  
from math import *  
  
a = [9, 4, 8, 3.1, 2, 7.5]  
print a  
a[3] = a[3]+0.6  
print a
```



おまじない

こういうことができるか？

27

## a[3]に0.6を足したい...

まず手始めに、aを縦に表示する。

```
#!/usr/bin/env python  
  
from sys import *  
from math import *  
  
a = [9, 4, 8, 3.1, 2, 7.5]  
  
print 'BEGIN'  
for j in range(0,len(a)):  
    print a[j]  
  
print 'END'
```



おまじない

ループ

28

## a[3]に0.6を足したい...

```
#!/usr/bin/env python

from sys import *
from math import *

a = [9, 4, 8, 3.1, 2, 7.5]

print 'BEGIN'
for j in range(0,len(a)):
    if (j == 3):
        print a[j]+0.6
    else:
        print a[j]

print 'END'
```

↑  
おまじない  
↓

## ループと条件文

29

## a[3]に0.6を足して、 a全体を別の変数bに入りたい

```
#!/usr/bin/env python

from sys import *
from math import *

a = [9, 4, 8, 3.1, 2, 7.5]
b = [ ]
for j in range(0,len(a)):
    if (j == 3):
        b.append(a[j]+0.6)
    else:
        b.append(a[j])

print a
print b
```

↑  
おまじない  
↓

## ループと条件文

等しい	==
異なる	!=
大きい	<
小さい	>

メソッドは来週詳しく勉強します。

30

# リスト([ ])にどのようなメソッドがあるか？

```
#!/usr/bin/env python

from sys import *
from math import *

a = ['AUG','GUU','CCC','UAG']

print a      # original list
a.sort()
print a      # sorted list
a.reverse()
print a      # reversed list
a.insert(3,'GAA')
print a      # GAA inserted list at 2
a.append('CGU')
print a      # CGU added list at the end
a.pop()
print a      # last-element-deleted list
a.remove('GUU')
print a      # GUU-deleted list
print 'GAA = ', a.index('GAA')    # find index for GAA
print 'GUU = ', a.count('GUU')    # count the number of GUU
```

31

## 他のファイルの中身を変数に読み込む

```
#!/usr/bin/env python
```

```
from sys import *
from math import *
```

```
a = ?
```

```
print a
```

おまじない

```
[MacPro2:~/Desktop] yura2% cat test2.py
#!/usr/bin/env python

from sys import *

f = argv[1]
print f

[MacPro2:~/Desktop] yura2% chmod 700 test2.py
[MacPro2:~/Desktop] yura2% ./test2.py cDNA.txt
cDNA.txt
[MacPro2:~/Desktop] yura2%
```

```
cDNA.txt
atgaattctatcatgggttcagggacacacactcacttttcatcaaggg
ctatcatgctactggagtaatgcgtctgcatcaggaccaggatcaactg
ctgacttcccaagcaggaggatgtcaacaacgaggcggattttctcaa
tatacttcaacgaatctctgtgccacggccacaggcgtgctcaaaagcc
ctgtgaagacctggcctgctaactctaccagaccccccttccccaaa
atcaagatttctatagaccagtgcaccccattcgaccccaagtccctcc
ttccaaagctgtcgtgagcagggtctctccagaaagggtctctgccttc
tcagacactttcagaactttctccagcgcactgtgaactggccctg
tcaacaacacacagaccgaagtgcgcacatcagacggccagtgctcac
tccgcagtgaggatgataactggacacggggtctaaacacgttaaa
cgagactttacacagtgaaagacatcgaatggatgagagtgaggagaa
atcagtcctcgtgcagtaagatacaactgggaaatgctcggaccgggag
ctgaagacgaaccacggacgagatctgacgaggacacttcgagtggagg
ctcgcgaactaatttcaacaagaacactcacgaacttgaaggaggt
ttcattcaacaataatctgactcgagccagacggatagagatcgcaaac
cctctccagctgagcgaacacaaagtgaagatctggttccagaacagagc
catgaaacagaagaaatgctcggggaaggcctagctcaaggattaatgc
tgattctggatgtgatgaggactcgaaaaaagtgcacttgttcactc
cctgattaa
```

32



# 他のファイルの中身を変数に読み込む

```
MacPro
[MacPro2:~/Desktop] yura2% cat test2.py
#!/usr/bin/env python
from sys import *
f = argv[1]
print f
[MacPro2:~/Desktop] yura2% chmod 700 test2.py
[MacPro2:~/Desktop] yura2% ./test2.py cDNA.txt
[MacPro2:~/Desktop] yura2%
```

```
#!/usr/bin/env python
```

```
from sys import *
from math import *
```

```
f = open(argv[1])
a = ""
for l in f:
    a += l
f.close()

print a
```

```
#!/usr/bin/env python
```

```
from sys import *
from math import *
```

```
f = open(argv[1])
a = ""
for l in f:
    a += l.strip()
f.close()

print a
```

33

## 練習しよう

```
cDNA.txt
atgaattcttatcatatgggttcagggacacaacgtcacttttcatcaagg
ctatcatgctactggagtaatgcgtctgcatcaggaccacggatcaactg
ctgacttccaagcagggaggatgtcaacaacagggcggattttcttcaa
tatcatttcacgaatctctgtgccagggccacagggcgtgctcaaaagcc
ctgtgaagaccctggcctgtaactctaccagaccccccttccccaaa
atcaagatttctatagaccagtgcacccaatcgacccaagtccctcc
ttccaagctgtcgtgagcagggctctccagaaaggcttctcgcttc
atcagacacttccagaacttctccagcgcactgtgaactcggcctg
tcaacaacacacagaccgaagtgcgcacatacagcggccagtgcgtcac
tccgagtgaggatgataaactggacacgggctctaaacacgttaaa
cgagactttacacagtggaaagacattcgaatggatgagagtgaggagaa
atcagtcctcgtgcagetaagatacaactgggaaaatgctcggaccgggag
ctgaagacgaaccacggacgagattctgacgaggacactcagagtggag
ctcgcgaactaattttacaacgaaacaactcacagaactgaaaaggagt
ttcatttcaacaaatctgactcagccagacggatagagatcgcaaac
cctctccagctgagcgaacacaagtgaagatctggtttcagaacagacg
catgaaacagaagaaatgctgcgggaaggcctagctcaaggattaatgc
tgatttctggatgtgatgaggactcgaaaaaaagtgcacttgttcatct
cctgattaa
```

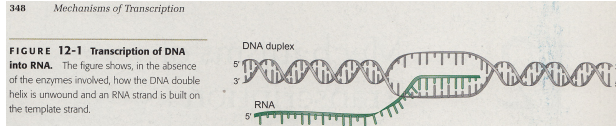
cDNA.txtに格納されているcDNAの中に、ctcとatgとgagは何個あるかを数えるプログラムを作成せよ？

34

# 関数

## ある作業をひとかたまりにする

from Molecular Biology of the Gene, 5th Edition



DNAを読み込んで、長さを測定して、正しいか（変な文字が入っていないか）チェックして、mRNAに変換する。

今までにならった方法で作成してみると...

煩雑（ぐちゃぐちゃ）  
コードを見ても、何を行うプログラムかわからない。

関数を利用して、  
これらの問題を解決する

```
[MacPro2:~/Desktop] yura2% cat RNApol.py
#!/usr/bin/env python

from sys import *
from math import *

f = open(argv[1])
dna = ''
for l in f:
    dna += l.strip()
f.close()

print dna

l = len(dna)
print 'Length = ', l

for i in range(0,l):
    if dna[i] != 'a' and dna[i] != 't' and dna[i] != 'g' and dna[i] != 'c':
        print 'non-standard nucleotide ', dna[i], ' at ', i+1
        exit

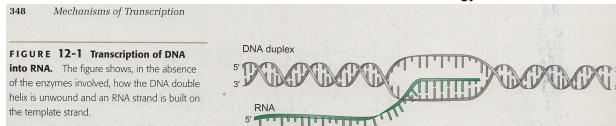
rna = ''
for i in range(1,l+1):
    if dna[-i] == 'a':
        rna += 'u'
    elif dna[-i] == 't':
        rna += 'a'
    elif dna[-i] == 'g':
        rna += 'c'
    elif dna[-i] == 'c':
        rna += 'g'

print rna
[MacPro2:~/Desktop] yura2% █
```

# 関数

## ある作業をひとかたまりにする

from Molecular Biology of the Gene, 5th Edition



```
#!/usr/bin/env python

from sys import *
from math import *

(中略)

#
# main program
#

dna = readin(argv[1])
print 'DNA = ', dna

l = len(dna)
print 'Length = ', l

checkdna(dna)
rna = RNApol(dna)

print rna

#EOF
```

おまじない

コメント文

関数

関数

関数

関数

コメント文

作業を自由にまとめて、  
それぞれを関数にする。

DNAを読み込んで、

長さを測定して、

正しいかチェックして、

mRNAに変換する。

すでに存在する関数を利用する。  
なければ自分で作成する。

# 関数

## 関数を自作する

### 返値がある関数

```
#
# read in DNA
#
def readin(file):
    f = open(file)
    d = ""
    for l in f:
        d += l.strip()
    f.close()
    return d
```

### 返値がない関数

```
#
# check nucleotide in DNA
#
def checkdna(d):
    l = len(d)
    for i in range(0,l):
        c = dna[i].lower()
        if c != 'a' and c != 't' and c != 'g' and c != 'c':
            print 'non-standard nucleotide ', c, ' at ', i+1
    return
```

```
dna = readin(argv[1]) ← 自分で作った関数
l = len(dna) ← 誰が作った関数?
checkdna(dna) ← 自分で作った関数
rna = RNAPol(dna) ← 自分で作った関数
```

37

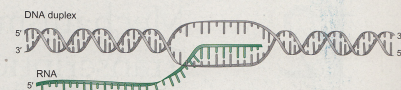
# 関数

## ある作業をひとかたまりにする

from Molecular Biology of the Gene, 5th Edition

348 Mechanisms of Transcription

**FIGURE 12-1 Transcription of DNA into RNA.** The figure shows, in the absence of the enzymes involved, how the DNA double helix is unwound and an RNA strand is built on the template strand.



DNAを読み込んで、長さを測定して、正しいか（変な文字が入っていないか）チェックして、mRNAに変換する。

## 関数を使って作成しよう

関数を使ってプログラムを書くと、

- 自分で作った関数を再利用できる。
- みんなでひとつのプログラム開発を分業できる。  
(引数と返値を決めておきさえすればよい)

38